IrisSupportLib

Version 1.0

Reference Guide

Chapter 1

IrisSupportLib Reference Guide

About this book

This book contains API reference documentation for IrisSupportLib. It was generated from the source code using Doxygen.
The IrisSupportLib library contains the code to create an IrisInstance object and helper classes to add functionality to the instance. It also contains the code to communicate with the Iris system using U64JSON and general support code used by the library, for example thread abstraction.
IrisSupportLib is built as a static library. It must be linked in to any executable or DSO that needs to connect to Iris. The library is provided pre-compiled in $IRIS_HOME/<OS_Compiler>/libIrisSupport.a|IrisSupport.lib. Headers are provided in the directory $IRIS_HOME/include/iris/ and the source code is provided in the directory $IRIS_HOME/↩ IrisSupportLib/.

Other information

For more information about Iris, see the *Iris User Guide*.
See the following locations for examples of Iris clients and plug-ins:

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

Feedback

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

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

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

- The title *IrisSupportLib Reference Guide*.
- The number 101319_0100_13_en.
- If applicable, the relevant page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.
This document includes language that can be offensive. We will replace this language in a future issue of this document.
To report offensive language in this document, email *terms@arm.com*.

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

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

Web Address
http://www.arm.com.

Release Information

Document History			
Issue	**Date**	**Confidentiality**	**Change**

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

Chapter 2

IrisSupportLib NAMESPACE macros

To allow multiple different versions of IrisSupportLib to be used by different components in the same executable, all IrisSupportLib code is defined in a hidden inner namespace. This namespace is constructed from the revision and fork from `iris/detail/IrisSupportLibRevision.h`. For example, if revision=0 and fork=master, this means IrisSupportLib code is in the namespace `iris::r0master`.
This is then imported into the namespace `iris` so all Iris code can be used without the hidden internal namespace. Make sure you include the Iris `NAMESPACE_` macros in any new source files, for example:

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

#include "iris/detail/IrisCommon.h"

NAMESPACE_IRIS_START

// Code goes here

NAMESPACE_IRIS_END

#endif // ARM_INCLUDE_MyHeader_h
```

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

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

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::*)(const ResourceInfo &, ResourceReadResult &) READER, IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) WRITER>
  void iris::IrisInstanceBuilder::setDefaultResourceDelegates (T *instance)

  Set both read and write resource delegates if they are defined in the same class.
- template<IrisErrorCode(*)(const ResourceInfo &, ResourceReadResult &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate ()

  Set default read access function for all subsequently added resources.
- void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate (ResourceReadDelegate delegate=ResourceReadDelegate())

  Set default read access function for all subsequently added resources.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
  void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate (T *instance)

  Set default read access function for all subsequently added resources.
- template<IrisErrorCode(*)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate ()

  Set default write access function for all subsequently added resources.
- void iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate (ResourceWriteDelegate delegate=ResourceWriteDelegate())

  Set default write access function for all subsequently added resources.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) METHOD>
  void iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate (T *instance)

  Set default write access function for all subsequently added resources.
- void iris::IrisInstanceBuilder::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()

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

Add metadata for one noValue resource.
Resource group: beginResourceGroup() must have been called before calling this function. The added resource is automatically added to the last group added by beginResourceGroup().
Type: The added resource is of type 'noValue'. Use addRegister() to add a register of type 'numeric' or 'numericFp'. Use addStringRegister() to add a register of type 'string'.
The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

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

Returns

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

7.2.2.2 addParameter()

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

Add numeric parameter.
Resource group: beginResourceGroup() must have been called before calling this function. The added parameter is automatically added to the last group added by beginResourceGroup().
Type: The added parameter is of type 'numeric'. Call setType("numericFp") on the returned ParameterBuilder to add a 'numericFp' (pure floating point) parameter. Use addStringParameter() to add a parameter of type 'string'.
The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

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

Returns

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

7.2.2.3 addRegister()

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

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

Add metadata for one numeric register resource.
Resource group: beginResourceGroup() must have been called before calling this function. The added resource is automatically added to the last group added by beginResourceGroup().
Type: The added resource is of type 'numeric'. Call setType("numericFp") on the returned RegisterBuilder to add a 'numericFp' (pure floating-point) register. Use addStringRegister() to add a register of type 'string'.
The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

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

Returns

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

Remarks

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

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

To set a caconicalRn of 2**64-1 use

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

7.2.2.4 addStringParameter()

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

Add string parameter.
Resource group: beginResourceGroup() must have been called before calling this function. The added parameter is automatically added to the last group added by beginResourceGroup().
Type: The added parameter is of type 'string'. Use addParameter() to add a parameter of a type 'numeric' or 'numericFp'.
The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

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

Returns

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

7.2.2.5 addStringRegister()

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

Add metadata for one string register resource.
Resource group: beginResourceGroup() must have been called before calling this function. The added resource is automatically added to the last group added by beginResourceGroup().
Type: The added resource is of type 'string'. Use addRegister() to add a register of type 'numeric'.
The returned builder object is only valid until another resource is added. It is only intended to modify the resource that was added last.

Parameters

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

Returns

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

7.2.2.6 beginResourceGroup()

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

Begin a new resource group.
This has the following effects:

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

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

Parameters

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

See also

addParameter
addStringParameter
addRegister
addStringRegister
addNoValueRegister

7.2.2.7 enhanceParameter()

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

Get ParameterBuilder to enhance a parameter.
This function can be used to add/set meta info to an existing parameter. There is no strong use case for this function as all meta info can be set/added by using chained calls to the set...()/add...() functions directly after adding the parameter.
Usage: irisInstance.getBuilder().enhanceParameter(rscId).setFoo(...).setBar(...);
The returned builder object is only valid until another resource is added. It is only intended to modify the specified resource and to add fields to this resource.

Parameters

rsc↩ Id	ResourceId of the parameter which is to be modified.

Returns

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

7.2.2.8 enhanceRegister()

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

Get RegisterBuilder to enhance register.
This function can be used to add sub-fields to register fields which is not possible in a chained call. The rscId can be retreieved 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

rsc↩ Id	ResourceId of the resource which is to be modified or to which fields are to be added.

Returns

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

7.2.2.9 getResourceInfo()

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

Get ResourceInfo of a previously added register.
The returned reference will only be valid until more resources are added.

Parameters

rsc↩ Id	Resource Id of the resource.

7.2.2.10 setDefaultResourceDelegates()

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

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

See also

setDefaultResourceReadDelegate

setDefaultResourceWriteDelegate

Template Parameters

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

Parameters

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

7.2.2.11 setDefaultResourceReadDelegate() [1/3]

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

Set default read access function for all subsequently added resources.
Resources that do not explicitly override the access function using
`addRegister(...).setReadDelegate(...)`
will use this delegate.
Usage: Pass in a global function to delegate resource reading to that function:

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

Template Parameters

FUNC	A function which is a resource read delegate.

7.2.2.12 setDefaultResourceReadDelegate() [2/3]

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

Set default read access function for all subsequently added resources.
Resources that do not explicitly override the access function using
`addRegister(...).setReadDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all resources.
Usage: Pass an instance of ResourceReadDelegate into this function to delegate reading to any class T:

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

Parameters

delegate	Delegate object which will be called to read resources.

7.2.2.13 setDefaultResourceReadDelegate() [3/3]

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

Set default read access function for all subsequently added resources.

Resources that do not explicitly override the access function using `addRegister(...).setReadDelegate(...)` will use this delegate.

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

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

Template Parameters

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

Parameters

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

7.2.2.14 setDefaultResourceWriteDelegate() [1/3]

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

Set default write access function for all subsequently added resources.

Resources that do not explicitly override the access function using `addRegister(...).setWriteDelegate(...)` will use this delegate.

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

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

Template Parameters

FUNC	A function that is a resource write delegate.

7.2.2.15 setDefaultResourceWriteDelegate() [2/3]

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

Set default write access function for all subsequently added resources.
Resources that do not explicitly override the access function using
`addRegister(...).setWriteDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all resources.
Usage: Pass an instance of class T where T::METHOD() is a resource write method:

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

Parameters

delegate	Delegate object which will be called to write resources.

7.2.2.16 setDefaultResourceWriteDelegate() [3/3]

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

Set default write access function for all subsequently added resources.
Resources that do not explicitly override the access function using
`addRegister(...).setWriteDelegate(...)`
will use this delegate.
Usage: Pass an instance of class T where T::METHOD() is a resource write method:

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

Template Parameters

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

Parameters

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

7.2.2.17 setNextSubRscId()

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

Set the rscId that will be used for the next resource to be added.
Resources that are added following this call are assigned subRscIds starting at nextSubRscId.

Parameters

nextSubRscId	The subRscId that is used for the next resource to be added.

7.2.2.18 setPropertyCanonicalRnScheme()

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

Set the register.canonicalRnScheme instance property.
This property is visible in the list of properties returned by instance_getProperties().
This property defines the scheme used by the 'canonicalRn' member of the RegisterInfo object. This should be called upon initialization, before other instances have a chance to call instance_getProperties().
When using the function setCanonicalRnElfDwarf() the property is set automatically to "ElfDwarf" and it is not necessary to call this function.
When not calling setCanonicalRn() for any register it is not necessary to call this function. In this case the property will not exist which is ok.
Custom scheme names (other than ElfDwarf) should always be of the form <comnapy-name>.com/<scheme-name> to avoid conflicts.

Parameters

canonicalRnScheme	Name of the canonical register number scheme used by this instance.

7.2.2.19 setTag()

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

Set a tag for a specific resource.

Parameters

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

See also

ResourceBuilder::setTag
RegisterBuilder::setTag

7.3 IrisInstanceBuilder event APIs

Set up event source metadata and event stream delegates.

Classes

- class iris::IrisInstanceBuilder::EventSourceBuilder

  Used to set metadata on an EventSource.

Functions

- EventSourceBuilder iris::IrisInstanceBuilder::addEventSource (const std::string &name, bool isHidden=false)

  Add metadata for an event source.
- EventSourceBuilder iris::IrisInstanceBuilder::addEventSource (const std::string &name, IrisEventEmitterBase &event_emitter, bool isHidden=false)

  Add metadata for an event source that uses an IrisEventEmitter.
- void iris::IrisInstanceBuilder::finalizeRegisterReadEvent ()
- void iris::IrisInstanceBuilder::finalizeRegisterUpdateEvent ()

  Finalize set up of an IrisEventEmitter.
- IrisInstanceEvent * iris::IrisInstanceBuilder::getIrisInstanceEvent ()
- void iris::IrisInstanceBuilder::resetRegisterReadEvent ()

  Reset the active register read event.
- void iris::IrisInstanceBuilder::resetRegisterUpdateEvent ()

  Reset the active register update event.
- template<IrisErrorCode(*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate ()

  Set the delegate that helps to create a new event stream for the simulation-specific event.
- void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate (EventStreamCreateDelegate delegate)

  Set the delegate that helps to create a new event stream for the simulation-specific event.
- template<typename T , IrisErrorCode(T::*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) METHOD>
  void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate (T *instance)

  Set the delegate that helps to create a new event stream for the simulation-specific event.
- EventSourceBuilder iris::IrisInstanceBuilder::setRegisterReadEvent (const std::string &name, const std::string &description=std::string())

  Add a new register read event source.
- EventSourceBuilder iris::IrisInstanceBuilder::setRegisterReadEvent (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)

  Add a new register read event source.
- EventSourceBuilder iris::IrisInstanceBuilder::setRegisterUpdateEvent (const std::string &name, const std::string &description=std::string())

  Add a new register update event source.
- EventSourceBuilder iris::IrisInstanceBuilder::setRegisterUpdateEvent (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)

  Add a new register update event source.

7.3.1 Detailed Description

Set up event source metadata and event stream delegates.

7.3.2 Function Documentation

7.3.2.1 addEventSource() [1/2]

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

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

Parameters

name	The name of the new event source.
isHidden	If true, the event source is hidden.

See also

EventSourceBuilder::setHidden

Returns

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

7.3.2.2 addEventSource() [2/2]

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

Add metadata for an event source that uses an IrisEventEmitter.

Parameters

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

See also

EventSourceBuilder::setHidden

Returns

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

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 finalizeRegisterUpdateEvent() 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 resetRegisterReadEvent()

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

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

7.3.2.7 resetRegisterUpdateEvent()

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

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

7.3.2.8 setDefaultEsCreateDelegate() [1/3]

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

Set the delegate that helps to create a new event stream for the simulation-specific event.
Consider using addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter) instead. Only use this if you want to implement a non-trivial trace source with its own event emitter handling.
Event sources that do not explicitly override the access function using
`addEventSource(...).setEventStreamCreateDelegate(...)`
use this delegate.
Usage: Pass in a global function to which to delegate event stream creation:

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

Template Parameters

FUNC	Global function to which to delegate event stream creation.

7.3.2.9 setDefaultEsCreateDelegate() [2/3]

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

Set the delegate that helps to create a new event stream for the simulation-specific event.
Consider using addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter) instead. Only use this if you want to implement a non-trivial trace source with its own event emitter handling.
Event sources that do not explicitly override the access function using
`addEventSource(...).setEventStreamCreateDelegate(...)`
use this delegate.
Usage: Pass an instance of class T where T::METHOD() is an event stream creation method:

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

Parameters

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

7.3.2.10 setDefaultEsCreateDelegate() [3/3]

```
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.
Consider using addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter) instead. Only use this if you want to implement a non-trivial trace source with its own event emitter handling.
Event sources that do not explicitly override the access function using
`addEventSource(...).setEventStreamCreateDelegate(...)`
use this delegate.
Usage: Pass an instance of class T where T::METHOD() is an event stream creation method:

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

Template Parameters

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

Parameters

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

7.3.2.11 setRegisterReadEvent() [1/2]

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

Add a new register read event source.
Any registers added after calling setRegisterReadEvent() and before the next call to setRegisterReadEvent() or finalizeRegisterReadEvent() are associated with this event.
A call to setRegisterReadEvent() implicitly calls finalizeRegisterReadEvent() on the event source with name `name` iff an event emitter object (type IrisRegisterEventEmitterBase) is provided as an argument.
If the register read event source already exists (identified by name), the active register read event source simply switches to it.
Register read events have three standard fields:

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

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 setRegisterReadEvent() [2/2]

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

Add a new register read event source.
Any registers added after calling setRegisterReadEvent() and before the next call to setRegisterReadEvent() or finalizeRegisterReadEvent() are associated with this event.
A call to setRegisterReadEvent() implicitly calls finalizeRegisterReadEvent() on the event source with name `name` iff an event emitter object (type IrisRegisterEventEmitterBase) is provided as an argument.
If the register read event source already exists (identified by name), the active register read event source simply switches to it.
Register read events have three standard fields:

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

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.

7.3.2.13 setRegisterUpdateEvent() [1/2]

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

Add a new register update event source.
Any registers added after calling setRegisterUpdateEvent() and before the next call to setRegisterUpdateEvent() or finalizeRegisterUpdateEvent() are associated with this event.
A call to setRegisterUpdateEvent implicitly calls finalizeRegisterUpdateEvent() on the event source with name `name` iff an event emitter object (type IrisRegisterEventEmitterBase) is provided as an argument.
If the register update event source (identified by name) already exists, the active register update event source simply switches to it.
Register update events have four standard fields:

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

Parameters

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.

7.3.2.14 setRegisterUpdateEvent() [2/2]

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

Add a new register update event source.
Any registers added after calling setRegisterUpdateEvent() and before the next call to setRegisterUpdateEvent() or finalizeRegisterUpdateEvent() are associated with this event.
A call to setRegisterUpdateEvent implicitly calls finalizeRegisterUpdateEvent() on the event source with name `name` iff an event emitter object (type IrisRegisterEventEmitterBase) is provided as an argument.
If the register update event source (identified by name) already exists, the active register update event source simply switches to it.
Register update events have four standard fields:

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

Parameters

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.

7.4 IrisInstanceBuilder breakpoint APIs

Set up breakpoint hit notifications and breakpoint delegates.

Functions

- void **iris::IrisInstanceBuilder::addBreakpointCondition** (const std::string &name, const std::string &type, const std::string &description, const std::vector< std::string > bpt_types=std::vector< std::string >())

  Add an optional component-specific condition.
- const BreakpointInfo * iris::IrisInstanceBuilder::getBreakpointInfo (BreakpointId bptId)

  Get the breakpoint information for a given breakpoint.
- void iris::IrisInstanceBuilder::notifyBreakpointHit (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId)

  Notify clients that a code breakpoint was hit.

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

  Notify clients that a data breakpoint was hit (`IRIS_BREAKPOINT_HIT`).
- void iris::IrisInstanceBuilder::notifyBreakpointHitRegister (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId, const std::string &accessRw, const std::vector< uint64_t > &data)

  Notify clients that a register breakpoint was hit (`IRIS_BREAKPOINT_HIT`).
- template<IrisErrorCode(*)(const BreakpointInfo &) FUNC>
  void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate ()

  Set the delegate that is called when a breakpoint is deleted.
- void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate (BreakpointDeleteDelegate delegate)

  Set the delegate that is called when a breakpoint is deleted.
- template<typename T , IrisErrorCode(T::*)(const BreakpointInfo &) METHOD>
  void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate (T *instance)

  Set the delegate that is called when a breakpoint is deleted.
- template<IrisErrorCode(*)(BreakpointInfo &) FUNC>
  void iris::IrisInstanceBuilder::setBreakpointSetDelegate ()

  Set the delegate that is called when a breakpoint is set.
- void iris::IrisInstanceBuilder::setBreakpointSetDelegate (BreakpointSetDelegate delegate)

  Set the delegate that is called when a breakpoint is set.
- template<typename T , IrisErrorCode(T::*)(BreakpointInfo &) METHOD>
  void iris::IrisInstanceBuilder::setBreakpointSetDelegate (T *instance)

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

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()

```
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

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

7.4.2.3 notifyBreakpointHitData()

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

Notify clients that a data breakpoint was hit (IRIS_BREAKPOINT_HIT).
This emits an (IRIS_BREAKPOINT_HIT) event.

Parameters

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

7.4.2.4 notifyBreakpointHitRegister()

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

Notify clients that a register breakpoint was hit (IRIS_BREAKPOINT_HIT).
This emits an (IRIS_BREAKPOINT_HIT) event.

Parameters

bptId	Breakpoint id for the breakpoint that was hit.
time	Simulation time at which the breakpoint was hit.

Parameters

pc	Value of the program counter when the breakpoint was hit.
pc↩ SpaceId	Memory space id for the PC when the breakpoint was hit.
accessRw	Access direction. Should be "r" for a read access or "w" for a write access.
data	The data transferred by the access that hit.

7.4.2.5 setBreakpointDeleteDelegate() [1/3]

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

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

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

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

Template Parameters

FUNC	Global function to call when a breakpoint is deleted.

7.4.2.6 setBreakpointDeleteDelegate() [2/3]

```
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:

```
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.

7.4.2.7 setBreakpointDeleteDelegate() [3/3]

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

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

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

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

Template Parameters

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

Parameters

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

7.4.2.8 setBreakpointSetDelegate() [1/3]

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

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

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

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

Template Parameters

FUNC	Global function to call when a breakpoint is set.

7.4.2.9 setBreakpointSetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setBreakpointSetDelegate (
            BreakpointSetDelegate delegate )  [inline]
```

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

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

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

Parameters

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

7.4.2.10 setBreakpointSetDelegate() [3/3]

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

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

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

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

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

Template Parameters

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

Parameters

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

7.5 IrisInstanceBuilder memory APIs

Set up address translation and memory space metadata and delegates.

Classes

- class iris::IrisInstanceBuilder::AddressTranslationBuilder

  Used to set metadata for an address translation.
- class iris::IrisInstanceBuilder::MemorySpaceBuilder

  Used to set metadata for a memory space.

Functions

- AddressTranslationBuilder iris::IrisInstanceBuilder::addAddressTranslation (MemorySpaceId inSpaceId, MemorySpaceId outSpaceId, const std::string &description)

  Add an address translation.
- MemorySpaceBuilder iris::IrisInstanceBuilder::addMemorySpace (const std::string &name)

  Add metadata for one memory space.
- template<IrisErrorCode(∗)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate ()

  Set the default address translation function for all subsequently added memory spaces.
- void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (MemoryAddressTranslateDelegate delegate=MemoryAddressTranslateDelegate())

  Set the default address translation function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::∗)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) METHOD>
  void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (T ∗instance)

  Set the default address translation function for all subsequently added memory spaces.
- template<IrisErrorCode(∗)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate ()

  Set the default sideband info function for all subsequently added memory spaces.
- void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate (MemoryGetSidebandInfoDelegate delegate)

  Set the default sideband info function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::∗)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) METHOD>
  void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate (T ∗instance)

  Set the default sideband info function for all subsequently added memory spaces.
- template<IrisErrorCode(∗)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate ()

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

- void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate (MemoryReadDelegate delegate=MemoryReadDelegate())

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

- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) METHOD>
  void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate (T *instance)

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

- template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate ()

  Set default write function for all subsequently added memory spaces.

- void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (MemoryWriteDelegate delegate=MemoryWriteDelegate())

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

- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
  void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (T *instance)

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

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

  Set the memory.canonicalMsnScheme instance property.

7.5.1 Detailed Description

Set up address translation and memory space metadata and delegates.

7.5.2 Function Documentation

7.5.2.1 addAddressTranslation()

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

Add an address translation.
Add metadata for the address translation from the memory space indicated by *inSpaceId* to the memory space indicated by *outSpaceId*.
By explicitly adding an address translation using this function, the Iris instance can tell clients which address translations are supported and a component can provide a specific delegate function to perform that translation.

Parameters

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()

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

Add metadata for one memory space.
Typical use pattern:

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

Parameters

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]

```
template<IrisErrorCode(*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &)
FUNC>
void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate ( )  [inline]
```

Set the default address translation function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
`addMemorySpace(...).setTranslationDelegate(...)`
will use this delegate.
Usage:

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

Template Parameters

FUNC	Global function to call to translate addresses.

7.5.2.4 setDefaultAddressTranslateDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (
            MemoryAddressTranslateDelegate delegate = MemoryAddressTranslateDelegate() )
[inline]
```

Set the default address translation function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
`addMemorySpace(...).setTranslationDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

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

Parameters

delegate	Delegate object which will be called to translate addresses.

7.5.2.5 setDefaultAddressTranslateDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslation↩
Result &) METHOD>
void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (
             T * instance )  [inline]
```

Set the default address translation function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
`addMemorySpace(...).setTranslationDelegate(...)`
will use this delegate.
Usage:

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

Template Parameters

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

Parameters

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

7.5.2.6 setDefaultGetMemorySidebandInfoDelegate() [1/3]

```
template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std↩
::vector< std::string > &, IrisValueMap &) FUNC>
void iris::IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate ( )  [inline]
```

Set the default sideband info function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the sideband function using
`addMemorySpace(...).setSidebandDelegate(...)`
will use this delegate.
Usage:

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

Template Parameters

FUNC	Global function to call to get sideband info.

7.5.2.7 setDefaultGetMemorySidebandInfoDelegate() [2/3]

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

Set the default sideband info function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the sideband function using
addMemorySpace(...).setSidebandDelegate(...)
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

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

Parameters

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

7.5.2.8 setDefaultGetMemorySidebandInfoDelegate() [3/3]

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

Set the default sideband info function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the sideband function using
addMemorySpace(...).setSidebandDelegate(...)
will use this delegate.
Usage:

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

Template Parameters

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

Parameters

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

7.5.2.9 setDefaultMemoryReadDelegate() [1/3]

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

Set the default read function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
`addMemorySpace(...).setReadDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩
not_implemented for all requests.
Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

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

Template Parameters

FUNC	A memory read delegate function.

7.5.2.10 setDefaultMemoryReadDelegate() [2/3]

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

Set the default read function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
`addMemorySpace(...).setReadDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩
not_implemented for all requests.
Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

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

Parameters

delegate	Delegate object which will be called to read memory.

7.5.2.11 setDefaultMemoryReadDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64↩
_t, const AttributeValueMap &, MemoryReadResult &) METHOD>
void iris::IrisInstanceBuilder::setDefaultMemoryReadDelegate (
              T * instance )  [inline]
```

Set the default read function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using

addMemorySpace(...).setReadDelegate(...)
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩
not_implemented for all requests.
Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

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

Template Parameters

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

Parameters

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

7.5.2.12 setDefaultMemoryWriteDelegate() [1/3]

```
template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const Attribute↩
ValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>
void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate ( )  [inline]
```

Set default write function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
addMemorySpace(...).setWriteDelegate(...)
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩
not_implemented for all requests.
Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

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

Template Parameters

FUNC	Global function to call to write memory.

7.5.2.13 setDefaultMemoryWriteDelegate() [2/3]

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

Set the default write function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
addMemorySpace(...).setWriteDelegate(...)
will use this delegate.

Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩
not_implemented for all requests.
Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

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

Parameters

delegate	Delegate object which will be called to write memory.

7.5.2.14 setDefaultMemoryWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64↩
_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (
             T * instance )  [inline]
```

Set the default write function for all subsequently added memory spaces.
Memory spaces that do not explicitly override the access function using
`addMemorySpace(...).setWriteDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩
not_implemented for all requests.
Usage: Pass an instance of class T, where T::METHOD() is a memory read method:

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

Template Parameters

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

Parameters

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

7.5.2.15 setPropertyCanonicalMsnScheme()

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

Set the memory.canonicalMsnScheme instance property.
This property is visible in the list of properties returned by instance_getProperties().
This property defines the scheme used by the 'canonicalMsn' member of the MemorySpaceInfo object. The default is 'arm.com/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_get↩Properties().

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

- 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::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::string &) FUNC>
  void iris::IrisInstanceBuilder::setLoadImageFileDelegate ()

  Set the delegate to load an image from a file.
- void iris::IrisInstanceBuilder::setLoadImageFileDelegate (ImageLoadFileDelegate delegate=ImageLoadFileDelegate())

  Set the delegate to load an image from a file.
- template<typename T , IrisErrorCode(T::∗)(const std::string &) METHOD>
  void iris::IrisInstanceBuilder::setLoadImageFileDelegate (T ∗instance)

  Set the delegate to load an image from a file.

7.6.1 Detailed Description

Set up image-loading delegates.

7.6.2 Function Documentation

7.6.2.1 setLoadImageDataDelegate() [1/3]

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

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

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

Template Parameters

FUNC	Global function to call for image loading.

7.6.2.2 setLoadImageDataDelegate() [2/3]

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

Set the delegate to load an image from the data provided.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

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

Parameters

delegate	Delegate object to call for image loading.

7.6.2.3 setLoadImageDataDelegate() [3/3]

```
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:

```
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.

7.6.2.4 setLoadImageFileDelegate() [1/3]

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

Set the delegate to load an image from a file.
Usage:

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

Template Parameters

FUNC	Global function to call for image loading.

7.6.2.5 setLoadImageFileDelegate() [2/3]

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

Set the delegate to load an image from a file.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

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

Parameters

delegate	Delegate object to call for image loading.

7.6.2.6 setLoadImageFileDelegate() [3/3]

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

Set the delegate to load an image from a file.
Usage:

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

Template Parameters

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.

7.7 IrisInstanceBuilder image readData callback APIs.

Open images for reading.

Functions

- uint64_t iris::IrisInstanceBuilder::openImage (const std::string &filename)

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

7.7.1 Detailed Description

Open images for reading.

7.7.2 Function Documentation

7.7.2.1 openImage()

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

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

Parameters

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

- template<IrisErrorCode(*)(uint64_t &, const std::string &) FUNC>
  void iris::IrisInstanceBuilder::setRemainingStepGetDelegate ()

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

  Set the delegate to get the remaining steps for this instance.
- template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>
  void iris::IrisInstanceBuilder::setRemainingStepGetDelegate (T *instance)

  Set the delegate to get the remaining steps for this instance.
- template<IrisErrorCode(*)(uint64_t, const std::string &) FUNC>
  void iris::IrisInstanceBuilder::setRemainingStepSetDelegate ()

  Set the delegate to set the remaining steps for this instance.
- void iris::IrisInstanceBuilder::setRemainingStepSetDelegate (RemainingStepSetDelegate delegate=RemainingStepSetDelegat

  Set the delegate to set the remaining steps for this instance.
- template<typename T , IrisErrorCode(T::*)(uint64_t, const std::string &) METHOD>
  void iris::IrisInstanceBuilder::setRemainingStepSetDelegate (T *instance)

  Set the delegate to set the remaining steps for this instance.
- template<IrisErrorCode(*)(uint64_t &, const std::string &) FUNC>
  void iris::IrisInstanceBuilder::setStepCountGetDelegate ()

  Set the delegate to get the step count for this instance.
- void iris::IrisInstanceBuilder::setStepCountGetDelegate (StepCountGetDelegate delegate=StepCountGetDelegate())

  Set the delegate to get the step count for this instance.
- template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>
  void iris::IrisInstanceBuilder::setStepCountGetDelegate (T *instance)

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

7.8.1 Detailed Description

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

7.8.2 Function Documentation

7.8.2.1 setRemainingStepGetDelegate() [1/3]

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

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

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

Template Parameters

FUNC	Global function to call to get the remaining steps.

7.8.2.2 setRemainingStepGetDelegate() [2/3]

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

Set the delegate to get the remaining steps for this instance.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getRemainingSteps(uint64_t &steps, const std::string &unit);
};
MyClass myInstanceOfMyClass;
iris::RemainingStepGetDelegate 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.

7.8.2.3 setRemainingStepGetDelegate() [3/3]

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

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

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

Template Parameters

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.

7.8.2.4 setRemainingStepSetDelegate() [1/3]

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

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

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

Template Parameters

FUNC	Global function to call to set the remaining steps.

7.8.2.5 setRemainingStepSetDelegate() [2/3]

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

Set the delegate to set the remaining steps for this instance.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

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

Parameters

delegate	Delegate object to call to set the remaining steps.

7.8.2.6 setRemainingStepSetDelegate() [3/3]

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

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

```
class MyClass
{
    ...
```

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

Template Parameters

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.

7.8.2.7 setStepCountGetDelegate() [1/3]

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

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

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

Template Parameters

FUNC	Global function to call to get the step count.

7.8.2.8 setStepCountGetDelegate() [2/3]

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

Set the delegate to get the step count for this instance.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

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

Parameters

delegate	Delegate object to call to get the step count.

7.8.2.9 setStepCountGetDelegate() [3/3]

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

```
            T * instance )  [inline]
```

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

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

Template Parameters

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

Parameters

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

7.9 Disassembler delegate functions

Set disassembler delegates.

Classes

- class iris::IrisInstanceDisassembler

  Disassembler add-on for IrisInstance.

Typedefs

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

  Get the disassembly for an individual opcode.
- typedef IrisDelegate< std::string & > iris::GetCurrentDisassemblyModeDelegate

  Get the current disassembly mode.
- typedef IrisDelegate< uint64_t, const std::string &, MemoryReadResult &, uint64_t, uint64_t, std::vector< DisassemblyLine > & > iris::GetDisassemblyDelegate

  Get the disassembly of a chunk of memory.

Functions

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

  Add a disassembly mode.
- void iris::IrisInstanceDisassembler::attachTo (IrisInstance *irisInstance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- iris::IrisInstanceDisassembler::IrisInstanceDisassembler (IrisInstance *irisInstance=nullptr)

  Construct an IrisInstanceDisassembler.
- void iris::IrisInstanceDisassembler::setDisassembleOpcodeDelegate (DisassembleOpcodeDelegate delegate)

  Set the delegate to get the disassembly of Opcode.
- void iris::IrisInstanceDisassembler::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

7.9.2.1 DisassembleOpcodeDelegate

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

Get the disassembly for an individual opcode.

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

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

7.9.2.2 GetCurrentDisassemblyModeDelegate

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

Get the current disassembly mode.

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

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

7.9.2.3 GetDisassemblyDelegate

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

Get the disassembly of a chunk of memory.

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

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

7.9.3 Function Documentation

7.9.3.1 addDisassemblyMode()

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

Add a disassembly mode.

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

Parameters

name	Name of the mode being added.
description	Description of the mode being added.

7.9.3.2 attachTo()

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

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

irisInstance	The IrisInstance to attach to.

7.9.3.3 IrisInstanceDisassembler()

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

Construct an IrisInstanceDisassembler.

Parameters

irisInstance	IrisInstance to attach this add-on to.

7.9.3.4 setDisassembleOpcodeDelegate()

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

Set the delegate to get the disassembly of Opcode.

Parameters

delegate	Delegate object that will be called to get the disassembly of an opcode.

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.

7.10.1 Detailed Description

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

Chapter 8

Class Documentation

8.1 iris::IrisInstanceBuilder::AddressTranslationBuilder Class Reference

Used to set metadata for an address translation.
`#include <IrisInstanceBuilder.h>`

Public Member Functions

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

  Set the delegate to perform an address translation.
- AddressTranslationBuilder & setTranslateDelegate (MemoryAddressTranslateDelegate delegate)

  Set the delegate to perform an address translation.
- template<typename T , IrisErrorCode(T::∗)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) METHOD>
  AddressTranslationBuilder & setTranslateDelegate (T ∗instance)

  Set the delegate to perform an address translation.

8.1.1 Detailed Description

Used to set metadata for an address translation.

8.1.2 Member Function Documentation

8.1.2.1 setTranslateDelegate() [1/3]

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

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

See also

IrisInstanceBuilder::setDefaultAddressTranslationDelegate

Template Parameters

FUNC	An address translation delegate function.

Returns

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

8.1.2.2 setTranslateDelegate() [2/3]

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

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

See also

IrisInstanceBuilder::setDefaultAddressTranslationDelegate

Parameters

delegate	MemoryAddressTranslateDelegate object.

Returns

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

8.1.2.3 setTranslateDelegate() [3/3]

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

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

See also

IrisInstanceBuilder::setDefaultAddressTranslationDelegate

Template Parameters

T	A class that defines a method with the right signature to be a memory address translation delegate.
METHOD	A memory address translation 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.

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.
- template<typename T >
  EventSourceBuilder & addOption (const std::string &name, const std::string &type, const T &defaultValue, bool optional, const std::string &description)

  Declare an option for event streams of an event source.
- **EventSourceBuilder** (IrisInstanceEvent::EventSourceInfoAndDelegate &info_)
- EventSourceBuilder & hasSideEffects (bool hasSideEffects_=true)

  Set hasSideEffects for this event source.
- EventSourceBuilder & setCounter (bool counter=true)

  Set the `counter` field.
- EventSourceBuilder & setDescription (const std::string &description)

  Set the `description` field.
- EventSourceBuilder & setEventStreamCreateDelegate (EventStreamCreateDelegate delegate)

  Set the delegate to create an event stream.
- template<typename T , IrisErrorCode(T::*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) METHOD>
  EventSourceBuilder & setEventStreamCreateDelegate (T *instance)

  Set the delegate to create an event stream.
- EventSourceBuilder & setFormat (const std::string &format)

  Set the `format` field.
- EventSourceBuilder & setHidden (bool hidden=true)

Hide/unhide this event source.

- EventSourceBuilder & setName (const std::string &name)

  Set the `name` field.

8.3.1 Detailed Description

Used to set metadata on an EventSource.

8.3.2 Member Function Documentation

8.3.2.1 addEnumElement()

```
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

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

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

name	The name of the field.
type	The type of the field.
size	The size of the field in bytes.
description	A human readable description of the field.

Returns

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

8.3.2.3 addOption()

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

Declare an option for event streams of an event source.
This method fills the 'options' member of EventSourceInfo. It may be called multiple times to add multiple options.

Parameters

name	The name of the field.
type	The type of the field.
defaultValue	The default value of the field.
optional	True if the field is optional, False otherwise.
description	A human readable description of the field.

Returns

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

8.3.2.4 hasSideEffects()

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

Set hasSideEffects for this event source.

Parameters

*hasSideEffects_*	If true, this event source has side effects. This is exotic. Normal event sources do not have side effects. For example semihosting events have side effects.

Returns

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

8.3.2.5 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.

8.3.2.6 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.

8.3.2.7 setEventStreamCreateDelegate() [1/2]

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

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

See also

IrisInstanceBuilder::setDefaultEsCreateDelegate

Parameters

delegate	EventStreamCreateDelegate object.

Returns

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

8.3.2.8 setEventStreamCreateDelegate() [2/2]

```
template<typename T , IrisErrorCode(T::*)(EventStream *&, const EventSourceInfo &, const std↩
::vector< std::string > &) METHOD>
EventSourceBuilder & iris::IrisInstanceBuilder::EventSourceBuilder::setEventStreamCreate↩
Delegate (
            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

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

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.

8.3.2.9 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.

8.3.2.10 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.

Returns

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

8.3.2.11 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.

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**
- bool **isProxy** {false}
- bool **isValid** {true}
- ProxyEventInfo **proxyEventInfo**

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.
`#include <IrisInstanceEvent.h>`
Inherited by iris::IrisEventStream.

Public Member Functions

- virtual IrisErrorCode action (const BreakpointAction &action_)
  Execute action on trace stream.
- void addField (const IrisU64StringConstant &field, bool value)
  Add a boolean field value.
- template<class T >
  void addField (const IrisU64StringConstant &field, const T &value)
  Add a field value.
- void addField (const IrisU64StringConstant &field, int64_t value)
  Add a sint field value.
- void addField (const IrisU64StringConstant &field, uint64_t value)
  Add a uint field value.
- void addFieldSlow (const std::string &field, bool value)
  Add a boolean field value.
- template<class T >
  void addFieldSlow (const std::string &field, const T &value)
  Add a field value.
- void addFieldSlow (const std::string &field, int64_t value)
  Add a sint field value.
- void addFieldSlow (const std::string &field, uint64_t value)
  Add a uint field value.
- bool checkRangePc (uint64_t pc) const
  Check the range for the PC.
- virtual IrisErrorCode disable ()=0
  Disable this event stream.
- void emitEventBegin (IrisRequest &req, uint64_t time, uint64_t pc=IRIS_UINT64_MAX)

Start to emit an event callback.

- void emitEventBegin (uint64_t time, uint64_t pc=IRIS_UINT64_MAX)

  Start to emit an event callback.
- void emitEventEnd (bool send=true)

  Emit the callback.
- virtual IrisErrorCode enable ()=0

  Enable this event stream.
- **EventStream** ()

  Construct a new event stream.
- virtual IrisErrorCode flush (RequestId requestId)

  Flush event stream.
- uint64_t getCountVal () const

  Get the current value of the counter.
- InstanceId getEcInstId () const

  Get the event callback instance id for this event stream.
- EventStreamId getEsId () const

  Get the Id of this event stream.
- const EventSourceInfo * getEventSourceInfo () const

  Get the event source info of this event stream.
- InstanceId getProxiedByInstanceId () const

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

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

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

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

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

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

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

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

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

  Initialize this event stream.
- void setProxiedByInstanceId (InstanceId instId)

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

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

  Set the trace ranges for this event stream.

Protected Attributes

- std::string **aspect**

  — members for range —
- bool **aspectFound**

  Found aspect in one of the fields.
- bool counter

  — members for a counter —
- EventCounterMode **counterMode**

  Specified counter mode.
- uint64_t **curAspectValue**

  The current aspect value.
- uint64_t **curVal**
- std::string **ecFunc**

  The event callback function name specified by eventEnable().
- 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**
- IrisInstance * irisInstance

  — basic members —
- bool **isProxyForOtherInstance** {false}

  Is this event stream a proxy for an event stream in another Iris instance?
- InstanceId proxiedByInstanceId {IRIS_UINT64_MAX}
- 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 action()

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

Execute action on trace stream.
This function is usually only ever called by breakpoints which have an action other than eventStream_enable or eventStream_disable.
This function is only implemented by very specific event streams.

Returns

An error code indicating whether the operation was successful.

8.5.2.2 addField() [1/4]

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

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

Parameters

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

8.5.2.3 addField() [2/4]

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

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

Parameters

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

8.5.2.4 addField() [3/4]

```
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

field	The name of the field whose value is set.

Parameters

value	The value of the field.

8.5.2.5 addField() [4/4]

```
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

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

8.5.2.6 addFieldSlow() [1/4]

```
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

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

8.5.2.7 addFieldSlow() [2/4]

```
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

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

8.5.2.8 addFieldSlow() [3/4]

```
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

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

8.5.2.9 addFieldSlow() [4/4]

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

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

Parameters

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

8.5.2.10 checkRangePc()

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

Check the range for the PC.
This can optionally be called before generating the callback request (before calling emitEventBegin()).

Parameters

pc	The program counter value to check.

Returns

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

8.5.2.11 disable()

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

Disable this event stream.
This function is only called when isEnabled()/enabled == true. It is not necessary to verify this inside the disable() method.

Returns

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

Implemented in iris::IrisEventStream.

8.5.2.12 emitEventBegin() [1/2]

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

Start to emit an event callback.

Parameters

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

8.5.2.13 emitEventBegin() [2/2]

```
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.

8.5.2.14 emitEventEnd()

```
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.

8.5.2.15 enable()

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

Enable this event stream.
This function is only called when isEnabled()/enabled == false. It is not necessary to verify this inside the enable() method.

Returns

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

Implemented in iris::IrisEventStream.

8.5.2.16 flush()

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

Flush event stream.
Supported in the derived classes for specific event sources.

Parameters

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

Returns

An error code indicating whether the operation was successful.

8.5.2.17 getCountVal()

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

Get the current value of the counter.

Returns

The current value of the event counter.

8.5.2.18 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.

8.5.2.19 getEsId()

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

Get the Id of this event stream.

Returns

The esId for this event stream.

8.5.2.20 getEventSourceInfo()

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

Get the event source info of this event stream.

Returns

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

8.5.2.21 getProxiedByInstanceId()

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

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

Returns

`The` instance ID of the Iris instance which is a proxy

8.5.2.22 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.

8.5.2.23 isCounter()

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

Is this event stream a counter?

Returns

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

8.5.2.24 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.

8.5.2.25 IsProxiedByOtherInstance()

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

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

Returns

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

8.5.2.26 IsProxyForOtherInstance()

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

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

Returns

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

8.5.2.27 selfRelease()

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

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

Note

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

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

8.5.2.28 setCounter()

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

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

Parameters

startVal	The starting value of the counter.
counterMode	The mode in which this counter operates.

8.5.2.29 setOptions()

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

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

Parameters

options	Map of options (key/value pairs).
eventStreamCreate	True: These are the options set by eventStream_create(). False: These are options set by eventStream_setOptions().
errorMessageOut	When this function returns an error it should set errorMessageOut to a meaningful error message.

Returns

An error code indicating whether the operation was successful.

8.5.2.30 setProperties()

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

Initialize this event stream.

Parameters

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

8.5.2.31 setProxiedByInstanceId()

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

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

Parameters

instId	The instance ID of the proxy Iris instance

8.5.2.32 setRanges()

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

Set the trace ranges for this event stream.

Parameters

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

Returns

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

8.5.3 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.

8.5.3.3 proxiedByInstanceId

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

An event stream in another Iris instance is a proxy for this event stream proxiedByInstanceId - the instance ID of the other Iris instance
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)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Set the `subRscId` field.
- FieldBuilder & setTag (const std::string &tag)

  Set the named boolean tag to true (e.g. isPc)
- FieldBuilder & setTag (const std::string &tag, const IrisValue &value)

  Set a tag to the specified value.
- FieldBuilder & setType (const std::string &type)

  Set the `type` field.
- template<IrisErrorCode(∗)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  FieldBuilder & setWriteDelegate ()

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

Set the delegate to write the resource.

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

  Set the delegate to write the resource.
- template<typename T >
  FieldBuilder & setWriteMask (std::initializer_list< T > &&t)

  Set the `writeMask` field for wide registers.
- FieldBuilder & setWriteMask (uint64_t value)

  Set the `writeMask` field to a value <= 64 bit.
- template<typename Container >
  FieldBuilder & setWriteMaskFromContainer (const Container &container)

  Set the `writeMask` field for wide registers.

Protected Attributes

- IrisInstanceResource::ResourceInfoAndAccess * **info** {}
- IrisInstanceBuilder * **instance_builder** {}
- RegisterBuilder * **parent_reg** {}

8.6.1 Detailed Description

Used to set metadata on a register field resource.

8.6.2 Member Function Documentation

8.6.2.1 addEnum()

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

Add a symbol to the enums field for numeric resources.
This should be called multiple times to add multiple symbols.

Parameters

symbol	The symbol string to be associated with the specified value.
value	The value of this symbol.
description	A description of this symbol.

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

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

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

architecture	ELF EM_* constant for architecture.
dwarfRegNum	DWARF register number for architecture.

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 setDescription()

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setDescription (
            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

parent↩ RscId	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]

```
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]

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

Set the delegate to read the resource.
Set a delegate which calls METHOD() on an instance of class T.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Template Parameters

T	A class that defines a method with the right signature to be a resource read delegate.
METHOD	A resource read 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.

8.6.2.21 setResetData() [1/2]

```
template<typename T >
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setResetData (
            std::initializer_list< T > && t )  [inline]
```

Set the `resetData` field for wide registers.
This function accepts a braced initializer-list and is otherwise idential 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.

8.6.2.22 setResetData() [2/2]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setResetData (
            uint64_t value )  [inline]
```

Set the `resetData` field to a value <= 64 bit.
If the register is wider than the passed value the value is zero extended.
If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

value	resetData value of the register.

Returns

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

8.6.2.23 setResetDataFromContainer()

```
template<typename Container >
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setResetDataFromContainer (
            const Container & container )  [inline]
```

Set the `resetData` field for wide registers.
Container must be a type which allows to iterate over uint64_t bit chunks of the value,
least significant bits first, for example std::array<uint64_t> or std::vector<uint64_t>.
Each element of the container will be promoted/narrowed to uint64_t.
If the register is wider than the passed value the value is zero extended.
If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

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()

```
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]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteDelegate (
            ResourceWriteDelegate writeDelegate )  [inline]
```

Set the delegate to write the resource.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Parameters

writeDelegate	ResourceWriteDelegate object.

Returns

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

8.6.2.32 setWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
METHOD>
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteDelegate (
            T * instance )  [inline]
```

Set the delegate to write the resource.
Set a delegate which calls METHOD() on an instance of class T.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Template Parameters

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.

8.6.2.33 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 idential to setWriteMaskFromContainer().
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.

8.6.2.34 setWriteMask() [2/2]

```
FieldBuilder & iris::IrisInstanceBuilder::FieldBuilder::setWriteMask (
            uint64_t value )  [inline]
```

Set the `writeMask` field to a value <= 64 bit.
If the register is wider than the passed value the value is zero extended.
If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

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 *channel_id_out)
- int64_t **IrisC_unregisterChannel** (uint64_t channel_id)
- **IrisCConnection** ()

  Construct an empty object. Used by subclasses that need to load a DSO and call init().

Protected Attributes

- void * **iris_c_context**

  Context pointer to use when calling IrisC_* functions. This is also needed by subclasses.

8.7.1 Detailed Description

Provide an IrisConnectionInterface which loads an IrisC library.

See also

IrisClient

IrisGlobalInstance

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

- IrisCConnection.h

8.8 iris::IrisClient Class Reference

Inherits IrisInterface, impl::IrisProcessEventsInterface, and IrisConnectionInterface.

Public Member Functions

- void connect (const std::string &connectionSpec)
- IrisErrorCode connect (const std::string &hostname, uint16_t port, unsigned timeoutInMs, std::string &error↩ ResponseOut)
- void connectSocketFd (SocketFd socketfd, unsigned timeoutInMs=1000)
- IrisErrorCode disconnect ()
- std::string **getConnectionStr** () const

  Get connection string, describing the Iris server we are connected to.
- impl::IrisRpcAdapterTcp::Format **getEffectiveSendingFormat** () const

  Get effective sending format that Rpc adapter uses.
- IrisInstance & getIrisInstance ()
- virtual IrisInterface ∗ **getIrisInterface** () override
- IrisInterface ∗ **getSendingInterface** ()

  Get interface for sending messages to the server.
- void initServiceServer (impl::IrisTcpSocket ∗socket_)
- **IrisClient** (const service::IrisServiceTcpServer ∗, const std::string &instName=std::string())

  Service constructor to initialize IrisService Server on IrisService side.
- IrisClient (const std::string &hostname, uint16_t port, const std::string &instName=std::string())

  Construct a connection to an Iris server.
- **IrisClient** (const std::string &instName=std::string(), const std::string &connectionSpec=std::string())

  Client constructor.
- bool **isConnected** () const

  Return true iff connected to a server.
- void loadPlugin (const std::string &plugin_name)
- virtual IrisErrorCode **processAsyncMessages** (bool waitForAMessage) override
- virtual void processEvents () override
- uint64_t **registerChannel** (IrisC_CommunicationChannel ∗channel)
- uint64_t **registerChannel** (IrisC_CommunicationChannel ∗channel, const ::std::string &path)
- virtual uint64_t **registerIrisInterfaceChannel** (IrisInterface ∗iris_interface) override
- void setInstanceName (const std::string &instName)
- void **setIrisMessageLogLevel** (unsigned level)

  Enable message logging.
- void **setPreferredSendingFormat** (impl::IrisRpcAdapterTcp::Format p)

  Set preferred sending format that Rpc adapter uses.
- void setSleepOnDestructionMs (uint64_t sleepOnDestructionMs_)
- void **setVerbose** (unsigned level, bool increaseOnly=false)

  Set verbose level.
- virtual void stopWaitForEvent () override
- void **unloadPlugin** ()
- void **unregisterChannel** (uint64_t channelId)
- virtual void **unregisterIrisInterfaceChannel** (uint64_t channelId) override
- virtual void waitForEvent () override
- virtual ∼**IrisClient** ()

  Destructor.

Public Attributes

- const std::string connectionHelpStr

  Connection help string.

8.8.1 Constructor & Destructor Documentation

8.8.1.1 IrisClient()

```
iris::IrisClient::IrisClient (
            const std::string & hostname,
            uint16_t port,
            const std::string & instName = std::string() )  [inline]
```

Construct a connection to an Iris server.

Parameters

hostname	Hostname of the Iris server. This can be an IP address. For example: • "192.168.0.5" IP address of a different host. • "127.0.0.1" Loopback IP address to connect to a server on the same machine. • "localhost" Hostname of the loopback interface. Port == 0 means to scan ports 7100 to 7109. • "foo.bar.com" Hostname of a remote machine.
port	Server port number to connect to on the host.

8.8.2 Member Function Documentation

8.8.2.1 connect() [1/2]

```
void iris::IrisClient::connect (
            const std::string & connectionSpec )  [inline]
```

Connect to an Iris server.
The connection details are specified as a string. See "connectionHelpStr" for syntax. This function is self documenting: Passing "help" will return a list of all supported connection types and their syntax, as an E_help_↩ message error.
This throws E_not_connected when connectionSpec was erroneous, and E_socket_error or E_connection_refused when the connection could not be established.

8.8.2.2 connect() [2/2]

```
IrisErrorCode iris::IrisClient::connect (
            const std::string & hostname,
            uint16_t port,
            unsigned timeoutInMs,
            std::string & errorResponseOut )  [inline]
```

Connect to server on hostname:port.
If hostname == "localhost" and port == 0 then a port scan on ports 7100 to 7109 is done.

8.8.2.3 connectSocketFd()

```
void iris::IrisClient::connectSocketFd (
            SocketFd socketfd,
            unsigned timeoutInMs = 1000 )  [inline]
```

Connect using an existing socketFd. All errors are reported by exceptions.

8.8.2.4 disconnect()

```
IrisErrorCode iris::IrisClient::disconnect ( )  [inline]
```

Disconnect from server. (Only for mode IRIS_TCP_CLIENT.)

8.8.2.5 getIrisInstance()

```
IrisInstance & iris::IrisClient::getIrisInstance ( )  [inline]
```
Get contained IrisInstance. This can be used as a generic client instance to call Iris functions.

8.8.2.6 initServiceServer()

```
void iris::IrisClient::initServiceServer (
            impl::IrisTcpSocket * socket_ )  [inline]
```
Initialize as an IrisService server, only used in IRIS_SERVICE_SERVER mode. This function will store pointer to IrisTcpSocket created by IrisService and initialize adapter as a server. -socket_ pointer to IrisTcpSocket created by IrisService when receiving new connection. (TODO safer memory management of this object) -return Nothing.

8.8.2.7 loadPlugin()

```
void iris::IrisClient::loadPlugin (
            const std::string & plugin_name )  [inline]
```
Load Plugin function, only used in IRIS_SERVICE_SERVER mode Only one plugin can be loaded at a a time

8.8.2.8 processEvents()

```
virtual void iris::IrisClient::processEvents ( )  [inline], [override], [virtual]
```
Client main processing function.

- Check for incoming requests/responses and process them .
- Check for pending outgoing requests/responses and process them. This function is ideal for integrating the client into other processing environments in one of the following ways: (1) Thread-less: Requests are only executed from within processEvents().
- pro: Iris request and responses are always synchronized with the rest of the code of the client. No explicit synchronization (mutexes etc.) necessary.
- con: No blocking Iris requests can be called from within received synchronous callbacks. (2) Asynchronous (handleRequestAsynchronously = true): Requests are executed in another thread
- pro: Blocking Iris requests can be called from within received synchronous callbacks transparently.
- con: Received Iris requests are called on another thread and they require explicit synchronization to be synchronized with the rest of the code of the client. It is harmless to call this function when there is nothing to do.

8.8.2.9 setInstanceName()

```
void iris::IrisClient::setInstanceName (
            const std::string & instName )  [inline]
```
Set instance name of the contained Iris instance returned by getIrisInstance. This must be called before connect().

8.8.2.10 setSleepOnDestructionMs()

```
void iris::IrisClient::setSleepOnDestructionMs (
            uint64_t sleepOnDestructionMs_ )  [inline]
```
Sleep a short time on destruction to de-interleave output by different processes. This has not functional impacto or purpose. It just beautifies the output on stdout.

8.8.2.11 stopWaitForEvent()

```
virtual void iris::IrisClient::stopWaitForEvent ( )  [inline], [override], [virtual]
```
Stop waiting in waitForEvent(). Return from waitForEvent() as soon as possible even without a socket event.

8.8.2.12 waitForEvent()

```
virtual void iris::IrisClient::waitForEvent ( )  [inline], [override], [virtual]
```

Wait for any event which would cause processEvents() to do some work. This function intentionally blocks until there is something useful to do. This function can be interrupted by calling stopWaitForEvent().

8.8.3 Member Data Documentation

8.8.3.1 connectionHelpStr

```
const std::string iris::IrisClient::connectionHelpStr
```

Initial value:

```
=
        "Supported connection types:\n"
        "tcp[=HOST][,port=PORT][,timeout=T]\n"
        "  Connect to an Iris TCP server on HOST:PORT.\n"
        "  The default for HOST is 'localhost' and the default for PORT is 0 if HOST is 'localhost' and 7100
       otherwise. If PORT is 0 then a port scan on ports 7100 to 7109 is done.\n"
        "  T is the connection timeout in ms (defaults to 100 if PORT==0, else 1000).\n"
        "\n"
        "socketfd=FD[,timeout=T]\n"
        "  Use socket file descriptor FD as an established UNIX domain socket connection.\n"
        "  T is the timeout for the Iris handshake in ms.\n"
        "\n"
        "General parameters:\n"
        "  verbose=N: Increase verbose level of IrisClient to level N (0..3).\n"
```

Connection help string.

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

- IrisClient.h

8.9 iris::IrisCommandLineParser Class Reference

```
#include <IrisCommandLineParser.h>
```

Classes

- struct Option

  Option container.

Public Member Functions

- Option & addOption (char shortOption, const std::string &longOption, const std::string &help, const std::string &formalArgumentName=std::string(), const std::string &defaultValue=std::string())
- void clear ()
- double getDbl (const std::string &longOption) const
- std::string getHelpMessage () const
- int64_t getInt (const std::string &longOption) const
- std::vector< std::string > **getList** (const std::string &longOption) const

  Get list of elements of a list option.
- std::map< std::string, std::string > getMap (const std::string &longOption) const
- const std::vector< std::string > & **getNonOptionArguments** () const

  Get non-option arguments.
- std::string **getStr** (const std::string &longOption) const

  Get string value.
- uint64_t **getSwitch** (const std::string &longOption) const

  Check how many times an option switch (an option without an argument) was specified.
- uint64_t getUint (const std::string &longOption) const

- **IrisCommandLineParser** (const std::string &programName_, const std::string &usageHeader_, const std::string &versionStr_)

  Constructor.
- bool isSpecified (const std::string &longOption) const
- void noNonOptionArguments ()
- int parseCommandLine (int argc, const char *argv[])
- void pleaseSpecifyOneOf (const std::vector< std::string > &options, const std::vector< std::string > &formalNonOptionArguments=std::vector< std::string >())
- int **printError** (const std::string &message) const

  Print error message (and do not exit).
- int printErrorAndExit (const IrisErrorException &e) const
- int printErrorAndExit (const std::string &message) const
- int printMessage (const std::string &message, int error=0, bool exit=false) const
- void setMessageFunc (const std::function< int(const std::string &message, int error, bool exit)> &messageFunc)
- void **setProgramName** (const std::string &programName_, bool append=false)

  Set/override program name.
- void setValue (const std::string &longOption, const std::string &value, bool append=false)
- void unsetValue (const std::string &longOption)

Static Public Member Functions

- static int defaultMessageFunc (const std::string &message, int error, bool exit)

8.9.1 Detailed Description

Generic command line parser.
This covers roughly all features supported by GNU getopt_long() and provides -h/–help and –version.
Usage:

1. Declare options by calling addOption() for each option.
2. Parse command line by calling parseCommandLine().
3. Retrieve command line option values by calling the get...() functions.

Example:

8.9.2 Member Function Documentation

8.9.2.1 addOption()

```
Option & iris::IrisCommandLineParser::addOption (
            char shortOption,
            const std::string & longOption,
            const std::string & help,
            const std::string & formalArgumentName = std::string(),
            const std::string & defaultValue = std::string() )
```

Add command line option. shortOption: Single character or 0 if no short option. longOption: Long option (mandatory, must be unique and non-empty). help: Description for –help. formalArgumentName: Empty means: This option has no argument (switch). Nonempty means: This option has an argument and this is named 'formalArgumentName' in the –help message. defaultValue: Default value of this option when not specified on the command line. When defaultValue is not specified: By default getSwitch(), getInt() and GetUint() return 0 and getStr() returns an empty string.

8.9.2.2 clear()

```
void iris::IrisCommandLineParser::clear ( )
```

Clear all values parsed by a previous parseCommandLine call. All options will be reset to their default values. All option definitions (addOption()) will be preserved.

8.9.2.3 defaultMessageFunc()

```
static int iris::IrisCommandLineParser::defaultMessageFunc (
            const std::string & message,
            int error,
            bool exit ) [static]
```

Default message function. The default message function prints message on stdout and exits with "error" status if exit==true, else it returns error status.

8.9.2.4 getDbl()

```
double iris::IrisCommandLineParser::getDbl (
            const std::string & longOption ) const
```

Get double value. (This will print an error and exit when there is a parse error.)

8.9.2.5 getHelpMessage()

```
std::string iris::IrisCommandLineParser::getHelpMessage ( ) const
```

Get help message. (parserCommandLine() automatically prints this on –help so there is usually no need to call this function.)

8.9.2.6 getInt()

```
int64_t iris::IrisCommandLineParser::getInt (
            const std::string & longOption ) const
```

Get integer value. (This will print an error and exit when there is a parse error.)

8.9.2.7 getMap()

```
std::map< std::string, std::string > iris::IrisCommandLineParser::getMap (
            const std::string & longOption ) const
```

Get NAME->VALUE map of elements of a list option. The elements are assumed to have the format "NAME=↩ VALUE" or "NAME". If "=VALUE" is missing then VALUE is the empty string.

8.9.2.8 getUint()

```
uint64_t iris::IrisCommandLineParser::getUint (
            const std::string & longOption ) const
```

Get unsigned integer value. (This will print an error and exit when there is a parse error.)

8.9.2.9 isSpecified()

```
bool iris::IrisCommandLineParser::isSpecified (
            const std::string & longOption ) const
```

Return true iff option is specified explicitly on the command line. (This can be used to detect whether an option was present on the command line even if it was just set to its default value.)

8.9.2.10 noNonOptionArguments()

```
void iris::IrisCommandLineParser::noNonOptionArguments ( )
```

Print an error for each non-option argument and exit if any non-option arguments are present. Call this after parseCommandLine() for programs which do not support any non-option arguments as these are otherwise silently ignored.

8.9.2.11 parseCommandLine()

```
int iris::IrisCommandLineParser::parseCommandLine (
            int argc,
            const char * argv[] )
```

Parse command line. After calling this function the named argument values can be retrieved by the get...() functions. All arguments after the first occurrence of a "--" argument are treated as non-option arguments. Also handles –help and –version and exit()s when these are specified.
argv[0] is ignored. The program name is passed in the constructor argument.
Calling parseCommandLine() again will ad and/or override options as if they were in a single command line.
Return value: By default parseCommandLine() exits (and so does not return) when it detects an error or when –help or –version was specified, so the return value can safely (and should) be ignored.
When the exit behavior is overridden by calling setMessageFunc() with a non-exiting function, then parseCommandLine() returns the return value of the message function or 0 when the message function was not called (no error and no –help/–version).
Note that parse errors in integers or doubles are only identified by the respective get∗() functions.

8.9.2.12 pleaseSpecifyOneOf()

```
void iris::IrisCommandLineParser::pleaseSpecifyOneOf (
            const std::vector< std::string > & options,
            const std::vector< std::string > & formalNonOptionArguments = std::vector< std↩
::string >() )
```

Check whether at least one of the options or non-option-arguments are specified and exit with an error message if not. Call this for programs which require at least one of these options or arguments to be set. If formalNonOption↩
Arguments is empty only options are checked.

8.9.2.13 printErrorAndExit() [1/2]

```
int iris::IrisCommandLineParser::printErrorAndExit (
            const IrisErrorException & e ) const  [inline]
```

Print error message and exit. Note that custom message functions may decide not to exit even on errors. In this case parseCommandLine() returns the return value of the message function.

8.9.2.14 printErrorAndExit() [2/2]

```
int iris::IrisCommandLineParser::printErrorAndExit (
            const std::string & message ) const
```

Print error message and exit. Note that custom message functions may decide not to exit even on errors. In this case parseCommandLine() returns the return value of the message function.

8.9.2.15 printMessage()

```
int iris::IrisCommandLineParser::printMessage (
            const std::string & message,
            int error = 0,
            bool exit = false ) const
```

Print message. This can be used by additional checks on the arguments to print warnings. This calls the message function set by setMessageFunc() or the defaultMessageFunc().

8.9.2.16 setMessageFunc()

```
void iris::IrisCommandLineParser::setMessageFunc (
            const std::function< int(const std::string &message, int error, bool exit)> &
messageFunc )
```

Set custom message function which prints errors (error!=0), –help and –version messages (error==0) and which potentially also exit()s (exit==true).
The default message function prints message on stdout and exits with "error" status if exit==true, else it returns error status.

Custom message functions may either exit, or they may return a value which is then returned by parserCommand↩ Line() for errors raised by parseCommandLine(). For errors in the get∗() functions the return value is ignored.

8.9.2.17 setValue()

```
void iris::IrisCommandLineParser::setValue (
            const std::string & longOption,
            const std::string & value,
            bool append = false )
```

Set/override command line option. By default overwrite the entire list for list options. Set append=true for list options to append to list.

8.9.2.18 unsetValue()

```
void iris::IrisCommandLineParser::unsetValue (
            const std::string & longOption )
```

Unset command line option. Set value to default value and mark as not specified.
The documentation for this class was generated from the following file:

- IrisCommandLineParser.h

8.10 iris::IrisEventEmitter< ARGS > Class Template Reference

A helper class for generating Iris events.
`#include <IrisEventEmitter.h>`
Inherits IrisEventEmitterBase.

Public Member Functions

- **IrisEventEmitter** ()

  Construct an event emitter.
- void operator() (ARGS... args)

  Emit an event.

8.10.1 Detailed Description

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

A helper class for generating Iris events.

Template Parameters

ARGS	Argument types corresponding to the fields in this event.

Use IrisEventEmitter with IrisInstanceBuilder to add events to your Iris instance:

```
// Declare an event emitter
iris::IrisEventEmitter<uint64_t, bool> my_event;
// Add it to an Iris instance
iris::IrisInstance my_instance(...);
my_instance->getBuilder()->addEventSource("MY_EVENT", my_event)
    .addField("FOO",  "uint", 8, "A value")
    .addField("FLAG", "bool", 1, "A flag");
// Emit an event
my_event(0x1234, true);
```

8.10.2 Member Function Documentation

8.10.2.1 operator()()

```
template<typename... ARGS>
void iris::IrisEventEmitter< ARGS >::operator() (
            ARGS... args ) [inline]
```

Emit an event.
The arguments to this function are the fields of the event source, in the same order that they appear in the template arguments to the IrisEventEmitter class.
The documentation for this class was generated from the following file:

- IrisEventEmitter.h

8.11 iris::IrisEventRegistry Class Reference

Class to register Iris event streams for an event.
`#include <IrisInstanceEvent.h>`

Public Types

- typedef std::set< EventStream * >::const_iterator **iterator**

Public Member Functions

- template<class T >
  void addField (const IrisU64StringConstant &field, const T &value) const

  Add a field value.
- template<class T >
  void addFieldSlow (const std::string &field, const T &value) const

  Add a field value.
- iterator begin () const

  Get an iterator to the beginning of the event stream set.
- void **emitEventBegin** (uint64_t time, uint64_t pc=IRIS_UINT64_MAX) const
- void emitEventEnd () const

  Emit the callback.
- bool empty () const

  Return true if no event streams are registered.
- iterator end () const

  Get an iterator to the end of the event stream set.
- bool registerEventStream (EventStream *evStream)

  Register an event stream.
- bool unregisterEventStream (EventStream *evStream)

  Unregister an event stream.

8.11.1 Detailed Description

Class to register Iris event streams for an event.

8.11.2 Member Function Documentation

8.11.2.1 addField()

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

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

Template Parameters

T	The type of `value`.

Parameters

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

8.11.2.2 addFieldSlow()

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

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

Template Parameters

T	The type of `value`.

Parameters

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

8.11.2.3 begin()

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

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

See also

end

Returns

An iterator to the beginning of the event stream set.

8.11.2.4 emitEventEnd()

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

Emit the callback.
This also checks the ranges and maintains the counter.

8.11.2.5 empty()

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

Return true if no event streams are registered.

Returns

`true` if no event streams are registered.

8.11.2.6 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.

8.11.2.7 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.

8.11.2.8 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:

- IrisInstanceEvent.h

8.12 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

8.12.1 Detailed Description

Event stream class for Iris-specific events.

8.12.2 Member Function Documentation

8.12.2.1 disable()

`virtual IrisErrorCode iris::IrisEventStream::disable ( ) [virtual]`
Disable this event stream.
This function is only called when isEnabled()/enabled == true. It is not necessary to verify this inside the disable() method.

Returns

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

Implements iris::EventStream.

8.12.2.2 enable()

`virtual IrisErrorCode iris::IrisEventStream::enable ( ) [virtual]`
Enable this event stream.
This function is only called when isEnabled()/enabled == false. It is not necessary to verify this inside the enable() method.

Returns

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

Implements iris::EventStream.
The documentation for this class was generated from the following file:

- IrisInstanceEvent.h

8.13 iris::IrisGlobalInstance Class Reference

Inherits IrisInterface, and IrisConnectionInterface.

Public Member Functions

- IrisInstance & getIrisInstance ()
- virtual IrisInterface * **getIrisInterface** () override

  Get the IrisInterface for this connection.
- **IrisGlobalInstance** ()

  Constructor.
- virtual void **irisHandleMessage** (const uint64_t *message) override

  Handle incoming Iris messages.
- virtual IrisErrorCode **processAsyncMessages** (bool waitForAMessage) override
- uint64_t registerChannel (IrisC_CommunicationChannel *channel, const std::string &connection_info="")
- virtual uint64_t registerIrisInterfaceChannel (IrisInterface *iris_interface) override
- virtual void **setIrisProxyInterface** (IrisProxyInterface *irisProxyInterface_) override

  Set proxy interface.
- void **setLogLevel** (unsigned level)
- void **unregisterChannel** (uint64_t channelId)

  Unregister a channel.
- virtual void unregisterIrisInterfaceChannel (uint64_t channelId) override
- ~**IrisGlobalInstance** ()

  Destructor.

8.13.1 Member Function Documentation

8.13.1.1 getIrisInstance()

```
IrisInstance & iris::IrisGlobalInstance::getIrisInstance ( )  [inline]
```

Get contained IrisInstance. This can be used as a generic client instance to call Iris functions.

8.13.1.2 registerChannel()

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

Register a channel. Returns an associated channel id.

8.13.1.3 registerIrisInterfaceChannel()

```
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.

8.13.1.4 unregisterIrisInterfaceChannel()

```
virtual void iris::IrisGlobalInstance::unregisterIrisInterfaceChannel (
            uint64_t channelId )  [inline], [override], [virtual]
```

Unregister a previously registered channel. This will automatically unregister all instances associated with that channel.

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

- IrisGlobalInstance.h

8.14 iris::IrisInstance Class Reference

Public Types

- using EventCallbackFunction = std::function< IrisErrorCode(EventStreamId, const IrisValueMap &, uint64_t, InstanceId, bool, std::string &)>

Public Member Functions

- void addCallback_IRIS_INSTANCE_REGISTRY_CHANGED (EventCallbackFunction f)
- void **clearCachedMetaInfo** ()

  Clear cached meta-information including the list of InstanceInfos for all instances in the system.
- std::vector< EventSourceInfo > findEventSources (const std::string &instancePathFilter="all")

  Find all event sources in the system.
- void findEventSourcesAndFields (const std::string &spec, std::vector< EventStreamInfo > &eventStream↩ InfosOut, InstanceId defaultInstId=IRIS_UINT64_MAX)

  Find specific event sources in the system.
- std::vector< InstanceInfo > findInstanceInfos (const std::string &instancePathFilter="all")

  Find instance infos of all instances in the system.
- IrisInstanceBuilder * getBuilder ()

  Get the IrisInstanceBuilder object for this instance. This can be used to set up metadata and callbacks for standard Iris functions.
- InstanceId getInstanceId (const std::string &instName)

  Get instance id for a specifid instance name.
- InstanceInfo getInstanceInfo (const std::string &instancePathFilter)

  Get instance info of a specific instance in the system.
- const InstanceInfo & getInstanceInfo (InstanceId instId)

  Get InstanceInfo including properties for a specific instId.
- const std::vector< InstanceInfo > & getInstanceList ()

  Get list of InstanceInfos of all instances in the system, including properties.
- const std::string & getInstanceName () const

  Get the instance name of this instance. This is valid after registerInstance() returns.
- std::string getInstanceName (InstanceId instId)

  Get instance name for a specifid instId.
- InstanceId getInstId () const

  Get the instance id of this instance. This is valid after registerInstance() returns.
- IrisInterface * getLocalIrisInterface ()

  Get the local IrisInterface of this instance. This is the interface that other instances use to send their requests and responses to this instance.
- const PropertyMap & getPropertyMap () const

  Get property map.
- IrisInterface * getRemoteIrisInterface ()

  Get the remote Iris interface.
- const std::vector< iris::ResourceGroupInfo > & **getResourceGroups** (InstanceId instId)

  Get list of resource groups.
- ResourceId getResourceId (InstanceId instId, const std::string &resourceSpec)

  Get resource id for a specific resource.
- const ResourceInfo & **getResourceInfo** (InstanceId instId, const std::string &resourceSpec)

  Get ResourceInfo for a specific resource.
- const ResourceInfo & **getResourceInfo** (InstanceId instId, ResourceId resourceId)

  Get ResourceInfo for a specific resource.
- const std::vector< iris::ResourceInfo > & **getResourceInfos** (InstanceId instId)

  Get list of resource infos.

- IrisCppAdapter & irisCall ()

  Get an IrisCppAdapter to call an Iris function of any other instance.
- IrisCppAdapter & irisCallNoThrow ()

  Get an IrisCppAdapter to call an Iris function of any other instance.
- IrisCppAdapter & irisCallThrow ()

  Get an IrisCppAdapter to call an Iris function of any other instance. When an Iris function returns an error response, this adapter always throws an exception. Usage:
- IrisInstance (IrisConnectionInterface *connection_interface=nullptr, const std::string &instName=std::string(), uint64_t flags=DEFAULT_FLAGS)

  Construct a new Iris instance.
- IrisInstance (IrisInstantiationContext *context)

  Construct a new Iris instance using an IrisInstantiationContext.
- bool **isAdapterInitialized** () const
- bool isRegistered () const
- bool isValidEvBufId (EventBufferId evBufId) const

  Check whether event buffer id is valid.
- void **notifyStateChanged** ()

  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 publishCppInterface (const std::string &interfaceName, T *pointer, const std::string &jsonDescription)

  Publish a C++ interface XYZ through a new instance_getCppInterfaceXYZ() function.
- void registerEventCallback (EventCallbackDelegate delegate, const std::string &name, const std::string &description, const std::string &dlgInstanceTypeStr)

  Register a general event callback using an EventCallbackDelegate.
- template<typename T , IrisErrorCode(T::*)(uint64_t, const AttributeValueMap &, uint64_t, uint64_t, bool, std::string &) METHOD>
  void registerEventCallback (T *instance, const std::string &name, const std::string &description, const std::string &dlgInstanceTypeStr)

  Register a general event callback using an EventCallbackDelegate.
- template<class T >
  void registerEventCallback (T *instance, const std::string &name, const std::string &description, void(T::*memberFunctionPtr)(IrisReceivedRequest &), const std::string &instanceTypeStr)

  Register a general event callback.
- template<class T >
  void registerFunction (T *instance, const std::string &name, void(T::*memberFunctionPtr)(IrisReceivedRequest &), const std::string &functionInfoJson, const std::string &instanceTypeStr)

  Register an Iris function implementation.
- IrisErrorCode registerInstance (const std::string &instName, uint64_t flags=DEFAULT_FLAGS)

  Register this instance if it was not registered when constructed.
- uint64_t resourceRead (InstanceId instId, const std::string &resourceSpec)

  Read numeric resource and return its value.
- uint64_t resourceReadCrn (InstanceId instId, uint64_t canonicalRegisterNumber)

  Read numeric resource and return its value (using the canonical register number aka DWARF register id).
- std::string resourceReadStr (InstanceId instId, const std::string &resourceSpec)

  Read string resource, or read other resources as string.
- void resourceWrite (InstanceId instId, const std::string &resourceSpec, uint64_t value)

  Write numeric resource.
- void resourceWriteCrn (InstanceId instId, uint64_t canonicalRegisterNumber, uint64_t value)

  Write numeric resource by canonical register number (aka DWARF register id).
- void resourceWriteStr (InstanceId instId, const std::string &resourceSpec, const std::string &value)

  Write string resource, or write numeric resource from string.

- bool sendRequest (IrisRequest &req)

  Send an Iris request or notification and potentially wait for a response.
- void sendResponse (const uint64_t *response)

  Send a response to the remote Iris interface.
- void **setAdapterInitialized** ()
- void setCallback_IRIS_SHUTDOWN_LEAVE (EventCallbackFunction f)
- void setCallback_IRIS_SIMULATION_TIME_EVENT (EventCallbackFunction f)
- void setConnectionInterface (IrisConnectionInterface *connection_interface)

  Set the remote connection interface.
- void **setEventHandler** (IrisInstanceEvent *handler)

  Set the event handler and enable the `IRIS_STATE_CHANGED` event.
- void **setInstId** (InstanceId instId)

  Internal function. Do not call. Set the instance id of this instance. The instId is automatically set after calling instane↩ Registry_registerInstance().
- void setPendingSyncStepResponse (RequestId requestId, EventBufferId evBufId)

  Set pending response to a step_syncStep() call.
- template<class 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 simulationTimeDisableEvents ()

  Disable the internal reception of IRIS_SIMULATION_TIME_EVENT events for performance reasons (e.g. during synchronous stepping).
- bool simulationTimeIsRunning ()

  Return true iff simulation is currently running.
- void simulationTimeRun ()

  Run simulation time and wait until simulation time started running.
- void simulationTimeRunUntilStop ()

  Run simulation time and wait until simulation time stopped again.
- void simulationTimeStop ()

  Stop simulation time and wait until simulation time stopped.
- void unpublishCppInterface (const std::string &interfaceName)

  Unpublish a previously published C++ interface.
- void **unregisterEventCallback** (const std::string &name)

  Unregister the named event callback function.
- void **unregisterFunction** (const std::string &name)

  Unregister a function that was previously registered with registerFunction() or irisRegisterFunction().
- IrisErrorCode unregisterInstance ()

  Unregister this instance.
- ~**IrisInstance** ()

  Destructor.

Static Public Attributes

- static const uint64_t **DEFAULT_FLAGS** = THROW_ON_ERROR

  Default flags used if not otherwise specified.
- static const 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.

8.14.1 Member Typedef Documentation

8.14.1.1 EventCallbackFunction

```
using iris::IrisInstance::EventCallbackFunction = std::function<IrisErrorCode(EventStreamId,
const IrisValueMap&, uint64_t, InstanceId, bool, std::string&)>
```

Event callback function type.
(Each IrisInstance can implicitly register two events which are used internally (IRIS_SIMULATION_TIME_EVENT and IRIS_SHUDOWN_LEAVE). Using the functions below clients can make use of these events without going through the effort of calling irisRegisterEventCallback()/registerEventCallback(), event_getEventSource() and eventStream_create(), and it also reduces the number of callbacks being called at runtime.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 IrisInstance() [1/2]

```
iris::IrisInstance::IrisInstance (
            IrisConnectionInterface * connection_interface = nullptr,
            const std::string & instName = std::string(),
            uint64_t flags = DEFAULT_FLAGS )
```

Construct a new Iris instance.

Parameters

connection_interface	The IrisConnectionInterface that this instance should use to connect to the simulation.
instName	Name of the instance. This should be prefixed with one of the following, as appropriate: • `"client."` • `"component."` • `"framework."`
flags	A bitwise OR of Instance Flags. Client instances should usually set the flag iris::IrisInstance::UNIQUIFY.

8.14.2.2 IrisInstance() [2/2]

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

Construct a new Iris instance using an IrisInstantiationContext.

Parameters

context	A context object that provides the necessary information to instantiate an instance.

8.14.3 Member Function Documentation

8.14.3.1 addCallback_IRIS_INSTANCE_REGISTRY_CHANGED()

```
void iris::IrisInstance::addCallback_IRIS_INSTANCE_REGISTRY_CHANGED (
            EventCallbackFunction f )
```

Add callback function for IRIS_INSTANCE_REGISTRY_CHANGED.

8.14.3.2 findEventSources()

```
std::vector< EventSourceInfo > iris::IrisInstance::findEventSources (
            const std::string & instancePathFilter = "all" )
```

Find all event sources in the system.
See filterInstanceInfos() (IrisObjects.h) for instancePathFilter semantics.

8.14.3.3 findEventSourcesAndFields()

```
void iris::IrisInstance::findEventSourcesAndFields (
            const std::string & spec,
            std::vector< EventStreamInfo > & eventStreamInfosOut,
            InstanceId defaultInstId = IRIS_UINT64_MAX )
```

Find specific event sources in the system.
Find all event sources in the system and/or in the instance defined by defaultInstId matching wildcard patterns.
All matching event sources are added to eventStreamInfosOut which is not cleared beforehand.
The following fields in each EventStreamInfo element are set to the meta-info of the events source: sInstId, evSrcId, evSrcName, fields, hasFields and eventSourceInfo
No event streams are created. The output is suitable as the eventStreamInfos argument for eventBuffer_create(). Alternatively, individual event streams can be created using eventStream_create() by looping over eventStream↩InfosOut.
The set of returned event sources is defined by the filters specified in "spec" which has the following format:

- [∼]EVENT_SOURCE ["(" [FIELD_OR_OPTION ["+" FIELD_OR_OPTION] ...] ")"] [":" ...]
- EVENT_SOURCE is a wildcard pattern matching on strings of the form <instance_path>.<event_source_↩name> (for all instances in the system) and on strings <event_source_name> for event sources of default↩InstId.
- FIELD_OR_OPTION is either a wildcard pattern matching on field names of the selected event sources, or it is of the format OPT=VAL setting option OPT to value VAL. Use *(+OPT=VAL) to set option and still emit all fields.*
- *Use ∼EVENT_SOURCE to remove any previously matched event sources. The adding and removing event sources is executed in the specified order, so usually removes should come at the end. This makes it easy to enable events using wildrads and then exclude certain events. Example: ∗:∼∗UTLB*: Enable all events in the system except all UTLB related events.
- Likewise, use ∼FIELD to remove any previously selected fields. When the first FIELD is a negative field matching starts with all fields.

Examples:

- ∗.INST:∗.CORE_WRITES (Trace INST and CORE_WRITES on all cores.)
- ∗.INST(PC+DISASS) (Only trace PC and disassembly of INST.)
- ∗:∼∗SEMIHOSTING∗:∼∗UTLB∗ (Enable all trace sources in the whole system except semihosting and UTLB related traces.)
- ∗.TRACE_DATA_FMT_V1_1(∗+bufferSize=100) (Enable trace stream in FMT1.1 format with buffer size 100.)

This may throw:

- E_syntax_error: Syntax error in spec (like missing closing parenthesis).
- E_unknown_event_source: A pattern in "evSrcName" did not match any instance and/or even source name.
- E_unknown_event_field: A pattern in "fields" did not match an field for its event source.

8.14.3.4 findInstanceInfos()

```
std::vector< InstanceInfo > iris::IrisInstance::findInstanceInfos (
            const std::string & instancePathFilter = "all" )
```

Find instance infos of all instances in the system.
This function uses instance info data cached in this instance. The cache can be cleared with clearCachedMetaInfo().
See filterInstanceInfos() (IrisObjects.h) for instancePathFilter semantics.

8.14.3.5 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.

8.14.3.6 getInstanceId()

```
InstanceId iris::IrisInstance::getInstanceId (
            const std::string & instName )
```

Get instance id for a specifid instance name.
If no such instance is known IrisErrorException(E_unknown_instance_name) is thrown.
This information is cached in this instance. The cache can be cleared with clearCachedMetaInfo().

Returns

Instance id.

8.14.3.7 getInstanceInfo() [1/2]

```
InstanceInfo iris::IrisInstance::getInstanceInfo (
            const std::string & instancePathFilter )
```

Get instance info of a specific instance in the system.
This function expects either a correct instance path or a pattern which just matches a single instance, for example "core" which always returns the first core, regardless of the number of cores in the system. If no instance is found or if more than one instances are found, IrisErrorException(E_unknown_instance_name) is thrown.
This function should only be used when the instance name is known upfront, or to get access to the first core only. Use findInstanceInfos() to discover arbitrary instances.
This function uses instance info data cached in this instance. The cache can be cleared with clearCachedMetaInfo().
See filterInstanceInfos() (IrisObjects.h) for instancePathFilter semantics.

8.14.3.8 getInstanceInfo() [2/2]

```
const InstanceInfo & iris::IrisInstance::getInstanceInfo (
            InstanceId instId )
```

Get InstanceInfo including properties for a specific instId.
This information is cached in this instance. The cache can be cleared with clearCachedMetaInfo().

Returns

InstanceInfo (including properties) for instId. Throws IrisErrorException(E_unknown_instance_id) if instId is unknown.

8.14.3.9 getInstanceList()

```
const std::vector< InstanceInfo > & iris::IrisInstance::getInstanceList ( )
```

Get list of InstanceInfos of all instances in the system, including properties.
Note that the index into the returned list is generally not the InstanceId. Use getInstanceInfo(instId) to get the InstanceInfo for a specific instance id.
This information is cached in this instance. The cache can be cleared with clearCachedMetaInfo().

Returns

InstanceInfos (including properties) for all instances in the system.

8.14.3.10 getInstanceName() [1/2]

```
const std::string & iris::IrisInstance::getInstanceName ( ) const  [inline]
```

Get the instance name of this instance. This is valid after registerInstance() returns.

Returns

The instance name of this instance. This is the same as the name parameter passed to the constructor or registerInstance() unless this instance was registered with the UNIQUIFY flag set and the name was modified to make it unique.

8.14.3.11 getInstanceName() [2/2]

```
std::string iris::IrisInstance::getInstanceName (
            InstanceId instId )
```

Get instance name for a specifid instId.
This function does not throw. It returns "instance.<instId>" for unknown instIds.
This information is cached in this instance. The cache can be cleared with clearCachedMetaInfo().

Returns

instance name or "instance.<instId>" instId is unknown.

8.14.3.12 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.

8.14.3.13 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.

8.14.3.14 getPropertyMap()

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

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

8.14.3.15 getRemoteIrisInterface()

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

Get the remote Iris interface.

Returns

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

8.14.3.16 getResourceId()

```
ResourceId iris::IrisInstance::getResourceId (
            InstanceId instId,
            const std::string & resourceSpec )
```

Get resource id for a specific resource.
See resourceRead() for semantics of resourceSpec.
Throws an error when resource is not found.

Returns

Resource id.

8.14.3.17 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()`.

8.14.3.18 irisCallNoThrow()

```
IrisCppAdapter & iris::IrisInstance::irisCallNoThrow ( )  [inline]
```

Get an IrisCppAdapter to call an Iris function of any other instance.
When an Iris function returns an error response, this adapter returns the error code and does not throw an exception.
Usage:

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

8.14.3.19 irisCallThrow()

```
IrisCppAdapter & iris::IrisInstance::irisCallThrow ( )  [inline]
```

Get an IrisCppAdapter to call an Iris function of any other instance. When an Iris function returns an error response, this adapter always throws an exception. Usage:

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

8.14.3.20 isRegistered()

```
bool iris::IrisInstance::isRegistered ( ) const  [inline]
```

Return true iff we are registered as an instance (= we have a valid instance id).

8.14.3.21 isValidEvBufId()

```
bool iris::IrisInstance::isValidEvBufId (
            EventBufferId evBufId ) const
```

Check whether event buffer id is valid.
This function is use to validate event buffer ids.

Returns

Returns true iff evBufId is a valid event buffer id.

8.14.3.22 publishCppInterface()

```
template<class T >
void iris::IrisInstance::publishCppInterface (
            const std::string & interfaceName,
            T * pointer,
            const std::string & jsonDescription )  [inline]
```

Publish a C++ interface XYZ through a new instance_getCppInterfaceXYZ() function.
Null pointers are silently ignored. An interface previously registered under the same name is silently overwritten.

Parameters

interfaceName	Class name or interface name of the interface to be published. This must be a C identifier without namespaces etc. The interface can betreieved with "instance_getCppInterface<interfaceName>()".
pointer	Pointer to the C++ class instance implementing this interface.
jsonDescription	Text for FunctionInfo.description. This must be a valid JSON string without enclosing quotes. This text is amended by generic notes aboud the compatibility of C++ pointers which are valid for every C++ interface.

8.14.3.23 registerEventCallback() [1/3]

```
void iris::IrisInstance::registerEventCallback (
            EventCallbackDelegate delegate,
            const std::string & name,
            const std::string & description,
            const std::string & dlgInstanceTypeStr )  [inline]
```

Register a general event callback using an EventCallbackDelegate.

Parameters

delegate	EventCallbackDelegate to call to handle the function.
name	Name of the function as it will be published.
description	Description of this event callback function.
dlgInstanceTypeStr	The name of the delegate type. This is only used for logging purposes.

8.14.3.24 registerEventCallback() [2/3]

```
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

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

8.14.3.25 registerEventCallback() [3/3]

```
template<class T >
void iris::IrisInstance::registerEventCallback (
            T * instance,
            const std::string & name,
            const std::string & description,
            void(T::*)(IrisReceivedRequest &) memberFunctionPtr,
            const std::string & instanceTypeStr )  [inline]
```

Register a general event callback.
Event callbacks have the same signature, only the description is different.

Parameters

instance	An instance of class T on which to call the member function.
name	Name of the function as it will be published.
description	Description of this event callback function.
memberFunctionPtr	Pointer to the C++ implementation of the function.
instanceTypeStr	The name of class T. This is only used for logging purposes.

8.14.3.26 registerFunction()

```
template<class T >
void iris::IrisInstance::registerFunction (
            T * instance,
            const std::string & name,
            void(T::*)(IrisReceivedRequest &) memberFunctionPtr,
            const std::string & functionInfoJson,
            const std::string & instanceTypeStr )  [inline]
```

Register an Iris function implementation.
The following macro can be used instead of calling this function to avoid specifying the function name twice↩
: irisRegisterFunction(instancePtr, instanceType, functionName, implFunctionName, functionInfoJson)

Parameters

instance	An instance of class T on which to call the member function.

Parameters

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.

8.14.3.27 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: • `"client."` • `"component."` • `"framework."`
flags	A bitwise OR of Instance Flags. Client instances should usually set the flag iris::IrisInstance::UNIQUIFY.

8.14.3.28 resourceRead()

```
uint64_t iris::IrisInstance::resourceRead (
            InstanceId instId,
            const std::string & resourceSpec )
```

Read numeric resource and return its value.
Resource spec may be:

- <resource_name>[.<child_name>...]
- <resource_group>.<resource_name>[.<child_name>...]
- tag:<tag> (e.g. "tag:isInstructionCounter" or "tag:isPc")
- crn:<canonical_register_number_in_decimal> (usage: resourceRead(instId, "crn:" + std::to_string(iris::Elf↩ Dwarf::ARM_R0)), see iris/IrisElfDwarfArm.h, consider using resourceReadCrn() instead)
- rscId:<resourceId> (fallback in case resourceId is already known, consider using irisCallThrow()->resource_read() instead)

If the resoure is not found or could not be read the appropriate error is thrown. If the resource is not a numeric resource E_type_mismatch is thrown.
This is a convenience function, intended to make reading well-known registers easy (e.g. PC, instruction counter). This intentionally does not handle the generic case (string registers, wide registers) to keep the usage simple. Use resource_read() to read any register which does not fit this function.
The resource meta-information is cached in this instance, but the value is not. The cache can be cleared with clearCachedMetaInfo().

Returns

Resource value.

8.14.3.29 resourceReadCrn()

```
uint64_t iris::IrisInstance::resourceReadCrn (
            InstanceId instId,
            uint64_t canonicalRegisterNumber )  [inline]
```

Read numeric resource and return its value (using the canonical register number aka DWARF register id).
See resourceRead() and the "crn:" case within.

Returns

Resource value.

8.14.3.30 resourceReadStr()

```
std::string iris::IrisInstance::resourceReadStr (
            InstanceId instId,
            const std::string & resourceSpec )
```

Read string resource, or read other resources as string.
Numeric resource values get converted to a string according to the type and bitWidth. Errors in the result.error fields are returned as string. noValue resources return an empty string.
See resourceRead() for semantics of resourceSpec, errors and limitations.

8.14.3.31 resourceWrite()

```
void iris::IrisInstance::resourceWrite (
            InstanceId instId,
            const std::string & resourceSpec,
            uint64_t value )
```

Write numeric resource.
If the resource is not a numeric resource E_type_mismatch is thrown.
See resourceRead() for semantics of resourceSpec, errors and limitations.

8.14.3.32 resourceWriteCrn()

```
void iris::IrisInstance::resourceWriteCrn (
            InstanceId instId,
            uint64_t canonicalRegisterNumber,
            uint64_t value )  [inline]
```

Write numeric resource by canonical register number (aka DWARF register id).
See resourceWrite() for semantics.

8.14.3.33 resourceWriteStr()

```
void iris::IrisInstance::resourceWriteStr (
            InstanceId instId,
            const std::string & resourceSpec,
            const std::string & value )
```

Write string resource, or write numeric resource from string.
If the resource is not a string the value is converted to a numeric value according to the resource type.
See resourceRead() for semantics of resourceSpec, errors and limitations.

8.14.3.34 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.

8.14.3.35 sendResponse()

```
void iris::IrisInstance::sendResponse (
            const uint64_t * response )  [inline]
```

Send a response to the remote Iris interface.
Call this from the function implementations registered with registerFunction() or irisRegisterFunction().

Parameters

response	The Iris response message to send.

8.14.3.36 setCallback_IRIS_SHUTDOWN_LEAVE()

```
void iris::IrisInstance::setCallback_IRIS_SHUTDOWN_LEAVE (
            EventCallbackFunction f )
```

Set callback function for IRIS_SHUTDOWN_LEAVE.

8.14.3.37 setCallback_IRIS_SIMULATION_TIME_EVENT()

```
void iris::IrisInstance::setCallback_IRIS_SIMULATION_TIME_EVENT (
            EventCallbackFunction f )
```

Set callback function for IRIS_SIMULATION_TIME_EVENT.

8.14.3.38 setConnectionInterface()

```
void iris::IrisInstance::setConnectionInterface (
            IrisConnectionInterface * connection_interface )
```

Set the remote connection interface.
Used to set the IrisConnectionInterface if it was not set in the constructor.

Parameters

connection_interface	The interface used to connect to an Iris simulation.

8.14.3.39 setPendingSyncStepResponse()

```
void iris::IrisInstance::setPendingSyncStepResponse (
            RequestId requestId,
            EventBufferId evBufId )
```

Set pending response to a step_syncStep() call.
This function is called when the step_syncStep() function is called and the response is delivered when the simulation time stopped.

8.14.3.40 setProperty()

```
template<class T >
void iris::IrisInstance::setProperty (
            const std::string & propertyName,
            const T & propertyValue )  [inline]
```

Set/add instance property.
This creates a new property or overwrites an existing one.
Properties (name and value) are defined by the instance that has them. Properties are not to be confused with parameters, whose values are defined by clients or by parent components and some parameters might change at runtime.
Properties are exposed by the function instance_getProperties(). This should only ever be called upon initialization, before other components have a chance to call instance_getProperties(). Properties are constant and should not be changed at runtime. T can be bool, uint64_t, int64_t, or std::string.

Parameters

propertyName	Name of the property.
propertyValue	Value of the property.

8.14.3.41 setThrowOnError()

```
void iris::IrisInstance::setThrowOnError (
            bool throw_on_error )  [inline]
```

Set default error behavior for irisCall().

Parameters

throw_on_error	If true, calls made using irisCall() that respond with an error response will throw an exception. This is the same behavior as irisCallThrow(). If false, calls made using irisCall() that respond with an error response will return the error code and not throw an exception. This is the same behavior as irisCallNoThrow().

8.14.3.42 simulationTimeDisableEvents()

```
void iris::IrisInstance::simulationTimeDisableEvents ( )
```

Disable the internal reception of IRIS_SIMULATION_TIME_EVENT events for performance reasons (e.g. during synchronous stepping).
The callback set with setCallback_IRIS_SIMULATION_TIME_EVENT() will no longer be called.
Internal IRIS_SIMULATION_TIME_EVENTs will automatically be re-enabled as soon as one of the other simulationTime∗() functions is called.
This function throws Iris errors.

8.14.3.43 simulationTimeIsRunning()

```
bool iris::IrisInstance::simulationTimeIsRunning ( )
```

Return true iff simulation is currently running.
Note that this information is always out of date if there is another simulation controller.
This function throws Iris errors.

8.14.3.44 simulationTimeRun()

```
void iris::IrisInstance::simulationTimeRun ( )
```

Run simulation time and wait until simulation time started running.

Does not wait until model stopped again. See simulationTimeRunUntilStop().
This function throws Iris errors.

8.14.3.45 simulationTimeRunUntilStop()

```
void iris::IrisInstance::simulationTimeRunUntilStop ( )
```

Run simulation time and wait until simulation time stopped again.
This function throws Iris errors.

8.14.3.46 simulationTimeStop()

```
void iris::IrisInstance::simulationTimeStop ( )
```

Stop simulation time and wait until simulation time stopped.
This function throws Iris errors.

8.14.3.47 unpublishCppInterface()

```
void iris::IrisInstance::unpublishCppInterface (
            const std::string & interfaceName )  [inline]
```

Unpublish a previously published C++ interface.
After calling this function the corresponding instance_getCppInterface...() function is no longer available. This is silently ignored If the interface was not previously published.

Parameters

interfaceName	Class name or interface name of the interface to be unpublished.

8.14.3.48 unregisterInstance()

```
IrisErrorCode iris::IrisInstance::unregisterInstance ( )
```

Unregister this instance.
Iris calls must not be made after the instance has been unregistered.
The documentation for this class was generated from the following file:

- IrisInstance.h

8.15 iris::IrisInstanceBreakpoint Class Reference

Breakpoint add-on for IrisInstance.
`#include <IrisInstanceBreakpoint.h>`

Public Member Functions

- void addCondition (const std::string &name, const std::string &type, const std::string &description, const std::vector< std::string > bpt_types=std::vector< std::string >())

  Add an optional component-specific condition that can be configured by clients.
- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- const BreakpointInfo * getBreakpointInfo (BreakpointId bptId) const

  Get BreakpointInfo for a breakpoint id.
- **IrisInstanceBreakpoint** (IrisInstance *irisInstance=nullptr)
- void notifyBreakpointHit (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId)

  Notify clients that a code breakpoint was hit.
- void notifyBreakpointHitData (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId, uint64_t accessAddr, uint64_t accessSize, const std::string &accessRw, const std::vector< uint64_t > &data)

Notify clients that a data breakpoint was hit.

- void notifyBreakpointHitRegister (BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId, const std::string &accessRw, const std::vector< uint64_t > &data)

  Notify clients that a register breakpoint was hit.
- void setBreakpointDeleteDelegate (BreakpointDeleteDelegate delegate)

  Set breakpoint delete delegate for all breakpoints deleted by this instance.
- void setBreakpointSetDelegate (BreakpointSetDelegate delegate)

  Set breakpoint set delegate for all breakpoints set by this instance.
- void setEventHandler (IrisInstanceEvent *handler)

  Set the event handler used to notify the clients that enable the IRIS_BREAKPOINT_HIT event.

8.15.1 Detailed Description

Breakpoint add-on for IrisInstance.
Instances use this class to support breakpoint functionality.
It implements all Iris breakpoint*() functions and maintains the breakpoint information that is set by breakpoint_set() and is exposed by breakpoint_getList().
Example usage:

```
irisInstanceBpt = new iris::IrisInstanceBreakpoint(irisInstance);
irisInstanceBpt->setBreakpointSetDelegate(bptSetDel);           // Use this delegate for breakpoint set.
irisInstanceBpt->setBreakpointDeleteDelegate(bptDeleteDel);     // Use this delegate for breakpoint delete.
// When a breakpoint is hit, notify the instances that enable the IRIS_BREAKPOINT_HIT event.
irisInstanceBpt->setEventHandler(irisInstanceEvent);
```

See DummyComponent.h for a working example.

8.15.2 Member Function Documentation

8.15.2.1 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

name	The name of the condition.
type	The type of the value that clients set to configure the condition.
description	A description of the condition.
bpt_types	A list of breakpoint types that this condition can be applied to. An empty list indicates all types.

8.15.2.2 attachTo()

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

Attach this IrisInstance add-on to a specific IrisInstance.
Only use this method if nullptr was passed to the constructor.

Parameters

irisInstance	The IrisInstance to attach to.

8.15.2.3 getBreakpointInfo()

```
const BreakpointInfo * iris::IrisInstanceBreakpoint::getBreakpointInfo (
            BreakpointId bptId ) const
```

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.

8.15.2.4 notifyBreakpointHit()

```
void iris::IrisInstanceBreakpoint::notifyBreakpointHit (
            BreakpointId bptId,
            uint64_t time,
            uint64_t pc,
            MemorySpaceId pcSpaceId )
```

Notify clients that a code breakpoint was hit.
It notifies clients by emitting an `IRIS_BREAKPOINT_HIT` event.

Parameters

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.

8.15.2.5 notifyBreakpointHitData()

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

Notify clients that a data breakpoint was hit.
It notifies clients by emitting an `IRIS_BREAKPOINT_HIT` event.

Parameters

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.

Parameters

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.

8.15.2.6 notifyBreakpointHitRegister()

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

Notify clients that a register breakpoint was hit.
It notifies clients by emitting an `IRIS_BREAKPOINT_HIT` event.

Parameters

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.
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.

8.15.2.7 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.

8.15.2.8 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.

8.15.2.9 setEventHandler()

```
void iris::IrisInstanceBreakpoint::setEventHandler (
            IrisInstanceEvent * handler )
```

Set the event handler used to notify the clients that enable the IRIS_BREAKPOINT_HIT event.
All breakpoint events are normal events and are handled through the same mechanism as other events.
The documentation for this class was generated from the following file:

- IrisInstanceBreakpoint.h

8.16 iris::IrisInstanceBuilder Class Reference

Builder interface to populate an IrisInstance with registers, memory etc.
`#include <IrisInstanceBuilder.h>`

Classes

- class AddressTranslationBuilder

  Used to set metadata for an address translation.
- class EventSourceBuilder

  Used to set metadata on an EventSource.
- class FieldBuilder

  Used to set metadata on a register field resource.
- class MemorySpaceBuilder

  Used to set metadata for a memory space.
- class ParameterBuilder

  Used to set metadata on a parameter.
- class RegisterBuilder

  Used to set metadata on a register resource.
- class SemihostingManager

  semihosting_apis IrisInstanceBuilder semihosting APIs
- class TableBuilder

  Used to set metadata for a table.
- class TableColumnBuilder

  Used to set metadata for a table column.

Public Member Functions

- AddressTranslationBuilder addAddressTranslation (MemorySpaceId inSpaceId, MemorySpaceId outSpace↩ Id, const std::string &description)

  Add an address translation.
- void **addBreakpointCondition** (const std::string &name, const std::string &type, const std::string &description, const std::vector< std::string > bpt_types=std::vector< std::string >())

  Add an optional component-specific condition.
- EventSourceBuilder addEventSource (const std::string &name, bool isHidden=false)

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

  Add metadata for an event source that uses an IrisEventEmitter.
- MemorySpaceBuilder addMemorySpace (const std::string &name)

  Add metadata for one memory space.

- RegisterBuilder addNoValueRegister (const std::string &name, const std::string &description, const std::string &format)

  Add metadata for one noValue resource.
- ParameterBuilder addParameter (const std::string &name, uint64_t bitWidth, const std::string &description)

  Add numeric parameter.
- RegisterBuilder addRegister (const std::string &name, uint64_t bitWidth, const std::string &description, uint64_t addressOffset=IRIS_UINT64_MAX, uint64_t canonicalRn=IRIS_UINT64_MAX)

  Add metadata for one numeric register resource.
- ParameterBuilder addStringParameter (const std::string &name, const std::string &description)

  Add string parameter.
- RegisterBuilder addStringRegister (const std::string &name, const std::string &description)

  Add metadata for one string register resource.
- TableBuilder addTable (const std::string &name)

  Add metadata for one table.
- void beginResourceGroup (const std::string &name, const std::string &description, uint64_t subRscId↩ Start=IRIS_UINT64_MAX, const std::string &cname=std::string())

  Begin a new resource group.
- ParameterBuilder enhanceParameter (ResourceId rscId)

  Get ParameterBuilder to enhance a parameter.
- RegisterBuilder enhanceRegister (ResourceId rscId)

  Get RegisterBuilder to enhance register.
- void finalizeRegisterReadEvent ()
- void finalizeRegisterUpdateEvent ()

  Finalize set up of an IrisEventEmitter.
- const BreakpointInfo ∗ getBreakpointInfo (BreakpointId bptId)

  Get the breakpoint information for a given breakpoint.
- IrisInstanceEvent ∗ getIrisInstanceEvent ()
- const ResourceInfo & getResourceInfo (ResourceId rscId)

  Get ResourceInfo 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 pcSpace↩ Id, 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 resetRegisterReadEvent ()

  Reset the active register read event.
- void resetRegisterUpdateEvent ()

  Reset the active register update event.
- template<IrisErrorCode(∗)(const BreakpointInfo &) FUNC>
  void setBreakpointDeleteDelegate ()

  Set the delegate that is called when a breakpoint is deleted.
- void setBreakpointDeleteDelegate (BreakpointDeleteDelegate delegate)

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

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

  Set the delegate that is called when a breakpoint is deleted.
- template<IrisErrorCode(*)(BreakpointInfo &) FUNC>
  void setBreakpointSetDelegate ()

  Set the delegate that is called when a breakpoint is set.
- void setBreakpointSetDelegate (BreakpointSetDelegate delegate)

  Set the delegate that is called when a breakpoint is set.
- template<typename T , IrisErrorCode(T::*)(BreakpointInfo &) METHOD>
  void setBreakpointSetDelegate (T *instance)

  Set the delegate that is called when a breakpoint is set.
- template<IrisErrorCode(*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FUNC>
  void setDefaultAddressTranslateDelegate ()

  Set the default address translation function for all subsequently added memory spaces.
- void setDefaultAddressTranslateDelegate (MemoryAddressTranslateDelegate delegate=MemoryAddressTranslateDelegate())

  Set the default address translation function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) METHOD>
  void setDefaultAddressTranslateDelegate (T *instance)

  Set the default address translation function for all subsequently added memory spaces.
- template<IrisErrorCode(*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) FUNC>
  void setDefaultEsCreateDelegate ()

  Set the delegate that helps to create a new event stream for the simulation-specific event.
- void setDefaultEsCreateDelegate (EventStreamCreateDelegate delegate)

  Set the delegate that helps to create a new event stream for the simulation-specific event.
- template<typename T , IrisErrorCode(T::*)(EventStream *&, const EventSourceInfo &, const std::vector< std::string > &) METHOD>
  void setDefaultEsCreateDelegate (T *instance)

  Set the delegate that helps to create a new event stream for the simulation-specific event.
- template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>
  void setDefaultGetMemorySidebandInfoDelegate ()

  Set the default sideband info function for all subsequently added memory spaces.
- void setDefaultGetMemorySidebandInfoDelegate (MemoryGetSidebandInfoDelegate delegate)

  Set the default sideband info function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) METHOD>
  void setDefaultGetMemorySidebandInfoDelegate (T *instance)

  Set the default sideband info function for all subsequently added memory spaces.
- template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) FUNC>
  void setDefaultMemoryReadDelegate ()

  Set the default read function for all subsequently added memory spaces.
- void setDefaultMemoryReadDelegate (MemoryReadDelegate delegate=MemoryReadDelegate())

  Set the default read function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) METHOD>
  void setDefaultMemoryReadDelegate (T *instance)

  Set the default read function for all subsequently added memory spaces.
- template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>
  void setDefaultMemoryWriteDelegate ()

  Set default write function for all subsequently added memory spaces.
- void setDefaultMemoryWriteDelegate (MemoryWriteDelegate delegate=MemoryWriteDelegate())

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

- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
  void setDefaultMemoryWriteDelegate (T *instance)

  Set the default write function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) READER, IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) WRITER>
  void setDefaultResourceDelegates (T *instance)

  Set both read and write resource delegates if they are defined in the same class.
- template<IrisErrorCode(*)(const ResourceInfo &, ResourceReadResult &) FUNC>
  void setDefaultResourceReadDelegate ()

  Set default read access function for all subsequently added resources.
- void setDefaultResourceReadDelegate (ResourceReadDelegate delegate=ResourceReadDelegate())

  Set default read access function for all subsequently added resources.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
  void setDefaultResourceReadDelegate (T *instance)

  Set default read access function for all subsequently added resources.
- template<IrisErrorCode(*)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  void setDefaultResourceWriteDelegate ()

  Set default write access function for all subsequently added resources.
- void setDefaultResourceWriteDelegate (ResourceWriteDelegate delegate=ResourceWriteDelegate())

  Set default write access function for all subsequently added resources.
- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &) METHOD>
  void setDefaultResourceWriteDelegate (T *instance)

  Set default write access function for all subsequently added resources.
- template<IrisErrorCode(*)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
  void setDefaultTableReadDelegate ()

  Set the default table read function for all subsequently added tables.
- template<typename T , IrisErrorCode(T::*)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) METHOD>
  void setDefaultTableReadDelegate (T *instance)

  Set the default table read function for all subsequently added tables.
- void setDefaultTableReadDelegate (TableReadDelegate delegate=TableReadDelegate())

  Set the default table read function for all subsequently added tables.
- template<IrisErrorCode(*)(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
  void setDefaultTableWriteDelegate ()

  Set the default table write function for all subsequently added tables.
- template<typename T , IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, TableWriteResult &) METHOD>
  void setDefaultTableWriteDelegate (T *instance)

  Set the default table write function for all subsequently added tables.
- void setDefaultTableWriteDelegate (TableWriteDelegate delegate=TableWriteDelegate())

  Set the default table write function for all subsequently added tables.
- template<IrisErrorCode(*)(bool &) FUNC>
  void setExecutionStateGetDelegate ()

  Set the delegate to get the execution state for this instance.
- void setExecutionStateGetDelegate (PerInstanceExecutionStateGetDelegate delegate)

  Set the delegate to get the execution state for this instance.
- template<typename T , IrisErrorCode(T::*)(bool &) METHOD>
  void setExecutionStateGetDelegate (T *instance)

  Set the delegate to get the execution state for this instance.
- template<IrisErrorCode(*)(bool) FUNC>
  void setExecutionStateSetDelegate ()

  Set the delegate to set the execution state for this instance.
- void setExecutionStateSetDelegate (PerInstanceExecutionStateSetDelegate delegate=PerInstanceExecutionStateSetDelegate

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

- template<typename T , IrisErrorCode(T::*)(bool) METHOD>
  void setExecutionStateSetDelegate (T *instance)

  Set the delegate to set the execution state for this instance.
- template<IrisErrorCode(*)(const std::vector< uint64_t > &, uint64_t) FUNC>
  void setLoadImageDataDelegate ()

  Set the delegate to load an image from the data provided.
- void setLoadImageDataDelegate (ImageLoadDataDelegate delegate=ImageLoadDataDelegate())

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

  Set the delegate to load an image from the data provided.
- template<IrisErrorCode(*)(const std::string &) FUNC>
  void setLoadImageFileDelegate ()

  Set the delegate to load an image from a file.
- void setLoadImageFileDelegate (ImageLoadFileDelegate delegate=ImageLoadFileDelegate())

  Set the delegate to load an image from a file.
- template<typename T , IrisErrorCode(T::*)(const std::string &) METHOD>
  void setLoadImageFileDelegate (T *instance)

  Set the delegate to load an image from a file.
- void setNextSubRscId (uint64_t nextSubRscId)

  Set the rscId that will be used for the next resource to be added.
- void setPropertyCanonicalMsnScheme (const std::string &canonicalMsnScheme)

  Set the memory.canonicalMsnScheme instance property.
- void setPropertyCanonicalRnScheme (const std::string &canonicalRnScheme)

  Set the register.canonicalRnScheme instance property.
- EventSourceBuilder setRegisterReadEvent (const std::string &name, const std::string &description=std::string())

  Add a new register read event source.
- EventSourceBuilder setRegisterReadEvent (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)

  Add a new register read event source.
- EventSourceBuilder setRegisterUpdateEvent (const std::string &name, const std::string &description=std::string())

  Add a new register update event source.
- EventSourceBuilder setRegisterUpdateEvent (const std::string &name, IrisRegisterEventEmitterBase &event_emitter)

  Add a new register update event source.
- template<IrisErrorCode(*)(uint64_t &, const std::string &) FUNC>
  void setRemainingStepGetDelegate ()

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

  Set the delegate to get the remaining steps for this instance.
- template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>
  void setRemainingStepGetDelegate (T *instance)

  Set the delegate to get the remaining steps for this instance.
- template<IrisErrorCode(*)(uint64_t, const std::string &) FUNC>
  void setRemainingStepSetDelegate ()

  Set the delegate to set the remaining steps for this instance.
- void setRemainingStepSetDelegate (RemainingStepSetDelegate delegate=RemainingStepSetDelegate())

  Set the delegate to set the remaining steps for this instance.
- template<typename T , IrisErrorCode(T::*)(uint64_t, const std::string &) METHOD>
  void setRemainingStepSetDelegate (T *instance)

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

- template<IrisErrorCode(*)(uint64_t &, const std::string &) FUNC>
  void setStepCountGetDelegate ()

  Set the delegate to get the step count for this instance.
- void setStepCountGetDelegate (StepCountGetDelegate delegate=StepCountGetDelegate())

  Set the delegate to get the step count for this instance.
- template<typename T , IrisErrorCode(T::*)(uint64_t &, const std::string &) METHOD>
  void setStepCountGetDelegate (T *instance)

  Set the delegate to get the step count for this instance.
- void setTag (ResourceId rscId, const std::string &tag)

  Set a tag for a specific resource.

- void setGetCurrentDisassemblyModeDelegate (GetCurrentDisassemblyModeDelegate delegate)

  disass_apis IrisInstanceBuilder disassembler APIs
- template<typename T , IrisErrorCode(T::*)(std::string &) METHOD>
  void **setGetCurrentDisassemblyModeDelegate** (T *instance)
- void **setGetDisassemblyDelegate** (GetDisassemblyDelegate delegate)

  Set the delegate to get the disassembly of a chunk of memory.
- template<typename T , IrisErrorCode(T::*)(uint64_t, const std::string &, MemoryReadResult &, uint64_t, uint64_t, std::vector< DisassemblyLine > &) METHOD>
  void **setGetDisassemblyDelegate** (T *instance)
- template<IrisErrorCode(*)(uint64_t, const std::string &, MemoryReadResult &, uint64_t, uint64_t, std::vector< DisassemblyLine > &) FUNC>
  void **setGetDisassemblyDelegate** ()
- void **setDisassembleOpcodeDelegate** (DisassembleOpcodeDelegate delegate)

  Set the delegate to get the disassembly of Opcode.
- template<typename T , IrisErrorCode(T::*)(const std::vector< uint64_t > &, uint64_t, const std::string &, DisassembleContext &, DisassemblyLine &) METHOD>
  void **setDisassembleOpcodeDelegate** (T *instance)
- template<IrisErrorCode(*)(const std::vector< uint64_t > &, uint64_t, const std::string &, DisassembleContext &, DisassemblyLine &) FUNC>
  void **setDisassembleOpcodeDelegate** ()
- void **addDisassemblyMode** (const std::string &name, const std::string &description)

  Add a disassembly mode.

- void setDbgStateSetRequestDelegate (DebuggableStateSetRequestDelegate delegate=DebuggableStateSetRequestDelegate

  debuggable_state_apis IrisInstanceBuilder debuggable state APIs
- template<typename T , IrisErrorCode(T::*)(bool) METHOD>
  void setDbgStateSetRequestDelegate (T *instance)

  Set the delegate to set the debuggable state request flag for this instance.
- template<IrisErrorCode(*)(bool) FUNC>
  void setDbgStateSetRequestDelegate ()

  Set the delegate to set the debuggable state request flag for this instance.
- void setDbgStateGetAcknowledgeDelegate (DebuggableStateGetAcknowledgeDelegate delegate=DebuggableStateGetAckno

  Set the delegate to get the debuggable state acknowledge flag for this instance.
- template<typename T , IrisErrorCode(T::*)(bool &) METHOD>
  void setDbgStateGetAcknowledgeDelegate (T *instance)

  Set the delegate to get the debuggable state acknowledge flag for this instance.
- template<IrisErrorCode(*)(bool &) FUNC>
  void setDbgStateGetAcknowledgeDelegate ()

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

- template<typename T , IrisErrorCode(T::*)(bool) SET_REQUEST, IrisErrorCode(T::*)(bool &) GET_ACKNOWLEDGE>
  void setDbgStateDelegates (T *instance)

  Set both the debuggable state delegates.
- void **setCheckpointSaveDelegate** (CheckpointSaveDelegate delegate=CheckpointSaveDelegate())

  Delegates for checkpointing.
- template<typename T , IrisErrorCode(T::*)(const std::string &) METHOD>
  void **setCheckpointSaveDelegate** (T *instance)
- void **setCheckpointRestoreDelegate** (CheckpointRestoreDelegate delegate=CheckpointRestoreDelegate())
- template<typename T , IrisErrorCode(T::*)(const std::string &) METHOD>
  void **setCheckpointRestoreDelegate** (T *instance)

- SemihostingManager enableSemihostingAndGetManager ()

  Enable semihosting functionality for this instance and get a manager object to make use of it.

8.16.1 Detailed Description

Builder interface to populate an IrisInstance with registers, memory etc.
See DummyComponent.h for a working example.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 IrisInstanceBuilder()

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

Construct an IrisInstanceBuilder for an Iris instance.

Parameters

iris_instance	The instance to build.

8.16.3 Member Function Documentation

8.16.3.1 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.

8.16.3.2 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.

8.16.3.3 setDbgStateDelegates()

```
template<typename T , IrisErrorCode(T::*)(bool) SET_REQUEST, IrisErrorCode(T::*)(bool &) GET_ACKNOWLEDGE>
void iris::IrisInstanceBuilder::setDbgStateDelegates (
            T * instance )  [inline]
```

Set both the debuggable state delegates.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
    iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateDelegates<MyClass,
                              &MyClass::setRequest,
                              &MyClass::getAcknowledgeFlag>(&myInstanceOfMyClass);
```

Template Parameters

T	Class that defines both a debuggable state request set and a get acknowledge delegate method.
SET_REQUEST	A method of class T which is a debuggable state request set delegate.
GET_ACKNOWLEDGE	A method of class T which is a debuggable state get acknowledge delegate.

Parameters

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

8.16.3.4 setDbgStateGetAcknowledgeDelegate() [1/3]

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

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

Usage:

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

Template Parameters

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

8.16.3.5 setDbgStateGetAcknowledgeDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setDbgStateGetAcknowledgeDelegate (
            DebuggableStateGetAcknowledgeDelegate delegate = DebuggableStateGetAcknowledgeDelegate()
)  [inline]
```

Set the delegate to get the debuggable state acknowledge flag for this instance.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
};
MyClass myInstanceOfMyClass;
iris::DebuggableStateGetAcknowledgeDelegate delegate =
    iris::DebuggableStateGetAcknowledgeDelegate::make<MyClass,
       &MyClass::getAcknowledgeFlag>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateGetAcknowledgeDelegate(delegate);
```

Parameters

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

8.16.3.6 setDbgStateGetAcknowledgeDelegate() [3/3]

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

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

```
class MyClass
{
    ...
    iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateGetAcknowledgeDelegate<MyClass, &MyClass::getAcknowledgeFlag>(&myInstanceOfMyClass);
```

Template Parameters

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.

8.16.3.7 setDbgStateSetRequestDelegate() [1/3]

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

Set the delegate to set the debuggable state request flag for this instance.
Usage:

```
iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
```

```
builder->setDbgStateSetRequestDelegate<&setRequestFlag>();
```

Template Parameters

FUNC	Global function to call to set the debuggable state request flag.

8.16.3.8 setDbgStateSetRequestDelegate() [2/3]

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

debuggable_state_apis IrisInstanceBuilder debuggable state APIs
Set the delegate to set the debuggable state request flag for this instance.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
};
MyClass myInstanceOfMyClass;
iris::DebuggableStateSetRequestDelegate delegate =
    iris::DebuggableStateSetRequestDelegate::make<MyClass, &MyClass::setRequestFlag>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateSetRequestDelegate(delegate);
```

Parameters

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

8.16.3.9 setDbgStateSetRequestDelegate() [3/3]

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

Set the delegate to set the debuggable state request flag for this instance.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateSetRequestDelegate<MyClass, &MyClass::setRequestFlag>(&myInstanceOfMyClass);
```

Template Parameters

T	Class that defines a debuggable state request set delegate method.
METHOD	A method of class T which is a debuggable state request set delegate.

Parameters

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

8.16.3.10 setDefaultTableReadDelegate() [1/3]

```
template<IrisErrorCode(*)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
void iris::IrisInstanceBuilder::setDefaultTableReadDelegate ( )  [inline]
```

Set the default table read function for all subsequently added tables.
Tables that do not explicitly override the access function using
`addTable(...).setReadDelegate(...)`
will use this delegate.
Usage:

```
iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index,
                              uint64_t count, iris::TableReadResult &result);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate<&readTable>();
builder->addTable(...); // Uses readTable
```

Template Parameters

FUNC	Global function to call to read a table.

8.16.3.11 setDefaultTableReadDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const TableInfo &, uint64_t, uint64_t, TableRead↩
Result &) METHOD>
void iris::IrisInstanceBuilder::setDefaultTableReadDelegate (
              T * instance )  [inline]
```

Set the default table read function for all subsequently added tables.
Tables that do not explicitly override the access function using
`addTable(...).setReadDelegate(...)`
will use this delegate.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index,
                                  uint64_t count, iris::TableReadResult &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate<MyClass, &MyClass::readTable>(&myInstanceOfMyClass);
builder->addTable(...); // Uses readTable
```

Template Parameters

T	Class that defines a table read delegate method.
METHOD	A method of class T which is a table read delegate.

Parameters

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

8.16.3.12 setDefaultTableReadDelegate() [3/3]

```
void iris::IrisInstanceBuilder::setDefaultTableReadDelegate (
              TableReadDelegate delegate = TableReadDelegate() )  [inline]
```

Set the default table read function for all subsequently added tables.
Tables that do not explicitly override the access function using
`addTable(...).setReadDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩

not_implemented for all requests.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode readTable(const iris::TableInfo &tableInfo, uint64_t index,
                                  uint64_t count, iris::TableReadResult &result);
};
MyClass myInstanceOfMyClass;
iris::TableReadDelegate delegate =
    iris::TableReadDelegate::make<MyClass, &MyClass::readTable>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableReadDelegate(delegate);
builder->addTable(...); // Uses readTable
```

Parameters

delegate	Delegate object to call to read a table.

8.16.3.13 setDefaultTableWriteDelegate() [1/3]

```
template<IrisErrorCode(*)(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
void iris::IrisInstanceBuilder::setDefaultTableWriteDelegate ( )  [inline]
```

Set the default table write function for all subsequently added tables.
Tables that do not explicitly override the access function using
`addTable(...).setWriteDelegate(...)`
will use this delegate.
Usage:

```
iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
                               const iris::TableRecords &records,
                               iris::TableWriteResult &result);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableWriteDelegate<&writeTable>();
builder->addTable(...); // Uses writeTable
```

Template Parameters

FUNC	Global function to call to write a table.

8.16.3.14 setDefaultTableWriteDelegate() [2/3]

```
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
`addTable(...).setWriteDelegate(...)`
will use this delegate.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
                                   const iris::TableRecords &records,
                                   iris::TableWriteResult &result);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableWriteDelegate<MyClass, &MyClass::writeTable>(&myInstanceOfMyClass);
builder->addTable(...); // Uses writeTable
```

Template Parameters

T	Class that defines a table write delegate method.

Template Parameters

METHOD	A method of class T which is a table write delegate.

Parameters

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

8.16.3.15 setDefaultTableWriteDelegate() [3/3]

```
void iris::IrisInstanceBuilder::setDefaultTableWriteDelegate (
            TableWriteDelegate delegate = TableWriteDelegate() )  [inline]
```

Set the default table write function for all subsequently added tables.
Tables that do not explicitly override the access function using
`addTable(...).setWriteDelegate(...)`
will use this delegate.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode writeTable(const iris::TableInfo &tableInfo,
                                   const iris::TableRecords &records,
                                   iris::TableWriteResult &result);
};
MyClass myInstanceOfMyClass;
iris::TableWriteDelegate delegate =
    iris::TableWriteDelegate::make<MyClass, &MyClass::writeTable>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDefaultTableWriteDelegate(delegate);
builder->addTable(...); // Uses writeTable
```

Parameters

delegate	Delegate object to call to write a table.

8.16.3.16 setExecutionStateGetDelegate() [1/3]

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

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

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

Template Parameters

FUNC	Global function to call to get the execution state.

8.16.3.17 setExecutionStateGetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setExecutionStateGetDelegate (
            PerInstanceExecutionStateGetDelegate delegate )  [inline]
```

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

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

```
class MyClass
{
    ...
    iris::IrisErrorCode getState(bool &execution_enabled);
};
MyClass myInstanceOfMyClass;
iris::PerInstanceExecutionStateGetDelegate delegate =
    iris::PerInstanceExecutionStateGetDelegate::make<MyClass, &MyClass::getState>(&myInstanceOfMyClass);
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateGetDelegate(delegate);
```

Parameters

delegate	Delegate object to call to get the execution state.

8.16.3.18 setExecutionStateGetDelegate() [3/3]

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

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

```
class MyClass
{
    ...
    iris::IrisErrorCode getState(bool &execution_enabled);
};
MyClass myInstanceOfMyClass;
iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setExecutionStateGetDelegate<MyClass, &MyClass::getState>(&myInstanceOfMyClass);
```

Template Parameters

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

Parameters

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

8.16.3.19 setExecutionStateSetDelegate() [1/3]

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

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

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

Template Parameters

FUNC	Global function to call to set the execution state.

8.16.3.20 setExecutionStateSetDelegate() [2/3]

```
void iris::IrisInstanceBuilder::setExecutionStateSetDelegate (
            PerInstanceExecutionStateSetDelegate delegate = PerInstanceExecutionStateSetDelegate()
)  [inline]
```

Set the delegate to set the execution state for this instance.
Passing an empty delegate (the default argument) restores the default implementation which always returns E_↩ not_implemented for all requests.
Usage:

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

Parameters

delegate	Delegate object to call to set the execution state.

8.16.3.21 setExecutionStateSetDelegate() [3/3]

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

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

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

Template Parameters

T	Class that defines a set execution state delegate method.
METHOD	A method of class T which is a set execution state delegate.

Parameters

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

8.16.3.22 setGetCurrentDisassemblyModeDelegate()

```
void iris::IrisInstanceBuilder::setGetCurrentDisassemblyModeDelegate (
            GetCurrentDisassemblyModeDelegate delegate )  [inline]
```

disass_apis IrisInstanceBuilder disassembler APIs
Set the delegates to get the current disassembly mode
The documentation for this class was generated from the following file:

- IrisInstanceBuilder.h

8.17 iris::IrisInstanceCheckpoint Class Reference

Checkpoint add-on for IrisInstance.

`#include <IrisInstanceCheckpoint.h>`

Public Member Functions

- void attachTo (IrisInstance *iris_instance_)

  Attach this IrisInstance add-on to a specific IrisInstance.
- **IrisInstanceCheckpoint** (IrisInstance *iris_instance=nullptr)
- void setCheckpointRestoreDelegate (CheckpointRestoreDelegate delegate)

  Set checkpoint restore delegate for all checkpoints related to this instance.
- void setCheckpointSaveDelegate (CheckpointSaveDelegate delegate)

  Set checkpoint save delegate for all checkpoints related to this instance.

8.17.1 Detailed Description

Checkpoint add-on for IrisInstance.

8.17.2 Member Function Documentation

8.17.2.1 attachTo()

```
void iris::IrisInstanceCheckpoint::attachTo (
            IrisInstance * iris_instance_ )
```

Attach this IrisInstance add-on to a specific IrisInstance.
Only use this method if nullptr was passed to the constructor.

Parameters

*iris_↩ instance_*	The IrisInstance to attach to.

8.17.2.2 setCheckpointRestoreDelegate()

```
void iris::IrisInstanceCheckpoint::setCheckpointRestoreDelegate (
            CheckpointRestoreDelegate delegate )
```

Set checkpoint restore delegate for all checkpoints related to this instance.

Parameters

delegate	A CheckpointRestoreDelegate to call when restoring a checkpoint.

8.17.2.3 setCheckpointSaveDelegate()

```
void iris::IrisInstanceCheckpoint::setCheckpointSaveDelegate (
            CheckpointSaveDelegate delegate )
```

Set checkpoint save delegate for all checkpoints related to this instance.

Parameters

delegate	A CheckpointSaveDelegate to call when saving a checkpoint.

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

- IrisInstanceCheckpoint.h

8.18 iris::IrisInstanceDebuggableState Class Reference

Debuggable-state add-on for IrisInstance.
`#include <IrisInstanceDebuggableState.h>`

Public Member Functions

- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- **IrisInstanceDebuggableState** (IrisInstance *iris_instance=nullptr)
- void setGetAcknowledgeDelegate (DebuggableStateGetAcknowledgeDelegate delegate)

  Set the get acknowledge flag delegate.
- void setSetRequestDelegate (DebuggableStateSetRequestDelegate delegate)

  Set the set request flag delegate.

8.18.1 Detailed Description

Debuggable-state add-on for IrisInstance.

8.18.2 Member Function Documentation

8.18.2.1 attachTo()

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

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

irisInstance	The IrisInstance to attach to.

8.18.2.2 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.

8.18.2.3 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

8.19 iris::IrisInstanceDisassembler Class Reference

Disassembler add-on for IrisInstance.
`#include <IrisInstanceDisassembler.h>`

Public Member Functions

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

  Add a disassembly mode.
- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- IrisInstanceDisassembler (IrisInstance *irisInstance=nullptr)

  Construct an IrisInstanceDisassembler.
- void setDisassembleOpcodeDelegate (DisassembleOpcodeDelegate delegate)

  Set the delegate to get the disassembly of Opcode.
- void setGetCurrentModeDelegate (GetCurrentDisassemblyModeDelegate delegate)

  Set the delegate to get the current disassembly mode.
- void setGetDisassemblyDelegate (GetDisassemblyDelegate delegate)

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

8.19.1 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

8.20 iris::IrisInstanceEvent Class Reference

Event add-on for IrisInstance.
`#include <IrisInstanceEvent.h>`

Classes

- struct EventSourceInfoAndDelegate

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

  Contains information for a single proxy EventSource.

Public Member Functions

- uint64_t addEventSource (const EventSourceInfoAndDelegate &info)

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

  Add metadata for an event source.
- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstanceEvent add-on to a specific IrisInstance.
- void deleteEventSource (const std::string &eventName)

  Delete metadata for an event source.
- void eventBufferClear (EventBufferId evBufId)

  Clear event buffer.
- const uint64_t * eventBufferGetSyncStepResponse (EventBufferId evBufId, RequestId requestId)

  Get response to step_syncStep(), containing event data.
- IrisInstanceEvent (IrisInstance *irisInstance=nullptr)

  Construct an IrisInstanceEvent add-on.
- bool isValidEvBufId (EventBufferId evBufId) const

  Check whether event buffer id is valid.
- void setDefaultEsCreateDelegate (EventStreamCreateDelegate delegate)

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

Friends

- struct **EventBuffer**

8.20.1 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.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 IrisInstanceEvent()

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

Construct an IrisInstanceEvent add-on.

Parameters

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

8.20.3 Member Function Documentation

8.20.3.1 addEventSource() [1/2]

```
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.

8.20.3.2 addEventSource() [2/2]

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

Add metadata for an event source.

Parameters

name	The name of the event source.
isHidden	If true, this event source is hidden. The EventSourceInfo is not included in the list of event sources returned by event_getEventSources() but can still be accessed by event_getEventSource() if the client knows the name of the hidden event.

Returns

A reference to an object which keeps the metadata and event-specific delegates (if applicable) for this event. The reference is valid until the next call to addEventSource().

8.20.3.3 attachTo()

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

Attach this IrisInstanceEvent add-on to a specific IrisInstance.
This should only be used if no instance was attached when this object was constructed.

Parameters

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

8.20.3.4 deleteEventSource()

```
void iris::IrisInstanceEvent::deleteEventSource (
            const std::string & eventName )
```

Delete metadata for an event source.

Parameters

eventName	The name of the event source.

8.20.3.5 eventBufferClear()

```
void iris::IrisInstanceEvent::eventBufferClear (
            EventBufferId evBufId )
```

Clear event buffer.
This is separate from eventBufferGetSyncStepResponse() so the message writer can be used to send the message without taking an unnecessary copy.

Parameters

evBufId	The event buffer which is to be cleared.

8.20.3.6 eventBufferGetSyncStepResponse()

```
const uint64_t * iris::IrisInstanceEvent::eventBufferGetSyncStepResponse (
            EventBufferId evBufId,
            RequestId requestId )
```

Get response to step_syncStep(), containing event data.

Parameters

evBufId	The data of this event buffer is returned. This is set beforehand with step_syncStepSetup().
requestId	This is the request id of the original step_syncStep() for which this function generates the answer.

Returns

Response message to step_syncStep() call, containing the event data.

8.20.3.7 isValidEvBufId()

```
bool iris::IrisInstanceEvent::isValidEvBufId (
            EventBufferId evBufId ) const
```

Check whether event buffer id is valid.
This function is use to validate event buffer ids.

Returns

Returns true iff evBufId is a valid event buffer id.

8.20.3.8 setDefaultEsCreateDelegate()

```
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

8.21 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 addHidenParameter (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.

8.21.1 Detailed Description

A builder class to construct instantiation parameter metadata.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 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.

8.21.3 Member Function Documentation

8.21.3.1 addBooleanParameter()

```
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

name	Name of the parameter.
description	Description of the parameter.

Returns

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

8.21.3.2 addHiddenBooleanParameter()

```
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

name	Name of the parameter.
description	Description of the parameter.

Returns

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

8.21.3.3 addHiddenStringParameter()

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

Add a new hidden string parameter.

Parameters

name	Name of the parameter.
description	Description of the parameter.

Returns

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

8.21.3.4 addHidenParameter()

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

Add a new hidden numeric parameter.

Parameters

name	Name of the parameter.
bitWidth	Width of the parameter in bits.
description	Description of the parameter.

Returns

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

8.21.3.5 addParameter()

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

Add a new numeric parameter.

Parameters

name	Name of the parameter.
bitWidth	Width of the parameter in bits.
description	Description of the parameter.

Returns

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

8.21.3.6 addStringParameter()

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

Add a new string parameter.

Parameters

name	Name of the parameter.
description	Description of the parameter.

Returns

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

8.21.3.7 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.

8.21.3.8 getParameterInfo()

```
const std::vector< ResourceInfo > & iris::IrisInstanceFactoryBuilder::getParameterInfo ( )
const  [inline]
```

Get all ResourceInfo for non-hidden parameters.

Returns

A vector of ResourceInfo. Iterators for this vector are invalidated if a new non-hidden parameter is added.

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

- IrisInstanceFactoryBuilder.h

8.22 iris::IrisInstanceImage Class Reference

Image loading add-on for IrisInstance.

```
#include <IrisInstanceImage.h>
```

Public Member Functions

- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- IrisInstanceImage (IrisInstance *irisInstance=0)

  Construct a new IrisInstanceImage.
- void setLoadImageDataDelegate (ImageLoadDataDelegate delegate)

  Set image loading from (pushed/pulled) data delegate.
- void setLoadImageFileDelegate (ImageLoadFileDelegate delegate)

  Set image loading from file delegate.

Static Public Member Functions

- static IrisErrorCode readFileData (const std::string &fileName, std::vector< uint64_t > &data, uint64_t &count)

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

8.22.1 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_clearMetaInfo↩ List().

See DummyComponent.h for a working example.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 IrisInstanceImage()

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

Construct a new IrisInstanceImage.

Parameters

irisInstance	The IrisInstance to attach this add-on to.

8.22.3 Member Function Documentation

8.22.3.1 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.

8.22.3.2 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.

Parameters

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.

8.22.3.3 setLoadImageDataDelegate()

```
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.

8.22.3.4 setLoadImageFileDelegate()

```
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

8.23 iris::IrisInstanceImage_Callback Class Reference

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

`#include <IrisInstanceImage.h>`

Public Member Functions

- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- IrisInstanceImage_Callback (IrisInstance *irisInstance=0)

  Construct an IrisInstanceImage_Callback add-on.
- uint64_t openImage (const std::string &fileName)

  Open an image for read.

Protected Member Functions

- void **impl_image_loadDataRead** (IrisReceivedRequest &request)

  Implementation of the Iris function image_loadDataRead().

8.23.1 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.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 IrisInstanceImage_Callback()

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

Construct an IrisInstanceImage_Callback add-on.

Parameters

irisInstance	The IrisInstance to attach this add-on to.

8.23.3 Member Function Documentation

8.23.3.1 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.

8.23.3.2 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

8.24 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.

8.24.1 Detailed Description

Memory add-on for IrisInstance.
This class is used by instances to expose their own memory.
It implements all Iris memory*() functions. It maintains/implements two main things:

- Memory space meta information (exposed by memory_getMemorySpaces()).
- Forwarding memory read/write and address translate accesses to functions with a simple prototype which is easy to implement by components, hiding a lot of the complexity of memory_read(), memory_write(), and memory_translateAddress().

Example usage:

```
irisInstance = new iris::IrisInstance(irisInterface, instanceName);
 irisInstanceMemory = new iris::IrisInstanceMemory(irisInstance);
// Use these delegates for read/write for all following memory spaces.
irisInstanceMemory->setDefaultReadDelegate<DummyComponent, &DummyComponent::readMemory>(this);
irisInstanceMemory->setDefaultWriteDelegate<DummyComponent, &DummyComponent::writeMemory>(this);
irisInstanceMemory->addMemorySpace("Memory"); // Add a memory address space.
```

See setDefaultReadDelegate() for an example of read/write delegates.
See DummyComponent.h for a working example.

See also

IrisInstanceBuilder memory APIs

8.24.2 Constructor & Destructor Documentation

8.24.2.1 IrisInstanceMemory()

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

Construct an IrisInstanceMemory.
Optionally attaches to an IrisInstance.

Parameters

irisInstance	The IrisInstance to attach to.

8.24.3 Member Function Documentation

8.24.3.1 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

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.

Returns

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

8.24.3.2 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.

8.24.3.3 attachTo()

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

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

irisInstance	The IrisInstance to attach to.

8.24.3.4 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.

8.24.3.5 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.

8.24.3.6 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.

8.24.3.7 setDefaultWriteDelegate()

```
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

8.25 iris::IrisInstancePerInstanceExecution Class Reference

Per-instance execution control add-on for IrisInstance.
`#include <IrisInstancePerInstanceExecution.h>`

Public Member Functions

- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstancePerInstanceExecution add-on to a specific IrisInstance.
- IrisInstancePerInstanceExecution (IrisInstance *irisInstance=nullptr)

  Construct an IrisInstancePerInstanceExecution add-on.
- void setExecutionStateGetDelegate (PerInstanceExecutionStateGetDelegate delegate)

  Set the delegate for getting execution state.
- void setExecutionStateSetDelegate (PerInstanceExecutionStateSetDelegate delegate)

  Set the delegate for setting execution state.

8.25.1 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.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 IrisInstancePerInstanceExecution()

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

Construct an IrisInstancePerInstanceExecution add-on.

Parameters

irisInstance	The IrisInstance to attach this add-on to.

8.25.3 Member Function Documentation

8.25.3.1 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.

8.25.3.2 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.

8.25.3.3 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

8.26 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 startSubRsc↩Id=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 calcHierarchicalNames (std::vector< ResourceInfo > &resourceInfos)

  Calculate hierarchicalName and hierarchicalCName for all RegisterInfos.
- static void makeNamesHierarchical (std::vector< ResourceInfo > &resourceInfos)

  Make name and cname of RegisterInfos hierarchical.

Protected Member Functions

- void **impl_resource_getList** (IrisReceivedRequest &request)
- void **impl_resource_getListOfResourceGroups** (IrisReceivedRequest &request)
- void **impl_resource_getResourceInfo** (IrisReceivedRequest &request)
- void **impl_resource_read** (IrisReceivedRequest &request)
- void **impl_resource_write** (IrisReceivedRequest &request)

8.26.1 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.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 IrisInstanceResource()

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

Construct an IrisInstanceResource.
Optionally attaches to an IrisInstance.

Parameters

irisInstance	The IrisInstance to attach to.

8.26.3 Member Function Documentation

8.26.3.1 addResource()

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

Add a new resource.

Parameters

type	The type of the resource. This should be one of: • "numeric" • "numericFp" • "String" • "noValue"
name	The name of the resource.
description	A human-readable description of the resource.

Returns

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

8.26.3.2 attachTo()

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

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

irisInstance	The IrisInstance to attach to.

8.26.3.3 beginResourceGroup()

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

Begin a new resource group.
This method has these effects:

- Add a resource group (only if it does not yet exist).
- Assign all resources that are added through addResource() calls to this group.

Parameters

name	The name of the resource group.
description	A description of this resource group.
startSub↩ RscId	If not IRIS_UINT64_MAX start counting from this subRscId when new resources are added.
cname	A C identifier version of the resource name if different from *name*.

8.26.3.4 calcHierarchicalNames()

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

Calculate hierarchicalName and hierarchicalCName for all RegisterInfos.
RegisterInfo.hierarchicalName and RegisterInfo.hierarchicalCName are set to the hierarchical name for each resource such that a child register X of parent FLAGS gets hierarchicalName=FLAGS.X and hierarchicalCName=FLAGS_X, similarly also for deeper nesting levels.
This functionality is not an Iris interface but just a convenience function for simple clients. The ResourceInfos returned by IrisInstance::getResourceInfo*() have already hierarchical names.
No errors are generated for missing parent resources. parentRscId links to missing parent resources are silently ignored. The intended usage is to call this function on a list containing all resources or all registers of an instance, so that all parent links can be resolved.

Parameters

resourceInfos	Array of all ResourceInfos of an instance.

8.26.3.5 getResourceInfo()

```
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.

8.26.3.6 makeNamesHierarchical()

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

Make name and cname of RegisterInfos hierarchical.
Legacy function overwriting ResourceInfo.name/cname.
This function calculates the hierarchical names using calcHierarchicalNames() and then copies ResourceInfo.hierarchicalName/hierarchicalCName into ResourceInfo.name/cname info, respectively.
Consider using calcHierarchicalNames() which does not alter the original resource information.

Parameters

resourceInfos	Array of all ResourceInfos of an instance.

8.26.3.7 setNextSubRscId()

```
void iris::IrisInstanceResource::setNextSubRscId (
            ResourceId nextSubRscId_ ) [inline]
```

Set next subRscId.

Resources that are added following this call are assigned subRscIds starting at nextSubRscId unless nextSubRscId is IRIS_UINT64_MAX, in which case all further resources are assigned IRIS_UINT64_MAX as the subRscId

Parameters

*nextSubRsc↩ Id_*	Next subRscId

8.26.3.8 setTag()

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

Set a tag for a specific resource.

Parameters

rsc↩ Id	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

8.27 iris::IrisInstanceSemihosting Class Reference

Public Member Functions

- void attachTo (IrisInstance *iris_instance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- void **enableExtensions** ()

  Instances that support semihosting extensions should call this method to enable the IRIS_SEMIHOSTING_↩ CALL_EXTENSION event.
- **IrisInstanceSemihosting** (IrisInstance *iris_instance=nullptr, IrisInstanceEvent *inst_event=nullptr)
- std::vector< uint8_t > readData (uint64_t fDes, uint64_t max_size=0, uint64_t flags=semihost::DEFAULT)

  Read data for a given file descriptor.
- std::pair< bool, uint64_t > semihostedCall (uint64_t operation, uint64_t parameter)

  Allow a client to perform a semihosting extension defined by operation and parameter.
- void setEventHandler (IrisInstanceEvent *handler)

  Set the corresponding IrisInstanceEvent object to use to manage semihosting events.
- void **unblock** ()

  Request premature exit from any blocking requests that are currently blocked.
- bool **writeData** (uint64_t fDes, const uint8_t *data, uint64_t size)

8.27.1 Member Function Documentation

8.27.1.1 attachTo()

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

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

iris_instance	The instance to attach to.

8.27.1.2 readData()

```
std::vector< uint8_t > iris::IrisInstanceSemihosting::readData (
            uint64_t fDes,
            uint64_t max_size = 0,
            uint64_t flags = semihost::DEFAULT )
```

Read data for a given file descriptor.
The exact behavior of this method depends on the value of the max_size and flags parameters. If the NONBLOCK flag is set, the method returns immediately with whatever data is already buffered, if any. If NONBLOCK is not set, the method blocks until data is available. Iris messages continue to be processed while this methods blocks. If max_size is not zero, then at most max_size bytes will be returned.

Parameters

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.

8.27.1.3 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

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. This meaning of this parameter is defined by the operation.

Returns

A pair of (bool success, uint64_t result). If status is true, a client performed the function and returned the value in result. If status is false, no client performed the function and result is 0.

8.27.1.4 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

handler	The event add-on for this Iris instance.

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

- IrisInstanceSemihosting.h

8.28 iris::IrisInstanceSimulation Class Reference

An IrisInstance add-on that adds simulation functions for the SimulationEngine instance.
`#include <IrisInstanceSimulation.h>`

Public Member Functions

- void attachTo (IrisInstance *iris_instance)

  Attach this IrisInstance add-on to a specific IrisInstance.
- void enterPostInstantiationPhase ()

  Move from the pre-instantiation to the post-instantiation phase.
- IrisInstanceSimulation (IrisInstance *iris_instance=nullptr, IrisConnectionInterface *connection_interface=nullptr)

  Construct an IrisInstanceSimulation add-on.
- void notifySimPhase (uint64_t time, IrisSimulationPhase phase)

  Emit an IRIS_SIM_PHASE event for the supplied phase.*
- void registerSimEventsOnGlobalInstance ()

  Register all simulation engine events as proxy events on the global iris instance.
- void setConnectionInterface (IrisConnectionInterface *connection_interface_)

  Set the IrisConnectionInterface to use for the instantiation.
- void setEventHandler (IrisInstanceEvent *handler)

  Set up IRIS_SIM_PHASE events.*
- template<IrisErrorCode(*)(std::vector< ResourceInfo > &) FUNC>
  void setGetParameterInfoDelegate (bool cache_result=true)

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

  Set the getParameterInfo() delegate.
- template<typename T , IrisErrorCode(T::*)(std::vector< ResourceInfo > &) METHOD>
  void setGetParameterInfoDelegate (T *instance, bool cache_result=true)

  Set the getParameterInfo() delegate.
- template<IrisErrorCode(*)(InstantiationResult &) FUNC>
  void setInstantiateDelegate ()

Set the instantiate() delegate.

- void setInstantiateDelegate (SimulationInstantiateDelegate delegate)

  Set the instantiate() delegate.

- template<typename T , IrisErrorCode(T::*)(InstantiationResult &) METHOD>
  void setInstantiateDelegate (T *instance)

  Set the instantiate() delegate.

- template<IrisErrorCode(*)() FUNC>
  void setRequestShutdownDelegate ()

  Set the requestShutdown() delegate.

- void setRequestShutdownDelegate (SimulationRequestShutdownDelegate delegate)

  Set the requestShutdown() delegate.

- template<typename T , IrisErrorCode(T::*)() METHOD>
  void setRequestShutdownDelegate (T *instance)

  Set the requestShutdown() delegate.

- template<IrisErrorCode(*)(const IrisSimulationResetContext &) FUNC>
  void setResetDelegate ()

  Set the reset() delegate.

- void setResetDelegate (SimulationResetDelegate delegate)

  Set the reset() delegate.

- template<typename T , IrisErrorCode(T::*)(const IrisSimulationResetContext &) METHOD>
  void setResetDelegate (T *instance)

  Set the reset() delegate.

- template<IrisErrorCode(*)(const InstantiationParameterValue &) FUNC>
  void setSetParameterValueDelegate ()

  Set the setParameterValue() delegate.

- void setSetParameterValueDelegate (SimulationSetParameterValueDelegate delegate)

  Set the setParameterValue() delegate.

- template<typename T , IrisErrorCode(T::*)(const InstantiationParameterValue &) METHOD>
  void setSetParameterValueDelegate (T *instance)

  Set the setParameterValue() delegate.

Static Public Member Functions

- static std::string getSimulationPhaseDescription (IrisSimulationPhase phase)

  Get dexcription string for a simulation phase.

- static std::string getSimulationPhaseName (IrisSimulationPhase phase)

  Get name of the enum symbol for name.

8.28.1 Detailed Description

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

8.28.2 Constructor & Destructor Documentation

8.28.2.1 IrisInstanceSimulation()

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

Construct an IrisInstanceSimulation add-on.

Parameters

iris_instance	The IrisInstance to attach this add-on to.
connection_interface	The connection interface that will be used when the simulation is instantiated.

8.28.3 Member Function Documentation

8.28.3.1 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.

8.28.3.2 enterPostInstantiationPhase()

```
void iris::IrisInstanceSimulation::enterPostInstantiationPhase ( )
```

Move from the pre-instantiation to the post-instantiation phase.
This effects which functions are published. Only call this function if the simulation is instantiated outside of Iris. This object automatically enters post-instantiation phase when the simulation is successfully instantiated by an Iris call to simulation_instantiate().

8.28.3.3 getSimulationPhaseDescription()

```
static std::string iris::IrisInstanceSimulation::getSimulationPhaseDescription (
            IrisSimulationPhase phase )  [static]
```

Get dexcription string for a simulation phase.
This is a free form single line text ending with a dot.

8.28.3.4 getSimulationPhaseName()

```
static std::string iris::IrisInstanceSimulation::getSimulationPhaseName (
            IrisSimulationPhase phase )  [static]
```

Get name of the enum symbol for name.
Example: getSimulationPhaseName(IRIS_SIM_PHASE_INIT) returns "IRIS_SIM_PHASE_INIT".

8.28.3.5 notifySimPhase()

```
void iris::IrisInstanceSimulation::notifySimPhase (
            uint64_t time,
            IrisSimulationPhase phase )
```

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.

8.28.3.6 registerSimEventsOnGlobalInstance()

```
void iris::IrisInstanceSimulation::registerSimEventsOnGlobalInstance ( )
```

Register all simulation engine events as proxy events on the global iris instance.
This function should be called after an iris instance has been attached to IrisInstanceSimulation object (IrisInstanceSimulation::attachTo). This will ensure that the simulation engine iris instance i.e. iris_instance is available to call the register API. This function should be called after event handler has been set for IrisInstanceSimulation object (IrisInstanceSimulation::setEventHandler). This will ensure that all simulation engine events are available in simulation engine event handler. This function should be called after an IrisIntanceEvent has been attached to iris_instance (IrisInstanceEvent::attachTo). This will ensure that event functions have been registered on simulation engine iris instance.

8.28.3.7 setConnectionInterface()

```
void iris::IrisInstanceSimulation::setConnectionInterface (
            IrisConnectionInterface * connection_interface_ )  [inline]
```

Set the IrisConnectionInterface to use for the instantiation.
This will be passed to the instantiate() delegate when the simulation is instantiated.

8.28.3.8 setEventHandler()

```
void iris::IrisInstanceSimulation::setEventHandler (
            IrisInstanceEvent * handler )
```

Set up IRIS_SIM_PHASE∗ events.

Parameters

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

8.28.3.9 setGetParameterInfoDelegate() [1/3]

```
template<IrisErrorCode(*)(std::vector< ResourceInfo > &) FUNC>
void iris::IrisInstanceSimulation::setGetParameterInfoDelegate (
            bool cache_result = true )  [inline]
```

Set the getParameterInfo() delegate.
Set the delegate to a global function.

Template Parameters

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.

8.28.3.10 setGetParameterInfoDelegate() [2/3]

```
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.

8.28.3.11 setGetParameterInfoDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(std::vector< ResourceInfo > &) METHOD>
void iris::IrisInstanceSimulation::setGetParameterInfoDelegate (
            T * instance,
            bool cache_result = true )  [inline]
```

Set the getParameterInfo() delegate.
Set the delegate to call a method in class T.

Template Parameters

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.

8.28.3.12 setInstantiateDelegate() [1/3]

```
template<IrisErrorCode(*)(InstantiationResult &) FUNC>
void iris::IrisInstanceSimulation::setInstantiateDelegate ( )  [inline]
```

Set the instantiate() delegate.
Set the delegate to a global function.

Template Parameters

FUNC	A function that is an instantiate delegate.

8.28.3.13 setInstantiateDelegate() [2/3]

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

Set the instantiate() delegate.

Parameters

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

8.28.3.14 setInstantiateDelegate() [3/3]

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

Set the instantiate() delegate.
Set the delegate to call a method in class T.

Template Parameters

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 *T* on which *METHOD* should be called.

8.28.3.15 setRequestShutdownDelegate() [1/3]

```
template<IrisErrorCode(*)() 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.

8.28.3.16 setRequestShutdownDelegate() [2/3]

```
void iris::IrisInstanceSimulation::setRequestShutdownDelegate (
            SimulationRequestShutdownDelegate delegate )  [inline]
```

Set the requestShutdown() delegate.

Parameters

delegate	A delegate object that will be called to request that the simulation be shut down.

8.28.3.17 setRequestShutdownDelegate() [3/3]

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

Set the requestShutdown() delegate.
Set the delegate to call a method in class T.

Template Parameters

T	Class that defines a requestShutdown delegate method.

Template Parameters

METHOD	A method of class *T* that is a requestShutdown delegate.

Parameters

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

8.28.3.18 setResetDelegate() [1/3]

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

Set the reset() delegate.
Set the delegate to a global function.

Template Parameters

FUNC	A function that is a reset delegate.

8.28.3.19 setResetDelegate() [2/3]

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

Set the reset() delegate.

Parameters

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

8.28.3.20 setResetDelegate() [3/3]

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

Set the reset() delegate.
Set the delegate to call a method in class T.

Template Parameters

T	Class that defines a reset delegate method.
METHOD	A method of class *T* that is a reset delegate.

Parameters

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

8.28.3.21 setSetParameterValueDelegate() [1/3]

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

Set the setParameterValue() delegate.
Set the delegate to a global function.

Template Parameters

FUNC	A function that is a setParameterValue delegate.

8.28.3.22 setSetParameterValueDelegate() [2/3]

```
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.

8.28.3.23 setSetParameterValueDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const InstantiationParameterValue &) METHOD>
void iris::IrisInstanceSimulation::setSetParameterValueDelegate (
            T * instance )  [inline]
```

Set the setParameterValue() delegate.
Set the delegate to call a method in class T.

Template Parameters

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.

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

- IrisInstanceSimulation.h

8.29 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 registerSimTimeEventsOnGlobalInstance ()

  Register all simulation time events as proxy events on the global iris instance.
- void setEventHandler (IrisInstanceEvent *handler)

  Set the event handler to use to send simulation time-related events.
- template<IrisErrorCode(*)(uint64_t &, uint64_t &, bool &) FUNC>
  void setSimTimeGetDelegate ()

  Set the getTime() delegate.
- void setSimTimeGetDelegate (SimulationTimeGetDelegate delegate)

  Set the getTime() delegate.
- template<typename T , IrisErrorCode(T::*)(uint64_t &, uint64_t &, bool &) METHOD>
  void setSimTimeGetDelegate (T *instance)

  Set the getTime() delegate.
- template<IrisErrorCode(*)() FUNC>
  void setSimTimeRunDelegate ()

  Set the run() delegate.
- void setSimTimeRunDelegate (SimulationTimeRunDelegate delegate)

  Set the run() delegate.
- template<typename T , IrisErrorCode(T::*)() METHOD>
  void setSimTimeRunDelegate (T *instance)

  Set the run() delegate.
- template<IrisErrorCode(*)() FUNC>
  void setSimTimeStopDelegate ()

  Set the stop() delegate.
- void setSimTimeStopDelegate (SimulationTimeStopDelegate delegate)

  Set the stop() delegate.
- template<typename T , IrisErrorCode(T::*)() METHOD>
  void setSimTimeStopDelegate (T *instance)

  Set the stop() delegate.

8.29.1 Detailed Description

Simulation time add-on for IrisInstance.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 IrisInstanceSimulationTime()

```
iris::IrisInstanceSimulationTime::IrisInstanceSimulationTime (
            IrisInstance * iris_instance = nullptr,
            IrisInstanceEvent * inst_event = nullptr )
```

Construct an IrisInstanceSimulationTime add-on.

Parameters

iris_instance	An IrisInstance to attach this add-on to.
inst_event	An IrisInstanceEvent add-on that is already attached to IrisInstance. This is used to set up simulation time events.

8.29.3 Member Function Documentation

8.29.3.1 attachTo()

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

Attach this IrisInstance add-on to a specific IrisInstance.

Parameters

irisInstance	An IrisInstance to attach this add-on to.

8.29.3.2 registerSimTimeEventsOnGlobalInstance()

```
void iris::IrisInstanceSimulationTime::registerSimTimeEventsOnGlobalInstance ( )
```

Register all simulation time events as proxy events on the global iris instance.
This function should be called after an iris instance has been attached to IrisInstanceSimulationTime object (IrisInstanceSimulationTime::attachTo). This will ensure that the simulation time iris instance i.e. iris_↩ instance is available to call the register API. This function should be called after event handler has been set for IrisInstanceSimulationTime object (IrisInstanceSimulationTime::setEventHandler). This will ensure that all simulation time events are available in simulation time event handler. This function should be called after an IrisIntance↩ Event has been attached to iris_instance (IrisInstanceEvent::attachTo). This will ensure that event functions have been registered on simulation time iris instance.

8.29.3.3 setEventHandler()

```
void iris::IrisInstanceSimulationTime::setEventHandler (
            IrisInstanceEvent * handler )
```

Set the event handler to use to send simulation time-related events.

Parameters

handler	An IrisInstanceEvent add-on that is already attached to IrisInstance. This is used to set up simulation time events.

8.29.3.4 setSimTimeGetDelegate() [1/3]

```
template<IrisErrorCode(*)(uint64_t &, uint64_t &, bool &) FUNC>
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate ( )  [inline]
```

Set the getTime() delegate.
Set the delegate to a global function.

Template Parameters

FUNC	A function that is a getTime delegate.

8.29.3.5 setSimTimeGetDelegate() [2/3]

```
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate (
            SimulationTimeGetDelegate delegate )  [inline]
```

Set the getTime() delegate.

Parameters

delegate	A delegate that is called to get the current simulation time.

8.29.3.6 setSimTimeGetDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(uint64_t &, uint64_t &, bool &) METHOD>
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate (
            T * instance )  [inline]
```

Set the getTime() delegate.

Template Parameters

T	Class that defines a getTime delegate method.
METHOD	A method of class *T* that is a getTime delegate.

Parameters

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

8.29.3.7 setSimTimeRunDelegate() [1/3]

```
template<IrisErrorCode(*)() 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.

8.29.3.8 setSimTimeRunDelegate() [2/3]

```
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.

8.29.3.9 setSimTimeRunDelegate() [3/3]

```
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.

8.29.3.10 setSimTimeStopDelegate() [1/3]

```
template<IrisErrorCode(*)() FUNC>
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.

8.29.3.11 setSimTimeStopDelegate() [2/3]

```
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.

8.29.3.12 setSimTimeStopDelegate() [3/3]

```
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.

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

- IrisInstanceSimulationTime.h

8.30 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.

8.30.1 Detailed Description

Step add-on for IrisInstance.
This is used by instances to support stepping functionality.
This class implements all Iris step*() functions.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 IrisInstanceStep()

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

Construct an IrisInstanceStep add-on.

Parameters

irisInstance	The IrisInstance to attach this add-on to.

8.30.3 Member Function Documentation

8.30.3.1 attachTo()

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

Attach this IrisInstanceStep add-on to a specific IrisInstance.
This should only be used if no instance was attached when this object was constructed.

Parameters

irisInstance	The IrisInstance to attach this add-on to.

8.30.3.2 setRemainingStepGetDelegate()

```
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.

8.30.3.3 setRemainingStepSetDelegate()

```
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.

8.30.3.4 setStepCountGetDelegate()

```
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

8.31 iris::IrisInstanceTable Class Reference

Table add-on for IrisInstance.

`#include <IrisInstanceTable.h>`

Classes

- struct TableInfoAndAccess

  Entry in 'tableInfos'.

Public Member Functions

- TableInfoAndAccess & addTableInfo (const std::string &name)

  Add metadata for one table.
- void attachTo (IrisInstance *irisInstance)

  Attach this IrisInstanceTable add-on to a specific IrisInstance.
- IrisInstanceTable (IrisInstance *irisInstance=nullptr)

  Construct an IrisInstanceTable add-on.

- void setDefaultReadDelegate (TableReadDelegate delegate=TableReadDelegate())

  Set the default delegate for reading table data.
- void setDefaultWriteDelegate (TableWriteDelegate delegate=TableWriteDelegate())

  Set the default delegate for writing table data.

8.31.1 Detailed Description

Table add-on for IrisInstance.
This is used by instances to support table functionality.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 IrisInstanceTable()

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

Construct an IrisInstanceTable add-on.

Parameters

irisInstance	The IrisInstance to attach this add-on to.

8.31.3 Member Function Documentation

8.31.3.1 addTableInfo()

```
TableInfoAndAccess & iris::IrisInstanceTable::addTableInfo (
            const std::string & name )
```

Add metadata for one table.

Parameters

name	The name of this table.

Returns

A reference to a TableInfoAndAccess object that can be used to set metadata and access delegates for this table.

8.31.3.2 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

irisInstance	The IrisInstance to attach this add-on to.

8.31.3.3 setDefaultReadDelegate()

```
void iris::IrisInstanceTable::setDefaultReadDelegate (
            TableReadDelegate delegate = TableReadDelegate() )  [inline]
```

Set the default delegate for reading table data.

Parameters

delegate	A delegate object that is called to read table data for tables in the attached instance that did not set a table-specific delegate.

8.31.3.4 setDefaultWriteDelegate()

```
void iris::IrisInstanceTable::setDefaultWriteDelegate (
            TableWriteDelegate delegate = TableWriteDelegate() )  [inline]
```

Set the default delegate for writing table data.

Parameters

delegate	A delegate object that is called to write table data for tables in the attached instance that did not set a table-specific delegate.

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

- IrisInstanceTable.h

8.32 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.
- void getParameter (const std::string &name, std::vector< uint64_t > &value)
  Get the value of a large numeric instantiation parameter.
- template<typename T >
  void getParameter (const std::string &name, T &value)
  Get the value of an instantiation parameter.
- uint64_t getRecommendedInstanceFlags () const
  Get the flags to use when registering the instance being instantiated.
- 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.

8.32.1 Detailed Description

Provides context when instantiating an Iris instance from a factory.

8.32.2 Member Function Documentation

8.32.2.1 error()

```
void void void iris::IrisInstantiationContext::error (
            const std::string & code,
            const char * format,
             ...  )
```

Add an error to the InstantiationResult.

See also

parameterError

Parameters

code	An error code symbol. This should be one of the codes specified for the InstantiationError object.
format	A printf-style format string.
...	Printf substitution arguments.

8.32.2.2 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().

8.32.2.3 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().

8.32.2.4 getParameter() [1/2]

```
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

name	The name of the parameter.
value	A reference to a value of type *T* that receives the value of the named parameter.

8.32.2.5 getParameter() [2/2]

```
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.

Parameters

name	The name of the parameter.
value	A reference to a value of type *T* that receives the value of the named parameter.

8.32.2.6 getRecommendedInstanceFlags()

```
uint64_t iris::IrisInstantiationContext::getRecommendedInstanceFlags ( ) const  [inline]
```

Get the flags to use when registering the instance being instantiated.

Returns

A value to use for the flags argument of IrisInstance::IrisInstance() or IrisInstance::registerInstance().

8.32.2.7 getSubcomponentContext()

```
IrisInstantiationContext * iris::IrisInstantiationContext::getSubcomponentContext (
            const std::string & child_name )
```

Get an IrisInstanceContext pointer for a subcomponent instance.
For example, you might call getSubcomponentContext("cpu0") on the context "component.cluster0" to get the context to instantiate "component.cluster0.cpu0". The object pointed to by the return value is owned by its parent context and has the same lifetime as the parent context.

Parameters

child_name	The name of a child instance.

Returns

A pointer to an IrisInstantiationContext object for the named child.

8.32.2.8 parameterError()

```
void void void void iris::IrisInstantiationContext::parameterError (
            const std::string & code,
            const std::string & parameterName,
            const char * format,
             ...  )
```

Add an error to the InstantiationResult.

See also

error

Parameters

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.

8.32.2.9 parameterWarning()

```
void void iris::IrisInstantiationContext::parameterWarning (
            const std::string & code,
            const std::string & parameterName,
            const char * format,
             ...  )
```

Add a warning to the InstantiationResult.

See also

warning

Parameters

code	An error code symbol. This should be one of the codes specified for the InstantiationError object.
parameterName	The name of the parameter this warning relates to.
format	A printf-style format string.
...	Printf substitution arguments.

8.32.2.10 warning()

```
void iris::IrisInstantiationContext::warning (
            const std::string & code,
            const char * format,
             ...  )
```

Add a warning to the InstantiationResult.

See also

parameterWarning

Parameters

code	An error code symbol. This should be one of the codes specified for the InstantiationError object.
format	A printf-style format string.
...	Printf substitution arguments.

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

- IrisInstantiationContext.h

8.33 iris::IrisParameterBuilder Class Reference

Helper class to construct instantiation parameters.
`#include <IrisParameterBuilder.h>`

Public Member Functions

- IrisParameterBuilder & addEnum (const std::string &symbol, const IrisValue &value, const std::string &description=std::string())

  Add an enum symbol for this parameter.
- IrisParameterBuilder & addStringEnum (const std::string &value, const std::string &description=std::string())

  Add a string enum symbol for this parameter.
- IrisParameterBuilder (ResourceInfo &info_)

  Construct a parameter builder for a given parameter resource.
- IrisParameterBuilder & setBitWidth (uint64_t bitWidth)

  Set the `bitWidth` field.
- IrisParameterBuilder & setDefault (const std::string &value)

  Set the default value for a string parameter.
- IrisParameterBuilder & setDefault (const std::vector< uint64_t > &value)

  Set the default value for a numeric parameter.
- IrisParameterBuilder & setDefault (uint64_t value)

  Set the default value for a numeric parameter.
- IrisParameterBuilder & setDefaultFloat (double value)

  Set the default value for a numericFp parameter.
- IrisParameterBuilder & setDefaultSigned (const std::vector< uint64_t > &value)

  Set the default value for a numericSigned parameter.
- IrisParameterBuilder & setDefaultSigned (int64_t value)

  Set the default value for a numericSigned parameter.
- IrisParameterBuilder & setDescr (const std::string &description)

  Set the `description` field.
- IrisParameterBuilder & setFormat (const std::string &format)

  Set the `format` field.
- IrisParameterBuilder & setHidden (bool hidden)

  Set the resource to hidden !
- IrisParameterBuilder & setInitOnly (bool value=true)

  Set the `initOnly` field.
- IrisParameterBuilder & setMax (const std::vector< uint64_t > &max)

Set the `max` field.

- IrisParameterBuilder & setMax (uint64_t max)

  Set the `max` field.
- IrisParameterBuilder & setMaxFloat (double max)

  Set the `max` field for floating-point parameters.
- IrisParameterBuilder & setMaxSigned (const std::vector< uint64_t > &max)

  Set the `max` field.
- IrisParameterBuilder & setMaxSigned (int64_t max)

  Set the `max` field.
- IrisParameterBuilder & setMin (const std::vector< uint64_t > &min)

  Set the `min` field.
- IrisParameterBuilder & setMin (uint64_t min)

  Set the `min` field.
- IrisParameterBuilder & setMinFloat (double min)

  Set the `min` field for floating-point parameters.
- IrisParameterBuilder & setMinSigned (const std::vector< uint64_t > &min)

  Set the `min` field.
- IrisParameterBuilder & setMinSigned (int64_t min)

  Set the `min` field.
- IrisParameterBuilder & setName (const std::string &name)

  Set the `name` field.
- IrisParameterBuilder & setRange (const std::vector< uint64_t > &min, const std::vector< uint64_t > &max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRange (uint64_t min, uint64_t max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRangeFloat (double min, double max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRangeSigned (const std::vector< uint64_t > &min, const std::vector< uint64_t > &max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRangeSigned (int64_t min, int64_t max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRwMode (const std::string &rwMode)

  Set the `rwMode` field.
- IrisParameterBuilder & setSubRscId (uint64_t subRscId)

  Set the `subRscId` field.
- IrisParameterBuilder & setTag (const std::string &tag)

  Set a boolean tag for this parameter resource.
- IrisParameterBuilder & setTag (const std::string &tag, const IrisValue &value)

  Set a tag for this parameter resource.
- IrisParameterBuilder & setTopology (bool value=true)

  Set the `topology` field.
- IrisParameterBuilder & setType (const std::string &type)

  Set the type of this parameter.

8.33.1 Detailed Description

Helper class to construct instantiation parameters.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 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.

8.33.3 Member Function Documentation

8.33.3.1 addEnum()

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

Add an enum symbol for this parameter.

Parameters

symbol	The enum symbol that is being added.
value	The value associated with the symbol.
description	A description explaining the meaning of the symbol.

Returns

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

8.33.3.2 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

value	The value associated with the symbol.
description	A description explaining the meaning of the symbol.

Returns

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

8.33.3.3 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.

8.33.3.4 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.

8.33.3.5 setDefault() [2/3]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefault (
            const std::vector< uint64_t > & value )  [inline]
```

Set the default value for a numeric parameter.
Use this variant for values that are >= 2**64.

Parameters

value	The `defaultData` field of the ParameterInfo object.

Returns

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

8.33.3.6 setDefault() [3/3]

```
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.

8.33.3.7 setDefaultFloat()

```
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.

8.33.3.8 setDefaultSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setDefaultSigned (
            const std::vector< uint64_t > & value ) [inline]
```

Set the default value for a numericSigned parameter.
Use this variant for values that are out of range for int64_t.

Parameters

value	The `defaultData` field of the ParameterInfo object.

Returns

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

8.33.3.9 setDefaultSigned() [2/2]

```
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.

8.33.3.10 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.

8.33.3.11 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.

8.33.3.12 setHidden()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setHidden (
            bool hidden )  [inline]
```

Set the resource to hidden !

Parameters

hidden	If true, this event source is not listed in resource_getList() calls but can still be accessed by resource_getResourceInfo() for clients that know the resource name. !

Returns

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

8.33.3.13 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.

8.33.3.14 setMax() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMax (
            const std::vector< uint64_t > & max )  [inline]
```

Set the max field.
Use this variant to set values that are >= 2**64.

Parameters

max	The max field of the ParameterInfo object.

Returns

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

8.33.3.15 setMax() [2/2]

```
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.

8.33.3.16 setMaxFloat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMaxFloat (
            double max )  [inline]
```

Set the max field for floating-point parameters.
This implies that the parameter type is "numericFp".

Parameters

max	The max field of the ParameterInfo object.

Returns

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

8.33.3.17 setMaxSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMaxSigned (
            const std::vector< uint64_t > & max )  [inline]
```

Set the max field.
This implies that the parameter type is "numericSigned". Use this variant for signed values that are out of range for int64_t.

Parameters

max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.18 setMaxSigned() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMaxSigned (
            int64_t max ) [inline]
```

Set the `max` field.
This implies that the parameter type is "numericSigned".

Parameters

max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.19 setMin() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMin (
            const std::vector< uint64_t > & min ) [inline]
```

Set the `min` field.
Use this variant to set values that are >= 2**64.

Parameters

min	The `min` field of the ParameterInfo object.

Returns

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

8.33.3.20 setMin() [2/2]

```
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.

8.33.3.21 setMinFloat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMinFloat (
            double min )  [inline]
```

Set the `min` field for floating-point parameters.
This implies that the parameter type is "numericFp".

Parameters

min	The `min` field of the ParameterInfo object.

Returns

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

8.33.3.22 setMinSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMinSigned (
            const std::vector< uint64_t > & min )  [inline]
```

Set the `min` field.
This implies that the parameter type is "numericSigned". Use this variant for signed values that are out of range for int64_t.

Parameters

min	The `min` field of the ParameterInfo object.

Returns

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

8.33.3.23 setMinSigned() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setMinSigned (
            int64_t min )  [inline]
```

Set the `min` field.
This implies that the parameter type is "numericSigned".

Parameters

min	The `min` field of the ParameterInfo object.

Returns

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

8.33.3.24 setName()

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

Set the `name` field.

Parameters

name	The `name` field of the ResourceInfo object.

Returns

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

8.33.3.25 setRange() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRange (
            const std::vector< uint64_t > & min,
            const std::vector< uint64_t > & max )  [inline]
```

Set both the `min` field and the `max` field.
Use this variant to set values that are >= 2**64.

Parameters

min	The `min` field of the ParameterInfo object.
max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.26 setRange() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRange (
            uint64_t min,
            uint64_t max )  [inline]
```

Set both the `min` field and the `max` field.

Parameters

min	The `min` field of the ParameterInfo object.
max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.27 setRangeFloat()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRangeFloat (
            double min,
            double max )  [inline]
```

Set both the `min` field and the `max` field.
This implies that the parameter type is "numericFp".

Parameters

min	The `min` field of the ParameterInfo object.
max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.28 setRangeSigned() [1/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRangeSigned (
            const std::vector< uint64_t > & min,
            const std::vector< uint64_t > & max )  [inline]
```

Set both the `min` field and the `max` field.
This implies that the parameter type is "numericSigned". Use this variant for signed values that are out of range for int64_t.

Parameters

min	The `min` field of the ParameterInfo object.
max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.29 setRangeSigned() [2/2]

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRangeSigned (
            int64_t min,
            int64_t max )  [inline]
```

Set both the `min` field and the `max` field.
This implies that the parameter type is "numericSigned".

Parameters

min	The `min` field of the ParameterInfo object.
max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.30 setRwMode()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setRwMode (
            const std::string & rwMode )  [inline]
```

Set the `rwMode` field.

Parameters

rwMode	The `rwMode` field of the ResourceInfo object.

Returns

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

8.33.3.31 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.

8.33.3.32 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.

8.33.3.33 setTag() [2/2]

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

Set a tag for this parameter resource.

Parameters

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.

8.33.3.34 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.

8.33.3.35 setType()

```
IrisParameterBuilder & iris::IrisParameterBuilder::setType (
            const std::string & type )  [inline]
```

Set the type of this parameter.
The bitWidth field must be set before setting the type.

Parameters

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

8.34 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

8.35 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.

8.35.1 Detailed Description

Set metadata for instantiating a plug-in instance.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 IrisPluginFactoryBuilder()

```
iris::IrisPluginFactoryBuilder::IrisPluginFactoryBuilder (
            const std::string & name )  [inline]
```

Parameters

name	The name of the plug-in to build.

8.35.3 Member Function Documentation

8.35.3.1 getDefaultInstanceName()

```
const std::string & iris::IrisPluginFactoryBuilder::getDefaultInstanceName ( ) const  [inline]
```

Get the default name to use for plug-in instances.

Returns

The default name for plug-in instances.

8.35.3.2 getInstanceNamePrefix()

```
const std::string & iris::IrisPluginFactoryBuilder::getInstanceNamePrefix ( ) const  [inline]
```

Get the prefix to use for instances of this plug-in.

Returns

The prefix to use for instances of this plug-in.

8.35.3.3 getPluginName()

```
const std::string & iris::IrisPluginFactoryBuilder::getPluginName ( ) const  [inline]
```

Get the plug-in name.

Returns

The name of the plug-in.

8.35.3.4 setDefaultInstanceName()

```
void iris::IrisPluginFactoryBuilder::setDefaultInstanceName (
            const std::string & name )  [inline]
```

Override the default instance name for plug-in instances.
The factory provides a sensible default for this name so it should only be overridden if there is a good reason to do so.

Parameters

name	The default name for plug-in instances.

8.35.3.5 setInstanceNamePrefix()

```
void iris::IrisPluginFactoryBuilder::setInstanceNamePrefix (
            const std::string & prefix )  [inline]
```

Override the instance name prefix. The default is "client.plugin".
The factory provides a sensible default for this prefix so it should only be overridden if there is a good reason to do so.

Parameters

prefix	The prefix that will be used for instances of this plug-in.

8.35.3.6 setPluginName()

```
void iris::IrisPluginFactoryBuilder::setPluginName (
            const std::string & name )  [inline]
```

Override the plug-in name.
The factory provides a sensible default for this name so it should only be overridden if there is a good reason to do so.

Parameters

name	The name of the plug-in.

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

- IrisPluginFactory.h

8.36 iris::IrisRegisterReadEventEmitter< REG_T, ARGS > Class Template Reference

An EventEmitter class for register read events.
`#include <IrisRegisterEventEmitter.h>`
Inherits IrisRegisterEventEmitterBase.

Public Member Functions

- void operator() (ResourceId rscId, bool debug, REG_T value, ARGS... args)

  Emit an event.

8.36.1 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;
 }
```

8.36.2 Member Function Documentation

8.36.2.1 operator()()

```
template<typename REG_T , typename...  ARGS>
void iris::IrisRegisterReadEventEmitter< REG_T, ARGS >::operator() (
             ResourceId rscId,
             bool debug,
             REG_T value,
             ARGS...  args )  [inline]
```

Emit an event.

Parameters

rscId	Resource id for the register that was accessed.
debug	True if this access originated from a debug access.
value	The register value that was read during this event.
args	Any additional custom fields for this event.

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

- IrisRegisterEventEmitter.h

8.37 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.

8.37.1 Detailed Description

template<typename REG_T, typename... ARGS>
class iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS >

An EventEmitter class for register update events.

Template Parameters

REG_T	The type of the register being read.
ARGS	Types of any custom fields that this event source defines, in addition to the standard fields defined for register update events.

Use IrisRegisterUpdateEventEmitter with IrisInstanceBuilder to add register update events to your Iris instance:

```
// Declare an event emitter
iris::IrisRegisterUpdateEventEmitter<uint64_t> reg_update_event;
// Add it to an Iris instance
iris::IrisInstance my_instance(...);
iris::IrisInstanceBuilder *builder = my_instance->getBuilder();
builder->setRegisterUpdateEvent("WRITE_REG", reg_update_event);
// Add some registers that will be traced by this event
builder->setNextRscId(0x1000);
builder->addRegister("X0", 64, "Register X0");
builder->addRegister("X1", 64, "Register X1");
builder->addRegister("X2", 64, "Register X2");
builder->addRegister("X3", 64, "Register X3");
// Now that the Instance builder has the metadata for the registers, we need
// to finalize the register update event to populate the event metadata.
builder->finalizeRegisterUpdateEvent();
void writeRegister(unsigned reg_index, bool is_debug, uint64_t new_value)
{
     uint64_t old_value = readRegValue(reg_index);
     writeRegValue(reg_index, new_value);
     // Emit an event
     reg_update_event(0x1000 | reg_index, is_debug, old_value, new_value);
 }
```

8.37.2 Member Function Documentation

8.37.2.1 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

rscId	Resource id for the register that was accessed.
debug	True if this access originated from a debug access.
old_value	The register value before the event.
new_value	The register value after the event.
args	Any additional custom fields for this event.

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

- IrisRegisterEventEmitter.h

8.38 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)

8.38.1 Detailed Description

Provides context to a reset delegate call.

8.38.2 Member Function Documentation

8.38.2.1 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

8.39 iris::IrisInstanceBuilder::MemorySpaceBuilder Class Reference

Used to set metadata for a memory space.
`#include <IrisInstanceBuilder.h>`

Public Member Functions

- MemorySpaceBuilder & addAttribute (const std::string &name, AttributeInfo attrib)

  Add an attribute to the `attrib` field.
- MemorySpaceId getSpaceId () const

  Get the memory space id for this memory space.
- **MemorySpaceBuilder** (IrisInstanceMemory::SpaceInfoAndAccess &info_)
- MemorySpaceBuilder & setAttributeDefault (const std::string &name, IrisValue value)

Set the default value for an attribute in the `attrib` field.

- MemorySpaceBuilder & 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 `minsAddr` field.
- MemorySpaceBuilder & setName (const std::string &name)

  Set the `name` field.
- template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) FUNC>
  MemorySpaceBuilder & setReadDelegate ()

  Set the delegate to read this memory space.
- MemorySpaceBuilder & setReadDelegate (MemoryReadDelegate delegate)

  Set the delegate to read this memory space.
- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) METHOD>
  MemorySpaceBuilder & setReadDelegate (T *instance)

  Set the delegate to read this memory space.
- template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>
  MemorySpaceBuilder & setSidebandDelegate ()

  Set the delegate to read sideband information.
- MemorySpaceBuilder & setSidebandDelegate (MemoryGetSidebandInfoDelegate delegate)

  Set the delegate to read sideband information.
- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) METHOD>
  MemorySpaceBuilder & setSidebandDelegate (T *instance)

  Set the delegate to read sideband information.
- template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) FUNC>
  MemorySpaceBuilder & setWriteDelegate ()

  Set the delegate to write to this memory space.
- MemorySpaceBuilder & setWriteDelegate (MemoryWriteDelegate delegate)

  Set the delegate to write to this memory space.
- template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
  MemorySpaceBuilder & setWriteDelegate (T *instance)

  Set the delegate to write to this memory space.

8.39.1 Detailed Description

Used to set metadata for a memory space.

8.39.2 Member Function Documentation

8.39.2.1 addAttribute()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::addAttribute (
            const std::string & name,
            AttributeInfo attrib ) [inline]
```

Add an attribute to the `attrib` field.

Parameters

name	The name of this attribute.
attrib	AttributeInfo for this attribute.

Returns

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

8.39.2.2 getSpaceId()

```
MemorySpaceId iris::IrisInstanceBuilder::MemorySpaceBuilder::getSpaceId ( ) const [inline]
```

Get the memory space id for this memory space.
This can be useful for setting up address translations and to map access requests to the correct memory space in memory access delegates.

Returns

The memory space id for this memory space.

8.39.2.3 setAttributeDefault()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setAttributeDefault (
            const std::string & name,
            IrisValue value ) [inline]
```

Set the default value for an attribute in the `attrib` field.

Parameters

name	The name of this attribute.
value	Default value of the named attribute.

Returns

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

8.39.2.4 setCanonicalMsn()

```
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.

8.39.2.5 setDescription()

```
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.

8.39.2.6 setEndianness()

```
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.

8.39.2.7 setMaxAddr()

```
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.

8.39.2.8 setMinAddr()

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setMinAddr (
            uint64_t minAddr )  [inline]
```

Set the `minsAddr` field.

Parameters

minAddr	The minAddr field of the MemorySpaceInfo object.

Returns

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

8.39.2.9 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.

8.39.2.10 setReadDelegate() [1/3]

```
template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const Attribute↩
ValueMap &, MemoryReadResult &) FUNC>
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setReadDelegate ( )  [inline]
```

Set the delegate to read this memory space.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultMemoryReadDelegate

Template Parameters

FUNC	A memory read delegate function.

Returns

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

8.39.2.11 setReadDelegate() [2/3]

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setReadDelegate (
            MemoryReadDelegate delegate )  [inline]
```

Set the delegate to read this memory space.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultMemoryReadDelegate

Parameters

delegate	MemoryReadDelegate object.

Returns

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

8.39.2.12 setReadDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64↩
_t, const AttributeValueMap &, MemoryReadResult &) METHOD>
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setReadDelegate (
            T * instance )  [inline]
```

Set the delegate to read this memory space.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultMemoryReadDelegate

Template Parameters

T	A class that defines a method with the right signature to be a memory read delegate.
METHOD	A memory read 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.

8.39.2.13 setSidebandDelegate() [1/3]

```
template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std↩
::vector< std::string > &, IrisValueMap &) FUNC>
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setSidebandDelegate ( )
[inline]
```

Set the delegate to read sideband information.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate

Template Parameters

FUNC	A memory sideband information delegate function.

Returns

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

8.39.2.14 setSidebandDelegate() [2/3]

```
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setSidebandDelegate (
            MemoryGetSidebandInfoDelegate delegate )  [inline]
```

Set the delegate to read sideband information.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate

Parameters

delegate	MemoryGetSidebandInfoDelegate object.

Returns

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

8.39.2.15 setSidebandDelegate() [3/3]

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

Set the delegate to read sideband information.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultGetMemorySidebandInfoDelegate

Template Parameters

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.

8.39.2.16 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const Attribute↩
ValueMap &, 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.

8.39.2.17 setWriteDelegate() [2/3]

MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setWriteDelegate (
MemoryWriteDelegate *delegate*) [inline]
Set the delegate to write to this memory space.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultMemoryWriteDelegate

Parameters

delegate	MemoryWriteDelegate object.

Returns

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

8.39.2.18 setWriteDelegate() [3/3]

template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
MemorySpaceBuilder & iris::IrisInstanceBuilder::MemorySpaceBuilder::setWriteDelegate (
T * *instance*) [inline]
Set the delegate to write to this memory space.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultMemoryWriteDelegate

Template Parameters

T	A class that defines a method with the right signature to be a memory write delegate.
METHOD	A memory write 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.

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

- IrisInstanceBuilder.h

8.40 iris::IrisCommandLineParser::Option Struct Reference

Option container.
`#include <IrisCommandLineParser.h>`

Public Member Functions

- Option & setList (char sep=',')

Friends

- class **IrisCommandLineParser**

8.40.1 Detailed Description

Option container.

8.40.2 Member Function Documentation

8.40.2.1 setList()

```
Option & iris::IrisCommandLineParser::Option::setList (
            char sep = ',' )  [inline]
```

Make this option a "list" option which can be specified multiple times. The value is stored as a single string and the elements are separated by "sep". Use getList() or getMap() to extract the elements.
The documentation for this struct was generated from the following file:

- IrisCommandLineParser.h

8.41 iris::IrisInstanceBuilder::ParameterBuilder Class Reference

Used to set metadata on a parameter.
`#include <IrisInstanceBuilder.h>`

Public Member Functions

- ParameterBuilder & addEnum (const std::string &symbol, const IrisValue &value, const std::string &description=std::string())

  Add a symbol to the enums field for numeric resources.
- ParameterBuilder & addStringEnum (const std::string &stringValue, const std::string &description=std↩ ::string())

  Add a symbol to the enums field for string resources.
- ResourceId 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.

- template<typename T >
  ParameterBuilder & setDefaultData (std::initializer_list< T > &&t)

  Set the `default` value for wide numeric parameters.

- ParameterBuilder & setDefaultData (uint64_t value)

  Set the `default` value for numeric parameter to a value <= 64 bit.

- template<typename Container >
  ParameterBuilder & setDefaultDataFromContainer (const Container &container)

  Set the `default` value for wide numeric parameters.

- ParameterBuilder & setDefaultString (const std::string &defaultString)

  Set the `defaultData` field for wide numeric parameters (bitWidth > 64 bit).

- ParameterBuilder & **setDescr** (const std::string &description)

  Obsolete alias for setDescription(). Do not use.

- ParameterBuilder & setDescription (const std::string &description)

  Set the `description` field.

- ParameterBuilder & setFormat (const std::string &format)

  Set the `format` field.

- ParameterBuilder & setHidden (bool hidden=true)

  Set the resource to hidden.

- ParameterBuilder & setInitOnly (bool initOnly=true)

  Set the `initOnly` flag of a parameter.

- template<typename T >
  ParameterBuilder & setMax (std::initializer_list< T > &&t)

  Set the `max` field for wide numeric parameters.

- ParameterBuilder & setMax (uint64_t value)

  Set the `max` field to a value <= 64 bit.

- template<typename Container >
  ParameterBuilder & setMaxFromContainer (const Container &container)

  Set the `max` field for wide numeric parameters.

- template<typename T >
  ParameterBuilder & setMin (std::initializer_list< T > &&t)

  Set the `min` field for wide numeric parameters.

- ParameterBuilder & setMin (uint64_t value)

  Set the `min` field to a value <= 64 bit.

- template<typename Container >
  ParameterBuilder & setMinFromContainer (const Container &container)

  Set the `min` field for wide numeric parameters.

- ParameterBuilder & setName (const std::string &name)

  Set the `name` field.

- ParameterBuilder & setParentRscId (ResourceId parentRscId)

  Set the `parentRscId` field.

- template<IrisErrorCode(*)(const ResourceInfo &, ResourceReadResult &) FUNC>
  ParameterBuilder & setReadDelegate ()

  Set the delegate to read the resource.

- ParameterBuilder & setReadDelegate (ResourceReadDelegate readDelegate)

Set the delegate to read the resource.

- template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, ResourceReadResult &) METHOD>
  ParameterBuilder & setReadDelegate (T *instance)

  Set the delegate to read the resource.
- ParameterBuilder & setRwMode (const std::string &rwMode)

  Set the `rwMode` field.
- ParameterBuilder & setSubRscId (uint64_t subRscId)

  Set the `subRscId` field.
- ParameterBuilder & setTag (const std::string &tag)

  Set the named boolean tag to true (e.g. isPc)
- ParameterBuilder & setTag (const std::string &tag, const IrisValue &value)

  Set a tag to the specified value.
- ParameterBuilder & setType (const std::string &type)

  Set the `type` field.
- template<IrisErrorCode(*)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  ParameterBuilder & setWriteDelegate ()

  Set the delegate to write the resource.
- ParameterBuilder & setWriteDelegate (ResourceWriteDelegate writeDelegate)

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

  Set the delegate to write the resource.

8.41.1 Detailed Description

Used to set metadata on a parameter.

8.41.2 Member Function Documentation

8.41.2.1 addEnum()

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

Add a symbol to the enums field for numeric resources.
This should be called multiple times to add multiple symbols.

Parameters

symbol	The symbol string to be associated with the specified value.
value	The value of this symbol.
description	A description of this symbol.

Returns

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

8.41.2.2 addStringEnum()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::addStringEnum (
```

```
            const std::string & stringValue,
            const std::string & description = std::string() )  [inline]
```

Add a symbol to the enums field for string resources.
This should be called multiple times to add multiple symbols.

Parameters

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

Returns

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

8.41.2.3 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.

8.41.2.4 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.

8.41.2.5 setBitWidth()

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

Set the `bitWidth` field.

Parameters

bitWidth	The bitWidth field of the ResourceInfo object.

Returns

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

8.41.2.6 setCname()

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

Set the `cname` field.

Parameters

cname	The cname field of the ResourceInfo object.

Returns

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

8.41.2.7 setDefaultData() [1/2]

```
template<typename T >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDefaultData (
            std::initializer_list< T > && t )  [inline]
```

Set the `default` value for wide numeric parameters.
This function accepts a braced initializer-list and is otherwise idential 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.

8.41.2.8 setDefaultData() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDefaultData (
            uint64_t value )  [inline]
```

Set the `default` value for numeric parameter to a value <= 64 bit.
If the parameter is wider than the passed value the value is zero extended.
If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

value	The defaultData field of the ParameterInfo object.

Returns

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

8.41.2.9 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.

8.41.2.10 setDefaultString()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDefaultString (
            const std::string & defaultString ) [inline]
```

Set the `defaultData` field for wide numeric parameters (bitWidth $>$ 64 bit).
Set the default value for string parameters.

Parameters

defaultString	The defaultString field of the ParameterInfo object.

Returns

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

8.41.2.11 setDescription()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setDescription (
            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.

8.41.2.12 setFormat()

```
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.

8.41.2.13 setHidden()

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

Set the resource to hidden.

Parameters

hidden	If true, this resource is not listed in resource_getList() calls

Returns

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

8.41.2.14 setInitOnly()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setInitOnly (
            bool initOnly = true )  [inline]
```

Set the `initOnly` flag of a parameter.
This also implicitly sets the parameter to read-only.

Parameters

initOnly	The initOnly flag of a parameter.

Returns

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

8.41.2.15 setMax() [1/2]

```
template<typename T >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMax (
            std::initializer_list< T > && t )  [inline]
```

Set the `max` field for wide numeric parameters.
This function accepts a braced initializer-list and is otherwise idential 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.

8.41.2.16 setMax() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMax (
            uint64_t value )  [inline]
```

Set the `max` field to a value <= 64 bit.
If the parameter is wider than the passed value the value is zero extended.

If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

value	Max value of the parameter.

Returns

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

8.41.2.17 setMaxFromContainer()

```
template<typename Container >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMaxFromContainer (
            const Container & container )  [inline]
```

Set the `max` field for wide numeric parameters.
Container must be a type which allows to iterate over uint64_t bit chunks of the value, least significant bits first, for example std::array<uint64_t> or std::vector<uint64_t>.
Each element of the container will be promoted/narrowed to uint64_t.
If the parameter is wider than the passed value the value is zero extended.
If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

container	Container containing the value in 64-bit chunks.

Returns

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

8.41.2.18 setMin() [1/2]

```
template<typename T >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMin (
            std::initializer_list< T > && t )  [inline]
```

Set the `min` field for wide numeric parameters.
This function accepts a braced initializer-list and is otherwise idential 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.

8.41.2.19 setMin() [2/2]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMin (
            uint64_t value )  [inline]
```

Set the `min` field to a value <= 64 bit.
If the parameter is wider than the passed value the value is zero extended.
If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

value	min value of the parameter.

Returns

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

8.41.2.20 setMinFromContainer()

```
template<typename Container >
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setMinFromContainer (
            const Container & container )  [inline]
```

Set the `min` field for wide numeric parameters.
Container must be a type which allows to iterate over uint64_t bit chunks of the value, least significant bits first, for example std::array<uint64_t> or std::vector<uint64_t>.
Each element of the container will be promoted/narrowed to uint64_t.
If the parameter is wider than the passed value the value is zero extended.
If the parameter is narrower than the passed value the superfluous bits are ignored.

Parameters

container	Container containing the value in 64-bit chunks.

Returns

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

8.41.2.21 setName()

```
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.

8.41.2.22 setParentRscId()

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setParentRscId (
            ResourceId parentRscId )  [inline]
```

Set the `parentRscId` field.
This function makes this register a child of the specified parent. It is not necessary to call this function when adding child registers using the addField() function.

Parameters

parent↩ RscId	The rscId of the parent register.

Returns

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

8.41.2.23 setReadDelegate() [1/3]

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

Set the delegate to read the resource.
Set a delegate which calls function FUNC().
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Template Parameters

FUNC	A resource read delegate function.

Returns

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

8.41.2.24 setReadDelegate() [2/3]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setReadDelegate (
            ResourceReadDelegate readDelegate )  [inline]
```

Set the delegate to read the resource.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Parameters

readDelegate	ResourceReadDelegate object.

Returns

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

8.41.2.25 setReadDelegate() [3/3]

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

Set the delegate to read the resource.
Set a delegate which calls METHOD() on an instance of class T.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Template Parameters

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

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.

8.41.2.26 setRwMode()

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

Set the `rwMode` field.

Parameters

rwMode	The rwMode field of the ResourceInfo object.

Returns

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

8.41.2.27 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.

8.41.2.28 setTag() [1/2]

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

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

Parameters

tag	The name of the tag to set.

Returns

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

8.41.2.29 setTag() [2/2]

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

Set a tag to the specified value.

Parameters

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.

8.41.2.30 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.

8.41.2.31 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate ( )  [inline]
```

Set the delegate to write the resource.
Set a delegate which calls function FUNC().
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Template Parameters

FUNC	A resource write delegate function.

Returns

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

8.41.2.32 setWriteDelegate() [2/3]

```
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate (
            ResourceWriteDelegate writeDelegate )  [inline]
```

Set the delegate to write the resource.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Parameters

writeDelegate	ResourceWriteDelegate object.

Returns

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

8.41.2.33 setWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
METHOD>
ParameterBuilder & iris::IrisInstanceBuilder::ParameterBuilder::setWriteDelegate (
            T * instance )  [inline]
```

Set the delegate to write the resource.
Set a delegate which calls METHOD() on an instance of class T.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Template Parameters

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 ParameterBuilder object allowing calls to be chained together.

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

- IrisInstanceBuilder.h

8.42 iris::IrisInstanceEvent::ProxyEventInfo Struct Reference

Contains information for a single proxy EventSource.
`#include <IrisInstanceEvent.h>`

Public Attributes

- std::vector< EventStreamId > **evStreamIds**
- EventSourceId **targetEvSrcId** {}
- InstanceId **targetInstId** {}

8.42.1 Detailed Description

Contains information for a single proxy EventSource.
The documentation for this struct was generated from the following file:

- IrisInstanceEvent.h

8.43 iris::IrisInstanceBuilder::RegisterBuilder Class Reference

Used to set metadata on a register resource.
`#include <IrisInstanceBuilder.h>`

Public Member Functions

- RegisterBuilder & addEnum (const std::string &symbol, const IrisValue &value, const std::string &description=std::string())

  Add a symbol to the enums field for numeric resources.
- FieldBuilder addField (const std::string &name, uint64_t lsbOffset, uint64_t bitWidth, const std::string &description)

  Add a subregister field to this register. By default, the field copies attributes from its parent register, but any field can be overridden.
- FieldBuilder addLogicalField (const std::string &name, uint64_t bitWidth, const std::string &description)

  Add a logical subregister field to this register. A logical field is a field which has a bitwidth, but which does not have an lsbOffset. It is usually used to represent non-contiguous fields which are distributed across multiple chunks in the parent register as a single contiguous register. This allows to attach enums to such a field.
- RegisterBuilder & addStringEnum (const std::string &stringValue, const std::string &description=std::string())

  Add a symbol to the enums field for string resources.
- ResourceId getRscId () const

  Return the rscId that was allocated for this resource.
- RegisterBuilder & getRscId (ResourceId &rscIdOut)

  Get the rscId that was allocated for this resource.
- **RegisterBuilder** (IrisInstanceResource::ResourceInfoAndAccess &info_, IrisInstanceResource *inst_resource_, IrisInstanceBuilder *instance_builder_)

- RegisterBuilder & setAddressOffset (uint64_t addressOffset)

  Set the `addressOffset` field.
- RegisterBuilder & setBitWidth (uint64_t bitWidth)

  Set the `bitWidth` field.
- RegisterBuilder & setCanonicalRn (uint64_t canonicalRn_)

  Set the `canonicalRn` field.
- RegisterBuilder & setCanonicalRnElfDwarf (uint16_t architecture, uint16_t dwarfRegNum)

  Set the `canonicalRn` field for "ElfDwarf" scheme.
- RegisterBuilder & setCname (const std::string &cname)

  Set the `cname` field.
- RegisterBuilder & **setDescr** (const std::string &description)

  Obsolete alias for setDescription(). Do not use.
- RegisterBuilder & setDescription (const std::string &description)

  Set the `description` field.
- RegisterBuilder & setFormat (const std::string &format)

  Set the `format` field.
- RegisterBuilder & setLsbOffset (uint64_t lsbOffset)

  Set the `lsbOffset` field.
- RegisterBuilder & setName (const std::string &name)

  Set the `name` field.
- RegisterBuilder & setParentRscId (ResourceId parentRscId)

  Set the `parentRscId` field.
- template<IrisErrorCode(*)(const ResourceInfo &, ResourceReadResult &) FUNC>
  RegisterBuilder & setReadDelegate ()

  Set the delegate to read the resource.
- RegisterBuilder & setReadDelegate (ResourceReadDelegate readDelegate)

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

  Set the delegate to read the resource.
- template<typename T >
  RegisterBuilder & setResetData (std::initializer_list< T > &&t)

  Set the `resetData` field for wide registers.
- RegisterBuilder & setResetData (uint64_t value)

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

  Set the `resetData` field for wide registers.
- RegisterBuilder & setResetString (const std::string &resetString)

  Set the `resetString` field.
- RegisterBuilder & setRwMode (const std::string &rwMode)

  Set the `rwMode` field.
- RegisterBuilder & setSubRscId (uint64_t subRscId)

  Set the `subRscId` field.
- RegisterBuilder & setTag (const std::string &tag)

  Set the named boolean tag to true (e.g. isPc)
- RegisterBuilder & setTag (const std::string &tag, const IrisValue &value)

  Set a tag to the specified value.
- RegisterBuilder & setType (const std::string &type)

  Set the `type` field.

- template<IrisErrorCode(*)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  RegisterBuilder & setWriteDelegate ()

  Set the delegate to write the resource.
- RegisterBuilder & setWriteDelegate (ResourceWriteDelegate writeDelegate)

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

  Set the delegate to write the resource.
- template<typename T >
  RegisterBuilder & setWriteMask (std::initializer_list< T > &&t)

  Set the `writeMask` field for wide registers.
- RegisterBuilder & setWriteMask (uint64_t value)

  Set the `writeMask` field to a value <= 64 bit.
- template<typename Container >
  RegisterBuilder & setWriteMaskFromContainer (const Container &container)

  Set the `writeMask` field for wide registers.

8.43.1 Detailed Description

Used to set metadata on a register resource.

8.43.2 Member Function Documentation

8.43.2.1 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

symbol	The symbol string to be associated with the specified value.
value	The value of this symbol.
description	A description of this symbol.

Returns

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

8.43.2.2 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

name	Name of the register field.
lsbOffset	The bit offset of this field inside its parent register.
bitWidth	The size of the field.
description	Description of this field.

Returns

A FieldBuilder object that allows the caller to set attributes for this field.

8.43.2.3 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

name	Name of the register field.
bitWidth	The size of the field.
description	Description of this field.

Returns

A FieldBuilder object that allows the caller to set attributes for this field.

8.43.2.4 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

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

Returns

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

8.43.2.5 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.

8.43.2.6 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.

8.43.2.7 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.

8.43.2.8 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.

8.43.2.9 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.

8.43.2.10 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.

8.43.2.11 setCname()

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

Set the `cname` field.

Parameters

cname	The cname field of the ResourceInfo object.

Returns

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

8.43.2.12 setDescription()

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

Set the `description` field.

Parameters

description	The description field of the ResourceInfo object.

Returns

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

8.43.2.13 setFormat()

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

Set the `format` field.

Parameters

format	The format field of the ResourceInfo object.

Returns

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

8.43.2.14 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.

8.43.2.15 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.

8.43.2.16 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

parent↩ RscId	The rscId of the parent register.

Returns

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

8.43.2.17 setReadDelegate() [1/3]

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

Set the delegate to read the resource.
Set a delegate which calls function FUNC().
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Template Parameters

FUNC	A resource read delegate function.

Returns

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

8.43.2.18 setReadDelegate() [2/3]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setReadDelegate (
            ResourceReadDelegate readDelegate )  [inline]
```

Set the delegate to read the resource.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Parameters

readDelegate	ResourceReadDelegate object.

Returns

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

8.43.2.19 setReadDelegate() [3/3]

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

Set the delegate to read the resource.
Set a delegate which calls METHOD() on an instance of class T.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceReadDelegate

Template Parameters

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

Parameters

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

Returns

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

8.43.2.20 setResetData() [1/2]

```
template<typename T >
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setResetData (
            std::initializer_list< T > && t )  [inline]
```

Set the `resetData` field for wide registers.
This function accepts a braced initializer-list and is otherwise idential to 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.

8.43.2.21 setResetData() [2/2]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setResetData (
            uint64_t value )  [inline]
```

Set the `resetData` field to a value <= 64 bit.
If the register is wider than the passed value the value is zero extended.
If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

value	resetData value of the register.

Returns

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

8.43.2.22 setResetDataFromContainer()

```
template<typename Container >
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setResetDataFromContainer (
            const Container & container )  [inline]
```

Set the `resetData` field for wide registers.
Container must be a type which allows to iterate over uint64_t bit chunks of the value,
least significant bits first, for example std::array<uint64_t> or std::vector<uint64_t>.
Each element of the container will be promoted/narrowed to uint64_t.
If the register is wider than the passed value the value is zero extended.
If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

container	Container containing the value in 64-bit chunks.

Returns

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

8.43.2.23 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.

8.43.2.24 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.

8.43.2.25 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.

8.43.2.26 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.

8.43.2.27 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.

8.43.2.28 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.

8.43.2.29 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteDelegate ( )  [inline]
```

Set the delegate to write the resource.
Set a delegate which calls function FUNC().
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Template Parameters

FUNC	A resource write delegate function.

Returns

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

8.43.2.30 setWriteDelegate() [2/3]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteDelegate (
            ResourceWriteDelegate writeDelegate )  [inline]
```

Set the delegate to write the resource.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Parameters

writeDelegate	ResourceWriteDelegate object.

Returns

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

8.43.2.31 setWriteDelegate() [3/3]

```
template<typename T , IrisErrorCode(T::*)(const ResourceInfo &, const ResourceWriteValue &)
METHOD>
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteDelegate (
            T * instance )  [inline]
```

Set the delegate to write the resource.
Set a delegate which calls METHOD() on an instance of class T.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultResourceWriteDelegate

Template Parameters

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 RegisterBuilder object allowing calls to be chained together.

8.43.2.32 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 idential 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.

8.43.2.33 setWriteMask() [2/2]

```
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteMask (
            uint64_t value )  [inline]
```

Set the `writeMask` field to a value <= 64 bit.
If the register is wider than the passed value the value is zero extended.
If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

value	writeMask value of the register.

Returns

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

8.43.2.34 setWriteMaskFromContainer()

```
template<typename Container >
RegisterBuilder & iris::IrisInstanceBuilder::RegisterBuilder::setWriteMaskFromContainer (
            const Container & container )  [inline]
```

Set the `writeMask` field for wide registers.
Container must be a type which allows to iterate over uint64_t bit chunks of the value,
least significant bits first, for example std::array<uint64_t> or std::vector<uint64_t>.
Each element of the container will be promoted/narrowed to uint64_t.
If the register is wider than the passed value the value is zero extended.
If the register is narrower than the passed value the superfluous bits are ignored.

Parameters

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

8.44 iris::IrisInstanceResource::ResourceInfoAndAccess Struct Reference

Entry in 'resourceInfos'.
`#include <IrisInstanceResource.h>`

Public Attributes

- ResourceReadDelegate **readDelegate**
- ResourceInfo **resourceInfo**
- ResourceWriteDelegate **writeDelegate**

8.44.1 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

8.45 iris::ResourceWriteValue Struct Reference

`#include <IrisInstanceResource.h>`

Public Attributes

- const uint64_t * **data** {}
- const std::string * **str** {}

  Non-null for non-string resources.

8.45.1 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

8.46 iris::IrisInstanceBuilder::SemihostingManager Class Reference

semihosting_apis IrisInstanceBuilder semihosting APIs
`#include <IrisInstanceBuilder.h>`

Public Member Functions

- void **enableExtensions** ()

  Instances that support semihosting extensions should call this function to enable the `IRIS_SEMIHOSTING_CALL_EXTENSION` event.
- std::vector< uint8_t > readData (uint64_t fDes, size_t max_size=0, uint64_t flags=semihost::DEFAULT)

  Read data for a given file descriptor.
- std::pair< bool, uint64_t > semihostedCall (uint64_t operation, uint64_t parameter)

  Allow a client to perform a semihosting extension defined by operation and parameter.
- **SemihostingManager** (IrisInstanceSemihosting *inst_semihost_)
- void **unblock** ()
- bool **writeData** (uint64_t fDes, const std::vector< uint8_t > &data)
- bool **writeData** (uint64_t fDes, const uint8_t *data, size_t size)

8.46.1 Detailed Description

semihosting_apis IrisInstanceBuilder semihosting APIs
Manage semihosting functionality

8.46.2 Member Function Documentation

8.46.2.1 readData()

```
std::vector< uint8_t > iris::IrisInstanceBuilder::SemihostingManager::readData (
            uint64_t fDes,
            size_t max_size = 0,
            uint64_t flags = semihost::DEFAULT )  [inline]
```

Read data for a given file descriptor.
The exact behavior of this method depends on the value of the max_size and flags parameters. If the NONBLOCK flag is set, the method returns immediately with whatever data is already buffered, if any. If NONBLOCK is not set, the method blocks until data is available. Iris messages continue to be processed while this methods blocks. If max_size is not zero, then at most max_size bytes will be returned.

Parameters

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.

8.46.2.2 semihostedCall()

```
std::pair< bool, uint64_t > iris::IrisInstanceBuilder::SemihostingManager::semihostedCall (
            uint64_t operation,
            uint64_t parameter )  [inline]
```

Allow a client to perform a semihosting extension defined by *operation* and *parameter*.
This might implement a user-defined operation or override the default implementation for a predefined operation.

Parameters

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

8.47 iris::IrisInstanceMemory::SpaceInfoAndAccess Struct Reference

Entry in 'spaceInfos'.
`#include <IrisInstanceMemory.h>`

Public Attributes

- MemoryReadDelegate **readDelegate**
- MemoryGetSidebandInfoDelegate **sidebandDelegate**
- MemorySpaceInfo **spaceInfo**
- MemoryWriteDelegate **writeDelegate**

8.47.1 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

8.48 iris::IrisInstanceBuilder::TableBuilder Class Reference

Used to set metadata for a table.
`#include <IrisInstanceBuilder.h>`

Public Member Functions

- TableColumnBuilder addColumn (const std::string &name)

  Add a new column.

- TableBuilder & addColumnInfo (const TableColumnInfo &columnInfo)

  Add a column with a preconstructed TableColumnInfo.
- TableBuilder & setDescription (const std::string &description)

  Set the `description` field.
- TableBuilder & setFormatLong (const std::string &format)

  Set the `formatLong` field.
- TableBuilder & setFormatShort (const std::string &format)

  Set the `formatShort` field.
- TableBuilder & setIndexFormatHint (const std::string &hint)

  Set the `indexFormatHint` field.
- TableBuilder & setMaxIndex (uint64_t maxIndex)

  Set the `maxIndex` field.
- TableBuilder & setMinIndex (uint64_t minIndex)

  Set the `minIndex` field.
- TableBuilder & setName (const std::string &name)

  Set the `name` field.
- template<IrisErrorCode(∗)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
  TableBuilder & setReadDelegate ()

  Set the delegate to read the table.
- template<typename T , IrisErrorCode(T::∗)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) METHOD>
  TableBuilder & setReadDelegate (T ∗instance)

  Set the delegate to read the table.
- TableBuilder & setReadDelegate (TableReadDelegate delegate)

  Set the delegate to read the table.
- template<IrisErrorCode(∗)(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
  TableBuilder & setWriteDelegate ()

  Set the delegate to write to the table.
- template<typename T , IrisErrorCode(T::∗)(const TableInfo &, const TableRecords &, TableWriteResult &) METHOD>
  TableBuilder & setWriteDelegate (T ∗instance)

  Set the delegate to write to the table.
- TableBuilder & setWriteDelegate (TableWriteDelegate delegate)

  Set the delegate to write to the table.
- **TableBuilder** (IrisInstanceTable::TableInfoAndAccess &info_)

8.48.1 Detailed Description

Used to set metadata for a table.

8.48.2 Member Function Documentation

8.48.2.1 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.

8.48.2.2 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.

8.48.2.3 setDescription()

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

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.

8.48.2.4 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.

8.48.2.5 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.

8.48.2.6 setIndexFormatHint()

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setIndexFormatHint (
            const std::string & hint )  [inline]
```

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.

8.48.2.7 setMaxIndex()

```
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.

8.48.2.8 setMinIndex()

```
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.

8.48.2.9 setName()

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

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.

8.48.2.10 setReadDelegate() [1/3]

```
template<IrisErrorCode(*)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setReadDelegate ( )  [inline]
```

Set the delegate to read the table.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultTableReadDelegate

Template Parameters

FUNC	A table read delegate function.

Returns

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

8.48.2.11 setReadDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const TableInfo &, uint64_t, uint64_t, TableRead↩
Result &) METHOD>
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setReadDelegate (
            T * instance )  [inline]
```

Set the delegate to read the table.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultTableReadDelegate

Template Parameters

T	A class that defines a method with the right signature to be a table read delegate.
METHOD	A table read delegate method in class T.

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.

8.48.2.12 setReadDelegate() [3/3]

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setReadDelegate (
            TableReadDelegate delegate ) [inline]
```

Set the delegate to read the table.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultTableReadDelegate

Parameters

delegate	TableReadDelegate object.

Returns

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

8.48.2.13 setWriteDelegate() [1/3]

```
template<IrisErrorCode(*)(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setWriteDelegate ( ) [inline]
```

Set the delegate to write to the table.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultTableWriteDelegate

Template Parameters

FUNC	A table write delegate function.

Returns

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

8.48.2.14 setWriteDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const TableInfo &, const TableRecords &, Table↩
WriteResult &) METHOD>
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setWriteDelegate (
            T * instance )  [inline]
```

Set the delegate to write to the table.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultTableWriteDelegate

Template Parameters

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

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.

8.48.2.15 setWriteDelegate() [3/3]

```
TableBuilder & iris::IrisInstanceBuilder::TableBuilder::setWriteDelegate (
            TableWriteDelegate delegate )  [inline]
```

Set the delegate to write to the table.
If this is not set, the default delegate is used.

See also

IrisInstanceBuilder::setDefaultTableWriteDelegate

Parameters

delegate	TableWriteDelegate object.

Returns

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

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

- IrisInstanceBuilder.h

8.49 iris::IrisInstanceBuilder::TableColumnBuilder Class Reference

Used to set metadata for a table column.
`#include <IrisInstanceBuilder.h>`

Public Member Functions

- TableColumnBuilder addColumn (const std::string &name)

  Add another new column.
- TableBuilder & addColumnInfo (const TableColumnInfo &columnInfo)

  Add another column with a preconstructed TableColumnInfo.
- TableBuilder & endColumn ()

  Stop building this column and go back to the parent table.
- TableColumnBuilder & setBitWidth (uint64_t bitWidth)

  Set the `bitWidth` field.
- TableColumnBuilder & setDescription (const std::string &description)

  Set the `description` field.
- TableColumnBuilder & setFormat (const std::string &format)

  Set the `format` field.
- TableColumnBuilder & setFormatLong (const std::string &format)

  Set the `formatLong` field.
- TableColumnBuilder & setFormatShort (const std::string &format)

  Set the `formatShort` field.
- TableColumnBuilder & setName (const std::string &name)

  Set the `name` field.
- TableColumnBuilder & setRwMode (const std::string &rwMode)

  Set the `rwMode` field.
- TableColumnBuilder & setType (const std::string &type)

  Set the `type` field.
- **TableColumnBuilder** (TableBuilder &parent_, TableColumnInfo &info_)

8.49.1 Detailed Description

Used to set metadata for a table column.

8.49.2 Member Function Documentation

8.49.2.1 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.

8.49.2.2 addColumnInfo()

```
TableBuilder & iris::IrisInstanceBuilder::TableColumnBuilder::addColumnInfo (
            const TableColumnInfo & columnInfo )  [inline]
```

Add another column with a preconstructed TableColumnInfo.

See also

TableBuilder::addColumnInfo

addColumn

Parameters

columnInfo	A preconstructed TableColumnInfo object for the new column.

Returns

A reference to the parent TableBuilder for this table.

8.49.2.3 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.

8.49.2.4 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.

8.49.2.5 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.

8.49.2.6 setFormat()

```
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.

8.49.2.7 setFormatLong()

```
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.

8.49.2.8 setFormatShort()

```
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.

8.49.2.9 setName()

```
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.

8.49.2.10 setRwMode()

```
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.

8.49.2.11 setType()

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

Set the `type` field.

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

8.50 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.

8.50.1 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 IrisCConnection.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisCConnection_h
8 #define ARM_INCLUDE_IrisCConnection_h
9
10 #include "iris/detail/IrisC.h"
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisErrorException.h"
13 #include "iris/detail/IrisInterface.h"
14
15 #include <string>
16
17 NAMESPACE_IRIS_START
18
24 class IrisCConnection : public IrisConnectionInterface
25 {
26 private:
28     IrisC_HandleMessageFunction   handleMessage_function;
29
30     IrisC_RegisterChannelFunction   registerChannel_function;
31     IrisC_UnregisterChannelFunction unregisterChannel_function;
32
33     IrisC_ProcessAsyncMessagesFunction processAsyncMessages_function;
34
35     class RemoteInterface : public IrisInterface
36     {
37     private:
```

```
38          IrisCConnection* irisc;
39
40      public:
41          RemoteInterface(IrisCConnection* irisc_)
42              : irisc(irisc_)
43          {
44          }
45
46      public: // IrisInterface
47          virtual void irisHandleMessage(const uint64_t* message) IRIS_OVERRIDE
48          {
49              // Forward to the IrisC library
50              int64_t status = irisc->IrisC_handleMessage(message);
51
52              if (status != E_ok)
53              {
54                  throw IrisErrorException(IrisErrorCode(status));
55              }
56          }
57      } remote_interface;
58
59      // Helper function to bridge IrisC_HandleMessageFunction to IrisInterface::irisHandleMessage
60      static int64_t handleMessageToIrisInterface(void* context, const uint64_t* message)
61      {
62          if (context == nullptr)
63          {
64              return E_invalid_context;
65          }
66          static_cast<IrisInterface*>(context)->irisHandleMessage(message);
67
68          return E_ok;
69      }
70
71  protected:
73      void* iris_c_context;
74
76      IrisCConnection()
77          : handleMessage_function(nullptr)
78          , registerChannel_function(nullptr)
79          , unregisterChannel_function(nullptr)
80          , processAsyncMessages_function(nullptr)
81          , remote_interface(this)
82          , iris_c_context(nullptr)
83      {
84      }
85
87
88      int64_t IrisC_handleMessage(const uint64_t* message)
89      {
90          return (*handleMessage_function)(iris_c_context, message);
91      }
92
93      int64_t IrisC_registerChannel(IrisC_CommunicationChannel* channel, uint64_t* channel_id_out)
94      {
95          return (*registerChannel_function)(iris_c_context, channel, channel_id_out);
96      }
97
98      int64_t IrisC_unregisterChannel(uint64_t channel_id)
99      {
100          return (*unregisterChannel_function)(iris_c_context, channel_id);
101      }
102
103      int64_t IrisC_processAsyncMessages(bool waitForAMessage)
104      {
105          return (*processAsyncMessages_function)(iris_c_context, waitForAMessage);
106      }
107
108 public:
109      IrisCConnection(IrisC_Functions* functions)
110          : handleMessage_function(functions->handleMessage_function)
111          , registerChannel_function(functions->registerChannel_function)
112          , unregisterChannel_function(functions->unregisterChannel_function)
113          , processAsyncMessages_function(functions->processAsyncMessages_function)
114          , remote_interface(this)
115          , iris_c_context(functions->iris_c_context)
116      {
117      }
118
119 public: // IrisConnectionInterface
124      virtual uint64_t registerIrisInterfaceChannel(IrisInterface* iris_interface) IRIS_OVERRIDE
125      {
126          IrisC_CommunicationChannel channel;
127
128          channel.CommunicationChannel_version = 0;
129          channel.handleMessage_function       = &IrisCConnection::handleMessageToIrisInterface;
130          channel.handleMessage_context        = static_cast<void*>(iris_interface);
131
```

```
132         uint64_t channelId = IRIS_UINT64_MAX;
133
134         IrisErrorCode status = static_cast<IrisErrorCode>(IrisC_registerChannel(&channel, &channelId));
135
136         if (status != E_ok)
137         {
138             throw IrisErrorException(status);
139         }
140
141         return channelId;
142     }
143
148     virtual void unregisterIrisInterfaceChannel(uint64_t channelId) IRIS_OVERRIDE
149     {
150         IrisErrorCode status = static_cast<IrisErrorCode>(IrisC_unregisterChannel(channelId));
151
152         if (status != E_ok)
153         {
154             throw IrisErrorException(status);
155         }
156     }
157
162     virtual IrisErrorCode processAsyncMessages(bool waitForAMessage) IRIS_OVERRIDE
163     {
164         return static_cast<IrisErrorCode>(IrisC_processAsyncMessages(waitForAMessage));
165     }
166
171     virtual IrisInterface* getIrisInterface() IRIS_OVERRIDE
172     {
173         return &remote_interface;
174     }
175 };
176
177 NAMESPACE_IRIS_END
178
179 #endif // ARM_INCLUDE_IrisCConnection_h
```

9.3 IrisClient.h File Reference

Iris client which supports multiple methods to connect to other Iris executables.

```
#include "iris/IrisInstance.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisErrorCode.h"
#include "iris/detail/IrisInterface.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisUtils.h"
#include "iris/detail/IrisCommaSeparatedParameters.h"
#include "iris/impl/IrisChannelRegistry.h"
#include "iris/impl/IrisMessageQueue.h"
#include "iris/impl/IrisPlugin.h"
#include "iris/impl/IrisProcessEventsThread.h"
#include "iris/impl/IrisRpcAdapterTcp.h"
#include "iris/impl/IrisTcpSocket.h"
#include <map>
#include <memory>
#include <mutex>
#include <queue>
#include <thread>
#include <vector>
```

Classes

- class iris::IrisClient

Functions

- **NAMESPACE_IRIS_INTERNAL_START** (service) class IrisServiceTcpServer

9.3.1 Detailed Description

Iris client which supports multiple methods to connect to other Iris executables.

Date

9.4 IrisClient.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisClient_h
8 #define ARM_INCLUDE_IrisClient_h
9
10 #include "iris/IrisInstance.h"
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisErrorCode.h"
14 #include "iris/detail/IrisInterface.h"
15 #include "iris/detail/IrisLogger.h"
16 #include "iris/detail/IrisUtils.h"
17 #include "iris/detail/IrisCommaSeparatedParameters.h"
18
19 #include "iris/impl/IrisChannelRegistry.h"
20 #include "iris/impl/IrisMessageQueue.h"
21 #include "iris/impl/IrisPlugin.h"
22 #include "iris/impl/IrisProcessEventsThread.h"
23 #include "iris/impl/IrisRpcAdapterTcp.h"
24 #include "iris/impl/IrisTcpSocket.h"
25 #include "iris/IrisInstance.h"
26
27 #include <map>
28 #include <memory>
29 #include <mutex>
30 #include <queue>
31 #include <thread>
32 #include <vector>
33
34 NAMESPACE_IRIS_INTERNAL_START(service)
35 class IrisServiceTcpServer;
36 NAMESPACE_IRIS_INTERNAL_END
37
38 NAMESPACE_IRIS_START
39
40 class IrisClient
41     : public IrisInterface
42     , public impl::IrisProcessEventsInterface
43     , public IrisConnectionInterface
44 {
45 public:
47     IrisClient(const std::string& instName = std::string(), const std::string& connectionSpec =
       std::string())
48     {
49         init(IRIS_TCP_CLIENT, instName);
50         if (!connectionSpec.empty())
51         {
52             connect(connectionSpec);
53         }
54     }
55
57     IrisClient(const service::IrisServiceTcpServer*, const std::string& instName = std::string())
58     {
59         init(IRIS_SERVICE_SERVER, instName);
60     }
61
72     IrisClient(const std::string& hostname, uint16_t port, const std::string& instName = std::string())
73     {
74         init(IRIS_TCP_CLIENT, instName);
75         std::string ignored_error;
76         IrisErrorCode status = connect(hostname, port, port ? 1000 : 100, ignored_error);
77         if (status != E_ok)
78         {
79             throw IrisErrorExceptionString(status, "Failed to connect to Iris TCP server");
80         }
81     }
82
84     virtual ~IrisClient()
85     {
86         disconnect();
87
88         // Do not rely on destructor order. The socket_thread expects this
89         // object to be fully alive.
```

```
90          if (socket_thread)
91          {
92              socket_thread->terminate();
93          }
94
95          switch (mode)
96          {
97          case IRIS_TCP_CLIENT:
98              socketSet.removeSocket(&sock);
99              break;
100
101         case IRIS_SERVICE_SERVER:
102             socketSet.removeSocket(service_socket);
103             // remove service_socket TODO safer memory management
104             delete service_socket;
105             break;
106         }
107
108         iris::sleepMs(sleepOnDestructionMs);
109     }
110
112     const std::string connectionHelpStr =
113         "Supported connection types:\n"
114         "tcp[=HOST][,port=PORT][,timeout=T]\n"
115         "  Connect to an Iris TCP server on HOST:PORT.\n"
116         "  The default for HOST is 'localhost' and the default for PORT is 0 if HOST is 'localhost' and
    7100 otherwise. If PORT is 0 then a port scan on ports 7100 to 7109 is done.\n"
117         "  T is the connection timeout in ms (defaults to 100 if PORT==0, else 1000).\n"
118         "\n"
119         "socketfd=FD[,timeout=T]\n"
120         "  Use socket file descriptor FD as an established UNIX domain socket connection.\n"
121         "  T is the timeout for the Iris handshake in ms.\n"
122         "\n"
123         "General parameters:\n"
124         "  verbose=N: Increase verbose level of IrisClient to level N (0..3).\n";
125
133     void connect(const std::string& connectionSpec)
134     {
135         IrisCommaSeparatedParameters params(connectionSpec, "1");
136
137         // Emit help message?
138         if (params.have("help"))
139         {
140             throw IrisErrorExceptionString(E_help_message, connectionHelpStr);
141         }
142
143         // Increase verbose level?
144         setVerbose(unsigned(params.getUint("verbose", 0)), /*increaseOnly=*/true);
145
146         // Validate connection type.
147         if (unsigned(params.have("tcp")) + unsigned(params.have("socketfd")) != 1)
148         {
149             throw IrisErrorExceptionString(E_not_connected, "Exactly one out of \"tcp\", \"socketfd\"
    and \"help\" must be specified (got \"" + connectionSpec + "\"). Specify \"help\" to get a list of
    all supported connection types.");
150         }
151
152         if (params.have("tcp"))
153         {
154             std::string hostname = params.getStr("tcp");
155             if (hostname == "1")
156             {
157                 hostname = "localhost";
158             }
159             uint16_t port = uint16_t(params.getUint("port", hostname == "localhost" ? 0 : 7100));
160             unsigned timeoutInMs = unsigned(params.getUint("timeout", port == 0 ? 100 : 1000));
161             if (params.haveUnusedParameters())
162             {
163                 throw IrisErrorExceptionString(E_not_connected, params.getUnusedParametersMessage("Error
    in 'tcp' connection parameters: "));
164             }
165             std::string errorResponse;
166             IrisErrorCode status = connect(hostname, port, timeoutInMs, errorResponse);
167             if (status != E_ok)
168             {
169                 throw IrisErrorExceptionString(status, errorResponse);
170             }
171         }
172
173         if (params.have("socketfd"))
174         {
175             SocketFd socketfd = SocketFd(params.getUint("socketfd"));
176             unsigned timeoutInMs = unsigned(params.getUint("timeout", 1000));
177             if (params.haveUnusedParameters())
178             {
179                 throw IrisErrorExceptionString(E_not_connected, params.getUnusedParametersMessage("Error
    in 'socketfd' connection parameters: "));
```

```
180             }
181             connectSocketFd(socketfd, timeoutInMs);
182         }
183     }
184
188     IrisErrorCode connect(const std::string& hostname, uint16_t port, unsigned timeoutInMs, std::string&
    errorResponseOut)
189     {
190         assert(mode == IRIS_TCP_CLIENT);
191
192         if (verbose)
193             log.info("IrisClient::connect(hostname=%s, port=%u, timeout=%u) enter\n", hostname.c_str(),
    port, timeoutInMs);
194
195         // Already connected?
196         IrisErrorCode error = E_ok;
197         if (adapter.isConnected() || sock.isConnected())
198         {
199             error = E_already_connected;
200             goto done;
201         }
202
203         // hostname==localhost and port==0 means port scan.
204         if ((hostname == "localhost") && (port == 0))
205         {
206             const uint16_t startport = 7100;
207             const uint16_t endport = 7109;
208             for (port = startport; port <= endport; port++)
209             {
210                 std::string errorMessage;
211                 if (connect(hostname, port, timeoutInMs, errorResponseOut) == iris::E_ok)
212                     return E_ok;
213             }
214             errorResponseOut = "No Iris TCP server found on ports " + std::to_string(startport) + ".." +
    std::to_string(endport) + "\n";
215             error = E_not_connected;
216             goto done;
217         }
218
219         if (!sock.isCreated())
220         {
221             sock.create();
222             sock.setNonBlocking();
223
224             // Unblock a potentially blocked worker thread which so far is waiting indefinitely
225             // on 'no socket'. This thread will block again on the socket we just created.
226             socketSet.stopWaitForEvent();
227         }
228
229         // Connect to server.
230         error = sock.connect(hostname, port, timeoutInMs);
231         if (error != E_ok)
232         {
233             errorResponseOut = "Error connecting to " + hostname + ":" + std::to_string(port);
234             sock.close();
235             goto done;
236         }
237
238         // Initialize client.
239         error = initClient(timeoutInMs, errorResponseOut);
240         if (error == E_ok)
241         {
242             connectionStr = hostname + ":" + std::to_string(port);
243         }
244
245         // Return error code (if any).
246     done:
247         if (verbose)
248             log.info("IrisClient::connect() leave (%s)\n", irisErrorCodeCStr(error));
249         return error;
250     }
251
254     void connectSocketFd(SocketFd socketfd, unsigned timeoutInMs = 1000)
255     {
256         assert(mode == IRIS_TCP_CLIENT);
257
258         if (verbose)
259             log.info("IrisClient::connectSocketFd(socketfd=%llu, timeout=%u)\n", (long long)socketfd,
    timeoutInMs);
260
261         // Already connected?
262         std::string errorResponse;
263         IrisErrorCode error = E_ok;
264         if (adapter.isConnected() || sock.isConnected())
265         {
266             throw IrisErrorExceptionString(E_already_connected, "Already connected.");
267         }
```

```
268
269         sock.setSocketFd(socketfd);
270         sock.setNonBlocking();
271
272         // Unblock a potentially blocked worker thread which so far is waiting indefinitely
273         // on 'no socket'. This thread will block again on the socket we just created.
274         socketSet.stopWaitForEvent();
275
276         // Initialize client.
277         error = initClient(timeoutInMs, errorResponse);
278         if (error != E_ok)
279         {
280             throw IrisErrorExceptionString(error, errorResponse);
281         }
282
283         connectionStr = "(connected via socketfd)";
284     }
285
288     IrisErrorCode disconnect()
289     {
290         // Tell IrisInstance to stop sending requests to us.
291         // All Iris calls (including the inevitable final
292         // instanceRegistry_unregisterInstance()) will return
293         // E_not_connected from now on.
294         irisInstance.setConnectionInterface(nullptr);
295
296         connectionStr = "(not connected)";
297
298         if (mode != IRIS_TCP_CLIENT)
299         {
300             return E_ok;
301         }
302
303         // We just close the TCP connection. This is a first-class operation which always must be
    handled gracefully by the server.
304         // The server needs to do all cleanup automatically.
305         IrisErrorCode errorCode = E_ok;
306         if (adapter.isConnected())
307             errorCode = adapter.closeConnection();
308         if (sock.isConnected())
309         {
310             if (errorCode != E_ok)
311                 sock.close();
312             else
313                 errorCode = sock.close();
314         }
315         return errorCode;
316     }
317
319     bool isConnected() const
320     {
321         return adapter.isConnected();
322     }
323
325     IrisInterface* getSendingInterface()
326     {
327         return this;
328     }
329
332     void setInstanceName(const std::string& instName)
333     {
334         if (irisInstance.isRegistered())
335         {
336             throw IrisErrorExceptionString(E_instance_already_registered, "IrisClient::setInstanceName()
    must be called before connect().");
337         }
338         irisInstanceInstName = instName;
339     }
340
343     IrisInstance& getIrisInstance() { return irisInstance; }
344
347     void setSleepOnDestructionMs(uint64_t sleepOnDestructionMs_)
348     {
349         sleepOnDestructionMs = sleepOnDestructionMs_;
350     }
351
352
353     // --- IrisProcessEventsInterface implementation ---
354
371     virtual void processEvents() override
372     {
373         if (verbose)
374             log.info("IrisClient::processEvents() enter\n");
375
376         // in IRIS_SERVICE_SERVER mode, the adapter should work as server and hence call
377         // function processEventsServer()
378         switch (mode)
```

```
379         {
380         case IRIS_TCP_CLIENT:
381             adapter.processEventsClient();
382             break;
383         case IRIS_SERVICE_SERVER:
384             adapter.processEventsServer();
385             break;
386         }
387 
388         if (verbose)
389             log.info("IrisClient::processEvents() leave\n");
390     }
391 
395     virtual void waitForEvent() override
396     {
397         if (verbose)
398             log.info("IrisClient::waitForEvent() enter\n");
399         socketSet.waitForEvent(1000);
400         if (verbose)
401             log.info("IrisClient::waitForEvent() leave\n");
402     }
403 
406     virtual void stopWaitForEvent() override
407     {
408         if (verbose)
409             log.info("IrisClient::stopWaitForEvent()\n");
410         socketSet.stopWaitForEvent();
411     }
412 
414     void setPreferredSendingFormat(impl::IrisRpcAdapterTcp::Format p)
415     {
416         adapter.setPreferredSendingFormat(p);
417     }
418 
420     impl::IrisRpcAdapterTcp::Format getEffectiveSendingFormat() const
421     {
422         return adapter.getEffectiveSendingFormat();
423     }
424 
426     void setVerbose(unsigned level, bool increaseOnly = false)
427     {
428         if (increaseOnly && (level < verbose))
429         {
430             return;
431         }
432 
433         verbose = level;
434         if (verbose)
435             log.info("IrisClient: verbose logging enabled (level %d)\n", verbose);
436         if (mode == IRIS_TCP_CLIENT)
437         {
438             sock.setVerbose(verbose);
439         }
440         socketSet.setVerbose(verbose);
441     }
442 
444     void setIrisMessageLogLevel(unsigned level) { irisMessageLogLevel = level;
      log.setIrisMessageLogLevel(irisMessageLogLevel); }
445 
447     std::string getConnectionStr() const { return connectionStr; }
448 
449 private:
451     enum Mode
452     {
453         IRIS_TCP_CLIENT,
454         IRIS_SERVICE_SERVER
455     };
456 
457     // Shared code for constructors in client mode.
458     void init(Mode mode_, const std::string& instName)
459     {
460         log.setLogContext("IrisTC");
461         mode = mode_;
462 
463         // Set instance name of contained IrisInstance.
464         if (instName.empty())
465         {
466             setInstanceName("client.IrisClient");
467         }
468         else
469         {
470             setInstanceName(instName);
471         }
472 
473         // Enable verbose logging?
474         setVerbose(static_cast<unsigned>(getEnvU64("IRIS_TCP_CLIENT_VERBOSE")), true);
475         irisMessageLogLevel = unsigned(getEnvU64("IRIS_TCP_CLIENT_LOG_MESSAGES"));
```

```
476         log.setIrisMessageLogLevel(irisMessageLogLevel);
477         log.setIrisMessageGetInstNameFunc([&](InstanceId instId){ return getInstName(instId); });
478 
479         if (mode == IRIS_TCP_CLIENT)
480         {
481             socketSet.addSocket(&sock);
482         }
483         sendingInterface = adapter.getSendingInterface();
484 
485         // Intercept all calls to the global instance since we must modify
    instanceRegistry_registerInstance() and
486         // instanceRegistry_unregisterInstance() and their responses.
487         instIdToInterface.push_back(&globalInstanceSendingInterface); // This must be index 0 in the
    vector (instId 0 == global instance).
488 
489         if (mode == IRIS_SERVICE_SERVER)
490         {
491             socket_thread = std::unique_ptr<impl::IrisProcessEventsThread>(new
    impl::IrisProcessEventsThread(this, "TcpSocket"));
492         }
493     }
494 
496     IrisErrorCode initClient(unsigned timeoutInMs, std::string& errorResponseOut)
497     {
498         assert(mode == IRIS_TCP_CLIENT);
499 
500         // Initialize IrisRpcAdapterTcp.
501         adapter.initClient(&sock, &socketSet, &receivingInterface, verbose);
502 
503         // Handshake.
504         IrisErrorCode error = adapter.handshakeClient(errorResponseOut, timeoutInMs);
505 
506         // Start a thread to process incoming data in the background.
507         socket_thread = std::unique_ptr<impl::IrisProcessEventsThread>(new
    impl::IrisProcessEventsThread(this, "TcpSocket"));
508 
509         // Initialize IrisInstance.
510         irisInstance.setConnectionInterface(this);
511         irisInstance.registerInstance(irisInstanceInstName, iris::IrisInstance::UNIQUIFY |
    iris::IrisInstance::THROW_ON_ERROR);
512 
513         return error;
514     }
515 
519     virtual void irisHandleMessage(const uint64_t* message) override
520     {
521         // Log message?
522         if (irisMessageLogLevel)
523         {
524             log.irisMessage(message);
525         }
526 
527         // This calls one of these:
528         // - this->globalInstanceSendingInterface_irisHandleMessage(); (for requests, instId == 0)
529         // - Iris interface of a local instance (if a local instance talks to a local instance)
530         // - sendingInterface (to send message to server using TCP)
531         findInterface(IrisU64JsonReader::getInstId(message))->irisHandleMessage(message);
532     }
533 
536     void globalInstanceSendingInterface_irisHandleMessage(const uint64_t* message)
537     {
538         // This is only ever called for instId == 0.
539         assert(IrisU64JsonReader::getInstId(message) == 0);
540         assert(IrisU64JsonReader::isRequestOrNotification(message));
541 
542         // Decode request.
543         IrisU64JsonReader          r(message);
544         IrisU64JsonReader::Request req    = r.openRequest();
545         std::string                method = req.getMethod();
546 
547         if (method == "instanceRegistry_registerInstance")
548         {
549             RequestId requestId = req.getRequestId();
550 
551             // We received an instanceRegistry_registerInstance() request from a local instance:
552             // - Create a new request id which is unique to this request for this TCP channel. (This is
    not required to be globally unique.)
553             // - Allocate an ongoingInstanceRegistryCalls slot for this new request id and remember the
    original request id and params.channelId in it.
554             // - Modify request id of request to the new request id so we can recognize the response
    later.
555             // - Send modified request.
556 
557             // Create a new request id which is unique to this request for this TCP channel. (This is
    not required to be globally unique.)
558             RequestId newRequestId = generateNewRequestIdForRegisterInstanceCall();
559 
```

```
560             // Get channelId.
561             uint64_t channelId = IRIS_UINT64_MAX;
562             if (!req.paramOptional(ISTR("channelId"), channelId))
563             {
564                 // Strange. 'params.channelId' is missing. This should never happen.
565                 log.error(
566                     "IrisClient::receivingInterface_irisHandleMessage():"
567                     " Received instanceRegistry_registerInstance() request without channelId
    parameter:\n%s\n",
568                     messageToString(message).c_str());
569                 goto send;
570             }
571 
572             {
573                 std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
574                 // Allocate an ongoingInstanceRegistryCalls slot for this new request id and remember
    the
575                 // original request id and params.channelId in it.
576                 ongoingInstanceRegistryCalls[newRequestId] = OngoingInstanceRegistryCallEntry(method,
577                                                                                               requestId,
578 
    channelId);
579             }
580 
581             // Create a modified reaquest that:
582             //  - sets the new request id so we can recognize the response later.
583             //  - removes the channelId parameter (it only has meaning in-process)
584             IrisU64JsonReader original_message(message);
585             IrisU64JsonWriter modified_message;
586 
587             {
588                 IrisU64JsonReader::Request original_req = original_message.openRequest();
589 
590                 IrisU64JsonWriter::Request new_req =
    modified_message.openRequest(original_req.getMethod(),
591 
    original_req.getInstId());
592                 new_req.setRequestId(newRequestId);
593 
594                 std::string param;
595                 while (original_req.readNextParam(param))
596                 {
597                     if ((param == "channelId") || (param == "instId"))
598                     {
599                         // Skip the params we want to remove (channelId)
600                         // and skip instId too because that will have already been filled in.
601                         // skip over the value to the next parameter
602                         original_message.skip();
603                     }
604                     else
605                     {
606                         new_req.paramSlow(param);
607 
608                         // Pass through the original value
609                         IrisValue value;
610                         persist(original_message, value);
611                         persist(modified_message, value);
612                     }
613                 }
614             }
615 
616             // Send modified request.
617             sendingInterface->irisHandleMessage(modified_message.getMessage());
618             return;
619         }
620         else if (method == "instanceRegistry_unregisterInstance")
621         {
622             // We received an instanceRegistry_unregisterInstance() request from a local instance:
623             // - Allocate an ongoingInstanceRegistryCalls slot for the request id and remember the
    instId of the unregistered instance in it.
624             // - Send request unmodified.
625 
626             // Get params.aInstId.
627             InstanceId aInstId = IRIS_UINT64_MAX;
628             if (!req.paramOptional(ISTR("aInstId"), aInstId))
629             {
630                 // Strange. 'params.aInstId' is missing. This should never happen.
631                 log.error(
632                     "IrisClient::receivingInterface_irisHandleMessage():"
633                     " Received instanceRegistry_unregisterInstance() request without aInstId
    parameter:\n%s\n",
634                     messageToString(message).c_str());
635                 goto send;
636             }
637 
638             if (!req.isNotification())
639             {
```

```
640                 RequestId requestId = req.getRequestId();
641
642                 if (aInstId == getCallerInstId(requestId))
643                 {
644                     std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
645                     // There will be a response to this request so we need to remember the interface to
    send it to.
646                     // Allocate an ongoingInstanceRegistryCalls slot for the request id and remember the
    instId of the unregistered instance in it.
647                     ongoingInstanceRegistryCalls[requestId] = OngoingInstanceRegistryCallEntry(method,
    aInstId);
648                     goto send;
649                 }
650             }
651
652             // There will be no more communication to the instance being unregistered.
653             // Remove instance from instIdToInterface.
654             assert(aInstId < InstanceId(instIdToInterface.size()));
655             // sendingInterface: Forward messages to unknown instIds to the server. The global instance
    may have reassigned the same instId to some other instance behind the server which exists.
656             instIdToInterface[aInstId] = sendingInterface;
657
658             // Intended fallthrough to send original request.
659         }
660         else if (method == "instanceRegistry_getList")
661         {
662             // We received an instanceRegistry_getList() request from a local instance:
663             // - We want to remember/snoop all returned instance names we get in the response (for
    logging).
664             // - Allocate an ongoingInstanceRegistryCalls slot for the request id in order to recognize
    the response.
665             // - Send request unmodified.
666
667             if (!req.isNotification())
668             {
669                 RequestId requestId = req.getRequestId();
670                 std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
671                 ongoingInstanceRegistryCalls[requestId] = OngoingInstanceRegistryCallEntry(method);
672             }
673
674             // Intended fallthrough to send original request.
675         }
676
677     send:
678         // Send original message.
679         sendingInterface->irisHandleMessage(message);
680     }
681
685     void receivingInterface_irisHandleResponse(const uint64_t* message)
686     {
687         {
688             std::lock_guard<std::mutex> lock(ongoingInstanceRegistryCallsMutex);
689
690             if (!ongoingInstanceRegistryCalls.empty())
691             {
692                 // Slow path is only used while a instanceRegistry_registerInstance() or
    instanceRegistry_unregisterInstance()
693                 // call is ongoing. This is usually only the case at startup and shutdown.
694
695                 // We need to check whether this is the response to either
696                 // instanceRegistry_registerInstance() or
697                 // instanceRegistry_unregisterInstance() or
698                 // any other response.
699
700                 // Decode response.
701                 IrisU64JsonReader           r(message);
702                 IrisU64JsonReader::Response resp = r.openResponse();
703                 RequestId requestId = resp.getRequestId();
704
705                 // Check whether this is a response to one of our pending requests.
706                 OngoingInstanceRegistryCallMap::iterator i =
    ongoingInstanceRegistryCalls.find(requestId);
707                 if (i == ongoingInstanceRegistryCalls.end())
708                 {
709                     goto send; // None of the pending responses. Handle in the normal way.
710                 }
711
712                 if (i->second.method == "instanceRegistry_registerInstance")
713                 {
714                     // This is a response to a previous instanceRegistry_registerInstance() call:
715
716                     IrisInterface* responseIfPtr = channel_registry.getChannel(i->second.channelId);
717
718                     if (resp.isError())
719                     {
720                         // The call failed, pass on the message.
721                         responseIfPtr->irisHandleMessage(message);
```

```
722                     }
723                     else
724                     {
725                         // The call succeeded:
726                         // - add new instId to our local instance registry
727                         // - translate request id back to the original request id
728                         // - send this modified response to the caller
729                         // - erase this entry in ongoingInstanceRegistryCalls
730 
731                         // Add instance to instIdToInterface.
732                         InstanceId newInstId;
733                         if (!resp.getResultReader().openObject().memberOptional(ISTR("instId"),
    newInstId))
734                         {
735                             // Strange. 'result.instId' is missing. This should never happen.
736                             log.error(
737                                 "IrisClient::receivingInterface_irisHandleResponse():"
738                                 " Received instanceRegistry_registerInstance() response without
    result.instId:\n%s\n",
739                                 messageToString(message).c_str());
740                         }
741                         else
742                         {
743                             // This is a valid response for instanceRegistry_registerInstance(): Enter
    newInstId into instIdToInterface.
744                             findInterface(newInstId);
745                             instIdToInterface[newInstId] = responseIfPtr;
746                         }
747 
748                         // Remember instance name.
749                         std::string newInstName;
750                         if (resp.getResultReader().openObject().memberOptional(ISTR("instName"),
    newInstName))
751                         {
752                             setInstName(newInstId, newInstName);
753                         }
754 
755                         // Translate the id back to the id of the original request and use the
    responseIfPtr to send the response.
756                         IrisU64JsonWriter modifiedMessageWriter;
757                         modifiedMessageWriter.copyMessageAndModifyId(message, i->second.id);
758 
759                         // Log message?
760                         if (irisMessageLogLevel)
761                         {
762                             log.irisMessage(modifiedMessageWriter.getMessage());
763                         }
764 
765                         responseIfPtr->irisHandleMessage(modifiedMessageWriter.getMessage());
766                     }
767 
768                     // Remove ongoingInstanceRegistryCalls entry now that we have seen the response.
769                     ongoingInstanceRegistryCalls.erase(i);
770                     return;
771                 }
772                 else if (i->second.method == "instanceRegistry_unregisterInstance")
773                 {
774                     // This is a response to a previous instanceRegistry_unregisterInstance() call:
775                     // - remove this instId from our local instance registry
776                     // - remove this entry from ongoingInstanceRegistryCalls
777                     // - send response to caller
778 
779                     InstanceId aInstId = i->second.id;
780 
781                     // Remeber the old response interface in case we need it after we override it
782                     IrisInterface* aInst_responseIf = instIdToInterface[aInstId];
783 
784                     // Remove instance from instIdToInterface.
785                     assert(aInstId < InstanceId(instIdToInterface.size()));
786                     // sendingInterface: Forward messages to unknown instIds to the server. The global
    instance may have reassigned the same instId to some other instance behind the server which exists.
787                     instIdToInterface[aInstId] = sendingInterface;
788                     setInstName(aInstId, ""); // IrisLogger will generate a default name for unknown
    instance ids.
789                     // Remove ongoingInstanceRegistryCalls entry.
790                     ongoingInstanceRegistryCalls.erase(i);
791 
792                     if (aInstId == resp.getInstId())
793                     {
794                         // An instance unregistered itself so we need to call it directly rather than
795                         // go through the normal message handler because we just set that to forward
796                         // messages to this instId to the server.
797                         aInst_responseIf->irisHandleMessage(message);
798                         return;
799                     }
800 
801                     // Intended fallthrough to irisHandleMessage(message).
```

```
802                 }
803                 else if (i->second.method == "instanceRegistry_getList")
804                 {
805                     // This is a response to a previous instanceRegistry_getList() call:
806                     // - remember all instance names (for logging)
807                     // - send response to caller
808
809                     // Remove ongoingInstanceRegistryCalls entry.
810                     ongoingInstanceRegistryCalls.erase(i);
811                     try
812                     {
813                         // Peek into instance list. We do not care whether this is just
814                         // a subset of all instances or not. We take what we can get.
815                         std::vector<InstanceInfo> instanceInfoList;
816                         resp.getResult(instanceInfoList);
817                         for (const auto& instanceInfo: instanceInfoList)
818                         {
819                             setInstName(instanceInfo.instId, instanceInfo.instName);
820                         }
821                     }
822                     catch(const IrisErrorException&)
823                     {
824                         // Silently ignore bogus responses. The caller will handle the error.
825                     }
826                     // Intended fallthrough to irisHandleMessage(message).
827                 }
828             }
829         }
830
831     send:
832         // Handle response in the normal way.
833         irisHandleMessage(message);
834     }
835
840     RequestId generateNewRequestIdForRegisterInstanceCall()
841     {
842         return nextInstIdForRegisterInstanceCall++;
843     }
844
850     IrisInterface* findInterface(InstanceId instId)
851     {
852         if (instId >= IrisMaxTotalInstances)
853         {
854             log.error("IrisClient::findInterface(instId=0x%08x): got ridiculously high instId",
    int(instId));
855             return sendingInterface;
856         }
857         if (instId >= InstanceId(instIdToInterface.size()))
858         {
859             instIdToInterface.resize(instId + 100, sendingInterface);
860         }
861         return instIdToInterface[instId];
862     }
863
865     class GlobalInstanceSendingInterface : public IrisInterface
866     {
867     public:
868         GlobalInstanceSendingInterface(IrisClient* parent_)
869             : parent(parent_)
870         {
871         }
872
876         virtual void irisHandleMessage(const uint64_t* message) override
877         {
878             if (IrisU64JsonReader::isRequestOrNotification(message))
879             {
880                 // Intercept requests to the global instance so we can snoop on
881                 // calls to instanceRegistry_registerInstance()
882                 parent->globalInstanceSendingInterface_irisHandleMessage(message);
883             }
884             else
885             {
886                 // This is called for responses sent from clients to the global instance.
887                 // Simply forward them as usual. Nothing to intercept.
888                 parent->sendingInterface->irisHandleMessage(message);
889             }
890         }
891
892     private:
893         IrisClient* const parent;
894     };
895
897     class ReceivingInterface : public IrisInterface
898     {
899     public:
900         ReceivingInterface(IrisLogger& log_, IrisClient* parent_)
901             : parent(parent_)
```

```
902             , log(log_)
903         {
904         }
905
907         virtual void irisHandleMessage(const uint64_t* message) override
908         {
909             InstanceId instId = IrisU64JsonReader::getInstId(message);
910
911             if (instId >= InstanceId(instId_to_thread_id.size()))
912             {
913                 // We do not have an entry for this instance therefore
914                 // we have not been asked to marshal requests to a specific
915                 // thread and should use the default.
916                 // Todo: Remove once IrisMessageQueue and IrisProcessEventsThread are gone
917                 setHandlerThread(instId, getDefaultThreadId());
918             }
919
920             // Todo: Refactor once IrisMessageQueue and IrisProcessEventsThread are gone
921             std::thread::id thread_id = instId_to_thread_id[instId];
922             if (thread_id == std::this_thread::get_id())
923             {
924                 // Message has already been marshalled, forward on
925                 if (IrisU64JsonReader::isRequestOrNotification(message))
926                 {
927                     parent->irisHandleMessage(message);
928                 }
929                 else
930                 {
931                     parent->receivingInterface_irisHandleResponse(message);
932                 }
933             }
934             else
935             {
936                 message_queue.push(message, thread_id);
937             }
938         }
939
940         void setHandlerThread(InstanceId instId, std::thread::id thread_id)
941         {
942             if (instId >= IrisMaxTotalInstances)
943             {
944                 log.error(
945                     "IrisClient::ReceivingInterface::setHandlerThread(instId=0x%08x):"
946                     " got ridiculously high instId",
947                     int(instId));
948             }
949             else if (instId >= InstanceId(instId_to_thread_id.size()))
950             {
951                 instId_to_thread_id.resize(instId + 100, getDefaultThreadId());
952             }
953
954             instId_to_thread_id[instId] = thread_id;
955         }
956
957         IrisErrorCode processMessagesForCurrentThread(bool waitForAMessage)
958         {
959             if (waitForAMessage)
960             {
961                 IrisErrorCode code = message_queue.waitForMessageForCurrentThread();
962                 if (code != E_ok)
963                 {
964                     return code;
965                 }
966             }
967             message_queue.processMessagesForCurrentThread();
968
969             return E_ok;
970         }
971
972     private:
974         std::thread::id getDefaultThreadId()
975         {
976             return process_events_thread.getThreadId();
977         }
978
979         IrisClient* const parent;
980
982         impl::IrisMessageQueue message_queue{this};
983
985         std::vector<std::thread::id> instId_to_thread_id;
986
987         IrisLogger& log;
988
991         impl::IrisProcessEventsThread process_events_thread{&message_queue, "ClientMsgHandlr"};
992     };
993
994 public: // IrisConnectionInterface
```

```
995     virtual uint64_t registerIrisInterfaceChannel(IrisInterface* iris_interface) override
996     {
997         return channel_registry.registerChannel(iris_interface);
998     }
999
1000     virtual void unregisterIrisInterfaceChannel(uint64_t channelId) override
1001     {
1002         IrisInterface* if_to_remove = channel_registry.getChannel(channelId);
1003
1004         std::vector<InstanceId> instIds_for_channel;
1005
1006         for (size_t i = 0; i < instIdToInterface.size(); i++)
1007         {
1008             if (instIdToInterface[i] == if_to_remove)
1009             {
1010                 InstanceId instId = InstanceId(i);
1011                 instIds_for_channel.push_back(instId);
1012             }
1013         }
1014         if (instIds_for_channel.size() > 0)
1015         {
1016             // Create an instance to call instanceRegistry_unregisterInstance() with.
1017             IrisInstance instance_killer(this, "framework.IrisClient.instance_killer",
1018                                          IrisInstance::UNIQUIFY);
1019             for (InstanceId instId : instIds_for_channel)
1020             {
1021                 instance_killer.irisCall().instanceRegistry_unregisterInstance(instId);
1022             }
1023         }
1024
1025         channel_registry.unregisterChannel(channelId);
1026     }
1027
1028     virtual IrisErrorCode processAsyncMessages(bool waitForAMessage) override
1029     {
1030         return receivingInterface.processMessagesForCurrentThread(waitForAMessage);
1031     }
1032
1033     virtual IrisInterface* getIrisInterface() override
1034     {
1035         return this;
1036     }
1037
1038     uint64_t registerChannel(IrisC_CommunicationChannel* channel)
1039     {
1040         return channel_registry.registerChannel(channel);
1041     }
1042
1043     void unregisterChannel(uint64_t channelId)
1044     {
1045         channel_registry.unregisterChannel(channelId);
1046     }
1047
1048     // function called by class IrisPlugin
1049     uint64_t registerChannel(IrisC_CommunicationChannel* channel, const ::std::string& path)
1050     {
1051         (void) path;
1052         return channel_registry.registerChannel(channel);
1053     }
1054
1055 public:
1058     void loadPlugin(const std::string& plugin_name)
1059     {
1060         assert(mode == IRIS_SERVICE_SERVER);
1061         assert(plugin == nullptr);
1062         plugin = std::unique_ptr<impl::IrisPlugin<IrisClient»(new impl::IrisPlugin<IrisClient>(this,
      plugin_name));
1063     }
1064
1065     void unloadPlugin()
1066     {
1067         assert(mode == IRIS_SERVICE_SERVER);
1068         plugin = nullptr;
1069     }
1070
1077     void initServiceServer(impl::IrisTcpSocket* socket_)
1078     {
1079         assert(mode == IRIS_SERVICE_SERVER);
1080         service_socket = socket_;
1081         socketSet.addSocket(service_socket);
1082         adapter.initServiceServer(service_socket, &socketSet, &receivingInterface, verbose);
1083     }
1084
1085 private:
1087     std::string getInstName(InstanceId instId)
1088     {
1089         // IrisLogger will generate a default name for unknown instances (empty string).
```

```
1090         return instId < instIdToInstName.size() ? instIdToInstName[instId] : std::string();
1091     }
1092
1094     void setInstName(InstanceId instId, const std::string& instName)
1095     {
1096         // Ignore ridiculously high instIds (prigramming errors).
1097         if (instId >= IrisMaxTotalInstances)
1098         {
1099             return;
1100         }
1101
1102         if (instId >= instIdToInstName.size())
1103         {
1104             instIdToInstName.resize(instId + 1, "");
1105         }
1106
1107         instIdToInstName[instId] = instName;
1108     }
1109
1110     // --- Private data. ---
1111
1113     IrisLogger log;
1114
1116     IrisInstance irisInstance;
1117
1119     std::string irisInstanceInstName;
1120
1122     GlobalInstanceSendingInterface globalInstanceSendingInterface{this};
1123
1125     ReceivingInterface receivingInterface{log, this};
1126
1128     impl::IrisTcpSocket sock{log, 0};
1129
1131     impl::IrisTcpSocket* service_socket{nullptr};
1132
1134     impl::IrisTcpSocketSet socketSet{log, 0};
1135
1137     std::vector<IrisInterface*> instIdToInterface;
1138
1140     std::vector<std::string> instIdToInstName;
1141
1143     impl::IrisChannelRegistry channel_registry{log};
1144
1146     IrisInterface* sendingInterface{nullptr};
1147
1150     uint32_t nextInstIdForRegisterInstanceCall{0};
1151
1153     struct OngoingInstanceRegistryCallEntry
1154     {
1155         OngoingInstanceRegistryCallEntry()
1156         {
1157         }
1158
1159         OngoingInstanceRegistryCallEntry(const std::string& method_, uint64_t id_ = IRIS_UINT64_MAX,
1160                                          uint64_t channelId_ = IRIS_UINT64_MAX)
1161             : method(method_)
1162             , id(id_)
1163             , channelId(channelId_)
1164         {
1165         }
1166
1167         std::string method;                  // instanceRegistry_registerInstance,
     instanceRegistry_unregisterInstance or instanceRegistry_getList().
1168         uint64_t id{IRIS_UINT64_MAX};        // For instanceRegistry_registerInstance(): Original
     request id. For instanceRegistry_unregisterInstance(): params.aInstId.
1169         uint64_t channelId{IRIS_UINT64_MAX}; // For instanceRegistry_registerInstance() only:
     params.channelId.
1170     };
1171
1175     typedef std::map<uint64_t, OngoingInstanceRegistryCallEntry> OngoingInstanceRegistryCallMap;
1176
1177     OngoingInstanceRegistryCallMap ongoingInstanceRegistryCalls;
1178
1180     std::mutex ongoingInstanceRegistryCallsMutex;
1181
1183     unsigned verbose{0};
1184
1186     unsigned irisMessageLogLevel{0};
1187
1189     impl::IrisRpcAdapterTcp adapter{log};
1190
1192     std::unique_ptr<impl::IrisProcessEventsThread> socket_thread{nullptr};
1193
1195     Mode mode;
1196
1198     std::string component_name;
1199
```

```
1201     std::unique_ptr<impl::IrisPlugin<IrisClient» plugin{nullptr};
1202
1204     std::string connectionStr{"(not connected)"};
1205
1208     uint64_t sleepOnDestructionMs{};
1209 };
1210
1211 NAMESPACE_IRIS_END
1212
1213 #endif // #ifndef ARM_INCLUDE_IrisClient_h
```

9.5 IrisCommandLineParser.h File Reference

Generic command line parser.

```
#include <cstdint>
#include <map>
#include <string>
#include <vector>
#include <functional>
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisErrorException.h"
```

Classes

- class iris::IrisCommandLineParser
- struct iris::IrisCommandLineParser::Option

  Option container.

9.5.1 Detailed Description

Generic command line parser.

Copyright

9.6 IrisCommandLineParser.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisCommandLineParser_h
8 #define ARM_INCLUDE_IrisCommandLineParser_h
9
10 #include <cstdint>
11 #include <map>
12 #include <string>
13 #include <vector>
14 #include <functional>
15
16 #include "iris/detail/IrisCommon.h"
17 #include "iris/detail/IrisErrorException.h"
18
19 NAMESPACE_IRIS_START
20
32 #if 0
33 #include <iostream>
34 #include "iris/IrisCommandLineParser.h"
35
36 int main(int argc, const char* argv[])
37 {
38     // Declare command line options.
39     iris::IrisCommandLineParser options("mytool", "Usage: mytool [OPTIONS]\n", "0.0.1");
40     options.addOption('v', "verbose", "Be more verbose (may be specified multiple times)."); // Switch
      option.
41     options.addOption(0, "port", "Specify local server port.", "PORT", "7999"); // Option with argument,
      without a short option.
42
43     // Parse command line.
44     options.parseCommandLine(argc, argv);
45
```

```
46      // Use options.
47      if (options.getSwitch("verbose"))
48      {
49          std::cout « "Verbose level: " « options.getSwitch("verbose") « "\n";
50      }
51      std::cout « "Port:  " « options.getInt("port") « "\n";
52      return 0;
53  }
54  #endif
55  class IrisCommandLineParser
56  {
57  public:
59      struct Option
60      {
61          // Public interface:
62
67          Option& setList(char sep = ',') { listSeparator = sep; return *this; }
68
69      private:
70          // Meta info:
71
73          char shortOption{};
74
77          std::string longOption;
78
80          std::string help;
81
85          std::string formalArgumentName;
86
88          std::string defaultValue;
89
91          char listSeparator{};
92
94          bool hasFormalArgument() const { return !formalArgumentName.empty(); }
95
96          // Actual values from command line:
97
101         std::string value;
102
104         bool isSpecified{};
105
107         void setValue(const std::string& v);
108
110         void unsetValue();
111
112         friend class IrisCommandLineParser;
113     };
114
116     IrisCommandLineParser(const std::string& programName_, const std::string& usageHeader_, const
        std::string& versionStr_);
117
125     Option& addOption(char shortOption, const std::string& longOption, const std::string& help, const
        std::string& formalArgumentName = std::string(), const std::string& defaultValue = std::string());
126
149     int parseCommandLine(int argc, const char* argv[]);
150
153     void noNonOptionArguments();
154
158     void pleaseSpecifyOneOf(const std::vector<std::string>& options, const std::vector<std::string>&
        formalNonOptionArguments = std::vector<std::string>());
159
161     std::string getStr(const std::string& longOption) const;
162
165     int64_t getInt(const std::string& longOption) const;
166
169     uint64_t getUint(const std::string& longOption) const;
170
173     double getDbl(const std::string& longOption) const;
174
176     uint64_t getSwitch(const std::string& longOption) const;
177
179     std::vector<std::string> getList(const std::string& longOption) const;
180
184     std::map<std::string,std::string> getMap(const std::string& longOption) const;
185
189     bool isSpecified(const std::string& longOption) const;
190
192     const std::vector<std::string>& getNonOptionArguments() const;
193
197     void clear();
198
203     int printMessage(const std::string& message, int error = 0, bool exit = false) const;
204
206     int printError(const std::string& message) const;
207
211     int printErrorAndExit(const std::string& message) const;
212
```

```
216     int printErrorAndExit(const IrisErrorException& e) const { return printErrorAndExit(e.errorMessage()
      + "\n"); }
217
229     void setMessageFunc(const std::function<int(const std::string& message, int error, bool exit)>&
      messageFunc);
230
234     static int defaultMessageFunc(const std::string& message, int error, bool exit);
235
239     std::string getHelpMessage() const;
240
244     void setValue(const std::string& longOption, const std::string& value, bool append = false);
245
248     void unsetValue(const std::string& longOption);
249
251     void setProgramName(const std::string& programName_, bool append = false);
252
253 private:
256     Option& getOption(const std::string& longOption);
257
259     const Option& getOption(const std::string& longOption) const;
260
262     std::string programName;
263
265     std::string usageHeader;
266
268     std::string versionStr;
269
271     std::vector<std::string> optionList;
272
275     std::map<std::string, Option> options;
276
278     std::vector<std::string> nonOptionArguments;
279
281     std::function<int(const std::string& message, int error, bool exit)> messageFunc;
282 };
283
284 NAMESPACE_IRIS_END
285
286 #endif // ARM_INCLUDE_IrisCommandLineParser_h
```

9.7 IrisElfDwarfArm.h File Reference

Constants for the register.canonicalRnScheme "ElfDwarf" for architecture Arm.

```
#include "iris/detail/IrisInterface.h"
#include "iris/detail/IrisCommon.h"
```

Enumerations

- enum **ElfDwarfArm** : uint64_t {
  ARM_R0 = 0x2800000000 , **ARM_R1** = 0x2800000001 , **ARM_R2** = 0x2800000002 , **ARM_R3** = 0x2800000003 ,
  ARM_R4 = 0x2800000004 , **ARM_R5** = 0x2800000005 , **ARM_R6** = 0x2800000006 , **ARM_R7** = 0x2800000007 ,
  ARM_R8 = 0x2800000008 , **ARM_R9** = 0x2800000009 , **ARM_R10** = 0x280000000a , **ARM_R11** = 0x280000000b ,
  ARM_R12 = 0x280000000c , **ARM_R13** = 0x280000000d , **ARM_R14** = 0x280000000e , **ARM_R15** = 0x280000000f ,
  ARM_SPSR = 0x2800000080 , **ARM_SPSR_fiq** = 0x2800000081 , **ARM_SPSR_irq** = 0x2800000082 , **ARM_SPSR_abt** = 0x2800000083 ,
  ARM_SPSR_und = 0x2800000084 , **ARM_SPSR_svc** = 0x2800000085 , **ARM_R8_fiq** = 0x2800000097 , **ARM_R9_fiq** = 0x2800000098 ,
  ARM_R10_fiq = 0x2800000099 , **ARM_R11_fiq** = 0x280000009a , **ARM_R12_fiq** = 0x280000009b , **ARM_R13_fiq** = 0x280000009c ,
  ARM_R14_fiq = 0x280000009d , **ARM_R13_irq** = 0x280000009e , **ARM_R14_irq** = 0x280000009f , **ARM_R13_abt** = 0x28000000a0 ,
  ARM_R14_abt = 0x28000000a1 , **ARM_R13_und** = 0x28000000a2 , **ARM_R14_und** = 0x28000000a3 , **ARM_R13_svc** = 0x28000000a4 ,
  ARM_R14_svc = 0x28000000a5 , **ARM_D0** = 0x2800000100 , **ARM_D1** = 0x2800000101 , **ARM_D2** = 0x2800000102 ,

ARM_D3 = 0x2800000103 , **ARM_D4** = 0x2800000104 , **ARM_D5** = 0x2800000105 , **ARM_D6** = 0x2800000106 ,
ARM_D7 = 0x2800000107 , **ARM_D8** = 0x2800000108 , **ARM_D9** = 0x2800000109 , **ARM_D10** = 0x280000010a ,
ARM_D11 = 0x280000010b , **ARM_D12** = 0x280000010c , **ARM_D13** = 0x280000010d , **ARM_D14** = 0x280000010e ,
ARM_D15 = 0x280000010f , **ARM_D16** = 0x2800000110 , **ARM_D17** = 0x2800000111 , **ARM_D18** = 0x2800000112 ,
ARM_D19 = 0x2800000113 , **ARM_D20** = 0x2800000114 , **ARM_D21** = 0x2800000115 , **ARM_D22** = 0x2800000116 ,
ARM_D23 = 0x2800000117 , **ARM_D24** = 0x2800000118 , **ARM_D25** = 0x2800000119 , **ARM_D26** = 0x280000011a ,
ARM_D27 = 0x280000011b , **ARM_D28** = 0x280000011c , **ARM_D29** = 0x280000011d , **ARM_D30** = 0x280000011e ,
ARM_D31 = 0x280000011f , **AARCH64_X0** = 0xb700000000 , **AARCH64_X1** = 0xb700000001 , **AARCH64_X2** = 0xb700000002 ,
AARCH64_X3 = 0xb700000003 , **AARCH64_X4** = 0xb700000004 , **AARCH64_X5** = 0xb700000005 , **AARCH64_X6** = 0xb700000006 ,
AARCH64_X7 = 0xb700000007 , **AARCH64_X8** = 0xb700000008 , **AARCH64_X9** = 0xb700000009 , **AARCH64_X10** = 0xb70000000a ,
AARCH64_X11 = 0xb70000000b , **AARCH64_X12** = 0xb70000000c , **AARCH64_X13** = 0xb70000000d , **AARCH64_X14** = 0xb70000000e ,
AARCH64_X15 = 0xb70000000f , **AARCH64_X16** = 0xb700000010 , **AARCH64_X17** = 0xb700000011 , **AARCH64_X18** = 0xb700000012 ,
AARCH64_X19 = 0xb700000013 , **AARCH64_X20** = 0xb700000014 , **AARCH64_X21** = 0xb700000015 , **AARCH64_X22** = 0xb700000016 ,
AARCH64_X23 = 0xb700000017 , **AARCH64_X24** = 0xb700000018 , **AARCH64_X25** = 0xb700000019 , **AARCH64_X26** = 0xb70000001a ,
AARCH64_X27 = 0xb70000001b , **AARCH64_X28** = 0xb70000001c , **AARCH64_X29** = 0xb70000001d , **AARCH64_X30** = 0xb70000001e ,
AARCH64_SP = 0xb70000001f , **AARCH64_ELR** = 0xb700000021 , **AARCH64_V0** = 0xb700000040 , **AARCH64_V1** = 0xb700000041 ,
AARCH64_V2 = 0xb700000042 , **AARCH64_V3** = 0xb700000043 , **AARCH64_V4** = 0xb700000044 , **AARCH64_V5** = 0xb700000045 ,
AARCH64_V6 = 0xb700000046 , **AARCH64_V7** = 0xb700000047 , **AARCH64_V8** = 0xb700000048 , **AARCH64_V9** = 0xb700000049 ,
AARCH64_V10 = 0xb70000004a , **AARCH64_V11** = 0xb70000004b , **AARCH64_V12** = 0xb70000004c , **AARCH64_V13** = 0xb70000004d ,
AARCH64_V14 = 0xb70000004e , **AARCH64_V15** = 0xb70000004f , **AARCH64_V16** = 0xb700000050 , **AARCH64_V17** = 0xb700000051 ,
AARCH64_V18 = 0xb700000052 , **AARCH64_V19** = 0xb700000053 , **AARCH64_V20** = 0xb700000054 , **AARCH64_V21** = 0xb700000055 ,
AARCH64_V22 = 0xb700000056 , **AARCH64_V23** = 0xb700000057 , **AARCH64_V24** = 0xb700000058 , **AARCH64_V25** = 0xb700000059 ,
AARCH64_V26 = 0xb70000005a , **AARCH64_V27** = 0xb70000005b , **AARCH64_V28** = 0xb70000005c , **AARCH64_V29** = 0xb70000005d ,
AARCH64_V30 = 0xb70000005e , **AARCH64_V31** = 0xb70000005f }

9.7.1 Detailed Description

Constants for the register.canonicalRnScheme "ElfDwarf" for architecture Arm.

Date

9.8 IrisElfDwarfArm.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisElfDwarfArm_h
9 #define ARM_INCLUDE_IrisElfDwarfArm_h
10
11 #include "iris/detail/IrisInterface.h" // uint64_t
12 #include "iris/detail/IrisCommon.h"    // namespace iris
13
14 NAMESPACE_IRIS_START
15
16 namespace ElfDwarf
17 {
18
19 enum ElfDwarfArm: uint64_t
20 {
21 //  Constant       canonicalRn      Register     Architecture ELF-Arch DwarfReg
22 //  ================================================================================
23     ARM_R0       = 0x2800000000, // R0           EM_ARM              40        0
24     ARM_R1       = 0x2800000001, // R1           EM_ARM              40        1
25     ARM_R2       = 0x2800000002, // R2           EM_ARM              40        2
26     ARM_R3       = 0x2800000003, // R3           EM_ARM              40        3
27     ARM_R4       = 0x2800000004, // R4           EM_ARM              40        4
28     ARM_R5       = 0x2800000005, // R5           EM_ARM              40        5
29     ARM_R6       = 0x2800000006, // R6           EM_ARM              40        6
30     ARM_R7       = 0x2800000007, // R7           EM_ARM              40        7
31     ARM_R8       = 0x2800000008, // R8           EM_ARM              40        8
32     ARM_R9       = 0x2800000009, // R9           EM_ARM              40        9
33     ARM_R10      = 0x280000000a, // R10          EM_ARM              40       10
34     ARM_R11      = 0x280000000b, // R11          EM_ARM              40       11
35     ARM_R12      = 0x280000000c, // R12          EM_ARM              40       12
36     ARM_R13      = 0x280000000d, // R13          EM_ARM              40       13
37     ARM_R14      = 0x280000000e, // R14          EM_ARM              40       14
38     ARM_R15      = 0x280000000f, // R15          EM_ARM              40       15
39     ARM_SPSR     = 0x2800000080, // SPSR         EM_ARM              40      128
40     ARM_SPSR_fiq = 0x2800000081, // SPSR_fiq     EM_ARM              40      129
41     ARM_SPSR_irq = 0x2800000082, // SPSR_irq     EM_ARM              40      130
42     ARM_SPSR_abt = 0x2800000083, // SPSR_abt     EM_ARM              40      131
43     ARM_SPSR_und = 0x2800000084, // SPSR_und     EM_ARM              40      132
44     ARM_SPSR_svc = 0x2800000085, // SPSR_svc     EM_ARM              40      133
45     ARM_R8_fiq   = 0x2800000097, // R8_fiq       EM_ARM              40      151
46     ARM_R9_fiq   = 0x2800000098, // R9_fiq       EM_ARM              40      152
47     ARM_R10_fiq  = 0x2800000099, // R10_fiq      EM_ARM              40      153
48     ARM_R11_fiq  = 0x280000009a, // R11_fiq      EM_ARM              40      154
49     ARM_R12_fiq  = 0x280000009b, // R12_fiq      EM_ARM              40      155
50     ARM_R13_fiq  = 0x280000009c, // R13_fiq      EM_ARM              40      156
51     ARM_R14_fiq  = 0x280000009d, // R14_fiq      EM_ARM              40      157
52     ARM_R13_irq  = 0x280000009e, // R13_irq      EM_ARM              40      158
53     ARM_R14_irq  = 0x280000009f, // R14_irq      EM_ARM              40      159
54     ARM_R13_abt  = 0x28000000a0, // R13_abt      EM_ARM              40      160
55     ARM_R14_abt  = 0x28000000a1, // R14_abt      EM_ARM              40      161
56     ARM_R13_und  = 0x28000000a2, // R13_und      EM_ARM              40      162
57     ARM_R14_und  = 0x28000000a3, // R14_und      EM_ARM              40      163
58     ARM_R13_svc  = 0x28000000a4, // R13_svc      EM_ARM              40      164
59     ARM_R14_svc  = 0x28000000a5, // R14_svc      EM_ARM              40      165
60     ARM_D0       = 0x2800000100, // D0           EM_ARM              40      256
61     ARM_D1       = 0x2800000101, // D1           EM_ARM              40      257
62     ARM_D2       = 0x2800000102, // D2           EM_ARM              40      258
63     ARM_D3       = 0x2800000103, // D3           EM_ARM              40      259
64     ARM_D4       = 0x2800000104, // D4           EM_ARM              40      260
65     ARM_D5       = 0x2800000105, // D5           EM_ARM              40      261
66     ARM_D6       = 0x2800000106, // D6           EM_ARM              40      262
67     ARM_D7       = 0x2800000107, // D7           EM_ARM              40      263
68     ARM_D8       = 0x2800000108, // D8           EM_ARM              40      264
69     ARM_D9       = 0x2800000109, // D9           EM_ARM              40      265
70     ARM_D10      = 0x280000010a, // D10          EM_ARM              40      266
71     ARM_D11      = 0x280000010b, // D11          EM_ARM              40      267
72     ARM_D12      = 0x280000010c, // D12          EM_ARM              40      268
73     ARM_D13      = 0x280000010d, // D13          EM_ARM              40      269
74     ARM_D14      = 0x280000010e, // D14          EM_ARM              40      270
75     ARM_D15      = 0x280000010f, // D15          EM_ARM              40      271
76     ARM_D16      = 0x2800000110, // D16          EM_ARM              40      272
77     ARM_D17      = 0x2800000111, // D17          EM_ARM              40      273
78     ARM_D18      = 0x2800000112, // D18          EM_ARM              40      274
79     ARM_D19      = 0x2800000113, // D19          EM_ARM              40      275
80     ARM_D20      = 0x2800000114, // D20          EM_ARM              40      276
81     ARM_D21      = 0x2800000115, // D21          EM_ARM              40      277
82     ARM_D22      = 0x2800000116, // D22          EM_ARM              40      278
83     ARM_D23      = 0x2800000117, // D23          EM_ARM              40      279
84     ARM_D24      = 0x2800000118, // D24          EM_ARM              40      280
85     ARM_D25      = 0x2800000119, // D25          EM_ARM              40      281
86     ARM_D26      = 0x280000011a, // D26          EM_ARM              40      282
87     ARM_D27      = 0x280000011b, // D27          EM_ARM              40      283
88     ARM_D28      = 0x280000011c, // D28          EM_ARM              40      284
89     ARM_D29      = 0x280000011d, // D29          EM_ARM              40      285
90     ARM_D30      = 0x280000011e, // D30          EM_ARM              40      286
91     ARM_D31      = 0x280000011f, // D31          EM_ARM              40      287
92     AARCH64_X0   = 0xb700000000, // X0           EM_AARCH64         183        0
93     AARCH64_X1   = 0xb700000001, // X1           EM_AARCH64         183        1
```

```
94      AARCH64_X2   = 0xb700000002, // X2          EM_AARCH64        183        2
95      AARCH64_X3   = 0xb700000003, // X3          EM_AARCH64        183        3
96      AARCH64_X4   = 0xb700000004, // X4          EM_AARCH64        183        4
97      AARCH64_X5   = 0xb700000005, // X5          EM_AARCH64        183        5
98      AARCH64_X6   = 0xb700000006, // X6          EM_AARCH64        183        6
99      AARCH64_X7   = 0xb700000007, // X7          EM_AARCH64        183        7
100     AARCH64_X8   = 0xb700000008, // X8          EM_AARCH64        183        8
101     AARCH64_X9   = 0xb700000009, // X9          EM_AARCH64        183        9
102     AARCH64_X10  = 0xb70000000a, // X10         EM_AARCH64        183       10
103     AARCH64_X11  = 0xb70000000b, // X11         EM_AARCH64        183       11
104     AARCH64_X12  = 0xb70000000c, // X12         EM_AARCH64        183       12
105     AARCH64_X13  = 0xb70000000d, // X13         EM_AARCH64        183       13
106     AARCH64_X14  = 0xb70000000e, // X14         EM_AARCH64        183       14
107     AARCH64_X15  = 0xb70000000f, // X15         EM_AARCH64        183       15
108     AARCH64_X16  = 0xb700000010, // X16         EM_AARCH64        183       16
109     AARCH64_X17  = 0xb700000011, // X17         EM_AARCH64        183       17
110     AARCH64_X18  = 0xb700000012, // X18         EM_AARCH64        183       18
111     AARCH64_X19  = 0xb700000013, // X19         EM_AARCH64        183       19
112     AARCH64_X20  = 0xb700000014, // X20         EM_AARCH64        183       20
113     AARCH64_X21  = 0xb700000015, // X21         EM_AARCH64        183       21
114     AARCH64_X22  = 0xb700000016, // X22         EM_AARCH64        183       22
115     AARCH64_X23  = 0xb700000017, // X23         EM_AARCH64        183       23
116     AARCH64_X24  = 0xb700000018, // X24         EM_AARCH64        183       24
117     AARCH64_X25  = 0xb700000019, // X25         EM_AARCH64        183       25
118     AARCH64_X26  = 0xb70000001a, // X26         EM_AARCH64        183       26
119     AARCH64_X27  = 0xb70000001b, // X27         EM_AARCH64        183       27
120     AARCH64_X28  = 0xb70000001c, // X28         EM_AARCH64        183       28
121     AARCH64_X29  = 0xb70000001d, // X29         EM_AARCH64        183       29
122     AARCH64_X30  = 0xb70000001e, // X30         EM_AARCH64        183       30
123     AARCH64_SP   = 0xb70000001f, // SP          EM_AARCH64        183       31
124     AARCH64_ELR  = 0xb700000021, // ELR         EM_AARCH64        183       33
125     AARCH64_V0   = 0xb700000040, // V0          EM_AARCH64        183       64
126     AARCH64_V1   = 0xb700000041, // V1          EM_AARCH64        183       65
127     AARCH64_V2   = 0xb700000042, // V2          EM_AARCH64        183       66
128     AARCH64_V3   = 0xb700000043, // V3          EM_AARCH64        183       67
129     AARCH64_V4   = 0xb700000044, // V4          EM_AARCH64        183       68
130     AARCH64_V5   = 0xb700000045, // V5          EM_AARCH64        183       69
131     AARCH64_V6   = 0xb700000046, // V6          EM_AARCH64        183       70
132     AARCH64_V7   = 0xb700000047, // V7          EM_AARCH64        183       71
133     AARCH64_V8   = 0xb700000048, // V8          EM_AARCH64        183       72
134     AARCH64_V9   = 0xb700000049, // V9          EM_AARCH64        183       73
135     AARCH64_V10  = 0xb70000004a, // V10         EM_AARCH64        183       74
136     AARCH64_V11  = 0xb70000004b, // V11         EM_AARCH64        183       75
137     AARCH64_V12  = 0xb70000004c, // V12         EM_AARCH64        183       76
138     AARCH64_V13  = 0xb70000004d, // V13         EM_AARCH64        183       77
139     AARCH64_V14  = 0xb70000004e, // V14         EM_AARCH64        183       78
140     AARCH64_V15  = 0xb70000004f, // V15         EM_AARCH64        183       79
141     AARCH64_V16  = 0xb700000050, // V16         EM_AARCH64        183       80
142     AARCH64_V17  = 0xb700000051, // V17         EM_AARCH64        183       81
143     AARCH64_V18  = 0xb700000052, // V18         EM_AARCH64        183       82
144     AARCH64_V19  = 0xb700000053, // V19         EM_AARCH64        183       83
145     AARCH64_V20  = 0xb700000054, // V20         EM_AARCH64        183       84
146     AARCH64_V21  = 0xb700000055, // V21         EM_AARCH64        183       85
147     AARCH64_V22  = 0xb700000056, // V22         EM_AARCH64        183       86
148     AARCH64_V23  = 0xb700000057, // V23         EM_AARCH64        183       87
149     AARCH64_V24  = 0xb700000058, // V24         EM_AARCH64        183       88
150     AARCH64_V25  = 0xb700000059, // V25         EM_AARCH64        183       89
151     AARCH64_V26  = 0xb70000005a, // V26         EM_AARCH64        183       90
152     AARCH64_V27  = 0xb70000005b, // V27         EM_AARCH64        183       91
153     AARCH64_V28  = 0xb70000005c, // V28         EM_AARCH64        183       92
154     AARCH64_V29  = 0xb70000005d, // V29         EM_AARCH64        183       93
155     AARCH64_V30  = 0xb70000005e, // V30         EM_AARCH64        183       94
156     AARCH64_V31  = 0xb70000005f, // V31         EM_AARCH64        183       95
157 }; // enum ElfDwarfArm
158
159 } // namespace ElfDwarf
160
161 NAMESPACE_IRIS_END
162
163 #endif // ARM_INCLUDE_IrisElfDwarfArm_h
164
```

9.9 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.

9.9.1 Detailed Description

A utility class for emitting Iris events.

Copyright

9.10 IrisEventEmitter.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisEventEmitter_h
9 #define ARM_INCLUDE_IrisEventEmitter_h
10
11 #include "iris/detail/IrisEventEmitterBase.h"
12
13 NAMESPACE_IRIS_START
14
35 template <typename... ARGS>
36 class IrisEventEmitter : public IrisEventEmitterBase
37 {
38 public:
40     IrisEventEmitter()
41         : IrisEventEmitterBase(sizeof...(ARGS))
42     {
43     }
44
52     void operator()(ARGS... args)
53     {
54         emitEvent(args...);
55     }
56 };
57
58 NAMESPACE_IRIS_END
59
60 #endif // ARM_INCLUDE_IrisEventEmitter_h
```

9.11 IrisGlobalInstance.h File Reference

Central instance which lives in the simulation engine and distributes all Iris messages.

```
#include "iris/IrisInstance.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisFunctionDecoder.h"
#include "iris/detail/IrisInterface.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisReceivedRequest.h"
#include "iris/impl/IrisChannelRegistry.h"
#include "iris/impl/IrisPlugin.h"
#include "iris/impl/IrisServiceClient.h"
#include "iris/impl/IrisTcpServer.h"
#include <atomic>
#include <list>
#include <map>
#include <memory>
#include <mutex>
#include <string>
#include <thread>
#include <unordered_map>
#include <vector>
```

Classes

- class iris::IrisGlobalInstance

9.11.1 Detailed Description

Central instance which lives in the simulation engine and distributes all Iris messages.

Date

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.

9.12 IrisGlobalInstance.h

Go to the documentation of this file.

```
1
10 #ifndef ARM_INCLUDE_IrisGlobalInstance_h
11 #define ARM_INCLUDE_IrisGlobalInstance_h
12
13 #include "iris/IrisInstance.h"
14 #include "iris/detail/IrisCommon.h"
15 #include "iris/detail/IrisFunctionDecoder.h"
16 #include "iris/detail/IrisInterface.h"
17 #include "iris/detail/IrisLogger.h"
18 #include "iris/detail/IrisObjects.h"
19 #include "iris/detail/IrisReceivedRequest.h"
20
21 #include "iris/impl/IrisChannelRegistry.h"
22 #include "iris/impl/IrisPlugin.h"
23 #include "iris/impl/IrisServiceClient.h"
24 #include "iris/impl/IrisTcpServer.h"
25
26 #include <atomic>
27 #include <list>
28 #include <map>
29 #include <memory>
30 #include <mutex>
31 #include <string>
32 #include <thread>
33 #include <unordered_map>
34 #include <vector>
35
36 NAMESPACE_IRIS_START
37
38 class IrisGlobalInstance : public IrisInterface
39     , public IrisConnectionInterface
40 {
41 public:
43     IrisGlobalInstance();
44
46     ~IrisGlobalInstance();
47
50     uint64_t registerChannel(IrisC_CommunicationChannel* channel, const std::string& connection_info =
        "");
51
53     void unregisterChannel(uint64_t channelId);
54
57     IrisInstance& getIrisInstance() { return irisInstance; }
58
59 public: // IrisConnectionInterface
64     virtual uint64_t registerIrisInterfaceChannel(IrisInterface* iris_interface) override;
65
68     virtual void unregisterIrisInterfaceChannel(uint64_t channelId) override
69     {
70         unregisterChannel(channelId);
71     }
72
73     virtual IrisErrorCode processAsyncMessages(bool waitForAMessage) override
74     {
75         return irisProxyInterface.load()->processAsyncMessagesInProxy(waitForAMessage);
76     }
77
79     virtual IrisInterface* getIrisInterface() override
80     {
81         return this;
82     }
83
```

```
85      virtual void setIrisProxyInterface(IrisProxyInterface* irisProxyInterface_) override
86      {
87          irisProxyInterface = irisProxyInterface_ ? irisProxyInterface_ : &defaultIrisProxyInterface;
88      }
89
90  public:
91      // IrisInterface implementation.
92
94      virtual void  irisHandleMessage(const uint64_t* message) override;
95
96      // Set log level for logging messages.
97      void setLogLevel(unsigned level);
98
99  private:
100     // --- Functions implemented locally in the global instance (registered in the functionDecoder). ---
101
103     void impl_instanceRegistry_registerInstance(IrisReceivedRequest& request);
104
106     void impl_instanceRegistry_unregisterInstance(IrisReceivedRequest& request);
107
109     void impl_instanceRegistry_getList(IrisReceivedRequest& request);
110
112     void impl_instanceRegistry_getInstanceInfoByInstId(IrisReceivedRequest& request);
113
115     void impl_instanceRegistry_getInstanceInfoByName(IrisReceivedRequest& request);
116
118     void impl_perInstanceExecution_setStateAll(IrisReceivedRequest& request);
119
121     void impl_perInstanceExecution_getStateAll(IrisReceivedRequest& request);
122
124     void impl_tcpServer_start(IrisReceivedRequest& request);
125
127     void impl_tcpServer_stop(IrisReceivedRequest& request);
128
130     void impl_tcpServer_getPort(IrisReceivedRequest& request);
131
133     void impl_plugin_load(IrisReceivedRequest& request);
134
136     void impl_service_connect(IrisReceivedRequest& request);
137
139     void impl_service_disconnect(IrisReceivedRequest& request);
140
141     // --- Private helpers ---
142
144     struct InstanceRegistryEntry
145     {
146         std::string    instName;
147         uint64_t       channelId{IRIS_UINT64_MAX}; // If this is IRIS_UINT64_MAX this means this entry
    is unused.
148         IrisInterface* iris_interface{nullptr};
149         std::string     connection_info;
150
151         bool empty() const
152         {
153             return channelId == IRIS_UINT64_MAX;
154         }
155
157         void clear()
158         {
159             instName        = "";
160             channelId       = IRIS_UINT64_MAX;
161             iris_interface  = nullptr;
162             connection_info = "";
163
164             assert(empty());
165         }
166     };
167
169     InstanceId registerInstance(std::string&   instName,
170                                 uint64_t       channel_id,
171                                 bool           uniquify,
172                                 IrisInterface* iris_interface);
173
175     void unregisterInstanceAndGenerateEvent(InstanceRegistryEntry*  entry,
176                                             InstanceId              aInstId,
177                                             uint64_t                time,
178                                             std::list<IrisRequest>& deferred_event_requests);
179
181     const InstanceRegistryEntry* findInstanceRegistryEntry(InstanceId instId) const
182     {
183         if (instId >= InstanceId(instanceRegistry.size()))
184             return nullptr;
185
186         if (instanceRegistry[instId].empty())
187             return nullptr;
188
189         return &instanceRegistry[instId];
```

```
190     }
191
195     InstanceId addNewInstance(const std::string& instName,
196                               uint64_t           channelId,
197                               IrisInterface*     iris_interface);
198
199     // Stop the Iris Server (if running)
200     void stopServer();
201
202     // stop the Iris Client (if running)
203     void stopClient();
204
205     void loadPlugin(const std::string& plugin_path);
206
208     IrisErrorCode createEventStream(EventStream*&, const EventSourceInfo&, const
        std::vector<std::string>&);
209
211     uint64_t getTimeForEvents();
212
215     std::string getInstName(InstanceId instId) const;
216
217
218     // --- Private data ---
219
224     class Instance : public IrisInstance
225     {
226     public:
227         Instance()
228             : IrisInstance()
229         {
230             thisInstanceInfo.instName = "framework.GlobalInstance";
231             thisInstanceInfo.instId   = IrisInstIdGlobalInstance;
232             setProperty("instName", getInstanceName());
233             setProperty("instId", getInstId());
234             // NOTE: This instance does not think it is registered.
235             //       This means it won't unregister itself when it is destroyed but that doesn't matter.
236             //       We will be cleaning up all that state anyway.
237         }
238
239         IrisInstanceEvent event_handler;
240     } irisInstance;
241
243     IrisEventRegistry instance_registry_changed_event_registry;
244
246     IrisEventRegistry shutdown_enter_event_registry;
247
249     IrisEventRegistry shutdown_leave_event_registry;
250
253     std::vector<InstanceRegistryEntry> instanceRegistry;
254
255     //
256     std::mutex instance_registry_mutex;
257
260     std::vector<InstanceId> freeInstIds;
261
263     typedef std::map<std::string, uint64_t> InstanceRegistryNameToIdMap;
264
266     InstanceRegistryNameToIdMap instanceRegistryNameToId;
267
275     unsigned logMessages;
276
278     IrisLogger log;
279
280     // TCP server. This won't start listening until startServer() is called.
281     impl::IrisTcpServer* tcp_server;
282
284     impl::IrisServiceClient* service_client;
285
286     // Create and manage communication channels
287     impl::IrisChannelRegistry channel_registry;
288
289     std::unordered_map<uint64_t, std::string> channel_connection_info;
290     std::mutex                                channel_connection_info_mutex;
291
292     // --- Load and manage plugins ---
293     using Plugin = impl::IrisPlugin<IrisGlobalInstance>;
294     std::unordered_map<std::string, std::unique_ptr<Plugin» plugins;
295
296     std::mutex plugins_mutex;
297
298     std::mutex log_mutex;
299
304     class DefaultIrisProxyInterface : public IrisProxyInterface
305     {
306     public:
307         virtual void          irisHandleMessageInProxy(IrisInterface* irisInterface, InstanceId instId,
        const uint64_t* message) override;
```

```
308         virtual IrisErrorCode processAsyncMessagesInProxy(bool waitForAMessage) override;
309     } defaultIrisProxyInterface;
310
318     std::atomic<IrisProxyInterface*> irisProxyInterface{&defaultIrisProxyInterface};
319 };
320
321 NAMESPACE_IRIS_END
322
323 #endif // #ifndef ARM_INCLUDE_IrisGlobalInstance_h
```

9.13 IrisInstance.h File Reference

Boilerplate code for an Iris instance, including clients and components.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisCppAdapter.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisFunctionDecoder.h"
#include "iris/detail/IrisObjects.h"
#include "iris/IrisInstanceEvent.h"
#include <cassert>
#include <mutex>
#include "iris/IrisInstanceBuilder.h"
```

Classes

- class iris::IrisInstance

Macros

- #define **irisRegisterEventCallback**(instancePtr, instanceType, functionName, description) registerEvent↩ Callback<instanceType, &instanceType::impl_##functionName>(instancePtr, #functionName, description, #instanceType)

  Register an event callback function using an EventCallbackDelegate.

- #define **irisRegisterFunction**(instancePtr, instanceType, functionName, functionInfoJson) register↩ Function(instancePtr, #functionName, &instanceType::impl_##functionName, functionInfoJson, #instance↩ Type)

  Register an Iris function implementation. The function can be implemented in this class or in any other class. The helper macro is here to avoid repeating the function name. The 'impl_' prefix limits namespace pollution.

Typedefs

- typedef IrisDelegate< uint64_t, const IrisValueMap &, uint64_t, uint64_t, bool, std::string & > iris::EventCallbackDelegate

  Event callback delegate.

9.13.1 Detailed Description

Boilerplate code for an Iris instance, including clients and components.

Copyright

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.

9.13.2 Typedef Documentation

9.13.2.1 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");
```

9.14 IrisInstance.h

Go to the documentation of this file.

```
1
19 #ifndef ARM_INCLUDE_IrisInstance_h
20 #define ARM_INCLUDE_IrisInstance_h
21
22 #include "iris/detail/IrisCommon.h"
23 #include "iris/detail/IrisCppAdapter.h"
24 #include "iris/detail/IrisDelegate.h"
25 #include "iris/detail/IrisFunctionDecoder.h"
26 #include "iris/detail/IrisObjects.h"
27 #include "iris/IrisInstanceEvent.h"
28
29 #include <cassert>
30 #include <mutex>
31
32 NAMESPACE_IRIS_START
33
62 typedef IrisDelegate<uint64_t, const IrisValueMap&, uint64_t, uint64_t, bool, std::string&>
      EventCallbackDelegate;
63
64 class IrisInstantiationContext;
65 class IrisInstanceBuilder;
66
67 class IrisInstance
68 {
69 public:
70 // --- Construction and destruction. ---
71
78 #define irisRegisterFunction(instancePtr, instanceType, functionName, functionInfoJson)
      registerFunction(instancePtr, #functionName, &instanceType::impl_##functionName, functionInfoJson,
      #instanceType)
79
81 #define irisRegisterEventCallback(instancePtr, instanceType, functionName, description)
      registerEventCallback<instanceType, &instanceType::impl_##functionName>(instancePtr, #functionName,
      description, #instanceType)
89     static const uint64_t UNIQUIFY = (1 << 0);
90
92     static const uint64_t THROW_ON_ERROR = (1 << 1);
93
95     static const uint64_t DEFAULT_FLAGS = THROW_ON_ERROR;
96
```

```
116     IrisInstance(IrisConnectionInterface* connection_interface = nullptr,
117                  const std::string&       instName             = std::string(),
118                  uint64_t                 flags                = DEFAULT_FLAGS);
119 
126     IrisInstance(IrisInstantiationContext* context);
127 
129     ~IrisInstance();
130 
138     void setConnectionInterface(IrisConnectionInterface* connection_interface);
139 
144     void processAsyncRequests();
145 
151     IrisInterface* getRemoteIrisInterface()
152     {
153         return remoteIrisInterface;
154     }
155 
166     void setThrowOnError(bool throw_on_error)
167     {
168         default_cppAdapter = throw_on_error ? &throw_cppAdapter : &nothrow_cppAdapter;
169     }
170 
184     IrisErrorCode registerInstance(const std::string& instName, uint64_t flags = DEFAULT_FLAGS);
185 
191     IrisErrorCode unregisterInstance();
192 
211     template <class T>
212     void setProperty(const std::string& propertyName, const T& propertyValue)
213     {
214         assert(!instance_getProperties_called);
215         propertyMap[propertyName].set(propertyValue);
216     }
217 
224     const PropertyMap& getPropertyMap() const
225     {
226         return propertyMap;
227     }
228 
229     // --- Interface for components. Provide functionality to clients. ---
230 
243     template <class T>
244     void registerFunction(T* instance, const std::string& name, void
    (T::*memberFunctionPtr)(IrisReceivedRequest&), const std::string& functionInfoJson, const
    std::string& instanceTypeStr)
245     {
246         functionDecoder.registerFunction(instance, name, memberFunctionPtr, functionInfoJson,
    instanceTypeStr);
247     }
248 
252     void unregisterFunction(const std::string& name)
253     {
254         functionDecoder.unregisterFunction(name);
255     }
256 
268     template <class T>
269     void registerEventCallback(T* instance, const std::string& name, const std::string& description,
270                                void (T::*memberFunctionPtr)(IrisReceivedRequest&),
271                                const std::string& instanceTypeStr)
272     {
273         std::string funcInfoJson = "{description:'" + description +
274             "',"
275             "args:{"
276             "  instId:{type:'NumberU64', description:'Target instance id.'},"
277             "  esId:{type:'NumberU64', description:'Event stream id.'},"
278             "  fields:{type:'Object', description:'Object which contains the names and values of event
    source fields.'},"
279             "  time:{type:'NumberU64', description:'Simulation time timestamp of the event.'},"
280             "  sInstId:{type:'NumberU64', description:'Source instId: Instance which generated and sent
    this event.'},"
281             "  syncEc:{type:'Boolean', description:'Synchronous callback behaviour.', optional:true},"
282             "},"
283             "retval:{type:'Null'}}";
284         functionDecoder.registerFunction(instance, name, memberFunctionPtr, funcInfoJson,
    instanceTypeStr);
285     }
286 
295     void registerEventCallback(EventCallbackDelegate delegate, const std::string& name,
296                                const std::string& description, const std::string& dlgInstanceTypeStr)
297     {
298         eventCallbacks[name] = ECD(delegate);
299         registerEventCallback(this, name, description, &IrisInstance::impl_eventCallback,
    dlgInstanceTypeStr);
300     }
301 
310     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, const AttributeValueMap&, uint64_t,
    uint64_t, bool, std::string&)>
311     void registerEventCallback(T* instance, const std::string& name, const std::string& description,
```

```
312                             const std::string& dlgInstanceTypeStr)
313     {
314         registerEventCallback(EventCallbackDelegate::make<T, METHOD>(instance),
315                               name, description, dlgInstanceTypeStr);
316     }
317
321     void unregisterEventCallback(const std::string& name);
322
333     using EventCallbackFunction = std::function<IrisErrorCode(EventStreamId, const IrisValueMap&,
    uint64_t, InstanceId, bool, std::string&)>;
334
338     void setCallback_IRIS_SIMULATION_TIME_EVENT(EventCallbackFunction f);
339
343     void setCallback_IRIS_SHUTDOWN_LEAVE(EventCallbackFunction f);
344
348     void addCallback_IRIS_INSTANCE_REGISTRY_CHANGED(EventCallbackFunction f);
349
357     void sendResponse(const uint64_t* response)
358     {
359         remoteIrisInterface->irisHandleMessage(response);
360     }
361
362     // --- Interface for clients. Access to other components. ---
363
369     IrisCppAdapter& irisCall() { return *default_cppAdapter; }
370
378     IrisCppAdapter& irisCallNoThrow() { return nothrow_cppAdapter; }
379
394     IrisCppAdapter& irisCallThrow() { return throw_cppAdapter; }
395
407     bool sendRequest(IrisRequest& req)
408     {
409         return irisCall().callAndPerhapsWaitForResponse(req);
410     }
411
412     // --- Misc functionality. ---
413
420     IrisInterface* getLocalIrisInterface() { return functionDecoder.getIrisInterface(); }
421
428     InstanceId getInstId() const { return thisInstanceInfo.instId; }
429
435     void setInstId(InstanceId instId) { thisInstanceInfo.instId = instId;
    cppAdapter_request_manager.setInstId(instId); }
436
446     const std::string& getInstanceName() const { return thisInstanceInfo.instName; }
447
450     bool isRegistered() const { return cppAdapter_request_manager.isRegistered(); }
451
458     IrisInstanceBuilder* getBuilder();
459
460     bool isAdapterInitialized() const { return is_adapter_initialized; }
461
462     void setAdapterInitialized() { is_adapter_initialized = true; }
463
467     void setEventHandler(IrisInstanceEvent* handler);
468
472     void notifyStateChanged();
473
484     template<class T>
485     void publishCppInterface(const std::string& interfaceName, T *pointer, const std::string&
    jsonDescription)
486     {
487         // Ignore null pointers: instance_getCppInterface...() promises to always return non-null
    pointers.
488         // (If there is no interface, do not publish it.)
489         if (pointer == nullptr)
490             return;
491
492         std::string functionInfoJson =
493             "{"
494             "    \"description\": \"" + jsonDescription + "\n"
495             "If this function is present it always returns a non-null pointer.\n"
496             "The caller of this function must make sure that the caller and callee use the same C++
    interface class layout and run in the same process. "
497             "This effectively means that they both must be compiled using the same compiler using the
    same header files. "
498             "The returned pointer is only meaningful if caller and callee run in the same process.\n"
499             "The meta-information provided alongside the returned pointer in CppInterfacePointer can
    (and should) be used to do minimal compatibility checking between caller and callee, see
    'CppInterfacePointer::isCompatibleWith()' in 'IrisObjects.h'.\","
500             "    \"args\": {"
501             "        \"instId\": {"
502             "            \"description\": \"Opaque number uniquely identifying the target instance.\","
503             "            \"type\": \"NumberU64\""
504             "        }"
505             "    },"
506             "    \"errors\": ["
```

```
507             "        \"E_unknown_instance_id\""
508             "    ],"
509             "    \"retval\": {"
510             "        \"description\": \"Pointer to the requested C++ interface (and associated
    meta-information) of this instance. Use 'CppInterfacePointer::isCompatibleWith()' to do a minimal
    compatibility check before using the pointer.\","
511             "        \"type\": \"CppInterfacePointer\""
512             "    }"
513             "}";
514         registerFunction(this, "instance_getCppInterface" + interfaceName,
    &IrisInstance::impl_instance_getCppInterface, functionInfoJson, "IrisInstance");
515         cppInterfaceRegistry[interfaceName].set(pointer);
516     }
517
526     void unpublishCppInterface(const std::string& interfaceName)
527     {
528         unregisterFunction("instance_getCppInterface" + interfaceName);
529         cppInterfaceRegistry.erase(interfaceName);
530     }
531
532     // --- Blocking simulation time functions ---
533
541     void simulationTimeRun();
542
548     void simulationTimeStop();
549
555     void simulationTimeRunUntilStop();
556
565     bool simulationTimeIsRunning();
566
579     void simulationTimeDisableEvents();
580
587     void setPendingSyncStepResponse(RequestId requestId, EventBufferId evBufId);
588
596     bool isValidEvBufId(EventBufferId evBufId) const;
597
639      void findEventSourcesAndFields(const std::string& spec, std::vector<EventStreamInfo>&
    eventStreamInfosOut, InstanceId defaultInstId = IRIS_UINT64_MAX);
640
648     std::vector<InstanceInfo> findInstanceInfos(const std::string& instancePathFilter = "all");
649
656     std::vector<EventSourceInfo> findEventSources(const std::string& instancePathFilter = "all");
657
665     const InstanceInfo& getInstanceInfo(InstanceId instId);
666
683     InstanceInfo getInstanceInfo(const std::string& instancePathFilter);
684
695     const std::vector<InstanceInfo>& getInstanceList();
696
706     std::string getInstanceName(InstanceId instId);
707
717     InstanceId getInstanceId(const std::string& instName);
718
728     ResourceId getResourceId(InstanceId instId, const std::string& resourceSpec);
729
753     uint64_t resourceRead(InstanceId instId, const std::string& resourceSpec);
754
762     uint64_t resourceReadCrn(InstanceId instId, uint64_t canonicalRegisterNumber)
763     {
764         return resourceRead(instId, "crn:" + std::to_string(canonicalRegisterNumber));
765     }
766
776     std::string resourceReadStr(InstanceId instId, const std::string& resourceSpec);
777
785     void resourceWrite(InstanceId instId, const std::string& resourceSpec, uint64_t value);
786
792     void resourceWriteCrn(InstanceId instId, uint64_t canonicalRegisterNumber, uint64_t value)
793     {
794         resourceWrite(instId, "crn:" + std::to_string(canonicalRegisterNumber), value);
795     }
796
805     void resourceWriteStr(InstanceId instId, const std::string& resourceSpec, const std::string& value);
806
810     const std::vector<iris::ResourceGroupInfo>& getResourceGroups(InstanceId instId);
811
815     const ResourceInfo& getResourceInfo(InstanceId instId, ResourceId resourceId);
816
820     const ResourceInfo& getResourceInfo(InstanceId instId, const std::string& resourceSpec);
821
825     const std::vector<iris::ResourceInfo>& getResourceInfos(InstanceId instId);
826
830     void clearCachedMetaInfo();
831
832 private:
833     void init(IrisConnectionInterface* connection_interface_ = nullptr,
834               const std::string&       instName              = std::string(),
835               uint64_t                 flags                 = DEFAULT_FLAGS);
```

```
836
838     struct InstanceMetaInfo
839     {
842         std::map<std::string,ResourceId> resourceSpecToResourceIdAll;
843
847         std::map<std::string,ResourceId> resourceSpecToResourceIdUsed;
848
850         std::vector<iris::ResourceGroupInfo> groupInfos;
851
853         std::vector<iris::ResourceInfo> resourceInfos;
854
856         std::map<ResourceId,uint64_t> resourceIdToIndex;
857     };
858
862     InstanceMetaInfo& getInstanceMetaInfo(InstanceId instId);
863
867     IrisInstance::InstanceMetaInfo& getResourceMetaInfo(InstanceId instId);
868
882     void expandWildcardsInEventStreamInfos(std::vector<EventStreamInfo>& eventStreamInfosInOut,
      InstanceId defaultInstId = IRIS_UINT64_MAX, const std::string& instancePathFilter = "all");
883
885     void enableSimulationTimeEvents();
886
888     void enableShutdownLeaveEvents();
889
891     void enableInstanceRegistryChangedEvent();
892
894     void simulationTimeWaitForRunning();
895
897     void simulationTimeWaitForStop();
898
900     void simulationTimeClearGotRunning();
901
905     std::string lookupInstanceNameLocal(InstanceId instId);
906
907     // --- Iris function implementations ---
908     void impl_instance_getProperties(IrisReceivedRequest& request);
909
910     void impl_instance_ping(IrisReceivedRequest& request);
911
912     void impl_instance_ping2(IrisReceivedRequest& request);
913
914     void impl_instance_getCppInterface(IrisReceivedRequest& request);
915
916     void impl_eventCallback(IrisReceivedRequest& request);
917
919     IrisErrorCode createEventStream(EventStream*& event_stream_out, const EventSourceInfo& info,
920                                     const std::vector<std::string>& fields);
921
923     IrisErrorCode impl_ec_IrisInstance_IRIS_SIMULATION_TIME_EVENT(EventStreamId esId, const
      IrisValueMap& fields, uint64_t time,
924                                                                   InstanceId sInstId, bool syncEc,
      std::string& errorMessageOut);
925
927     IrisErrorCode impl_ec_IrisInstance_IRIS_SHUTDOWN_LEAVE(EventStreamId esId, const IrisValueMap&
      fields, uint64_t time,
928                                                            InstanceId sInstId, bool syncEc, std::string&
      errorMessageOut);
929
931     IrisErrorCode impl_ec_IrisInstance_IRIS_INSTANCE_REGISTRY_CHANGED(EventStreamId esId, const
      IrisValueMap& fields, uint64_t time,
932                                                                       InstanceId sInstId, bool syncEc,
      std::string& errorMessageOut);
933
934     // --- Iris specific data and state ---
935
937     IrisFunctionDecoder functionDecoder{log, this};
938
940     IrisCppAdapter::RequestManager cppAdapter_request_manager{log};
941
943     IrisCppAdapter throw_cppAdapter{&cppAdapter_request_manager, /*throw_on_error=*/true};
944
946     IrisCppAdapter nothrow_cppAdapter{&cppAdapter_request_manager, /*throw_on_error=*/false};
947
949     IrisCppAdapter* default_cppAdapter{&throw_cppAdapter};
950
954     IrisConnectionInterface* connection_interface{nullptr};
955
958     IrisInterface* remoteIrisInterface{nullptr};
959
960 protected:
962     InstanceInfo thisInstanceInfo{};
963
964 private:
966     bool instance_getProperties_called{false};
967
968     bool registered{false};
```

```
969
970     bool is_adapter_initialized{false};
971
972     uint64_t channelId{IRIS_UINT64_MAX};
973
975     IrisLogger log;
976
977     // --- Instance specific data and state ---
978
980     PropertyMap propertyMap{};
981
983     struct ECD
984     {
985         // Work around symbol length limits in Visual Studio (warning C4503)
986         EventCallbackDelegate dlg;
987         ECD() {}
988         ECD(EventCallbackDelegate dlg_)
989             : dlg(dlg_)
990         {
991         }
992     };
993     typedef std::map<std::string, ECD> EventCallbackMap;
994     EventCallbackMap                   eventCallbacks{};
995
997     IrisInstanceBuilder* builder{nullptr};
998
1000     IrisEventRegistry* state_changed_event_registry{nullptr};
1001
1003     IrisInstanceEvent *irisInstanceEvent{};
1004
1008     typedef std::map<std::string, CppInterfacePointer> CppInterfaceRegistryMap;
1009     CppInterfaceRegistryMap cppInterfaceRegistry{};
1010
1012     bool simulationTimeIsRunning_{};
1013
1015     bool simulationTimeGotRunningTrue{};
1016
1018     bool simulationTimeGotRunningFalse{};
1019
1021     std::mutex simulationTimeIsRunningMutex;
1022
1024     std::condition_variable simulationTimeIsRunningChanged;
1025
1027     EventStreamId simulationTimeEsId = IRIS_UINT64_MAX;
1028
1030     EventStreamId shutdownLeaveEsId = IRIS_UINT64_MAX;
1031
1033     EventStreamId instanceRegistryChangedEsId = IRIS_UINT64_MAX;
1034
1036     EventCallbackFunction simulationTimeCallbackFunction;
1037
1039     EventCallbackFunction shutdownLeaveCallbackFunction;
1040
1041     // List of callback functions for IRIS_INSTANCE_REGISTRY_CHANGED.
1042     std::vector<EventCallbackFunction> instanceRegistryChangedFunctions;
1043
1045     struct PendingSyncStepResponse
1046     {
1048         void set(RequestId requestId_, EventBufferId evBufId_)
1049         {
1050             requestId = requestId_;
1051             evBufId = evBufId_;
1052         }
1053
1055         bool isPending() const
1056         {
1057             return requestId != IRIS_UINT64_MAX;
1058         }
1059
1061         void clear()
1062         {
1063             requestId = IRIS_UINT64_MAX;
1064             evBufId = 0;
1065         }
1066
1069         RequestId requestId{IRIS_UINT64_MAX};
1070
1072         EventBufferId evBufId{};
1073     };
1074
1076     PendingSyncStepResponse pendingSyncStepResponse;
1077
1079     std::vector<InstanceInfo> instanceInfos;
1080
1083     std::vector<uint64_t> instIdToIndex;
1084
1086     std::map<InstanceId,InstanceMetaInfo> instIdToMetaInfo;
```

```
1087 };
1088
1089
1090 NAMESPACE_IRIS_END
1091
1092 #endif // #ifndef ARM_INCLUDE_IrisInstance_h
1093
1094 // Convenience #include.
1095 // (IrisInstanceBuilder needs the complete type of IrisInstance.)
1096 #include "iris/IrisInstanceBuilder.h"
1097
```

9.15 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.

9.15.1 Detailed Description

Breakpoint add-on to IrisInstance.

Copyright

Copyright (C) 2016-2020 Arm Limited. All rights reserved.

The IrisInstanceBreakpoint class:

- Implements all breakpoint-related Iris functions.
- Maintains and provides breakpoint information, for example type, address, and rscId.
- Converts between Iris breakpoint functions (breakpoint∗()) and various C++ access functions.

9.15.2 Typedef Documentation

9.15.2.1 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.

9.15.2.2 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.

9.16 IrisInstanceBreakpoint.h

Go to the documentation of this file.

```
1
12 #ifndef ARM_INCLUDE_IrisInstanceBreakpoint_h
13 #define ARM_INCLUDE_IrisInstanceBreakpoint_h
14
15 #include "iris/detail/IrisCommon.h"
16 #include "iris/detail/IrisDelegate.h"
17 #include "iris/detail/IrisLogger.h"
18 #include "iris/detail/IrisObjects.h"
19
20 #include <cstdio>
21
22 NAMESPACE_IRIS_START
23
24 class IrisInstance;
25 class IrisInstanceEvent;
26 class IrisEventRegistry;
27 class IrisReceivedRequest;
28
29 class EventStream;
30 struct EventSourceInfo;
31
45 typedef IrisDelegate<BreakpointInfo&> BreakpointSetDelegate;
46
57 typedef IrisDelegate<const BreakpointInfo&> BreakpointDeleteDelegate;
58
79 class IrisInstanceBreakpoint
80 {
81
82 public:
83     // --- Construction and destruction. ---
84     IrisInstanceBreakpoint(IrisInstance* irisInstance = nullptr);
85     ~IrisInstanceBreakpoint();
86
94     void attachTo(IrisInstance* irisInstance);
95
101     void setBreakpointSetDelegate(BreakpointSetDelegate delegate);
102
108     void setBreakpointDeleteDelegate(BreakpointDeleteDelegate delegate);
109
115     void setEventHandler(IrisInstanceEvent* handler);
116
128     void notifyBreakpointHit(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId);
129
147     void notifyBreakpointHitData(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId
        pcSpaceId,
148                                  uint64_t accessAddr, uint64_t accessSize,
149                                  const std::string& accessRw, const std::vector<uint64_t>& data);
150
166     void notifyBreakpointHitRegister(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId
        pcSpaceId,
167                                      const std::string& accessRw, const std::vector<uint64_t>& data);
168
176     const BreakpointInfo* getBreakpointInfo(BreakpointId bptId) const;
177
187     void addCondition(const std::string& name, const std::string& type, const std::string& description,
188                       const std::vector<std::string> bpt_types = std::vector<std::string>());
189
190 private:
191     void impl_breakpoint_set(IrisReceivedRequest& request);
192
193     void impl_breakpoint_delete(IrisReceivedRequest& request);
194
195     void impl_breakpoint_getList(IrisReceivedRequest& request);
196
197     void impl_breakpoint_getAdditionalConditions(IrisReceivedRequest& request);
198
199     bool validateInterceptionParameters(IrisReceivedRequest& request, const InterceptionParams&
        interceptionParams);
```

```
200
203     bool beginBreakpointHit(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId);
204
206     IrisErrorCode createEventStream(EventStream*&, const EventSourceInfo&, const
      std::vector<std::string>&);
207
209     IrisErrorCode deleteBreakpoint(BreakpointId bpt);
210
211     void register_ec_IRIS_INSTANCE_REGISTRY_CHANGED();
212     IrisErrorCode ec_IRIS_INSTANCE_REGISTRY_CHANGED(EventStreamId esId, const IrisValueMap& fields,
      uint64_t time,
213                                                     InstanceId sInstId, bool syncEc, std::string&
      errorMessageOut);
214
216
218     IrisInstance* irisInstance;
219
221     IrisEventRegistry* breakpoint_hit_registry;
222
225     std::vector<BreakpointInfo> bptInfos;
226
229     std::vector<uint64_t> freeBptIds;
230
232     std::map<uint64_t, BreakpointAction> bptActions;
233
235     std::vector<BreakpointConditionInfo> additional_conditions;
236
238     BreakpointSetDelegate bptSetDelegate;
239
241     BreakpointDeleteDelegate bptDeleteDelegate;
242
244     IrisLogger log;
245
250     bool instance_registry_changed_registered{};
251 };
252
253 NAMESPACE_IRIS_END
254
255 #endif // #ifndef ARM_INCLUDE_IrisInstanceBreakpoint_h
```

9.17 IrisInstanceBuilder.h File Reference

A high level interface to build up functionality on an IrisInstance.

```
#include "iris/IrisEventEmitter.h"
#include "iris/IrisInstance.h"
#include "iris/IrisInstanceBreakpoint.h"
#include "iris/IrisInstanceDebuggableState.h"
#include "iris/IrisInstanceDisassembler.h"
#include "iris/IrisInstanceEvent.h"
#include "iris/IrisInstanceImage.h"
#include "iris/IrisInstanceMemory.h"
#include "iris/IrisInstancePerInstanceExecution.h"
#include "iris/IrisInstanceResource.h"
#include "iris/IrisInstanceSemihosting.h"
#include "iris/IrisInstanceCheckpoint.h"
#include "iris/IrisInstanceStep.h"
#include "iris/IrisInstanceTable.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisElfDwarf.h"
#include <cassert>
```

Classes

- class iris::IrisInstanceBuilder::AddressTranslationBuilder

  Used to set metadata for an address translation.
- class iris::IrisInstanceBuilder::EventSourceBuilder

  Used to set metadata on an EventSource.
- class iris::IrisInstanceBuilder::FieldBuilder

  Used to set metadata on a register field resource.

- class iris::IrisInstanceBuilder

  Builder interface to populate an IrisInstance with registers, memory etc.
- class iris::IrisInstanceBuilder::MemorySpaceBuilder

  Used to set metadata for a memory space.
- class iris::IrisInstanceBuilder::ParameterBuilder

  Used to set metadata on a parameter.
- class iris::IrisInstanceBuilder::RegisterBuilder

  Used to set metadata on a register resource.
- class iris::IrisInstanceBuilder::SemihostingManager

  semihosting_apis IrisInstanceBuilder semihosting APIs
- class iris::IrisInstanceBuilder::TableBuilder

  Used to set metadata for a table.
- class iris::IrisInstanceBuilder::TableColumnBuilder

  Used to set metadata for a table column.

9.17.1 Detailed Description

A high level interface to build up functionality on an IrisInstance.

Copyright

9.18 IrisInstanceBuilder.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisInstanceBuilder_h
9 #define ARM_INCLUDE_IrisInstanceBuilder_h
10
11 #include "iris/IrisEventEmitter.h"
12 #include "iris/IrisInstance.h"
13 #include "iris/IrisInstanceBreakpoint.h"
14 #include "iris/IrisInstanceDebuggableState.h"
15 #include "iris/IrisInstanceDisassembler.h"
16 #include "iris/IrisInstanceEvent.h"
17 #include "iris/IrisInstanceImage.h"
18 #include "iris/IrisInstanceMemory.h"
19 #include "iris/IrisInstancePerInstanceExecution.h"
20 #include "iris/IrisInstanceResource.h"
21 #include "iris/IrisInstanceSemihosting.h"
22 #include "iris/IrisInstanceCheckpoint.h"
23 #include "iris/IrisInstanceStep.h"
24 #include "iris/IrisInstanceTable.h"
25 #include "iris/detail/IrisCommon.h"
26 #include "iris/detail/IrisElfDwarf.h"
27
28 #include <cassert>
29
30 NAMESPACE_IRIS_START
31
32 class IrisRegisterEventEmitterBase;
33
39 class IrisInstanceBuilder
40 {
41 private:
48     template <typename T, T* (IrisInstanceBuilder::*INIT_METHOD)()>
49     class LazyAddOn
50     {
51     private:
52         IrisInstanceBuilder* parent;
53         T*                   add_on;
54
55     public:
56         LazyAddOn(IrisInstanceBuilder* parent_)
57             : parent(parent_)
58             , add_on(nullptr)
59         {
60         }
61
62         ~LazyAddOn()
```

```
63          {
64              delete add_on;
65          }
66
67          T* operator->()
68          {
69              if (add_on == nullptr)
70              {
71                  init();
72              }
73
74              return add_on;
75          }
76
77          operator T*()
78          {
79              if (add_on == nullptr)
80              {
81                  init();
82              }
83
84              return add_on;
85          }
86
87          T* getPtr()
88          {
89              return add_on;
90          }
91
92          void init()
93          {
94              assert(add_on == nullptr);
95              add_on = (parent->*INIT_METHOD)();
96          }
97      };
98      IrisInstance* iris_instance;
99  #define INTERNAL_LAZY(addon) \
100     addon* init##addon();    \
101     LazyAddOn<addon, &IrisInstanceBuilder::init##addon>
102     INTERNAL_LAZY(IrisInstanceResource)
103     inst_resource;
104     INTERNAL_LAZY(IrisInstanceEvent)
105     inst_event;
106     INTERNAL_LAZY(IrisInstanceBreakpoint)
107     inst_breakpoint;
108     INTERNAL_LAZY(IrisInstanceMemory)
109     inst_memory;
110     INTERNAL_LAZY(IrisInstanceImage)
111     inst_image;
112     INTERNAL_LAZY(IrisInstanceImage_Callback)
113     inst_image_cb;
114     INTERNAL_LAZY(IrisInstanceStep)
115     inst_step;
116     INTERNAL_LAZY(IrisInstancePerInstanceExecution)
117     inst_per_inst_exec;
118     INTERNAL_LAZY(IrisInstanceTable)
119     inst_table;
120     INTERNAL_LAZY(IrisInstanceDisassembler)
121     inst_disass;
122     INTERNAL_LAZY(IrisInstanceDebuggableState)
123     inst_dbg_state;
124     INTERNAL_LAZY(IrisInstanceSemihosting)
125     inst_semihost;
126     INTERNAL_LAZY(IrisInstanceCheckpoint)
127     inst_checkpoint;
128 #undef INTERNAL_LAZY
129
131
132
134     ResourceReadDelegate  default_reg_read_delegate;
135     ResourceWriteDelegate default_reg_write_delegate;
136
139     bool canonicalRnSchemeIsAlreadySet{};
141
143
144     struct RegisterEventInfo
145     {
146         IrisInstanceEvent::EventSourceInfoAndDelegate event_info;
147
148         typedef std::vector<uint64_t> RscIdList;
149         RscIdList                     rscId_list;
150         IrisRegisterEventEmitterBase* event_emitter;
151
152         RegisterEventInfo()
153             : event_emitter(nullptr)
154         {
155         }
```

```
156     };
157
158     std::vector<RegisterEventInfo*> register_read_event_info_list;
159     std::vector<RegisterEventInfo*> register_update_event_info_list;
160
161     RegisterEventInfo* active_register_read_event_info{};
162     RegisterEventInfo* active_register_update_event_info{};
163
164     RegisterEventInfo* find_register_event(const std::vector<RegisterEventInfo*>&
      register_event_info_list,
165                                            const std::string&                    name);
166
167     RegisterEventInfo* initRegisterReadEventInfo(const std::string& name);
168     RegisterEventInfo* initRegisterUpdateEventInfo(const std::string& name);
169
172     void finalizeRegisterEvent(RegisterEventInfo* event_info, bool is_read);
174     void associateRegisterWithTraceEvents(ResourceId rscId);
176
178
179     IrisErrorCode setBreakpoint(BreakpointInfo& info);
180     IrisErrorCode deleteBreakpoint(const BreakpointInfo& info);
181
182     struct RegisterEventEmitterPair
183     {
184         IrisRegisterEventEmitterBase* read;
185         IrisRegisterEventEmitterBase* update;
186
187         RegisterEventEmitterPair()
188             : read(nullptr)
189             , update(nullptr)
190         {
191         }
192     };
193     typedef std::map<uint64_t, RegisterEventEmitterPair> RscIdEventEmitterMap;
194     RscIdEventEmitterMap                                 register_event_emitter_map;
195
196     BreakpointSetDelegate    user_setBreakpoint;
197     BreakpointDeleteDelegate user_deleteBreakpoint;
199
200 public:
206     IrisInstanceBuilder(IrisInstance* iris_instance);
207
208     /* No destructor: IrisInstanceBuilder objects live as long as the instance
209      * they belong to. Do not key anything to the destructor.
210      */
211
218 #define INTERNAL_RESOURCE_BUILDER_MIXIN(TYPE)
                \
219                                                                                 \
220                                                 \
221                     \
222     TYPE& setName(const std::string& name)
                \
223     {
                \
224         info->resourceInfo.name = name;
                \
225         return *this;
                \
226     }
                \
227                                                                                 \
228                                                 \
229                     \
230     TYPE& setCname(const std::string& cname)
                \
231     {
                \
232         info->resourceInfo.cname = cname;
                \
233         return *this;
                \
234     }
                \
235                                                                                 \
236                                     \
237                     \
238     TYPE& setDescription(const std::string& description)
                \
239     {
                \
240         info->resourceInfo.description = description;
                \
241         return *this;
                \
242     }
                \
```

```
243                                                 \
244     /* [[deprecated("Inconsistently named function. Use setDescription() instead.")]] */
                \
245     TYPE& setDescr(const std::string& description)
                \
246     {
                \
247         return setDescription(description);
                \
248     }
                \
249                                                                         \
250                                           \
251                   \
252     TYPE& setFormat(const std::string& format)
                \
253     {
                \
254         info->resourceInfo.format = format;
                \
255         return *this;
                \
256     }
                \
257                                                                       \
258                                       \
259                   \
260     TYPE& setBitWidth(uint64_t bitWidth)
                \
261     {
                \
262         info->resourceInfo.bitWidth = bitWidth;
                \
263         return *this;
                \
264     }
                \
265                                                                           \
266                                                 \
267                   \
268     TYPE& setType(const std::string& type)
                \
269     {
                \
270         info->resourceInfo.type = type;
                \
271         return *this;
                \
272     }
                \
273                                                                         \
274                                           \
275                   \
276     TYPE& setRwMode(const std::string& rwMode)
                \
277     {
                \
278         info->resourceInfo.rwMode = rwMode;
                \
279         return *this;
                \
280     }
                \
281                                                                       \
282                                       \
283                   \
284     TYPE& setSubRscId(uint64_t subRscId)
                \
285     {
                \
286         info->resourceInfo.subRscId = subRscId;
                \
287         return *this;
                \
288     }
                \
289                                           \
290                                           \
291                               \
292                                                       \
293                                                   \
294                   \
295     TYPE& addEnum(const std::string& symbol, const IrisValue& value, const std::string& description =
    std::string())      \
296     {
                \
297         info->resourceInfo.enums.push_back(EnumElementInfo(value, symbol, description));
```

```
                    \
298         return *this;
                \
299     }
                \
300                                             \
301                                              \
302                 \
303                                                         \
304                     \
305     TYPE& addStringEnum(const std::string& stringValue, const std::string& description = std::string())
                    \
306     {
                \
307         info->resourceInfo.enums.push_back(EnumElementInfo(IrisValue(stringValue), std::string(),
    description));          \
308         return *this;
                \
309     }
                \
310                                                     \
311                                                             \
312                     \
313     TYPE& setTag(const std::string& tag)
                \
314     {
                \
315         info->resourceInfo.tags[tag] = IrisValue(true);
                \
316         return *this;
                \
317     }
            \
318                                                                 \
319                                                                  \
320                                                                 \
321                     \
322     TYPE& setTag(const std::string& tag, const IrisValue& value)
                \
323     {
                \
324         info->resourceInfo.tags[tag] = value;
                \
325         return *this;
                \
326     }
                \
327                                                             \
328                                                         \
329                                                    \
330                                                       \
331                     \
332     TYPE& setReadDelegate(ResourceReadDelegate readDelegate)
                \
333     {
                \
334         info->readDelegate = readDelegate;
                \
335         return *this;
                \
336     }
                \
337                                                         \
338                                                     \
339                                                \
340                                                   \
341                     \
342     TYPE& setWriteDelegate(ResourceWriteDelegate writeDelegate)
                \
343     {
                \
344         info->writeDelegate = writeDelegate;
                \
345         return *this;
                \
346     }
                \
347                                                             \
348                                             \
349                                                     \
350                                                \
351   \
352                                                 \
353                                         \
354                 \
355     template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, ResourceReadResult&)>
                \
```

```
356     TYPE& setReadDelegate(T* instance)
                \
357     {
                \
358         return setReadDelegate(ResourceReadDelegate::make<T, METHOD>(instance));
                \
359     }
                \
360                                                             \
361                                                              \
362                                                           \
363                                                      \
364                                                         \
365                 \
366     template <IrisErrorCode (*FUNC)(const ResourceInfo&, ResourceReadResult&)>
                \
367     TYPE& setReadDelegate()
                \
368     {
                \
369         return setReadDelegate(ResourceReadDelegate::make<FUNC>());
                \
370     }
                \
371                                                           \
372                                           \
373                                                    \
374                                                \
375 \
376                                             \
377                                         \
378                   \
379     template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, const ResourceWriteValue&)>
                \
380     TYPE& setWriteDelegate(T* instance)
                \
381     {
                \
382         return setWriteDelegate(ResourceWriteDelegate::make<T, METHOD>(instance));
                \
383     }
                \
384                                                           \
385                                                            \
386                                                        \
387                                                   \
388                                                      \
389                   \
390     template <IrisErrorCode (*FUNC)(const ResourceInfo&, const ResourceWriteValue&)>
                \
391     TYPE& setWriteDelegate()
                \
392     {
                \
393         return setWriteDelegate(ResourceWriteDelegate::make<FUNC>());
                \
394     }
                \
395                                                                     \
396       \
397                                       \
398                                             \
399                   \
400     TYPE& setParentRscId(ResourceId parentRscId)
                \
401     {
                \
402         info->resourceInfo.parentRscId = parentRscId;
                \
403         return *this;
                \
404     }
                \
405                                           \
406                                         \
407     ResourceId getRscId() const
                \
408     {
                \
409         return info->resourceInfo.rscId;
                \
410     }
                \
411                                                \
412                           \
413                                                                  \
414                   \
```

```
415     TYPE& getRscId(ResourceId &rscIdOut)
                \
416     {
                \
417         rscIdOut = info->resourceInfo.rscId;
                \
418         return *this;
                \
419     }
420
421 #define INTERNAL_REGISTER_BUILDER_MIXIN(TYPE)
    \
422                                                                 \
423                         \
424  \
425     TYPE& setLsbOffset(uint64_t lsbOffset)
    \
426     {
    \
427         info->resourceInfo.registerInfo.lsbOffset = lsbOffset;
    \
428         return *this;
    \
429     }
    \
430                                                             \
431                           \
432                       \
433  \
434     TYPE& setCanonicalRn(uint64_t canonicalRn_)
    \
435     {
    \
436         info->resourceInfo.registerInfo.canonicalRn    = canonicalRn_;
    \
437         info->resourceInfo.registerInfo.hasCanonicalRn = true;
    \
438         return *this;
    \
439     }
    \
440                                     \
441                                     \
442                                 \
443  \
444     TYPE& setCanonicalRnElfDwarf(uint16_t architecture, uint16_t dwarfRegNum)
    \
445     {
    \
446         if (!instance_builder->canonicalRnSchemeIsAlreadySet) /* Only set property if not already set.
    */\
447         {
    \
448             if (getWithDefault(instance_builder->iris_instance->getPropertyMap(),
    "register.canonicalRnScheme", "").getAsString().empty()) \
449             {
    \
450                 instance_builder->setPropertyCanonicalRnScheme("ElfDwarf");
    \
451             }
    \
452             instance_builder->canonicalRnSchemeIsAlreadySet = true;
    \
453         }
    \
454         return setCanonicalRn(makeCanonicalRnElfDwarf(architecture, dwarfRegNum));
    \
455     }
    \
456                                       \
457                   \
458            \
459                                       \
460  \
461     TYPE& setWriteMask(uint64_t value)
    \
462     {
    \
463         info->resourceInfo.setVector(info->resourceInfo.registerInfo.writeMask, value);
    \
464         return *this;
    \
465     }
    \
466                                         \
467        \
468       \
```

```
469                         \
470                     \
471             \
472                         \
473 \
474     template<typename Container>
    \
475     TYPE& setWriteMaskFromContainer(const Container& container)
    \
476     {
    \
477         info->resourceInfo.setVectorFromContainer(info->resourceInfo.registerInfo.writeMask, container);
    \
478         return *this;
    \
479     }
    \
480                                         \
481                     \                                                   \
482                                                                             \
483                                           \
484                                               \
485 \
486     template<typename T>
    \
487     TYPE& setWriteMask(std::initializer_list<T>&& t)
    \
488     {
    \
489         setWriteMaskFromContainer(std::forward<std::initializer_list<T»(t));
    \
490         return *this;
    \
491     }
    \
492                                       \
493                     \
494             \
495                                       \
496 \
497     TYPE& setResetData(uint64_t value)
    \
498     {
    \
499         info->resourceInfo.setVector(info->resourceInfo.registerInfo.resetData, value);
    \
500         return *this;
    \
501     }
    \
502                                         \
503         \
504       \
505                           \
506                     \
507             \
508                           \
509 \
510     template<typename Container>
    \
511     TYPE& setResetDataFromContainer(const Container& container)
    \
512     {
    \
513         info->resourceInfo.setVectorFromContainer(info->resourceInfo.registerInfo.resetData, container);
    \
514         return *this;
    \
515     }
    \
516                                         \
517                     \                                                   \
518                                                                             \
519                                           \
520                                               \
521 \
522     template<typename T>
    \
523     TYPE& setResetData(std::initializer_list<T>&& t)
    \
524     {
    \
525         setResetDataFromContainer(std::forward<std::initializer_list<T»(t));
    \
526         return *this;
    \
527     }
```

```
    \
528                                                         \
529                                                     \
530                     \
531 \
532     TYPE& setResetString(const std::string& resetString)
    \
533     {
    \
534         info->resourceInfo.registerInfo.resetString = resetString;
    \
535         return *this;
    \
536     }
    \
537                                                         \
538                     \
539 \
540     TYPE& setAddressOffset(uint64_t addressOffset)
    \
541     {
    \
542         info->resourceInfo.registerInfo.addressOffset    = addressOffset;
    \
543         info->resourceInfo.registerInfo.hasAddressOffset = true;
    \
544         return *this;
    \
545     }
546
547 #define INTERNAL_PARAMETER_BUILDER_MIXIN(TYPE)
    \
548                     \
549                     \
550           \
551                       \
552 \
553     TYPE& setDefaultData(uint64_t value)
    \
554     {
    \
555         info->resourceInfo.setVector(info->resourceInfo.parameterInfo.defaultData, value);
    \
556         return *this;
    \
557     }
    \
558                                     \
559       \
560       \
561                           \
562                     \
563           \
564                       \
565 \
566     template<typename Container>
    \
567     TYPE& setDefaultDataFromContainer(const Container& container)
    \
568     {
    \
569         info->resourceInfo.setVectorFromContainer(info->resourceInfo.parameterInfo.defaultData,
    container); \
570         return *this;
    \
571     }
    \
572                                     \
573                   \
574                                                             \
575                                           \
576                                               \
577 \
578     template<typename T>
    \
579     TYPE& setDefaultData(std::initializer_list<T>&& t)
    \
580     {
    \
581         setDefaultDataFromContainer(std::forward<std::initializer_list<T>>(t));
    \
582         return *this;
    \
583     }
    \
584           \
585                                                 \
```

```
586                   \
587 \
588     TYPE& setDefaultString(const std::string& defaultString)
        \
589     {
        \
590         info->resourceInfo.parameterInfo.defaultString = defaultString;
        \
591         return *this;
        \
592     }
        \
593                                                 \
594                                           \
595                                       \
596 \
597     TYPE& setInitOnly(bool initOnly = true)
        \
598     {
        \
599         info->resourceInfo.parameterInfo.initOnly = initOnly;
        \
600         /* Implicitly set read-only to make clear that parameter cannot be modified at run-time. */
        \
601         info->resourceInfo.rwMode = initOnly ? "r" : std::string(); /* =rw */
        \
602         return *this;
        \
603     }
        \
604                                                                     \
605                 \
606     /*                  but can still be accessed by resource_getResourceInfo() for clients that know the
    */\
607     /*                  resource name. */
        \
608           \
609     TYPE& setHidden(bool hidden = true)
        \
610     {
        \
611         info->resourceInfo.isHidden = hidden;
        \
612         return *this;
        \
613     }
        \
614                                               \
615                   \
616           \
617                                             \
618 \
619     TYPE& setMax(uint64_t value)
        \
620     {
        \
621         info->resourceInfo.setVector(info->resourceInfo.parameterInfo.max, value);
        \
622         return *this;
        \
623     }
        \
624                                       \
625         \
626       \
627                           \
628                   \
629           \
630                           \
631 \
632     template<typename Container>
        \
633     TYPE& setMaxFromContainer(const Container& container)
        \
634     {
        \
635         info->resourceInfo.setVectorFromContainer(info->resourceInfo.parameterInfo.max, container);
        \
636         return *this;
        \
637     }
        \
638                                       \
639                   \
640                                                                       \
641                                           \
642                                               \
```

```
643 \
644     template<typename T>
    \
645     TYPE& setMax(std::initializer_list<T>&& t)
    \
646     {
    \
647         setMaxFromContainer(std::forward<std::initializer_list<T>>(t));
    \
648         return *this;
    \
649     }
    \
650                                             \
651                     \
652           \
653                                             \
654 \
655     TYPE& setMin(uint64_t value)
    \
656     {
    \
657         info->resourceInfo.setVector(info->resourceInfo.parameterInfo.min, value);
    \
658         return *this;
    \
659     }
    \
660                                         \
661         \
662       \
663                           \
664                     \
665           \
666                       \
667 \
668     template<typename Container>
    \
669     TYPE& setMinFromContainer(const Container& container)
    \
670     {
    \
671         info->resourceInfo.setVectorFromContainer(info->resourceInfo.parameterInfo.min, container);
    \
672         return *this;
    \
673     }
    \
674                                         \
675                   \
676                                                                       \
677                                             \
678                                                 \
679 \
680     template<typename T>
    \
681     TYPE& setMin(std::initializer_list<T>&& t)
    \
682     {
    \
683         setMinFromContainer(std::forward<std::initializer_list<T>>(t));
    \
684         return *this;
    \
685     }
    \
686
690     class ParameterBuilder
691     {
692     private:
693         IrisInstanceResource::ResourceInfoAndAccess* info;
694
695     public:
696         ParameterBuilder(IrisInstanceResource::ResourceInfoAndAccess& info_)
697             : info(&info_)
698         {
699             info->resourceInfo.isParameter = true;
700         }
701
702         ParameterBuilder()
703             : info(nullptr)
704         {
705         }
706
707         INTERNAL_RESOURCE_BUILDER_MIXIN(ParameterBuilder)
708         INTERNAL_PARAMETER_BUILDER_MIXIN(ParameterBuilder)
709     };
```

```
710
711     class FieldBuilder;
712
716     class RegisterBuilder
717     {
718     private:
719         IrisInstanceResource::ResourceInfoAndAccess* info{};
720         IrisInstanceResource*                        inst_resource{};
721         IrisInstanceBuilder*                         instance_builder{};
722
723     public:
724         RegisterBuilder(IrisInstanceResource::ResourceInfoAndAccess& info_, IrisInstanceResource*
    inst_resource_, IrisInstanceBuilder *instance_builder_)
725             : info(&info_)
726             , inst_resource(inst_resource_)
727             , instance_builder(instance_builder_)
728         {
729             info->resourceInfo.isRegister = true;
730         }
731
732         RegisterBuilder()
733         {
734         }
735
736         INTERNAL_RESOURCE_BUILDER_MIXIN(RegisterBuilder)
737         INTERNAL_REGISTER_BUILDER_MIXIN(RegisterBuilder)
738
748         FieldBuilder addField(const std::string& name, uint64_t lsbOffset, uint64_t bitWidth, const
    std::string& description);
749
763         FieldBuilder addLogicalField(const std::string& name, uint64_t bitWidth, const std::string&
    description);
764     };
765
769     class FieldBuilder
770     {
771     protected:
772         IrisInstanceResource::ResourceInfoAndAccess* info{};
773         RegisterBuilder*                             parent_reg{};
774         IrisInstanceBuilder*                         instance_builder{};
775
776     public:
777         FieldBuilder(IrisInstanceResource::ResourceInfoAndAccess& info_, RegisterBuilder* parent_reg_,
    IrisInstanceBuilder *instance_builder_)
778             : info(&info_)
779             , parent_reg(parent_reg_)
780             , instance_builder(instance_builder_)
781         {
782         }
783
784         FieldBuilder()
785         {
786         }
787
788         INTERNAL_RESOURCE_BUILDER_MIXIN(FieldBuilder)
789         INTERNAL_REGISTER_BUILDER_MIXIN(FieldBuilder)
790
795         RegisterBuilder& parent()
796         {
797             return *parent_reg;
798         }
799
804         FieldBuilder addField(const std::string& name, uint64_t lsbOffset, uint64_t bitWidth, const
    std::string& description)
805         {
806             return parent().addField(name, lsbOffset, bitWidth, description);
807         }
808
813         FieldBuilder addLogicalField(const std::string& name, uint64_t bitWidth, const std::string&
    description)
814         {
815             return parent().addLogicalField(name, bitWidth, description);
816         }
817     };
818
819 #undef INTERNAL_RESOURCE_BUILDER_MIXIN
820 #undef INTERNAL_REGISTER_BUILDER_MIXIN
821 #undef INTERNAL_PARAMETER_BUILDER_MIXIN
822
853     void setDefaultResourceReadDelegate(ResourceReadDelegate delegate = ResourceReadDelegate())
854     {
855         default_reg_read_delegate = delegate;
856     }
857
885     template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, ResourceReadResult&)>
886     void setDefaultResourceReadDelegate(T* instance)
887     {
```

```
888         setDefaultResourceReadDelegate(ResourceReadDelegate::make<T, METHOD>(instance));
889     }
890
910     template <IrisErrorCode (*FUNC)(const ResourceInfo&, ResourceReadResult&)>
911     void setDefaultResourceReadDelegate()
912     {
913         setDefaultResourceReadDelegate(ResourceReadDelegate::make<FUNC>());
914     }
915
945     void setDefaultResourceWriteDelegate(ResourceWriteDelegate delegate = ResourceWriteDelegate())
946     {
947         default_reg_write_delegate = delegate;
948     }
949
976     template <typename T, IrisErrorCode (T::*METHOD)(const ResourceInfo&, const ResourceWriteValue&)>
977     void setDefaultResourceWriteDelegate(T* instance)
978     {
979         setDefaultResourceWriteDelegate(ResourceWriteDelegate::make<T, METHOD>(instance));
980     }
981
1000     template <IrisErrorCode (*FUNC)(const ResourceInfo&, const ResourceWriteValue&)>
1001     void setDefaultResourceWriteDelegate()
1002     {
1003         setDefaultResourceWriteDelegate(ResourceWriteDelegate::make<*FUNC>());
1004     }
1005
1015     template <typename T, IrisErrorCode (T::*READER)(const ResourceInfo&, ResourceReadResult&),
1016               IrisErrorCode (T::*WRITER)(const ResourceInfo&, const ResourceWriteValue&)>
1017     void setDefaultResourceDelegates(T* instance)
1018     {
1019         setDefaultResourceReadDelegate(ResourceReadDelegate::make<T, READER>(instance));
1020         setDefaultResourceWriteDelegate(ResourceWriteDelegate::make<T, WRITER>(instance));
1021     }
1022
1045     void beginResourceGroup(const std::string& name,
1046                             const std::string& description,
1047                             uint64_t           subRscIdStart = IRIS_UINT64_MAX,
1048                             const std::string& cname         = std::string());
1049
1072     ParameterBuilder addParameter(const std::string& name, uint64_t bitWidth, const std::string&
    description);
1073
1092     ParameterBuilder addStringParameter(const std::string& name, const std::string& description);
1093
1127     RegisterBuilder addRegister(const std::string& name, uint64_t bitWidth, const std::string&
    description,
1128                                 uint64_t addressOffset = IRIS_UINT64_MAX, uint64_t canonicalRn =
    IRIS_UINT64_MAX);
1129
1148     RegisterBuilder addStringRegister(const std::string& name, const std::string& description);
1149
1170     RegisterBuilder addNoValueRegister(const std::string& name, const std::string& description, const
    std::string& format);
1171
1190     ParameterBuilder enhanceParameter(ResourceId rscId)
1191     {
1192         return ParameterBuilder(*(inst_resource->getResourceInfo(rscId)));
1193     }
1194
1216     RegisterBuilder enhanceRegister(ResourceId rscId)
1217     {
1218         return RegisterBuilder(*(inst_resource->getResourceInfo(rscId)), inst_resource, this);
1219     }
1220
1243     void setPropertyCanonicalRnScheme(const std::string& canonicalRnScheme);
1244
1252     void setNextSubRscId(uint64_t nextSubRscId)
1253     {
1254         inst_resource->setNextSubRscId(nextSubRscId);
1255     }
1256
1266     void setTag(ResourceId rscId, const std::string& tag);
1267
1275     const ResourceInfo &getResourceInfo(ResourceId rscId)
1276     {
1277         return inst_resource->getResourceInfo(rscId)->resourceInfo;
1278     }
1279
1280
1294     class EventSourceBuilder
1295     {
1296     private:
1297         IrisInstanceEvent::EventSourceInfoAndDelegate& info;
1298
1299     public:
1300         EventSourceBuilder(IrisInstanceEvent::EventSourceInfoAndDelegate& info_)
1301             : info(info_)
```

```
1302        {
1303        }
1304
1310        EventSourceBuilder& setName(const std::string& name)
1311        {
1312            info.info.name = name;
1313            return *this;
1314        }
1315
1321        EventSourceBuilder& setDescription(const std::string& description)
1322        {
1323            info.info.description = description;
1324            return *this;
1325        }
1326
1332        EventSourceBuilder& setFormat(const std::string& format)
1333        {
1334            info.info.format = format;
1335            return *this;
1336        }
1337
1343        EventSourceBuilder& setCounter(bool counter = true)
1344        {
1345            info.info.counter = counter;
1346            return *this;
1347        }
1348
1356        EventSourceBuilder& setHidden(bool hidden = true)
1357        {
1358            info.info.isHidden = hidden;
1359            return *this;
1360        }
1361
1368        EventSourceBuilder& hasSideEffects(bool hasSideEffects_ = true)
1369        {
1370            info.info.hasSideEffects = hasSideEffects_;
1371            return *this;
1372        }
1373
1386        EventSourceBuilder& addField(const std::string& name, const std::string& type, uint64_t size,
1387                                     const std::string& description)
1388        {
1389            info.info.addField(name, type, size, description);
1390            return *this;
1391        }
1392
1403        EventSourceBuilder& addEnumElement(uint64_t value, const std::string& symbol, const
    std::string& description = "")
1404        {
1405            if (info.info.fields.size() > 0)
1406            {
1407                info.info.fields.back().addEnumElement(value, symbol, description);
1408                return *this;
1409            }
1410            else
1411            {
1412                throw IrisInternalError("EventSourceInfo has no fields to add an enum element to.");
1413            }
1414        }
1415
1425        EventSourceBuilder& setEventStreamCreateDelegate(EventStreamCreateDelegate delegate)
1426        {
1427            info.createEventStream = delegate;
1428            return *this;
1429        }
1430
1443        template <typename T,
1444                  IrisErrorCode (T::*METHOD)(EventStream*&, const EventSourceInfo&, const
    std::vector<std::string>&)>
1445        EventSourceBuilder& setEventStreamCreateDelegate(T* instance)
1446        {
1447            return setEventStreamCreateDelegate(EventStreamCreateDelegate::make<T, METHOD>(instance));
1448        }
1449
1463        template<typename T>
1464        EventSourceBuilder& addOption(const std::string& name, const std::string& type, const T&
    defaultValue,
1465                                      bool optional, const std::string& description)
1466        {
1467            info.info.addOption(name, type, defaultValue, optional, description);
1468            return *this;
1469        }
1470    };
1471
1486    EventSourceBuilder addEventSource(const std::string& name, bool isHidden = false)
1487    {
1488        return EventSourceBuilder(inst_event->addEventSource(name, isHidden));
```

```
1489     }
1490
1502     EventSourceBuilder addEventSource(const std::string& name, IrisEventEmitterBase& event_emitter,
     bool isHidden = false)
1503     {
1504         IrisInstanceEvent::EventSourceInfoAndDelegate& info = inst_event->addEventSource(name,
     isHidden);
1505
1506         event_emitter.setIrisInstance(iris_instance);
1507         event_emitter.setEvSrcId(info.info.evSrcId);
1508         info.createEventStream = EventStreamCreateDelegate::make<IrisEventEmitterBase,
1509
     &IrisEventEmitterBase::createEventStream>(&event_emitter);
1510
1511         return EventSourceBuilder(info);
1512     }
1513
1539     EventSourceBuilder setRegisterReadEvent(const std::string& name, const std::string& description =
     std::string());
1540
1566     EventSourceBuilder setRegisterReadEvent(const std::string& name, IrisRegisterEventEmitterBase&
     event_emitter);
1567
1574     void finalizeRegisterReadEvent();
1575
1602     EventSourceBuilder setRegisterUpdateEvent(const std::string& name, const std::string& description =
     std::string());
1603
1630     EventSourceBuilder setRegisterUpdateEvent(const std::string& name, IrisRegisterEventEmitterBase&
     event_emitter);
1631
1638     void finalizeRegisterUpdateEvent();
1639
1646     void resetRegisterReadEvent();
1647
1654     void resetRegisterUpdateEvent();
1655
1687     void setDefaultEsCreateDelegate(EventStreamCreateDelegate delegate)
1688     {
1689         inst_event->setDefaultEsCreateDelegate(delegate);
1690     }
1691
1722     template <typename T, IrisErrorCode (T::*METHOD)(EventStream*&, const EventSourceInfo&, const
     std::vector<std::string>&)>
1723     void setDefaultEsCreateDelegate(T* instance)
1724     {
1725         setDefaultEsCreateDelegate(EventStreamCreateDelegate::make<T, METHOD>(instance));
1726     }
1727
1750     template <IrisErrorCode (*FUNC)(EventStream*&, const EventSourceInfo&, const
     std::vector<std::string>&)>
1751     void setDefaultEsCreateDelegate()
1752     {
1753         setDefaultEsCreateDelegate(EventStreamCreateDelegate::make<FUNC>());
1754     }
1755
1762     IrisInstanceEvent* getIrisInstanceEvent() { return inst_event; }
1763
1795     void setBreakpointSetDelegate(BreakpointSetDelegate delegate)
1796     {
1797         if (inst_breakpoint.getPtr() == nullptr)
1798         {
1799             // Ensure the underlying IrisInstanceBreakpoint object is initialised too.
1800             inst_breakpoint.init();
1801         }
1802         user_setBreakpoint = delegate;
1803     }
1804
1826     template <typename T, IrisErrorCode (T::*METHOD)(BreakpointInfo&)>
1827     void setBreakpointSetDelegate(T* instance)
1828     {
1829         setBreakpointSetDelegate(BreakpointSetDelegate::make<T, METHOD>(instance));
1830     }
1831
1845     template <IrisErrorCode (*FUNC)(BreakpointInfo&)>
1846     void setBreakpointSetDelegate()
1847     {
1848         setBreakpointSetDelegate(BreakpointSetDelegate::make<FUNC>());
1849     }
1850
1872     void setBreakpointDeleteDelegate(BreakpointDeleteDelegate delegate)
1873     {
1874         if (inst_breakpoint.getPtr() == nullptr)
1875         {
1876             // Ensure the underlying IrisInstanceBreakpoint object is initialised too.
1877             inst_breakpoint.init();
1878         }
```

```
1879         user_deleteBreakpoint = delegate;
1880     }
1881
1903     template <typename T, IrisErrorCode (T::*METHOD)(const BreakpointInfo&)>
1904     void setBreakpointDeleteDelegate(T* instance)
1905     {
1906         setBreakpointDeleteDelegate(BreakpointDeleteDelegate::make<T, METHOD>(instance));
1907     }
1908
1922     template <IrisErrorCode (*FUNC)(const BreakpointInfo&)>
1923     void setBreakpointDeleteDelegate()
1924     {
1925         setBreakpointDeleteDelegate(BreakpointDeleteDelegate::make<FUNC>());
1926     }
1927
1938     void notifyBreakpointHit(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId pcSpaceId)
1939     {
1940         inst_breakpoint->notifyBreakpointHit(bptId, time, pc, pcSpaceId);
1941     }
1942
1958     void notifyBreakpointHitData(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId
    pcSpaceId,
1959                                  uint64_t accessAddr, uint64_t accessSize,
1960                                  const std::string& accessRw, const std::vector<uint64_t>& data)
1961     {
1962         inst_breakpoint->notifyBreakpointHitData(bptId, time, pc, pcSpaceId, accessAddr, accessSize,
    accessRw, data);
1963     }
1964
1978     void notifyBreakpointHitRegister(BreakpointId bptId, uint64_t time, uint64_t pc, MemorySpaceId
    pcSpaceId,
1979                                      const std::string& accessRw, const std::vector<uint64_t>& data)
1980     {
1981         inst_breakpoint->notifyBreakpointHitRegister(bptId, time, pc, pcSpaceId, accessRw, data);
1982     }
1983
1991     const BreakpointInfo* getBreakpointInfo(BreakpointId bptId)
1992     {
1993         return inst_breakpoint->getBreakpointInfo(bptId);
1994     }
1995
1997     void addBreakpointCondition(const std::string& name, const std::string& type, const std::string&
    description,
1998                                 const std::vector<std::string> bpt_types = std::vector<std::string>())
1999     {
2000         inst_breakpoint->addCondition(name, type, description, bpt_types);
2001     }
2002
2016     class MemorySpaceBuilder
2017     {
2018     private:
2019         IrisInstanceMemory::SpaceInfoAndAccess& info;
2020
2021     public:
2022         MemorySpaceBuilder(IrisInstanceMemory::SpaceInfoAndAccess& info_)
2023             : info(info_)
2024         {
2025         }
2026
2033         MemorySpaceBuilder& setName(const std::string& name)
2034         {
2035             info.spaceInfo.name = name;
2036             return *this;
2037         }
2038
2045         MemorySpaceBuilder& setDescription(const std::string& description)
2046         {
2047             info.spaceInfo.description = description;
2048             return *this;
2049         }
2050
2057         MemorySpaceBuilder& setMinAddr(uint64_t minAddr)
2058         {
2059             info.spaceInfo.minAddr = minAddr;
2060             return *this;
2061         }
2062
2069         MemorySpaceBuilder& setMaxAddr(uint64_t maxAddr)
2070         {
2071             info.spaceInfo.maxAddr = maxAddr;
2072             return *this;
2073         }
2074
2081         MemorySpaceBuilder& setCanonicalMsn(uint64_t canonicalMsn)
2082         {
2083             info.spaceInfo.canonicalMsn = canonicalMsn;
2084             return *this;
```

```
2085         }
2086
2093         MemorySpaceBuilder& setEndianness(const std::string& endianness)
2094         {
2095             info.spaceInfo.endianness = endianness;
2096             return *this;
2097         }
2098
2106         MemorySpaceBuilder& addAttribute(const std::string& name, AttributeInfo attrib)
2107         {
2108             info.spaceInfo.attrib[name] = attrib;
2109             return *this;
2110         }
2111
2119         MemorySpaceBuilder& setAttributeDefault(const std::string& name, IrisValue value)
2120         {
2121             info.spaceInfo.attribDefaults[name] = value;
2122             return *this;
2123         }
2124
2135         MemorySpaceBuilder& setReadDelegate(MemoryReadDelegate delegate)
2136         {
2137             info.readDelegate = delegate;
2138             return *this;
2139         }
2140
2151         MemorySpaceBuilder& setWriteDelegate(MemoryWriteDelegate delegate)
2152         {
2153             info.writeDelegate = delegate;
2154             return *this;
2155         }
2156
2167         MemorySpaceBuilder& setSidebandDelegate(MemoryGetSidebandInfoDelegate delegate)
2168         {
2169             info.sidebandDelegate = delegate;
2170             return *this;
2171         }
2172
2186         template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
    uint64_t, const AttributeValueMap&, MemoryReadResult&)>
2187         MemorySpaceBuilder& setReadDelegate(T* instance)
2188         {
2189             return setReadDelegate(MemoryReadDelegate::make<T, METHOD>(instance));
2190         }
2191
2205         template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
    uint64_t, const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2206         MemorySpaceBuilder& setWriteDelegate(T* instance)
2207         {
2208             return setWriteDelegate(MemoryWriteDelegate::make<T, METHOD>(instance));
2209         }
2210
2224         template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, const
    IrisValueMap&, const std::vector<std::string>&, IrisValueMap&)>
2225         MemorySpaceBuilder& setSidebandDelegate(T* instance)
2226         {
2227             return setSidebandDelegate(MemoryGetSidebandInfoDelegate::make<T, METHOD>(instance));
2228         }
2229
2240         template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2241                                         const AttributeValueMap&, MemoryReadResult&)>
2242         MemorySpaceBuilder& setReadDelegate()
2243         {
2244             return setReadDelegate(MemoryReadDelegate::make<FUNC>());
2245         }
2246
2257         template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2258                                         const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2259         MemorySpaceBuilder& setWriteDelegate()
2260         {
2261             return setWriteDelegate(MemoryWriteDelegate::make<FUNC>());
2262         }
2263
2274         template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, const IrisValueMap&,
2275                                         const std::vector<std::string>&, IrisValueMap&)>
2276         MemorySpaceBuilder& setSidebandDelegate()
2277         {
2278             return setSidebandDelegate(MemoryGetSidebandInfoDelegate::make<FUNC>());
2279         }
2280
2289         MemorySpaceId getSpaceId() const
2290         {
2291             return info.spaceInfo.spaceId;
2292         }
2293     };
2294
2298     class AddressTranslationBuilder
```

```
2299    {
2300    private:
2301        IrisInstanceMemory::AddressTranslationInfoAndAccess& info;
2302
2303    public:
2304        AddressTranslationBuilder(IrisInstanceMemory::AddressTranslationInfoAndAccess& info_)
2305            : info(info_)
2306        {
2307        }
2308
2319        AddressTranslationBuilder& setTranslateDelegate(MemoryAddressTranslateDelegate delegate)
2320        {
2321            info.translateDelegate = delegate;
2322            return *this;
2323        }
2324
2338        template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, uint64_t, uint64_t,
    MemoryAddressTranslationResult&)>
2339        AddressTranslationBuilder& setTranslateDelegate(T* instance)
2340        {
2341            return setTranslateDelegate(MemoryAddressTranslateDelegate::make<T, METHOD>(instance));
2342        }
2343
2354        template <IrisErrorCode (*FUNC)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&)>
2355        AddressTranslationBuilder& setTranslateDelegate()
2356        {
2357            return setTranslateDelegate(MemoryAddressTranslateDelegate::make<FUNC>());
2358        }
2359    };
2360
2373    void setPropertyCanonicalMsnScheme(const std::string& canonicalMsnScheme);
2374
2407    void setDefaultMemoryReadDelegate(MemoryReadDelegate delegate = MemoryReadDelegate())
2408    {
2409        inst_memory->setDefaultReadDelegate(delegate);
2410    }
2411
2444    template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
    uint64_t, const AttributeValueMap&, MemoryReadResult&)>
2445    void setDefaultMemoryReadDelegate(T* instance)
2446    {
2447        setDefaultMemoryReadDelegate(MemoryReadDelegate::make<T, METHOD>(instance));
2448    }
2449
2475    template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2476                                    const AttributeValueMap&, MemoryReadResult&)>
2477    void setDefaultMemoryReadDelegate()
2478    {
2479        setDefaultMemoryReadDelegate(MemoryReadDelegate::make<FUNC>());
2480    }
2481
2515    void setDefaultMemoryWriteDelegate(MemoryWriteDelegate delegate = MemoryWriteDelegate())
2516    {
2517        inst_memory->setDefaultWriteDelegate(delegate);
2518    }
2519
2553    template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, uint64_t,
    uint64_t, const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2554    void setDefaultMemoryWriteDelegate(T* instance)
2555    {
2556        setDefaultMemoryWriteDelegate(MemoryWriteDelegate::make<T, METHOD>(instance));
2557    }
2558
2584    template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
2585                                    const AttributeValueMap&, const uint64_t*, MemoryWriteResult&)>
2586    void setDefaultMemoryWriteDelegate()
2587    {
2588        setDefaultMemoryWriteDelegate(MemoryWriteDelegate::make<FUNC>());
2589    }
2590
2609    MemorySpaceBuilder addMemorySpace(const std::string& name)
2610    {
2611        return MemorySpaceBuilder(inst_memory->addMemorySpace(name));
2612    }
2613
2645    void setDefaultAddressTranslateDelegate(MemoryAddressTranslateDelegate delegate =
    MemoryAddressTranslateDelegate())
2646    {
2647        inst_memory->setDefaultTranslateDelegate(delegate);
2648    }
2649
2677    template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, uint64_t, uint64_t,
    MemoryAddressTranslationResult&)>
2678    void setDefaultAddressTranslateDelegate(T* instance)
2679    {
2680        setDefaultAddressTranslateDelegate(MemoryAddressTranslateDelegate::make<T, METHOD>(instance));
2681    }
```

```
2682
2702     template <IrisErrorCode (*FUNC)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&)>
2703     void setDefaultAddressTranslateDelegate()
2704     {
2705         setDefaultAddressTranslateDelegate(MemoryAddressTranslateDelegate::make<FUNC>());
2706     }
2707
2724     AddressTranslationBuilder addAddressTranslation(MemorySpaceId inSpaceId, MemorySpaceId outSpaceId,
2725                                                     const std::string& description)
2726     {
2727         return AddressTranslationBuilder(inst_memory->addAddressTranslation(inSpaceId, outSpaceId,
     description));
2728     }
2729
2762     void setDefaultGetMemorySidebandInfoDelegate(MemoryGetSidebandInfoDelegate delegate)
2763     {
2764         inst_memory->setDefaultGetSidebandInfoDelegate(delegate);
2765     }
2766
2795     template <typename T, IrisErrorCode (T::*METHOD)(const MemorySpaceInfo&, uint64_t, const
     IrisValueMap&, const std::vector<std::string>&, IrisValueMap&)>
2796     void setDefaultGetMemorySidebandInfoDelegate(T* instance)
2797     {
2798         setDefaultGetMemorySidebandInfoDelegate(MemoryGetSidebandInfoDelegate::make<T,
     METHOD>(instance));
2799     }
2800
2821     template <IrisErrorCode (*FUNC)(const MemorySpaceInfo&, uint64_t, const IrisValueMap&,
2822                                     const std::vector<std::string>&, IrisValueMap&)>
2823     void setDefaultGetMemorySidebandInfoDelegate()
2824     {
2825         setDefaultGetMemorySidebandInfoDelegate(MemoryGetSidebandInfoDelegate::make<FUNC>());
2826     }
2827
2862     void setLoadImageFileDelegate(ImageLoadFileDelegate delegate = ImageLoadFileDelegate())
2863     {
2864         inst_image->setLoadImageFileDelegate(delegate);
2865     }
2866
2887     template <typename T, IrisErrorCode (T::*METHOD)(const std::string&)>
2888     void setLoadImageFileDelegate(T* instance)
2889     {
2890         setLoadImageFileDelegate(ImageLoadFileDelegate::make<T, METHOD>(instance));
2891     }
2892
2905     template <IrisErrorCode (*FUNC)(const std::string&)>
2906     void setLoadImageFileDelegate()
2907     {
2908         setLoadImageFileDelegate(ImageLoadFileDelegate::make<FUNC>());
2909     }
2910
2935     void setLoadImageDataDelegate(ImageLoadDataDelegate delegate = ImageLoadDataDelegate())
2936     {
2937         inst_image->setLoadImageDataDelegate(delegate);
2938     }
2939
2960     template <typename T, IrisErrorCode (T::*METHOD)(const std::vector<uint64_t>&, uint64_t)>
2961     void setLoadImageDataDelegate(T* instance)
2962     {
2963         setLoadImageDataDelegate(ImageLoadDataDelegate::make<T, METHOD>(instance));
2964     }
2965
2978     template <IrisErrorCode (*FUNC)(const std::vector<uint64_t>&, uint64_t)>
2979     void setLoadImageDataDelegate()
2980     {
2981         setLoadImageDataDelegate(ImageLoadDataDelegate::make<FUNC>());
2982     }
2983
2999     uint64_t openImage(const std::string& filename)
3000     {
3001         return inst_image_cb->openImage(filename);
3002     }
3003
3038     void setRemainingStepSetDelegate(RemainingStepSetDelegate delegate = RemainingStepSetDelegate())
3039     {
3040         inst_step->setRemainingStepSetDelegate(delegate);
3041     }
3042
3067     void setRemainingStepGetDelegate(RemainingStepGetDelegate delegate)
3068     {
3069         inst_step->setRemainingStepGetDelegate(delegate);
3070     }
3071
3092     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, const std::string&)>
3093     void setRemainingStepSetDelegate(T* instance)
3094     {
3095         setRemainingStepSetDelegate(RemainingStepSetDelegate::make<T, METHOD>(instance));
```

```
3096     }
3097
3118     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t&, const std::string&)>
3119     void setRemainingStepGetDelegate(T* instance)
3120     {
3121         setRemainingStepGetDelegate(RemainingStepGetDelegate::make<T, METHOD>(instance));
3122     }
3123
3136     template <IrisErrorCode (*FUNC)(uint64_t, const std::string&)>
3137     void setRemainingStepSetDelegate()
3138     {
3139         setRemainingStepSetDelegate(RemainingStepSetDelegate::make<FUNC>());
3140     }
3141
3154     template <IrisErrorCode (*FUNC)(uint64_t&, const std::string&)>
3155     void setRemainingStepGetDelegate()
3156     {
3157         setRemainingStepGetDelegate(RemainingStepGetDelegate::make<FUNC>());
3158     }
3159
3184     //
3185     void setStepCountGetDelegate(StepCountGetDelegate delegate = StepCountGetDelegate())
3186     {
3187         inst_step->setStepCountGetDelegate(delegate);
3188     }
3189
3210     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t&, const std::string&)>
3211     void setStepCountGetDelegate(T* instance)
3212     {
3213         setStepCountGetDelegate(RemainingStepGetDelegate::make<T, METHOD>(instance));
3214     }
3215
3228     template <IrisErrorCode (*FUNC)(uint64_t&, const std::string&)>
3229     void setStepCountGetDelegate()
3230     {
3231         setStepCountGetDelegate(RemainingStepGetDelegate::make<FUNC>());
3232     }
3233
3238     /*
3239      * @brief exec_apis IrisInstanceBuilder per-instance execution APIs
3240      * @{
3241      */
3242
3267     void setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate delegate =
    PerInstanceExecutionStateSetDelegate())
3268     {
3269         inst_per_inst_exec->setExecutionStateSetDelegate(delegate);
3270     }
3271
3292     template <typename T, IrisErrorCode (T::*METHOD)(bool)>
3293     void setExecutionStateSetDelegate(T* instance)
3294     {
3295         setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate::make<T, METHOD>(instance));
3296     }
3297
3310     template <IrisErrorCode (*FUNC)(bool)>
3311     void setExecutionStateSetDelegate()
3312     {
3313         setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate::make<FUNC>());
3314     }
3315
3340     void setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate delegate)
3341     {
3342         inst_per_inst_exec->setExecutionStateGetDelegate(delegate);
3343     }
3344
3365     template <typename T, IrisErrorCode (T::*METHOD)(bool&)>
3366     void setExecutionStateGetDelegate(T* instance)
3367     {
3368         setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate::make<T, METHOD>(instance));
3369     }
3370
3383     template <IrisErrorCode (*FUNC)(bool&)>
3384     void setExecutionStateGetDelegate()
3385     {
3386         setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate::make<FUNC>());
3387     }
3388
3393     /*
3394      * @brief table_apis IrisInstanceBuilder table APIs
3395      * @{
3396      */
3397
3398     class TableColumnBuilder;
3399
3403     class TableBuilder
3404     {
```

```
3405    private:
3406        IrisInstanceTable::TableInfoAndAccess& info;
3407
3408    public:
3409        TableBuilder(IrisInstanceTable::TableInfoAndAccess& info_)
3410            : info(info_)
3411        {
3412        }
3413
3419        TableBuilder& setName(const std::string& name)
3420        {
3421            info.tableInfo.name = name;
3422            return *this;
3423        }
3424
3430        TableBuilder& setDescription(const std::string& description)
3431        {
3432            info.tableInfo.description = description;
3433            return *this;
3434        }
3435
3441        TableBuilder& setMinIndex(uint64_t minIndex)
3442        {
3443            info.tableInfo.minIndex = minIndex;
3444            return *this;
3445        }
3446
3452        TableBuilder& setMaxIndex(uint64_t maxIndex)
3453        {
3454            info.tableInfo.maxIndex = maxIndex;
3455            return *this;
3456        }
3457
3463        TableBuilder& setIndexFormatHint(const std::string& hint)
3464        {
3465            info.tableInfo.indexFormatHint = hint;
3466            return *this;
3467        }
3468
3474        TableBuilder& setFormatShort(const std::string& format)
3475        {
3476            info.tableInfo.formatShort = format;
3477            return *this;
3478        }
3479
3485        TableBuilder& setFormatLong(const std::string& format)
3486        {
3487            info.tableInfo.formatLong = format;
3488            return *this;
3489        }
3490
3500        TableBuilder& setReadDelegate(TableReadDelegate delegate)
3501        {
3502            info.readDelegate = delegate;
3503            return *this;
3504        }
3505
3515        TableBuilder& setWriteDelegate(TableWriteDelegate delegate)
3516        {
3517            info.writeDelegate = delegate;
3518            return *this;
3519        }
3520
3532        template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, uint64_t, uint64_t,
    TableReadResult&)>
3533        TableBuilder& setReadDelegate(T* instance)
3534        {
3535            return setReadDelegate(TableReadDelegate::make<T, METHOD>(instance));
3536        }
3537
3549        template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, const TableRecords&,
    TableWriteResult&)>
3550        TableBuilder& setWriteDelegate(T* instance)
3551        {
3552            return setWriteDelegate(TableWriteDelegate::make<T, METHOD>(instance));
3553        }
3554
3564        template <IrisErrorCode (*FUNC)(const TableInfo&, uint64_t, uint64_t, TableReadResult&)>
3565        TableBuilder& setReadDelegate()
3566        {
3567            return setReadDelegate(TableReadDelegate::make<FUNC>());
3568        }
3569
3579        template <IrisErrorCode (*FUNC)(const TableInfo&, const TableRecords&, TableWriteResult&)>
3580        TableBuilder& setWriteDelegate()
3581        {
3582            return setWriteDelegate(TableWriteDelegate::make<FUNC>());
```

```
3583         }
3584
3595         TableBuilder& addColumnInfo(const TableColumnInfo& columnInfo)
3596         {
3597             info.tableInfo.columns.push_back(columnInfo);
3598             return *this;
3599         }
3600
3612         TableColumnBuilder addColumn(const std::string& name);
3613     };
3614
3618     class TableColumnBuilder
3619     {
3620     private:
3621         TableBuilder&    parent;
3622         TableColumnInfo& info;
3623
3624     public:
3625         TableColumnBuilder(TableBuilder& parent_, TableColumnInfo& info_)
3626             : parent(parent_)
3627             , info(info_)
3628         {
3629         }
3630
3640         TableBuilder& addColumnInfo(const TableColumnInfo& columnInfo)
3641         {
3642             return parent.addColumnInfo(columnInfo);
3643         }
3644
3656         TableColumnBuilder addColumn(const std::string& name) { return parent.addColumn(name); }
3657
3666         TableBuilder& endColumn()
3667         {
3668             return parent;
3669         }
3670
3677         TableColumnBuilder& setName(const std::string& name)
3678         {
3679             info.name = name;
3680             return *this;
3681         }
3682
3689         TableColumnBuilder& setDescription(const std::string& description)
3690         {
3691             info.description = description;
3692             return *this;
3693         }
3694
3701         TableColumnBuilder& setFormat(const std::string& format)
3702         {
3703             info.format = format;
3704             return *this;
3705         }
3706
3713         TableColumnBuilder& setType(const std::string& type)
3714         {
3715             info.type = type;
3716             return *this;
3717         }
3718
3725         TableColumnBuilder& setBitWidth(uint64_t bitWidth)
3726         {
3727             info.bitWidth = bitWidth;
3728             return *this;
3729         }
3730
3737         TableColumnBuilder& setFormatShort(const std::string& format)
3738         {
3739             info.formatShort = format;
3740             return *this;
3741         }
3742
3749         TableColumnBuilder& setFormatLong(const std::string& format)
3750         {
3751             info.formatLong = format;
3752             return *this;
3753         }
3754
3761         TableColumnBuilder& setRwMode(const std::string& rwMode)
3762         {
3763             info.rwMode = rwMode;
3764             return *this;
3765         }
3766     };
3767
3790     TableBuilder addTable(const std::string& name)
3791     {
```

```
3792         return TableBuilder(inst_table->addTableInfo(name));
3793     }
3794
3825     void setDefaultTableReadDelegate(TableReadDelegate delegate = TableReadDelegate())
3826     {
3827         inst_table->setDefaultReadDelegate(delegate);
3828     }
3829
3861     void setDefaultTableWriteDelegate(TableWriteDelegate delegate = TableWriteDelegate())
3862     {
3863         inst_table->setDefaultWriteDelegate(delegate);
3864     }
3865
3892     template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, uint64_t, uint64_t,
    TableReadResult&)>
3893     void setDefaultTableReadDelegate(T* instance)
3894     {
3895         setDefaultTableReadDelegate(TableReadDelegate::make<T, METHOD>(instance));
3896     }
3897
3925     template <typename T, IrisErrorCode (T::*METHOD)(const TableInfo&, const TableRecords&,
    TableWriteResult&)>
3926     void setDefaultTableWriteDelegate(T* instance)
3927     {
3928         setDefaultTableWriteDelegate(TableWriteDelegate::make<T, METHOD>(instance));
3929     }
3930
3949     template <IrisErrorCode (*FUNC)(const TableInfo&, uint64_t, uint64_t, TableReadResult&)>
3950     void setDefaultTableReadDelegate()
3951     {
3952         setDefaultTableReadDelegate(TableReadDelegate::make<FUNC>());
3953     }
3954
3974     template <IrisErrorCode (*FUNC)(const TableInfo&, const TableRecords&, TableWriteResult&)>
3975     void setDefaultTableWriteDelegate()
3976     {
3977         setDefaultTableWriteDelegate(TableWriteDelegate::make<FUNC>());
3978     }
3979
3990     void setGetCurrentDisassemblyModeDelegate(GetCurrentDisassemblyModeDelegate delegate)
3991     {
3992         inst_disass->setGetCurrentModeDelegate(delegate);
3993     }
3994
3995     template <typename T, IrisErrorCode (T::*METHOD)(std::string&)>
3996     void setGetCurrentDisassemblyModeDelegate(T* instance)
3997     {
3998         setGetCurrentDisassemblyModeDelegate(GetCurrentDisassemblyModeDelegate::make<T,
    METHOD>(instance));
3999     }
4000
4002     void setGetDisassemblyDelegate(GetDisassemblyDelegate delegate)
4003     {
4004         inst_disass->setGetDisassemblyDelegate(delegate);
4005     }
4006
4007     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t, const std::string&, MemoryReadResult&,
    uint64_t, uint64_t, std::vector<DisassemblyLine>&)>
4008     void setGetDisassemblyDelegate(T* instance)
4009     {
4010         setGetDisassemblyDelegate(GetDisassemblyDelegate::make<T, METHOD>(instance));
4011     }
4012
4013     template <IrisErrorCode (*FUNC)(uint64_t, const std::string&, MemoryReadResult&,
4014                                     uint64_t, uint64_t, std::vector<DisassemblyLine>&)>
4015     void setGetDisassemblyDelegate()
4016     {
4017         setGetDisassemblyDelegate(GetDisassemblyDelegate::make<FUNC>());
4018     }
4019
4021     void setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate delegate)
4022     {
4023         inst_disass->setDisassembleOpcodeDelegate(delegate);
4024     }
4025
4026     template <typename T, IrisErrorCode (T::*METHOD)(const std::vector<uint64_t>&, uint64_t, const
    std::string&, DisassembleContext&, DisassemblyLine&)>
4027     void setDisassembleOpcodeDelegate(T* instance)
4028     {
4029         setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate::make<T, METHOD>(instance));
4030     }
4031
4032     template <IrisErrorCode (*FUNC)(const std::vector<uint64_t>&, uint64_t, const std::string&,
4033                                     DisassembleContext&, DisassemblyLine&)>
4034     void setDisassembleOpcodeDelegate()
4035     {
4036         setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate::make<FUNC>());
```

```
4037     }
4038
4040     void addDisassemblyMode(const std::string& name, const std::string& description)
4041     {
4042         inst_disass->addDisassemblyMode(name, description);
4043     }
4044
4078     void setDbgStateSetRequestDelegate(DebuggableStateSetRequestDelegate delegate =
     DebuggableStateSetRequestDelegate())
4079     {
4080         inst_dbg_state->setSetRequestDelegate(delegate);
4081     }
4082
4103     template <typename T, IrisErrorCode (T::*METHOD)(bool)>
4104     void setDbgStateSetRequestDelegate(T* instance)
4105     {
4106         setDbgStateSetRequestDelegate(DebuggableStateSetRequestDelegate::make<T, METHOD>(instance));
4107     }
4108
4121     template <IrisErrorCode (*FUNC)(bool)>
4122     void setDbgStateSetRequestDelegate()
4123     {
4124         setDbgStateSetRequestDelegate(DebuggableStateSetRequestDelegate::make<FUNC>());
4125     }
4126
4151     void setDbgStateGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate delegate =
     DebuggableStateGetAcknowledgeDelegate())
4152     {
4153         inst_dbg_state->setGetAcknowledgeDelegate(delegate);
4154     }
4155
4176     template <typename T, IrisErrorCode (T::*METHOD)(bool&)>
4177     void setDbgStateGetAcknowledgeDelegate(T* instance)
4178     {
4179         setDbgStateGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate::make<T,
     METHOD>(instance));
4180     }
4181
4194     template <IrisErrorCode (*FUNC)(bool&)>
4195     void setDbgStateGetAcknowledgeDelegate()
4196     {
4197         setDbgStateGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate::make<FUNC>());
4198     }
4199
4227     template <typename T, IrisErrorCode (T::*SET_REQUEST)(bool), IrisErrorCode
     (T::*GET_ACKNOWLEDGE)(bool&)>
4228     void setDbgStateDelegates(T* instance)
4229     {
4230         setDbgStateSetRequestDelegate<T, SET_REQUEST>(instance);
4231         setDbgStateGetAcknowledgeDelegate<T, GET_ACKNOWLEDGE>(instance);
4232     }
4233
4235     void setCheckpointSaveDelegate(CheckpointSaveDelegate delegate = CheckpointSaveDelegate())
4236     {
4237         inst_checkpoint->setCheckpointSaveDelegate(delegate);
4238     }
4239
4240     template <typename T, IrisErrorCode (T::*METHOD)(const std::string&)>
4241     void setCheckpointSaveDelegate(T* instance)
4242     {
4243         setCheckpointSaveDelegate(CheckpointSaveDelegate::make<T, METHOD>(instance));
4244     }
4245
4246     void setCheckpointRestoreDelegate(CheckpointRestoreDelegate delegate = CheckpointRestoreDelegate())
4247     {
4248         inst_checkpoint->setCheckpointRestoreDelegate(delegate);
4249     }
4250
4251     template <typename T, IrisErrorCode (T::*METHOD)(const std::string&)>
4252     void setCheckpointRestoreDelegate(T* instance)
4253     {
4254         setCheckpointRestoreDelegate(CheckpointRestoreDelegate::make<T, METHOD>(instance));
4255     }
4256
4269     class SemihostingManager
4270     {
4271     private:
4272         IrisInstanceSemihosting* inst_semihost;
4273
4274     public:
4275         SemihostingManager(IrisInstanceSemihosting* inst_semihost_)
4276             : inst_semihost(inst_semihost_)
4277         {
4278         }
4279
4280         ~SemihostingManager()
4281         {
```

```
4282             // Interrupt any requests that are currently blocked
4283             unblock();
4284         }
4285 
4290         void enableExtensions()
4291         {
4292             inst_semihost->enableExtensions();
4293         }
4294 
4309         std::vector<uint8_t> readData(uint64_t fDes, size_t max_size = 0, uint64_t flags =
     semihost::DEFAULT)
4310         {
4311             return inst_semihost->readData(fDes, max_size, flags);
4312         }
4313 
4314         /*
4315          * @brief Write data for a given file descriptor
4316          *
4317          * @param   fDes        File descriptor to write to. Usually semihost::STDOUT or
     semihost::STDERR.
4318          * @param   data        Buffer containing the data to write.
4319          * @param   size        Size of the data buffer in bytes.
4320          * @return              Returns false if no client is registered for IRIS_SEMIHOSTING_OUTPUT
     events.
4321          */
4322         bool writeData(uint64_t fDes, const uint8_t* data, size_t size)
4323         {
4324             return inst_semihost->writeData(fDes, data, size);
4325         }
4326 
4327         /*
4328          * @brief Write data for a given file descriptor
4329          *
4330          * @param   fDes        File descriptor to write to. Usually semihost::STDOUT or
     semihost::STDERR.
4331          * @param   data        Buffer containing the data to write.
4332          * @return              Returns false if no client is registered for IRIS_SEMIHOSTING_OUTPUT
     events.
4333          */
4334         bool writeData(uint64_t fDes, const std::vector<uint8_t>& data)
4335         {
4336             return writeData(fDes, &data.front(), data.size());
4337         }
4338 
4353         std::pair<bool, uint64_t> semihostedCall(uint64_t operation, uint64_t parameter)
4354         {
4355             return inst_semihost->semihostedCall(operation, parameter);
4356         }
4357 
4358         /*
4359          * @brief Request premature exit from any blocking requests that are currently blocked.
4360          */
4361         void unblock()
4362         {
4363             return inst_semihost->unblock();
4364         }
4365     };
4366 
4374     SemihostingManager enableSemihostingAndGetManager()
4375     {
4376         inst_semihost.init();
4377         return SemihostingManager(inst_semihost);
4378     }
4379 
4383 };
4384 
4385 inline IrisInstanceBuilder::TableColumnBuilder IrisInstanceBuilder::TableBuilder::addColumn(const
     std::string& name)
4386 {
4387     // Add a new column with default info
4388     info.tableInfo.columns.resize(info.tableInfo.columns.size() + 1);
4389     TableColumnInfo& col = info.tableInfo.columns.back();
4390 
4391     col.name = name;
4392 
4393     return TableColumnBuilder(*this, col);
4394 }
4395 
4396 NAMESPACE_IRIS_END
4397 
4398 #endif // ARM_INCLUDE_IrisInstanceBuilder_h
```

9.19 IrisInstanceCheckpoint.h File Reference

Checkpoint add-on to IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
```

Classes

- class iris::IrisInstanceCheckpoint

  Checkpoint add-on for IrisInstance.

Typedefs

- typedef IrisDelegate< const std::string & > iris::CheckpointRestoreDelegate

  Restore the checkpoint corresponding to the given information.
- typedef IrisDelegate< const std::string & > iris::CheckpointSaveDelegate

  Save a checkpoint corresponding to the given information.

9.19.1 Detailed Description

Checkpoint add-on to IrisInstance.

Date

Copyright ARM Limited 2019 All Rights Reserved.

9.19.2 Typedef Documentation

9.19.2.1 CheckpointRestoreDelegate

```
typedef IrisDelegate<const std::string&> iris::CheckpointRestoreDelegate
```

Restore the checkpoint corresponding to the given information.

```
IrisErrorCode checkpoint_restore(const std::string & checkpoint_dir)
```

Error: Return E_* error code if it failed to restore the checkpoint.

9.19.2.2 CheckpointSaveDelegate

```
typedef IrisDelegate<const std::string&> iris::CheckpointSaveDelegate
```

Save a checkpoint corresponding to the given information.

```
IrisErrorCode checkpoint_save(const std::string & checkpoint_dir)
```

Error: Return E_* error code if it failed to save the checkpoint.

9.20 IrisInstanceCheckpoint.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisInstanceCheckpoint_h
8 #define ARM_INCLUDE_IrisInstanceCheckpoint_h
9
10 #include "iris/detail/IrisCommon.h"
11 #include "iris/detail/IrisDelegate.h"
12
13 NAMESPACE_IRIS_START
14
15 class IrisInstance;
16 class IrisReceivedRequest;
17
26 typedef IrisDelegate<const std::string&> CheckpointSaveDelegate;
27
36 typedef IrisDelegate<const std::string&> CheckpointRestoreDelegate;
37
```

```
41 class IrisInstanceCheckpoint
42 {
43
44 public:
45     IrisInstanceCheckpoint(IrisInstance* iris_instance = nullptr);
46
54     void attachTo(IrisInstance* iris_instance_);
55
61     void setCheckpointSaveDelegate(CheckpointSaveDelegate delegate);
62
68     void setCheckpointRestoreDelegate(CheckpointRestoreDelegate delegate);
69
70 private:
71     void impl_checkpoint_save(IrisReceivedRequest& request);
72
73     void impl_checkpoint_restore(IrisReceivedRequest& request);
74
76
78     IrisInstance* iris_instance;
79
81     CheckpointSaveDelegate save_delegate;
82
84     CheckpointRestoreDelegate restore_delegate;
85 };
86
87 NAMESPACE_IRIS_END
88
89 #endif // #ifndef ARM_INCLUDE_IrisInstanceCheckpoint_h
```

9.21 IrisInstanceDebuggableState.h File Reference

IrisInstance add-on to implement debuggableState functions.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
```

Classes

- class iris::IrisInstanceDebuggableState

  Debuggable-state add-on for IrisInstance.

Typedefs

- typedef IrisDelegate< bool & > iris::DebuggableStateGetAcknowledgeDelegate

  Interface to stop the simulation time progress.
- typedef IrisDelegate< bool > iris::DebuggableStateSetRequestDelegate

  Delegate to set the debuggable-state-request flag.

9.21.1 Detailed Description

IrisInstance add-on to implement debuggableState functions.

Copyright

9.21.2 Typedef Documentation

9.21.2.1 DebuggableStateGetAcknowledgeDelegate

```
typedef IrisDelegate<bool&> iris::DebuggableStateGetAcknowledgeDelegate
```

Interface to stop the simulation time progress.

```
IrisErrorCode getAcknowledge(bool &acknowledge_out);
```

9.21.2.2 DebuggableStateSetRequestDelegate

```
typedef IrisDelegate<bool> iris::DebuggableStateSetRequestDelegate
```

Delegate to set the debuggable-state-request flag.

```
IrisErrorCode setRequest(bool request);
```

9.22 IrisInstanceDebuggableState.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisInstanceDebuggableState_h
9 #define ARM_INCLUDE_IrisInstanceDebuggableState_h
10
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisDelegate.h"
13
14 NAMESPACE_IRIS_START
15
22 typedef IrisDelegate<bool> DebuggableStateSetRequestDelegate;
23
30 typedef IrisDelegate<bool&> DebuggableStateGetAcknowledgeDelegate;
31
32 class IrisInstance;
33 class IrisReceivedRequest;
34
38 class IrisInstanceDebuggableState
39 {
40 private:
42     IrisInstance* iris_instance;
43
45     DebuggableStateSetRequestDelegate     setRequest;
46     DebuggableStateGetAcknowledgeDelegate getAcknowledge;
47
48 public:
49     IrisInstanceDebuggableState(IrisInstance* iris_instance = nullptr);
50
56     void attachTo(IrisInstance* irisInstance);
57
65     void setSetRequestDelegate(DebuggableStateSetRequestDelegate delegate)
66     {
67         setRequest = delegate;
68     }
69
77     void setGetAcknowledgeDelegate(DebuggableStateGetAcknowledgeDelegate delegate)
78     {
79         getAcknowledge = delegate;
80     }
81
82 private:
84     void impl_debuggableState_setRequest(IrisReceivedRequest& request);
85
87     void impl_debuggableState_getAcknowledge(IrisReceivedRequest& request);
88 };
89
90 NAMESPACE_IRIS_END
91
92 #endif // ARM_INCLUDE_IrisInstanceSimulationTime_h
```

9.23 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.

9.23.1 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.

9.24 IrisInstanceDisassembler.h

Go to the documentation of this file.

```
1
9 #ifndef ARM_INCLUDE_IrisInstanceDisassembler_h
10 #define ARM_INCLUDE_IrisInstanceDisassembler_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisLogger.h"
15 #include "iris/detail/IrisObjects.h"
16
17 #include <cstdio>
18
19 NAMESPACE_IRIS_START
20
21 class IrisInstance;
22 class IrisReceivedRequest;
23
39 typedef IrisDelegate<std::string&> GetCurrentDisassemblyModeDelegate;
40
51 typedef IrisDelegate<uint64_t, const std::string&, MemoryReadResult&,
52                      uint64_t, uint64_t, std::vector<DisassemblyLine>&>
53     GetDisassemblyDelegate;
54
65 typedef IrisDelegate<const std::vector<uint64_t>&, uint64_t, const std::string&,
66                      DisassembleContext&, DisassemblyLine&>
67     DisassembleOpcodeDelegate;
68
69 /*
70  * @}
71  */
72
90 class IrisInstanceDisassembler
91 {
92 public:
98     IrisInstanceDisassembler(IrisInstance* irisInstance = nullptr);
99
105     void attachTo(IrisInstance* irisInstance);
106
114     void setGetCurrentModeDelegate(GetCurrentDisassemblyModeDelegate delegate)
115     {
116         getCurrentMode = delegate;
117     }
118
126     void setGetDisassemblyDelegate(GetDisassemblyDelegate delegate)
127     {
128         getDisassembly = delegate;
129     }
130
138     void setDisassembleOpcodeDelegate(DisassembleOpcodeDelegate delegate)
139     {
140         disassembleOpcode = delegate;
141     }
142
```

```
152     void addDisassemblyMode(const std::string& name, const std::string& description);
153
154 private:
155     void impl_disassembler_getModes(IrisReceivedRequest& request);
156
157     void impl_disassembler_getCurrentMode(IrisReceivedRequest& request);
158
159     void impl_disassembler_getDisassembly(IrisReceivedRequest& request);
160
161     void impl_disassembler_disassembleOpcode(IrisReceivedRequest& request);
162
165     void checkDisassemblyMode(std::string& mode, bool& isValidMode);
166
168
170     IrisInstance* irisInstance;
171
173     GetCurrentDisassemblyModeDelegate getCurrentMode;
174
176     GetDisassemblyDelegate getDisassembly;
177
179     DisassembleOpcodeDelegate disassembleOpcode;
180
181     std::vector<DisassemblyMode> disassemblyModes;
183     IrisLogger log;
184 };
185
186 NAMESPACE_IRIS_END
187
188 #endif // #ifndef ARM_INCLUDE_IrisInstanceDisassembler_h
```

9.25 IrisInstanceEvent.h File Reference

Event add-on to IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisRequest.h"
#include <cstdio>
#include <set>
```

Classes

- struct iris::IrisInstanceEvent::EventSourceInfoAndDelegate

  Contains the metadata and delegates for a single EventSource.
- class iris::EventStream

  Base class for event streams.
- class iris::IrisEventRegistry

  Class to register Iris event streams for an event.
- class iris::IrisEventStream

  Event stream class for Iris-specific events.
- class iris::IrisInstanceEvent

  Event add-on for IrisInstance.
- struct iris::IrisInstanceEvent::ProxyEventInfo

  Contains information for a single proxy EventSource.

Typedefs

- typedef IrisDelegate< EventStream *&, const EventSourceInfo &, const std::vector< std::string > & > iris::EventStreamCreateDelegate

  Delegate to create an EventStream.

9.25.1 Detailed Description

Event add-on to IrisInstance.

Copyright

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.

9.25.2 Typedef Documentation

9.25.2.1 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.

9.26 IrisInstanceEvent.h

Go to the documentation of this file.

```
1
12 #ifndef ARM_INCLUDE_IrisInstanceEvent_h
13 #define ARM_INCLUDE_IrisInstanceEvent_h
14
15 #include "iris/detail/IrisCommon.h"
16 #include "iris/detail/IrisDelegate.h"
17 #include "iris/detail/IrisLogger.h"
18 #include "iris/detail/IrisObjects.h"
19 #include "iris/detail/IrisRequest.h"
20
21 #include <cstdio>
22 #include <set>
23
24 NAMESPACE_IRIS_START
25
26 class IrisInstance;
27 class IrisReceivedRequest;
28
29 class EventStream;
30 class IrisEventRegistry;
31
44 typedef IrisDelegate<EventStream*&, const EventSourceInfo&, const std::vector<std::string>&>
    EventStreamCreateDelegate;
45
63 class IrisInstanceEvent
64 {
65 public:
66
67     /* ! What is a proxy event source?
68        - The event source in actual does not belong to this Iris instance, but instead belongs to another
    Iris instance (target).
69        - The event source is registered as a proxy in this Iris instance using Iris interface -
    event_registerProxyEventSource()
70        - This Iris instance acts as a proxy for those registered events.
71        - All interface calls (for example, eventStream_create) on the proxy event source are forwarded to
    the target instance.
72        - Similarly, all the created event streams in this Iris instance for the proxy event source are
    tagged as proxyForOtherInstance
73        - All the interface calls (for example, eventStream_enable) on such proxy event streams are
    forwarded to the target instance.
```

```
74        - Finally, the proxy event source can be deregistered using Iris interface -
      event_unregisterProxyEventSource()
75      */
76
80      struct ProxyEventInfo
81      {
82          InstanceId targetInstId{};      //target Iris instance Id
83          EventSourceId targetEvSrcId{};  //event source ID in target Iris instance
84          std::vector<EventStreamId> evStreamIds; //list of created event stream IDs
85          //Important note: When we create an event stream, we use the same esID for both - this and target
      Iris instance
86      };
87
91      struct EventSourceInfoAndDelegate
92      {
93          EventSourceInfo           info;
94          EventStreamCreateDelegate createEventStream;
95
96          bool isValid{true}; //deleteEventSource() sets isValid to false
97          bool isProxy{false};
98          ProxyEventInfo proxyEventInfo; //contains proper values only if isProxy=true
99      };
100
106     IrisInstanceEvent(IrisInstance* irisInstance = nullptr);
107     ~IrisInstanceEvent();
108
116     void attachTo(IrisInstance* irisInstance);
117
125     void setDefaultEsCreateDelegate(EventStreamCreateDelegate delegate);
126
139     EventSourceInfoAndDelegate& addEventSource(const std::string& name, bool isHidden = false);
140
148     uint64_t addEventSource(const EventSourceInfoAndDelegate& info);
149
155     void deleteEventSource(const std::string& eventName);
156
164     const uint64_t *eventBufferGetSyncStepResponse(EventBufferId evBufId, RequestId requestId);
165
174     void eventBufferClear(EventBufferId evBufId);
175
183     bool isValidEvBufId(EventBufferId evBufId) const;
184
185 private:
186     // --- Iris function implementations ---
187
188     void impl_event_getEventSources(IrisReceivedRequest& request);
189
190     void impl_event_getEventSource(IrisReceivedRequest& request);
191
192     void impl_eventStream_create(IrisReceivedRequest& request);
193
194     void impl_eventStream_destroy(IrisReceivedRequest& request);
195
196     void impl_eventStream_enable(IrisReceivedRequest& request);
197
198     void impl_eventStream_disable(IrisReceivedRequest& request);
199
200     void impl_eventStream_getCounter(IrisReceivedRequest& request);
201
202     void impl_eventStream_setTraceRanges(IrisReceivedRequest& request);
203
204     void impl_eventStream_getState(IrisReceivedRequest& request);
205
206     void impl_eventStream_flush(IrisReceivedRequest& request);
207
208     void impl_eventStream_setOptions(IrisReceivedRequest& request);
209
210     void impl_eventStream_action(IrisReceivedRequest& request);
211
212     void impl_eventBuffer_create(IrisReceivedRequest& request);
213
214     void impl_eventBuffer_flush(IrisReceivedRequest& request);
215
216     void impl_eventBuffer_destroy(IrisReceivedRequest& request);
217
218     void impl_ec_eventBuffer(IrisReceivedRequest& request);
219
220     void register_ec_IRIS_INSTANCE_REGISTRY_CHANGED();
221     IrisErrorCode ec_IRIS_INSTANCE_REGISTRY_CHANGED(EventStreamId esId, const IrisValueMap& fields,
      uint64_t time,
222                                                     InstanceId sInstId, bool syncEc, std::string&
      errorMessageOut);
223
225
226     void impl_event_registerProxyEventSource(IrisReceivedRequest& request);
227
228     void impl_event_unregisterProxyEventSource(IrisReceivedRequest& request);
```

```
229
230     void impl_eventStream_create_proxy(IrisReceivedRequest& request);
231
232     IrisErrorCode impl_eventStream_destroy_target(IrisReceivedRequest& request, EventStream* evStream);
233
234     void impl_eventStream_enable_proxy(IrisReceivedRequest& request, EventStream* evStream);
235
236     void impl_eventStream_disable_proxy(IrisReceivedRequest& request, EventStream* evStream);
237
238     void impl_eventStream_getCounter_proxy(IrisReceivedRequest& request, EventStream* evStream);
239
240     void impl_eventStream_setTraceRanges_proxy(IrisReceivedRequest& request, EventStream* evStream);
241
242     void impl_eventStream_getState_proxy(IrisReceivedRequest& request, EventStream* evStream);
243
244     void impl_eventStream_flush_proxy(IrisReceivedRequest& request, EventStream* evStream);
245
246     void impl_eventStream_setOptions_proxy(IrisReceivedRequest& request, EventStream* evStream);
247
248     void impl_eventStream_action_proxy(IrisReceivedRequest& request, EventStream* evStream);
249
250     ProxyEventInfo& getProxyEventInfo(EventStream* evStream);
251
252     InstanceId getTargetInstId(EventStream* evStream);
253
255
256     EventStream* getEventStream(EventStreamId esId);
257
258     struct EventBufferStreamInfo;
259     struct EventBuffer;
260
263     const EventBufferStreamInfo* getEventBufferStreamInfo(InstanceId sInstId, EventStreamId esId) const;
264
266     EventBuffer* getEventBuffer(EventBufferId evBufId) const;
267
270     void eventBufferSend(EventBuffer *eventBuffer, bool flush);
271
273     void eventBufferDestroy(EventBufferId evBufId);
274
275     //Find a free event stream ID where a new EventStream can be added
276     //The returned ID is greater than or equal to 'minEsId'
277     EventStreamId findFreeEventStreamId(EventStreamId minEsId);
278
280
282     IrisInstance* irisInstance;
283
285     std::vector<EventSourceInfoAndDelegate> eventSources;
286
288     std::map<std::string, uint64_t>         srcNameToId;
289
292     std::vector<EventStream*> eventStreams;
293
296     std::vector<EventStreamId> freeEsIds;
297
299     EventStreamCreateDelegate defaultEsCreateDelegate;
300
302     IrisLogger log;
303
308     bool instance_registry_changed_registered{};
309
311     struct EventStreamOriginInfo
312     {
313         EventStreamId esId;
314         InstanceId sInstId;
315     };
316
318     struct EventBuffer
319     {
321         EventBuffer(const std::string& mode, uint64_t bufferSize, const std::string& ebcFunc, InstanceId
        ebcInstId, bool syncEbc, EventBufferId evBufId, IrisInstanceEvent *parent);
322
324         ~EventBuffer();
325
327         void clear();
328
330         const uint64_t* getResponse(RequestId requestId);
331
335         void getRequest(bool flush);
336
338         void addEventData(EventStreamInfoId esInfoId, uint64_t time, const uint64_t *fieldsU64Json);
339
341         std::string mode;
342
344         uint64_t bufferSizeU64{};
345
347         std::string ebcFunc;
348
```

```
350         InstanceId ebcInstId{IRIS_UINT64_MAX};
351
353         bool syncEbc{};
354
356         std::vector<EventStreamOriginInfo> eventStreams;
357
383         IrisU64JsonWriter writer;
384
386         uint64_t numEvents{};
387
389         size_t eventDataStartPos{};
390
393         IrisU64JsonWriter responseHeader;
394         size_t responseStartPos{};
395         size_t responseObjectPos{};
396         size_t responseArrayPos{};
397
400         IrisU64JsonWriter requestHeader;
401         size_t requestStartPos{};
402         size_t requestParamsPos{};
403         size_t requestReasonPos{};
404         size_t requestArrayPos{};
405
406         const uint64_t reasonSend  = 0x200000646E657304; // == "send"
407         const uint64_t reasonFlush = 0x20006873756C6605; // == "flush"
408
410         IrisInstanceEvent *parent{};
411     };
412     friend struct EventBuffer;
413
417     std::vector<EventBuffer*> eventBuffers;
418
421     std::vector<EventBufferId> freeEventBufferIds;
422
425     struct EventBufferStreamInfo
426     {
427         EventBuffer* eventBuffer{};
428         EventStreamInfoId esInfoId{};
429     };
430
438     std::vector<std::vector<EventBufferStreamInfo>> eventCallbackInfoToEventBufferStreamInfo;
439 };
440
446 class EventStream
447 {
448 public:
452     EventStream()
453         : enabled(false)
454         , req(nullptr)
455         , internal_req(nullptr)
456         , counter(false)
457         , isWaiting(false)
458         , selfReleaseAfterWaiting(false)
459     {
460     }
461
462     virtual ~EventStream() {}
463
475     void selfRelease()
476     {
477         // Disable the event stream if it is still enabled.
478         if (isEnabled())
479         {
480             disable();
481         }
482
483         // The request to destroy this event stream is nested and processed in the delegate to
484         // wait for the response, so it is not multi-threaded and no need to protect the variables.
485         if (!isWaiting)
486         {
487             delete this;
488             return;
489         }
490
491         // It is waiting for the response of the current request.
492         // Cancel the wait and release this object later (after the end of the wait).
493         req->cancel();
494         selfReleaseAfterWaiting = true;
495     }
496
507     virtual IrisErrorCode enable() = 0;
508
519     virtual IrisErrorCode disable() = 0;
520
530     virtual IrisErrorCode getState(IrisValueMap& fields)
531     {
532         (void) fields;
```

```
533         return E_not_supported_for_event_source;
534     }
535
545     virtual IrisErrorCode flush(RequestId requestId)
546     {
547         (void)requestId;
548         return E_not_supported_for_event_source;
549     }
550
568     virtual IrisErrorCode setOptions(const AttributeValueMap& options, bool eventStreamCreate,
        std::string& errorMessageOut)
569     {
570         (void)options;
571         (void)eventStreamCreate;
572         (void)errorMessageOut;
573
574         // Event streams which do not suuport options happily accept an empty options map.
575         return options.empty() ? E_ok : E_not_supported_for_event_source;
576     }
577
588     virtual IrisErrorCode action(const BreakpointAction& action_)
589     {
590         (void)action_;
591         return E_not_supported_for_event_source;
592     }
593
594     // Temporary: Keep PVModelLib happy. TODO: Remove.
595     virtual IrisErrorCode insertTrigger()
596     {
597         return E_not_supported_for_event_source;
598     }
599
600
601     // --- Functions for basic properties ---
602
618     void setProperties(IrisInstance* irisInstance, const EventSourceInfo* srcInfo,
619                        InstanceId ecInstId, const std::string& ecFunc, EventStreamId esId,
620                        bool syncEc);
621
627     bool isEnabled() const
628     {
629         return enabled;
630     }
631
637     EventStreamId getEsId() const
638     {
639         return esId;
640     }
641
647     const EventSourceInfo* getEventSourceInfo() const
648     {
649         return srcInfo;
650     }
651
658     InstanceId getEcInstId() const
659     {
660         return ecInstId;
661     }
662
663     // --- Functions for the counter mode ---
664
671     void setCounter(uint64_t startVal, const EventCounterMode& counterMode);
672
678     bool isCounter() const
679     {
680         return counter;
681     }
682
687     void setProxyForOtherInstance()
688     {
689         isProxyForOtherInstance = true;
690     }
691
697     bool IsProxyForOtherInstance() const
698     {
699         return isProxyForOtherInstance;
700     }
701
707     void setProxiedByInstanceId(InstanceId instId)
708     {
709         proxiedByInstanceId = instId;
710     }
711
717     bool IsProxiedByOtherInstance() const
718     {
719         return proxiedByInstanceId != IRIS_UINT64_MAX;
720     }
```

```
721
727     InstanceId getProxiedByInstanceId() const
728     {
729         return proxiedByInstanceId;
730     }
731
737     uint64_t getCountVal() const
738     {
739         return curVal;
740     }
741
742     // --- Functions for event stream with ranges
743
752     IrisErrorCode setRanges(const std::string& aspect, const std::vector<uint64_t>& ranges);
753
762     bool checkRangePc(uint64_t pc) const
763     {
764         return ranges.empty() || (aspect != ":pc") || checkRangesHelper(pc, ranges);
765     }
766
767     // --- Functions to emit the event callback ---
768     // Usage (example):
769     //     emitEventBegin(time, pc);    // Start to emit the callback.
770     //     addField(...);           // Add field value.
771     //     addField(...);           // Add field value.
772     //     ...
773     //     emitEventEnd();              // Emit the callback.
774
782     void emitEventBegin(IrisRequest& req, uint64_t time, uint64_t pc = IRIS_UINT64_MAX);
783
790     void emitEventBegin(uint64_t time, uint64_t pc = IRIS_UINT64_MAX);
791
801     void addField(const IrisU64StringConstant& field, uint64_t value)
802     {
803         addFieldRangeHelper(field, value);
804     }
805
815     void addField(const IrisU64StringConstant& field, int64_t value)
816     {
817         addFieldRangeHelper(field, value);
818     }
819
829     void addField(const IrisU64StringConstant& field, bool value)
830     {
831         addFieldRangeHelper(field, value);
832     }
833
843     template <class T>
844     void addField(const IrisU64StringConstant& field, const T& value)
845     {
846         fieldObj.member(field, value);
847     }
848
858     void addFieldSlow(const std::string& field, uint64_t value)
859     {
860         addFieldSlowRangeHelper(field, value);
861     }
862
872     void addFieldSlow(const std::string& field, int64_t value)
873     {
874         addFieldSlowRangeHelper(field, value);
875     }
876
886     void addFieldSlow(const std::string& field, bool value)
887     {
888         addFieldSlowRangeHelper(field, value);
889     }
890
900     template <class T>
901     void addFieldSlow(const std::string& field, const T& value)
902     {
903         fieldObj.memberSlow(field, value);
904     }
905
915     void emitEventEnd(bool send = true);
916
917 private:
919
923     bool counterTrigger();
924
926     bool checkRanges() const
927     {
928         return !aspectFound || checkRangesHelper(curAspectValue, ranges);
929     }
930
932     static bool checkRangesHelper(uint64_t value, const std::vector<uint64_t>& ranges);
933
```

```
935     template <typename T>
936     void addFieldRangeHelper(const IrisU64StringConstant& field, T value)
937     {
938         if (!aspect.empty() && aspect == toString(field))
939         {
940             aspectFound    = true;
941             curAspectValue = static_cast<uint64_t>(value);
942         }
943 
944         fieldObj.member(field, value);
945     }
946 
948     template <typename T>
949     void addFieldSlowRangeHelper(const std::string& field, T value)
950     {
951         if (aspect == field)
952         {
953             aspectFound    = true;
954             curAspectValue = static_cast<uint64_t>(value);
955         }
956 
957         fieldObj.memberSlow(field, value);
958     }
959 
960 protected:
962 
964     IrisInstance* irisInstance;
965 
967     const EventSourceInfo* srcInfo;
968 
970     InstanceId ecInstId;
971 
973     std::string ecFunc;
974 
976     EventStreamId esId;
977 
979     bool syncEc;
980 
982     bool enabled;
983 
985     IrisRequest*              req;
986     IrisRequest*              internal_req;
987     IrisU64JsonWriter::Object fieldObj;
988 
990 
992     bool counter;
993 
995     uint64_t startVal;
996     uint64_t curVal;
997 
999     EventCounterMode counterMode;
1000 
1002 
1003     std::string           aspect;
1004     std::vector<uint64_t> ranges;
1005 
1007     bool aspectFound;
1008 
1010     uint64_t curAspectValue;
1011 
1013     bool isProxyForOtherInstance{false};
1014 
1017     InstanceId proxiedByInstanceId{IRIS_UINT64_MAX};
1018 
1019 private:
1022     bool isWaiting;
1023 
1025     bool selfReleaseAfterWaiting;
1026 };
1027 
1031 class IrisEventStream : public EventStream
1032 {
1033 public:
1034     IrisEventStream(IrisEventRegistry* registry_);
1035 
1036     virtual IrisErrorCode enable() IRIS_OVERRIDE;
1037 
1038     virtual IrisErrorCode disable() IRIS_OVERRIDE;
1039 
1040 private:
1041     IrisEventRegistry* registry;
1042 };
1043 
1047 class IrisEventRegistry
1048 {
1049 public:
1055     bool empty() const
```

```
1056     {
1057         return esSet.empty();
1058     }
1059 
1066     bool registerEventStream(EventStream* evStream);
1067 
1074     bool unregisterEventStream(EventStream* evStream);
1075 
1076     // --- Functions to emit the callback of all registered event streams ---
1077     // Usage (example):
1078     //     emitEventBegin(time, pc);    // Start to emit the callback.
1079     //     addField(...);           // Add field value.
1080     //     addField(...);           // Add field value.
1081     //     ...
1082     //     emitEventEnd();              // Emit the callback.
1083 
1084     void emitEventBegin(uint64_t time, uint64_t pc = IRIS_UINT64_MAX) const;
1085 
1096     template <class T>
1097     void addField(const IrisU64StringConstant& field, const T& value) const
1098     {
1099         for (std::set<EventStream*>::const_iterator i = esSet.begin(), e = esSet.end(); i != e; i++)
1100             (*i)->addField(field, value);
1101     }
1102 
1113     template <class T>
1114     void addFieldSlow(const std::string& field, const T& value) const
1115     {
1116         for (std::set<EventStream*>::const_iterator i = esSet.begin(), e = esSet.end(); i != e; i++)
1117             (*i)->addFieldSlow(field, value);
1118     }
1119 
1125     void emitEventEnd() const;
1126 
1127     typedef std::set<EventStream*>::const_iterator iterator;
1128 
1136     iterator begin() const
1137     {
1138         return esSet.begin();
1139     }
1140 
1148     iterator end() const
1149     {
1150         return esSet.end();
1151     }
1152 
1153     ~IrisEventRegistry()
1154     {
1155         // Disable any remaining event streams.
1156         // Calling disable() on an EventStream will cause esSet to be modified so we need to loop
     without
1157         // using iterators which become invalidated.
1158         while (!esSet.empty())
1159         {
1160             (*esSet.begin())->disable();
1161         }
1162     }
1163 
1164 private:
1165     // All registered event streams
1166     std::set<EventStream*> esSet;
1167 };
1168 
1169 NAMESPACE_IRIS_END
1170 
1171 #endif // #ifndef ARM_INCLUDE_IrisInstanceBreakpoint_h
```

9.27 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.

9.27.1 Detailed Description

A helper class to build instantiation parameter metadata.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.28 IrisInstanceFactoryBuilder.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisInstanceFactoryBuilder_h
8 #define ARM_INCLUDE_IrisInstanceFactoryBuilder_h
9
10 #include "iris/IrisParameterBuilder.h"
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisObjects.h"
13
14 #include <string>
15 #include <vector>
16
17 NAMESPACE_IRIS_START
18
22 class IrisInstanceFactoryBuilder
23 {
24 private:
26     std::vector<ResourceInfo> parameters;
27
29     std::vector<ResourceInfo> hidden_parameters;
30
32     std::string parameter_prefix;
33
34     ResourceInfo& addParameterInternal(const std::string& name, uint64_t bitWidth, const std::string&
      description,
35                                        const std::string& type, bool hidden)
36     {
37         std::vector<ResourceInfo>& param_list = hidden ? hidden_parameters : parameters;
38         param_list.resize(parameters.size() + 1);
39         ResourceInfo& info = param_list.back();
40
41         info.name     = name;
42         info.bitWidth = bitWidth;
43         info.description    = description;
44         info.type     = type;
45
46         return info;
47     }
48
49 public:
55     IrisInstanceFactoryBuilder(const std::string& prefix)
56         : parameter_prefix(prefix)
57     {
58     }
59
68     IrisParameterBuilder addParameter(const std::string& name, uint64_t bitWidth, const std::string&
      description)
69     {
70         return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, bitWidth, description,
      "" /*numeric*/, false));
71     }
72
81     IrisParameterBuilder addHidenParameter(const std::string& name, uint64_t bitWidth, const std::string&
      description)
82     {
83         return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, bitWidth, description,
      "" /*numeric*/, true));
84     }
85
93     IrisParameterBuilder addStringParameter(const std::string& name, const std::string& description)
94     {
95         return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, 0, description,
      "string", false));
96     }
97
```

```
105     IrisParameterBuilder addHiddenStringParameter(const std::string& name, const std::string&
        description)
106     {
107         return IrisParameterBuilder(addParameterInternal(parameter_prefix + name, 0, description,
        "string", true));
108     }
109
120     IrisParameterBuilder addBooleanParameter(const std::string& name, const std::string& description)
121     {
122         ResourceInfo& info = addParameterInternal(parameter_prefix + name, 1, description, "numeric",
        false);
123
124         // Be explicit about the range even though there are only two possible values anyway.
125         info.parameterInfo.min.push_back(0);
126         info.parameterInfo.max.push_back(1);
127
128         // Add enum strings for the values
129         info.enums.push_back(EnumElementInfo(IrisValue(0), "false", ""));
130         info.enums.push_back(EnumElementInfo(IrisValue(1), "true", ""));
131
132         return IrisParameterBuilder(info);
133     }
134
145     IrisParameterBuilder addHiddenBooleanParameter(const std::string& name, const std::string&
        description)
146     {
147         ResourceInfo& info = addParameterInternal(parameter_prefix + name, 1, description, "numeric",
        true);
148
149         // Be explicit about the range even though there are only two possible values anyway.
150         info.parameterInfo.min.push_back(0);
151         info.parameterInfo.max.push_back(1);
152
153         // Add enum strings for the values
154         info.enums.push_back(EnumElementInfo(IrisValue(0), "false", ""));
155         info.enums.push_back(EnumElementInfo(IrisValue(1), "true", ""));
156
157         return IrisParameterBuilder(info);
158     }
159
166     const std::vector<ResourceInfo>& getParameterInfo() const
167     {
168         return parameters;
169     }
170
177     const std::vector<ResourceInfo>& getHiddenParameterInfo() const
178     {
179         return hidden_parameters;
180     }
181 };
182
183 NAMESPACE_IRIS_END
184
185 #endif // ARM_INCLUDE_IrisInstanceFactoryBuilder_h
```

9.29 IrisInstanceImage.h File Reference

Image-loading add-on to IrisInstance and image-loading callback add-on to the caller.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdio>
```

Classes

- class iris::IrisInstanceImage

  Image loading add-on for IrisInstance.
- class iris::IrisInstanceImage_Callback

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

Typedefs

- typedef IrisDelegate< const std::vector< 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.

9.29.1 Detailed Description

Image-loading add-on to IrisInstance and image-loading callback add-on to the caller.

Copyright

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.

9.29.2 Typedef Documentation

9.29.2.1 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.

9.29.2.2 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.

9.30 IrisInstanceImage.h

Go to the documentation of this file.

```
1
13 #ifndef ARM_INCLUDE_IrisInstanceImage_h
14 #define ARM_INCLUDE_IrisInstanceImage_h
15
16 #include "iris/detail/IrisCommon.h"
17 #include "iris/detail/IrisDelegate.h"
```

```
18 #include "iris/detail/IrisLogger.h"
19 #include "iris/detail/IrisObjects.h"
20
21 #include <cstdio>
22
23 NAMESPACE_IRIS_START
24
25 class IrisInstance;
26 class IrisReceivedRequest;
27
44 typedef IrisDelegate<const std::string&> ImageLoadFileDelegate;
45
61 typedef IrisDelegate<const std::vector<uint64_t>&, uint64_t> ImageLoadDataDelegate;
62
79 class IrisInstanceImage
80 {
81
82 public:
88     IrisInstanceImage(IrisInstance* irisInstance = 0);
89
95     void attachTo(IrisInstance* irisInstance);
96
102     void setLoadImageFileDelegate(ImageLoadFileDelegate delegate);
103
109     void setLoadImageDataDelegate(ImageLoadDataDelegate delegate);
110
119     static IrisErrorCode readFileData(const std::string& fileName, std::vector<uint64_t>& data,
      uint64_t& count);
120
121 private:
123     void loadImageFromData(IrisReceivedRequest& request, const ImageReadResult& imageData);
124
126
128     void impl_image_loadFile(IrisReceivedRequest& request);
129
131     void impl_image_loadData(IrisReceivedRequest& request);
132
134     void impl_image_loadDataPull(IrisReceivedRequest& request);
135
136     void impl_image_getMetaInfoList(IrisReceivedRequest& request);
137
138     void impl_image_clearMetaInfoList(IrisReceivedRequest& request);
139
141     void writeRawDataToMemory(IrisReceivedRequest& request, const std::vector<uint64_t>& data, uint64_t
      count, uint64_t rawAddr, MemorySpaceId rawSpaceId);
142
144     IrisErrorCode pullData(InstanceId callerId, uint64_t tag, ImageReadResult& result);
145
147
149     IrisInstance* irisInstance;
150
154     typedef std::vector<ImageMetaInfo> ImageMetaInfoList;
155     ImageMetaInfoList                  metaInfos;
156
158     IrisLogger log;
159
160     ImageLoadFileDelegate loadFileDelegate;
161     ImageLoadDataDelegate loadDataDelegate;
162 };
163
174 class IrisInstanceImage_Callback
175 {
176 public:
182     IrisInstanceImage_Callback(IrisInstance* irisInstance = 0);
183
184     ~IrisInstanceImage_Callback();
185
191     void attachTo(IrisInstance* irisInstance);
192
202     uint64_t openImage(const std::string& fileName);
203
204 protected:
206     void impl_image_loadDataRead(IrisReceivedRequest& request);
207
208 private:
210     IrisErrorCode readImageData(uint64_t tag, uint64_t position, uint64_t size, bool end,
      ImageReadResult& result);
211
213     IrisInstance* irisInstance;
214
216     IrisLogger log;
217
219     typedef std::vector<FILE*> ImageList;
220     ImageList                  images;
221 };
222
223 NAMESPACE_IRIS_END
```

```
224
225 #endif // #ifndef ARM_INCLUDE_IrisInstanceImage_h
```

9.31 IrisInstanceMemory.h File Reference

Memory add-on to IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
```

Classes

- struct iris::IrisInstanceMemory::AddressTranslationInfoAndAccess

  Contains static address translation information.
- class iris::IrisInstanceMemory

  Memory add-on for IrisInstance.
- struct iris::IrisInstanceMemory::SpaceInfoAndAccess

  Entry in 'spaceInfos'.

Typedefs

- typedef IrisDelegate< uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult & > iris::MemoryAddressTranslateDelegat

  Delegate to translate an address.
- typedef IrisDelegate< const MemorySpaceInfo &, uint64_t, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap & > iris::MemoryGetSidebandInfoDelegate
- typedef IrisDelegate< const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult & > iris::MemoryReadDelegate

  Delegate to read memory data.
- typedef IrisDelegate< const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult & > iris::MemoryWriteDelegate

  Delegate to write memory data.

9.31.1 Detailed Description

Memory add-on to IrisInstance.

Copyright

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.

9.31.2 Typedef Documentation

9.31.2.1 MemoryAddressTranslateDelegate

```
typedef IrisDelegate<uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&> iris::MemoryAddressTransl
```

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.

9.31.2.2 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.

9.31.2.3 MemoryReadDelegate

```
typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t, const Attribute↩
ValueMap&, 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.

9.31.2.4 MemoryWriteDelegate

```
typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t, const Attribute↩
ValueMap&, const uint64_t*, MemoryWriteResult&> iris::MemoryWriteDelegate
```

Delegate to write memory data.

```
IrisErrorCode write(const MemorySpaceInfo &spaceInfo, uint64_t address, uint64_t byteWidth,
                    uint64_t count, const AttributeValueMap &attrib, const uint64_t *data, MemoryWriteResult
      &result)
```

See also

MemoryReadDelegate data contains the data elements to be written in the same format as MemoryRead↩ Result.data for reads.

9.32 IrisInstanceMemory.h

Go to the documentation of this file.

```
1
14 #ifndef ARM_INCLUDE_IrisInstanceMemory_h
15 #define ARM_INCLUDE_IrisInstanceMemory_h
16
17 #include "iris/detail/IrisCommon.h"
18 #include "iris/detail/IrisDelegate.h"
19 #include "iris/detail/IrisLogger.h"
20 #include "iris/detail/IrisObjects.h"
21
22 NAMESPACE_IRIS_START
23
24 class IrisInstance;
25 class IrisReceivedRequest;
26
47 typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
48                      const AttributeValueMap&, MemoryReadResult&>
49     MemoryReadDelegate;
50
61 typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, uint64_t, uint64_t,
62                      const AttributeValueMap&, const uint64_t*, MemoryWriteResult&>
63     MemoryWriteDelegate;
64
85 typedef IrisDelegate<uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult&>
       MemoryAddressTranslateDelegate;
86
99 typedef IrisDelegate<const MemorySpaceInfo&, uint64_t, const IrisValueMap&,
100                      const std::vector<std::string>&, IrisValueMap&>
101     MemoryGetSidebandInfoDelegate;
102
129 class IrisInstanceMemory
130 {
131 public:
137     struct SpaceInfoAndAccess
138     {
139         MemorySpaceInfo               spaceInfo;
140         MemoryReadDelegate            readDelegate;     // May be empty. In this case
       defaultReadDelegate is used.
141         MemoryWriteDelegate           writeDelegate;    // May be empty. In this case
       defaultWriteDelegate is used.
142         MemoryGetSidebandInfoDelegate sidebandDelegate; // May be empty. In this case sidebandDelegate
       is used.
143     };
144
148     struct AddressTranslationInfoAndAccess
149     {
150         AddressTranslationInfoAndAccess(MemorySpaceId inSpaceId, MemorySpaceId outSpaceId, const
       std::string& description)
151             : translationInfo(inSpaceId, outSpaceId, description)
152         {
153         }
154
155         MemorySupportedAddressTranslationResult translationInfo;
156         MemoryAddressTranslateDelegate          translateDelegate;
157     };
158
165     IrisInstanceMemory(IrisInstance* irisInstance = 0);
166
172     void attachTo(IrisInstance* irisInstance);
173
179     void setDefaultReadDelegate(MemoryReadDelegate delegate = MemoryReadDelegate())
180     {
181         memReadDelegate = delegate;
182     }
183
189     void setDefaultWriteDelegate(MemoryWriteDelegate delegate = MemoryWriteDelegate())
190     {
191         memWriteDelegate = delegate;
192     }
193
201     SpaceInfoAndAccess& addMemorySpace(const std::string& name);
202
213     AddressTranslationInfoAndAccess& addAddressTranslation(MemorySpaceId inSpaceId, MemorySpaceId
       outSpaceId,
214                                                            const std::string& description);
215
221     void setDefaultTranslateDelegate(MemoryAddressTranslateDelegate delegate =
       MemoryAddressTranslateDelegate())
222     {
223         translateDelegate = delegate;
224     }
225
231     void setDefaultGetSidebandInfoDelegate(MemoryGetSidebandInfoDelegate delegate =
       MemoryGetSidebandInfoDelegate())
```

```
232     {
233         if (delegate.empty())
234         {
235             delegate = MemoryGetSidebandInfoDelegate::make<IrisInstanceMemory,
    &IrisInstanceMemory::getDefaultSidebandInfo>(this);
236         }
237
238         sidebandDelegate = delegate;
239     }
240
241 private:
243
244     void impl_memory_getMemorySpaces(IrisReceivedRequest& request);
245
246     void impl_memory_read(IrisReceivedRequest& request);
247
248     void impl_memory_write(IrisReceivedRequest& request);
249
250     void impl_memory_translateAddress(IrisReceivedRequest& request);
251
252     void impl_memory_getUsefulAddressTranslations(IrisReceivedRequest& request);
253
254     void impl_memory_getSidebandInfo(IrisReceivedRequest& request);
256
258     IrisErrorCode getDefaultSidebandInfo(const MemorySpaceInfo& spaceInfo, uint64_t address,
259                                          const IrisValueMap&            attrib,
260                                          const std::vector<std::string>& request,
261                                          IrisValueMap&                  result);
262     // --- state ---
263
265     IrisInstance* irisInstance;
266
268     typedef std::vector<SpaceInfoAndAccess> SpaceInfoList;
269     SpaceInfoList                           spaceInfos;
270
272     typedef std::vector<AddressTranslationInfoAndAccess> SupportedTranslations;
273     SupportedTranslations                                supportedTranslations;
274
276     MemoryReadDelegate            memReadDelegate;
277     MemoryWriteDelegate           memWriteDelegate;
278     MemoryAddressTranslateDelegate translateDelegate;
279
282     MemoryGetSidebandInfoDelegate sidebandDelegate;
283
285     IrisLogger log;
286 };
287
288 NAMESPACE_IRIS_END
289
290 #endif // #ifndef ARM_INCLUDE_IrisInstanceMemory_h
```

9.33 IrisInstancePerInstanceExecution.h File Reference

Per-instance execution control add-on to IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdio>
```

Classes

- class iris::IrisInstancePerInstanceExecution

  Per-instance execution control add-on for IrisInstance.

Typedefs

- typedef IrisDelegate< bool & > iris::PerInstanceExecutionStateGetDelegate

  Get the execution state.

- typedef IrisDelegate< bool > iris::PerInstanceExecutionStateSetDelegate

  Delegate to set the execution state.

9.33.1 Detailed Description

Per-instance execution control add-on to IrisInstance.

Copyright

Implements all per-instance execution control-related Iris functions.

9.33.2 Typedef Documentation

9.33.2.1 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.

9.33.2.2 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.

9.34 IrisInstancePerInstanceExecution.h

Go to the documentation of this file.

```
1
9 #ifndef ARM_INCLUDE_IrisInstancePerInstanceExecution_h
10 #define ARM_INCLUDE_IrisInstancePerInstanceExecution_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisLogger.h"
15 #include "iris/detail/IrisObjects.h"
16
17 #include <cstdio>
18
19 NAMESPACE_IRIS_START
20
21 class IrisInstance;
22 class IrisReceivedRequest;
23
33 typedef IrisDelegate<bool> PerInstanceExecutionStateSetDelegate;
34
44 typedef IrisDelegate<bool&> PerInstanceExecutionStateGetDelegate;
45
53 class IrisInstancePerInstanceExecution
54 {
55 public:
61     IrisInstancePerInstanceExecution(IrisInstance* irisInstance = nullptr);
62
70     void attachTo(IrisInstance* irisInstance);
71
78     void setExecutionStateSetDelegate(PerInstanceExecutionStateSetDelegate delegate);
79
86     void setExecutionStateGetDelegate(PerInstanceExecutionStateGetDelegate delegate);
87
88 private:
89     void impl_perInstanceExecution_setState(IrisReceivedRequest& request);
90
91     void impl_perInstanceExecution_getState(IrisReceivedRequest& request);
92
94
96     IrisInstance* irisInstance;
97
99     PerInstanceExecutionStateSetDelegate execStateSet;
100     PerInstanceExecutionStateGetDelegate execStateGet;
```

```
101
103     IrisLogger log;
104 };
105
106 NAMESPACE_IRIS_END
107
108 #endif // #ifndef ARM_INCLUDE_IrisInstancePerInstanceExecution_h
```

9.35 IrisInstanceResource.h File Reference

Resource add-on to IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cassert>
```

Classes

- class iris::IrisInstanceResource

  Resource add-on for IrisInstance.
- struct iris::IrisInstanceResource::ResourceInfoAndAccess

  Entry in 'resourceInfos'.
- struct iris::ResourceWriteValue

Typedefs

- typedef IrisDelegate< const ResourceInfo &, ResourceReadResult & > iris::ResourceReadDelegate

  Delegate to read resources.
- typedef IrisDelegate< const ResourceInfo &, const ResourceWriteValue & > iris::ResourceWriteDelegate

  Delegate to write resources.

Functions

- uint64_t iris::resourceReadBitField (uint64_t parentValue, const ResourceInfo &resourceInfo)
- template<class T >
  void iris::resourceWriteBitField (T &parentValue, uint64_t fieldValue, const ResourceInfo &resourceInfo)

9.35.1 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.

9.35.2 Typedef Documentation

9.35.2.1 ResourceReadDelegate

```
typedef IrisDelegate<const ResourceInfo&, ResourceReadResult&> iris::ResourceReadDelegate
```

Delegate to read resources.

```
IrisErrorCode read(const ResourceInfo &resourceInfo, ResourceReadResult &result)
```

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 read and then append the read data to result:

- Return data (no undefined bits):
  - Append data to result.data, which is a vector<uint64_t>. Append one uint64_t if resource is <= 64 bits.
  - Append multiple uint64_t for wider resources, least significant uint64_t first.
- Return data with undefined bits:
  - Same as above, but in addition, append a mask which contains 1 bit for all undefined bits to result.↩ undefinedBits (same format and length as result.data) and set all undefined bits to 0 in result.data.

Error: If the resource could not be read, return E_* error code, for example E_error_reading_write_only_resource, E_error_reading_resource, or E_not_implemented, and leave result unchanged.

9.35.2.2 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.

9.35.3 Function Documentation

9.35.3.1 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.

9.35.3.2 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.

9.36 IrisInstanceResource.h

Go to the documentation of this file.

```
1
14 #ifndef ARM_INCLUDE_IrisInstanceResource_h
15 #define ARM_INCLUDE_IrisInstanceResource_h
16
17 #include "iris/detail/IrisCommon.h"
18 #include "iris/detail/IrisDelegate.h"
19 #include "iris/detail/IrisLogger.h"
20 #include "iris/detail/IrisObjects.h"
21
22 #include <cassert>
23
24 NAMESPACE_IRIS_START
25
26 class IrisInstance;
27 class IrisReceivedRequest;
28
32 inline uint64_t resourceReadBitField(uint64_t parentValue, const ResourceInfo& resourceInfo)
33 {
34     return (resourceInfo.registerInfo.lsbOffset < 64) ?
35         ((parentValue » resourceInfo.registerInfo.lsbOffset) & maskWidthLsb(resourceInfo.bitWidth, 0))
36         : 0;
37 }
38
39
43 template<class T>
44 inline void resourceWriteBitField(T& parentValue, uint64_t fieldValue, const ResourceInfo& resourceInfo)
45 {
46     T mask = T(maskWidthLsb(resourceInfo.bitWidth, resourceInfo.registerInfo.lsbOffset));
47     parentValue &= ~mask;
48     parentValue |= (resourceInfo.registerInfo.lsbOffset < 64) ?
49         ((fieldValue « resourceInfo.registerInfo.lsbOffset) & mask)
50         : 0;
51 }
52
53
58 struct ResourceWriteValue
59 {
60     const uint64_t*    data{};
61     const std::string* str{};
63 };
64
65
89 typedef IrisDelegate<const ResourceInfo&, ResourceReadResult&> ResourceReadDelegate;
90
106 typedef IrisDelegate<const ResourceInfo&, const ResourceWriteValue&> ResourceWriteDelegate;
107
120 class IrisInstanceResource
121 {
122 public:
128     struct ResourceInfoAndAccess
129     {
130         ResourceInfo          resourceInfo;
131         ResourceReadDelegate  readDelegate;  // May be invalid. In this case defaultReadDelegate is
      used.
132         ResourceWriteDelegate writeDelegate; // May be invalid. In this case defaultWriteDelegate is
      used.
133     };
134
141     IrisInstanceResource(IrisInstance* irisInstance = 0);
142
148     void attachTo(IrisInstance* irisInstance);
149
163     ResourceInfoAndAccess& addResource(const std::string& type,
164                                        const std::string& name,
165                                        const std::string& description);
166
179     void beginResourceGroup(const std::string& name,
180                             const std::string& description,
181                             uint64_t           startSubRscId = IRIS_UINT64_MAX,
182                             const std::string& cname         = std::string());
183
193     void setNextSubRscId(ResourceId nextSubRscId_)
194     {
195         nextSubRscId = nextSubRscId_;
196     }
197
206     void setTag(ResourceId rscId, const std::string& tag);
207
216     ResourceInfoAndAccess* getResourceInfo(ResourceId rscId);
217
238     static void calcHierarchicalNames(std::vector<ResourceInfo>& resourceInfos);
239
254     static void makeNamesHierarchical(std::vector<ResourceInfo>& resourceInfos);
```

```
255
256 protected:
257     // --- Iris function implementations ---
258
259     void impl_resource_getList(IrisReceivedRequest& request);
260
261     void impl_resource_getListOfResourceGroups(IrisReceivedRequest& request);
262
263     void impl_resource_getResourceInfo(IrisReceivedRequest& request);
264
265     void impl_resource_read(IrisReceivedRequest& request);
266
267     void impl_resource_write(IrisReceivedRequest& request);
268
269 private:
270
276     static void calcHierarchicalNamesInternal(std::vector<ResourceInfo>& resourceInfos, const
        std::map<ResourceId,size_t>& rscIdToIndex, std::vector<bool>& done, size_t index);
277
278     // --- State ---
279
281     IrisInstance* irisInstance;
282
284     IrisLogger log;
285
288     typedef std::vector<ResourceInfoAndAccess> ResourceInfoList;
289     ResourceInfoList                           resourceInfos;
290
292     typedef std::vector<ResourceGroupInfo> GroupInfoList;
293     GroupInfoList                          groupInfos;
294
296     typedef std::map<std::string, size_t> GroupNameToIndex;
297     GroupNameToIndex                      groupNameToIndex;
298
300     ResourceGroupInfo* currentAddGroup;
301
303     uint64_t nextSubRscId{IRIS_UINT64_MAX};
304 };
305
306 NAMESPACE_IRIS_END
307
308 #endif // #ifndef ARM_INCLUDE_IrisInstanceResource_source
```

9.37 IrisInstanceSemihosting.h File Reference

IrisInstance add-on to implement semihosting functionality.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/IrisInstanceEvent.h"
#include <mutex>
#include <queue>
```

Classes

- class iris::IrisInstanceSemihosting

9.37.1 Detailed Description

IrisInstance add-on to implement semihosting functionality.

Copyright

9.38 IrisInstanceSemihosting.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisInstanceSemihosting_h
9 #define ARM_INCLUDE_IrisInstanceSemihosting_h
10
```

```
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisLogger.h"
13 #include "iris/detail/IrisObjects.h"
14
15 #include "iris/IrisInstanceEvent.h"
16
17 #include <mutex>
18 #include <queue>
19
20 NAMESPACE_IRIS_START
21
22 class IrisInstance;
23 class IrisInstanceEvent;
24 class IrisReceivedRequest;
25
26 namespace semihost
27 {
28
38     static const uint64_t COOKED = (0 « 0);
39
43     static const uint64_t RAW = (1 « 0);
44
50     static const uint64_t BLOCK = (0 « 1);
51
55     static const uint64_t NONBLOCK = (1 « 1);
56
60     static const uint64_t EMIT_EVENT = (0 « 2);
61
65     static const uint64_t NO_EVENT = (1 « 2);
66
70     static const uint64_t DEFAULT = COOKED | BLOCK | EMIT_EVENT;
71
79     static const uint64_t STDIN = 0;
80
84     static const uint64_t STDOUT = 1;
85
89     static const uint64_t STDERR = 2;
90
91 } // namespace semihost
92
93 class IrisInstanceSemihosting
94 {
95 private:
97     IrisInstance* iris_instance{nullptr};
98
101     IrisInstanceEvent* inst_event{nullptr};
102
104     std::map<uint64_t, unsigned> evSrcId_map{};
105
107     std::vector<IrisEventRegistry> event_registries{};
108
110     std::map<uint64_t, std::queue<uint8_t» buffered_input_data{};
111
113     std::mutex buffer_mutex{};
114
116     std::mutex extension_mutex{};
117
118     uint64_t extension_retval{0};
119
120     IrisLogger log{};
121
123     std::atomic<bool> unblock_requested{false};
124
125     enum ExtensionState
126     {
127         XS_DISABLED,          // Semihosting extensions are not supported
128         XS_DORMANT,           // No ongoing semihosting extension call in progress
129         XS_WAITING_FOR_REPLY, // Event has been emitted, waiting for a reply for a client
130         XS_RETURNED,          // A client instance has called semihosting_return()
131         XS_NOT_IMPLEMENTED    // A client instance has called semihosting_notImplemented()
132
133     } extension_state{XS_DISABLED};
134
135 public:
136     IrisInstanceSemihosting(IrisInstance* iris_instance = nullptr, IrisInstanceEvent* inst_event =
    nullptr);
137
138     ~IrisInstanceSemihosting();
139
145     void attachTo(IrisInstance* iris_instance);
146
155     void setEventHandler(IrisInstanceEvent* handler);
156
171     std::vector<uint8_t> readData(uint64_t fDes, uint64_t max_size = 0, uint64_t flags =
    semihost::DEFAULT);
172
173     /*
```

```
174      * @brief Write data for a given file descriptor
175      *
176      * @param   fDes        File descriptor to write to. Usually semihost::STDOUT or semihost::STDERR.
177      * @param   data        Buffer containing the data to write.
178      * @param   size        Size of the data buffer in bytes.
179      * @return              Returns false if no client is registered for IRIS_SEMIHOSTING_OUTPUT events.
180      */
181     bool writeData(uint64_t fDes, const uint8_t* data, uint64_t size);
182
187     void enableExtensions();
188
203     std::pair<bool, uint64_t> semihostedCall(uint64_t operation, uint64_t parameter);
204
208     void unblock();
209
210 private:
212     void impl_semihosting_provideInputData(IrisReceivedRequest& request);
213
215     void impl_semihosting_return(IrisReceivedRequest& request);
216
218     void impl_semihosting_notImplemented(IrisReceivedRequest& request);
219
221     IrisErrorCode createEventStream(EventStream*& stream_out, const EventSourceInfo& info,
222                                     const std::vector<std::string>& requested_fields);
223
225     void notifyCall(uint64_t operation, uint64_t parameter);
226
227     class SemihostingEventStream;
228
229     IrisErrorCode enableEventStream(EventStream* stream, unsigned event_type);
230     IrisErrorCode disableEventStream(EventStream* stream, unsigned event_type);
231 };
232
233 NAMESPACE_IRIS_END
234
235 #endif // ARM_INCLUDE_IrisInstanceSemihosting_h
```

9.39 IrisInstanceSimulation.h File Reference

IrisInstance add-on to implement simulation_* functions.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include "iris/IrisInstantiationContext.h"
#include <map>
#include <mutex>
#include <string>
#include <vector>
```

Classes

- class iris::IrisInstanceSimulation

  An IrisInstance add-on that adds simulation functions for the SimulationEngine instance.
- class iris::IrisSimulationResetContext

  Provides context to a reset delegate call.

Typedefs

- typedef IrisDelegate< std::vector< ResourceInfo > & > iris::SimulationGetParameterInfoDelegate

  Delegate to get a list of parameter information.
- typedef IrisDelegate< InstantiationResult & > iris::SimulationInstantiateDelegate

  Delegate to instantiate the simulation.
- typedef IrisDelegate iris::SimulationRequestShutdownDelegate

  Delegate to request that the simulation be shut down.
- typedef IrisDelegate< const IrisSimulationResetContext & > iris::SimulationResetDelegate

  Delegate to reset the simulation.

- typedef IrisDelegate< const InstantiationParameterValue & > iris::SimulationSetParameterValueDelegate

  Delegate to set the value of an instantiation parameter.

Enumerations

- enum iris::IrisSimulationPhase {
  IRIS_SIM_PHASE_INITIAL_PLUGIN_LOADING_COMPLETE , **IRIS_SIM_PHASE_INSTANTIATE_ENTER** , **IRIS_SIM_PHASE_INSTANTIATE** , **IRIS_SIM_PHASE_INSTANTIATE_LEAVE** ,
  IRIS_SIM_PHASE_INIT_ENTER , **IRIS_SIM_PHASE_INIT** , **IRIS_SIM_PHASE_INIT_LEAVE** , **IRIS_SIM_PHASE_BEFORE_END_OF_ELABORATION** ,
  IRIS_SIM_PHASE_END_OF_ELABORATION , **IRIS_SIM_PHASE_INITIAL_RESET_ENTER** , **IRIS_SIM_PHASE_INITIAL_RESET** , **IRIS_SIM_PHASE_INITIAL_RESET_LEAVE** ,
  IRIS_SIM_PHASE_START_OF_SIMULATION , **IRIS_SIM_PHASE_RESET_ENTER** , **IRIS_SIM_PHASE_RESET** , **IRIS_SIM_PHASE_RESET_LEAVE** ,
  IRIS_SIM_PHASE_END_OF_SIMULATION , **IRIS_SIM_PHASE_TERMINATE_ENTER** , **IRIS_SIM_PHASE_TERMINATE** , **IRIS_SIM_PHASE_TERMINATE_LEAVE** ,
  IRIS_SIM_PHASE_NUM }

  List of IRIS_SIMULATION_PHASE events.

9.39.1 Detailed Description

IrisInstance add-on to implement simulation_∗ functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.39.2 Typedef Documentation

9.39.2.1 SimulationGetParameterInfoDelegate

```
typedef IrisDelegate<std::vector<ResourceInfo>&> iris::SimulationGetParameterInfoDelegate
```

Delegate to get a list of parameter information.

```
IrisErrorCode getInstantiationParameterInfo(std::vector<ResourceInfo> &parameters_out)
```

9.39.2.2 SimulationInstantiateDelegate

```
typedef IrisDelegate<InstantiationResult&> iris::SimulationInstantiateDelegate
```

Delegate to instantiate the simulation.

```
IrisErrorCode instantiate(InstantiationResult &result_out)
```

9.39.2.3 SimulationRequestShutdownDelegate

```
typedef IrisDelegate iris::SimulationRequestShutdownDelegate
```

Delegate to request that the simulation be shut down.

```
IrisErrorCode requestShutdown()
```

9.39.2.4 SimulationResetDelegate

```
typedef IrisDelegate<const IrisSimulationResetContext&> iris::SimulationResetDelegate
```

Delegate to reset the simulation.

```
IrisErrorCode reset(const IrisSimulationResetContext &)
```

9.39.2.5 SimulationSetParameterValueDelegate

```
typedef IrisDelegate<const InstantiationParameterValue&> iris::SimulationSetParameterValueDelegate
```

Delegate to set the value of an instantiation parameter.

```
IrisErrorCode setInstantiationParameterValue(const InstantiationParameterValue &value)
```

9.40 IrisInstanceSimulation.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisInstanceSimulation_h
9 #define ARM_INCLUDE_IrisInstanceSimulation_h
10
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisDelegate.h"
13 #include "iris/detail/IrisLogger.h"
14 #include "iris/detail/IrisObjects.h"
15
16 #include "iris/IrisInstantiationContext.h"
17
18 #include <map>
19 #include <mutex>
20 #include <string>
21 #include <vector>
22
23 NAMESPACE_IRIS_START
24
25 class IrisInstance;
26 class IrisReceivedRequest;
27 class IrisInstanceEvent;
28 class IrisEventRegistry;
29
30 class EventStream;
31
39 typedef IrisDelegate<InstantiationResult&> SimulationInstantiateDelegate;
40
44 class IrisSimulationResetContext
45 {
46 private:
48     static const uint64_t ALLOW_PARTIAL = (1 << 0);
49
51     uint64_t flags;
52
54     bool getFlag(uint64_t mask) const
55     {
56         return (flags & mask) != 0;
57     }
58
60     void setFlag(uint64_t mask, bool value)
61     {
62         flags &= ~mask;
63         flags |= (value ? mask : 0);
64     }
65
66 public:
67     IrisSimulationResetContext()
68         : flags(0)
69     {
70     }
71
77     bool getAllowPartialReset() const
78     {
79         return getFlag(ALLOW_PARTIAL);
80     }
81
82     // Set/clear the allowPartialReset flag.
83     void setAllowPartialReset(bool value = true)
84     {
85         setFlag(ALLOW_PARTIAL, value);
86     }
87 };
88
94 typedef IrisDelegate<const IrisSimulationResetContext&> SimulationResetDelegate;
95
101 typedef IrisDelegate<> SimulationRequestShutdownDelegate;
102
107 typedef IrisDelegate<std::vector<ResourceInfo>&> SimulationGetParameterInfoDelegate;
108
113 typedef IrisDelegate<const InstantiationParameterValue&> SimulationSetParameterValueDelegate;
114
118 enum IrisSimulationPhase
119 {
120     IRIS_SIM_PHASE_INITIAL_PLUGIN_LOADING_COMPLETE,
```

```
121     IRIS_SIM_PHASE_INSTANTIATE_ENTER,
122     IRIS_SIM_PHASE_INSTANTIATE,
123     IRIS_SIM_PHASE_INSTANTIATE_LEAVE,
124     IRIS_SIM_PHASE_INIT_ENTER,
125     IRIS_SIM_PHASE_INIT,
126     IRIS_SIM_PHASE_INIT_LEAVE,
127     IRIS_SIM_PHASE_BEFORE_END_OF_ELABORATION,
128     IRIS_SIM_PHASE_END_OF_ELABORATION,
129     IRIS_SIM_PHASE_INITIAL_RESET_ENTER,
130     IRIS_SIM_PHASE_INITIAL_RESET,
131     IRIS_SIM_PHASE_INITIAL_RESET_LEAVE,
132     IRIS_SIM_PHASE_START_OF_SIMULATION,
133     IRIS_SIM_PHASE_RESET_ENTER,
134     IRIS_SIM_PHASE_RESET,
135     IRIS_SIM_PHASE_RESET_LEAVE,
136     IRIS_SIM_PHASE_END_OF_SIMULATION,
137     IRIS_SIM_PHASE_TERMINATE_ENTER,
138     IRIS_SIM_PHASE_TERMINATE,
139     IRIS_SIM_PHASE_TERMINATE_LEAVE,
140     IRIS_SIM_PHASE_NUM
141 };
142 static const size_t IrisSimulationPhase_total = IRIS_SIM_PHASE_NUM;
143
147 class IrisInstanceSimulation
148 {
149 private:
151     IrisInstance* iris_instance;
152
155     IrisConnectionInterface* connection_interface;
156
158     SimulationInstantiateDelegate instantiate;
159
161     SimulationResetDelegate reset;
162
164     SimulationRequestShutdownDelegate requestShutdown;
165
167     SimulationGetParameterInfoDelegate getParameterInfo;
168
170     SimulationSetParameterValueDelegate setParameterValue;
171
174     enum
175     {
176         CACHE_DISABLED,
177         CACHE_EMPTY,
178         CACHE_SET
179     } parameter_info_cache_state;
180
182     std::vector<ResourceInfo> cached_parameter_info;
183
185     std::mutex mutex;
186
188     std::vector<IrisEventRegistry*> simulation_phase_event_registries;
189
191     std::map<uint64_t, IrisSimulationPhase> evSrcId_to_phase;
192
193     IrisLogger log;
194
197     bool simulation_has_been_initialised;
198
200     std::vector<uint64_t> requests_waiting_for_instantiation;
201
202 public:
210     IrisInstanceSimulation(IrisInstance*           iris_instance        = nullptr,
211                            IrisConnectionInterface* connection_interface = nullptr);
212     ~IrisInstanceSimulation();
213
219     void attachTo(IrisInstance* iris_instance);
220
226     void setConnectionInterface(IrisConnectionInterface* connection_interface_)
227     {
228         connection_interface = connection_interface_;
229     }
230
236     void setInstantiateDelegate(SimulationInstantiateDelegate delegate)
237     {
238         instantiate = delegate;
239     }
240
250     template <typename T, IrisErrorCode (T::*METHOD)(InstantiationResult&)>
251     void setInstantiateDelegate(T* instance)
252     {
253         setInstantiateDelegate(SimulationInstantiateDelegate::make<T, METHOD>(instance));
254     }
255
263     template <IrisErrorCode (*FUNC)(InstantiationResult&)>
264     void setInstantiateDelegate()
265     {
```

```
266         setInstantiateDelegate(SimulationInstantiateDelegate::make<FUNC>());
267     }
268
274     void setResetDelegate(SimulationResetDelegate delegate)
275     {
276         reset = delegate;
277     }
278
288     template <typename T, IrisErrorCode (T::*METHOD)(const IrisSimulationResetContext&)>
289     void setResetDelegate(T* instance)
290     {
291         setResetDelegate(SimulationResetDelegate::make<T, METHOD>(instance));
292     }
293
301     template <IrisErrorCode (*FUNC)(const IrisSimulationResetContext&)>
302     void setResetDelegate()
303     {
304         setResetDelegate(SimulationResetDelegate::make<FUNC>());
305     }
306
313     void setRequestShutdownDelegate(SimulationRequestShutdownDelegate delegate)
314     {
315         requestShutdown = delegate;
316     }
317
327     template <typename T, IrisErrorCode (T::*METHOD)()>
328     void setRequestShutdownDelegate(T* instance)
329     {
330         setRequestShutdownDelegate(SimulationRequestShutdownDelegate::make<T, METHOD>(instance));
331     }
332
340     template <IrisErrorCode (*FUNC)()>
341     void setRequestShutdownDelegate()
342     {
343         setRequestShutdownDelegate(SimulationRequestShutdownDelegate::make<FUNC>());
344     }
345
356     void setGetParameterInfoDelegate(SimulationGetParameterInfoDelegate delegate, bool cache_result =
    true)
357     {
358         getParameterInfo           = delegate;
359         parameter_info_cache_state = cache_result ? CACHE_EMPTY : CACHE_DISABLED;
360         cached_parameter_info.clear();
361     }
362
376     template <typename T, IrisErrorCode (T::*METHOD)(std::vector<ResourceInfo>&)>
377     void setGetParameterInfoDelegate(T* instance, bool cache_result = true)
378     {
379         typedef SimulationGetParameterInfoDelegate D;
380         setGetParameterInfoDelegate(D::make<T, METHOD>(instance), cache_result);
381     }
382
394     template <IrisErrorCode (*FUNC)(std::vector<ResourceInfo>&)>
395     void setGetParameterInfoDelegate(bool cache_result = true)
396     {
397         typedef SimulationGetParameterInfoDelegate D;
398         setGetParameterInfoDelegate(D::make<FUNC>(), cache_result);
399     }
400
407     void setSetParameterValueDelegate(SimulationSetParameterValueDelegate delegate)
408     {
409         setParameterValue = delegate;
410     }
411
421     template <typename T, IrisErrorCode (T::*METHOD)(const InstantiationParameterValue&)>
422     void setSetParameterValueDelegate(T* instance)
423     {
424         setSetParameterValueDelegate(SimulationSetParameterValueDelegate::make<T, METHOD>(instance));
425     }
426
434     template <IrisErrorCode (*FUNC)(const InstantiationParameterValue&)>
435     void setSetParameterValueDelegate()
436     {
437         setSetParameterValueDelegate(SimulationSetParameterValueDelegate::make<FUNC>());
438     }
439
448     void enterPostInstantiationPhase();
449
455     void setEventHandler(IrisInstanceEvent* handler);
456
463     void notifySimPhase(uint64_t time, IrisSimulationPhase phase);
464
476     void registerSimEventsOnGlobalInstance();
477
483     static std::string getSimulationPhaseName(IrisSimulationPhase phase);
484
490     static std::string getSimulationPhaseDescription(IrisSimulationPhase phase);
```

```
491
492 private:
494     void impl_simulation_getInstantiationParameterInfo(IrisReceivedRequest& request);
495
497     void impl_simulation_setInstantiationParameterValues(IrisReceivedRequest& request);
498
500     void impl_simulation_instantiate(IrisReceivedRequest& request);
501
503     void impl_simulation_reset(IrisReceivedRequest& request);
504
506     void impl_simulation_requestShutdown(IrisReceivedRequest& request);
507
509     void impl_simulation_waitForInstantiation(IrisReceivedRequest& request);
510
512     IrisErrorCode createEventStream(EventStream*& event_stream_out, const EventSourceInfo& info,
513                                     const std::vector<std::string>& fields);
514 };
515
516 NAMESPACE_IRIS_END
517
518 #endif // ARM_INCLUDE_IrisInstanceSimulation_h
```

9.41 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.

9.41.1 Detailed Description

IrisInstance add-on to implement simulationTime functions.

Copyright

9.41.2 Typedef Documentation

9.41.2.1 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);
```

9.41.2.2 SimulationTimeRunDelegate

```
typedef IrisDelegate iris::SimulationTimeRunDelegate
```

Delegate to resume the simulation time progress.

```
IrisErrorCode run();
```

9.41.2.3 SimulationTimeStopDelegate

```
typedef IrisDelegate iris::SimulationTimeStopDelegate
```

Delegate to stop the simulation time progress.

```
IrisErrorCode stop();
```

9.41.3 Enumeration Type Documentation

9.41.3.1 TIME_EVENT_REASON

```
enum iris::TIME_EVENT_REASON
```

The reasons why the simulation time stopped.

Enumerator

TIME_EVENT_UNKNOWN	Simulation stopped for a different reason.
TIME_EVENT_STOP	simulationTime_stop() was called.
TIME_EVENT_BREAKPOINT	Breakpoint was hit.
TIME_EVENT_TRACE_COUNTER_OVERFLOW	CounterMode.overflowStopSim.

9.42 IrisInstanceSimulationTime.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisInstanceSimulationTime_h
9 #define ARM_INCLUDE_IrisInstanceSimulationTime_h
10
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisDelegate.h"
13
14 #include <string>
15 #include <vector>
16
17 NAMESPACE_IRIS_START
18
23 typedef IrisDelegate<> SimulationTimeRunDelegate;
24
29 typedef IrisDelegate<> SimulationTimeStopDelegate;
30
35 typedef IrisDelegate<uint64_t&, uint64_t&, bool&> SimulationTimeGetDelegate;
36
40 enum TIME_EVENT_REASON
41 {
42     TIME_EVENT_UNKNOWN,
43     TIME_EVENT_STOP,
44     TIME_EVENT_BREAKPOINT,
45     TIME_EVENT_TRACE_COUNTER_OVERFLOW
46 };
47
48 class IrisInstance;
49 class IrisInstanceEvent;
```

```
50 class IrisEventRegistry;
51 class IrisReceivedRequest;
52
53 class EventStream;
54 struct EventSourceInfo;
55
59 class IrisInstanceSimulationTime
60 {
61 private:
63     IrisInstance* iris_instance;
64
66     IrisEventRegistry* simulation_time_event_registry;
67
69     SimulationTimeRunDelegate  run_delegate;
70     SimulationTimeStopDelegate stop_delegate;
71     SimulationTimeGetDelegate  get_time_delegate;
72
73 public:
81     IrisInstanceSimulationTime(IrisInstance* iris_instance = nullptr, IrisInstanceEvent* inst_event =
      nullptr);
82     ~IrisInstanceSimulationTime();
83
89     void attachTo(IrisInstance* irisInstance);
90
97     void setEventHandler(IrisInstanceEvent* handler);
98
104     void setSimTimeRunDelegate(SimulationTimeRunDelegate delegate)
105     {
106         run_delegate = delegate;
107     }
108
116     template <typename T, IrisErrorCode (T::*METHOD)()>
117     void setSimTimeRunDelegate(T* instance)
118     {
119         setSimTimeRunDelegate(SimulationTimeRunDelegate::make<T, METHOD>(instance));
120     }
121
129     template <IrisErrorCode (*FUNC)()>
130     void setSimTimeRunDelegate()
131     {
132         setSimTimeRunDelegate(SimulationTimeRunDelegate::make<FUNC>());
133     }
134
140     void setSimTimeStopDelegate(SimulationTimeStopDelegate delegate)
141     {
142         stop_delegate = delegate;
143     }
144
152     template <typename T, IrisErrorCode (T::*METHOD)()>
153     void setSimTimeStopDelegate(T* instance)
154     {
155         setSimTimeStopDelegate(SimulationTimeStopDelegate::make<T, METHOD>(instance));
156     }
157
165     template <IrisErrorCode (*FUNC)()>
166     void setSimTimeStopDelegate()
167     {
168         setSimTimeStopDelegate(SimulationTimeStopDelegate::make<FUNC>());
169     }
170
176     void setSimTimeGetDelegate(SimulationTimeGetDelegate delegate)
177     {
178         get_time_delegate = delegate;
179     }
180
188     template <typename T, IrisErrorCode (T::*METHOD)(uint64_t&, uint64_t&, bool&)>
189     void setSimTimeGetDelegate(T* instance)
190     {
191         setSimTimeGetDelegate(SimulationTimeGetDelegate::make<T, METHOD>(instance));
192     }
193
201     template <IrisErrorCode (*FUNC)(uint64_t&, uint64_t&, bool&)>
202     void setSimTimeGetDelegate()
203     {
204         setSimTimeGetDelegate(SimulationTimeGetDelegate::make<FUNC>());
205     }
206
208     void notifySimulationTimeEvent(TIME_EVENT_REASON reason = TIME_EVENT_UNKNOWN);
209
221     void registerSimTimeEventsOnGlobalInstance();
222
223 private:
225     void impl_simulationTime_run(IrisReceivedRequest& request);
226     void impl_simulationTime_stop(IrisReceivedRequest& request);
227     void impl_simulationTime_get(IrisReceivedRequest& request);
228
229     IrisErrorCode createEventStream(EventStream*&, const EventSourceInfo&, const
```

```
      std::vector<std::string>&);
230 };
231
232 NAMESPACE_IRIS_END
233
234 #endif // ARM_INCLUDE_IrisInstanceSimulationTime_h
```

9.43 IrisInstanceStep.h File Reference

Stepping-related add-on to an IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
#include "iris/detail/IrisLogger.h"
#include "iris/detail/IrisObjects.h"
#include <cstdio>
```

Classes

- class iris::IrisInstanceStep

  Step add-on for IrisInstance.

Typedefs

- typedef IrisDelegate< uint64_t &, const std::string & > iris::RemainingStepGetDelegate

  Delegate to get the value of the currently remaining steps.
- typedef IrisDelegate< uint64_t, const std::string & > iris::RemainingStepSetDelegate

  Delegate to set the remaining steps measured in the specified unit.
- typedef IrisDelegate< uint64_t &, const std::string & > iris::StepCountGetDelegate

  Delegate to get the value of the step count.

9.43.1 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.

9.43.2 Typedef Documentation

9.43.2.1 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.

9.43.2.2 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.

9.43.2.3 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.

9.44 IrisInstanceStep.h

Go to the documentation of this file.

```
1
9 #ifndef ARM_INCLUDE_IrisInstanceStep_h
10 #define ARM_INCLUDE_IrisInstanceStep_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisLogger.h"
15 #include "iris/detail/IrisObjects.h"
16
17 #include <cstdio>
18
19 NAMESPACE_IRIS_START
20
21 class IrisInstance;
22 class IrisReceivedRequest;
23
31 typedef IrisDelegate<uint64_t, const std::string&> RemainingStepSetDelegate;
32
40 typedef IrisDelegate<uint64_t&, const std::string&> RemainingStepGetDelegate;
41
49 typedef IrisDelegate<uint64_t&, const std::string&> StepCountGetDelegate;
50
58 class IrisInstanceStep
59 {
60 public:
66     IrisInstanceStep(IrisInstance* irisInstance = nullptr);
67
75     void attachTo(IrisInstance* irisInstance);
76
83     void setRemainingStepSetDelegate(RemainingStepSetDelegate delegate);
84
91     void setRemainingStepGetDelegate(RemainingStepGetDelegate delegate);
92
99     void setStepCountGetDelegate(StepCountGetDelegate delegate);
100
101 private:
102     void impl_step_setup(IrisReceivedRequest& request);
103
104     void impl_step_getRemainingSteps(IrisReceivedRequest& request);
105
106     void impl_step_getStepCounterValue(IrisReceivedRequest& request);
107
108     void impl_step_syncStep(IrisReceivedRequest& request);
109
110     void impl_step_syncStepSetup(IrisReceivedRequest& request);
111
113
115     IrisInstance* irisInstance;
116
118     RemainingStepSetDelegate stepSetDel;
119     RemainingStepGetDelegate stepGetDel;
120
122     StepCountGetDelegate stepCountGetDel;
123
125     IrisLogger log;
126
128     EventBufferId evBufId{IRIS_UINT64_MAX};
129 };
130
131 NAMESPACE_IRIS_END
132
133 #endif // #ifndef ARM_INCLUDE_IrisInstanceStep_h
```

9.45 IrisInstanceTable.h File Reference

Table add-on to IrisInstance.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisDelegate.h"
```

```
#include "iris/detail/IrisObjects.h"
```

Classes

- class iris::IrisInstanceTable

  Table add-on for IrisInstance.
- struct iris::IrisInstanceTable::TableInfoAndAccess

  Entry in 'tableInfos'.

Typedefs

- typedef IrisDelegate< const TableInfo &, uint64_t, uint64_t, TableReadResult & > iris::TableReadDelegate

  Delegate to read table data.
- typedef IrisDelegate< const TableInfo &, const TableRecords &, TableWriteResult & > iris::TableWriteDelegate

  Delegate to write table data.

9.45.1 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.

9.45.2 Typedef Documentation

9.45.2.1 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.

9.45.2.2 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.

9.46 IrisInstanceTable.h

Go to the documentation of this file.

```
1
9 #ifndef ARM_INCLUDE_IrisInstanceTable_h
10 #define ARM_INCLUDE_IrisInstanceTable_h
11
12 #include "iris/detail/IrisCommon.h"
13 #include "iris/detail/IrisDelegate.h"
14 #include "iris/detail/IrisObjects.h"
15
16 NAMESPACE_IRIS_START
17
18 class IrisInstance;
19 class IrisReceivedRequest;
20
```

```
31 typedef IrisDelegate<const TableInfo&, uint64_t, uint64_t, TableReadResult&> TableReadDelegate;
32
43 typedef IrisDelegate<const TableInfo&, const TableRecords&, TableWriteResult&> TableWriteDelegate;
44
50 class IrisInstanceTable
51 {
52 public:
58     struct TableInfoAndAccess
59     {
60         TableInfo          tableInfo;
61         TableReadDelegate  readDelegate;
62         TableWriteDelegate writeDelegate;
63     };
64
70     IrisInstanceTable(IrisInstance* irisInstance = nullptr);
71
79     void attachTo(IrisInstance* irisInstance);
80
88     TableInfoAndAccess& addTableInfo(const std::string& name);
89
96     void setDefaultReadDelegate(TableReadDelegate delegate = TableReadDelegate())
97     {
98         defaultReadDelegate = delegate;
99     }
100
107     void setDefaultWriteDelegate(TableWriteDelegate delegate = TableWriteDelegate())
108     {
109         defaultWriteDelegate = delegate;
110     }
111
112 private:
113     void impl_table_getList(IrisReceivedRequest& request);
114
115     void impl_table_read(IrisReceivedRequest& request);
116
117     void impl_table_write(IrisReceivedRequest& request);
118
120
122     IrisInstance* irisInstance;
123
125     typedef std::vector<TableInfoAndAccess> TableInfoAndAccessList;
126     TableInfoAndAccessList                  tableInfos;
127
129     TableReadDelegate  defaultReadDelegate;
130     TableWriteDelegate defaultWriteDelegate;
131 };
132
133 NAMESPACE_IRIS_END
134
135 #endif // #ifndef ARM_INCLUDE_IrisInstanceTable_h
```

9.47 IrisInstantiationContext.h File Reference

Helper class used to instantiate Iris instances from generic factories.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisUtils.h"
#include <string>
#include <vector>
```

Classes

- class iris::IrisInstantiationContext

  Provides context when instantiating an Iris instance from a factory.

9.47.1 Detailed Description

Helper class used to instantiate Iris instances from generic factories.

Copyright

9.48 IrisInstantiationContext.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisInstantiationContext_h
8 #define ARM_INCLUDE_IrisInstantiationContext_h
9
10 #include "iris/detail/IrisCommon.h"
11 #include "iris/detail/IrisObjects.h"
12 #include "iris/detail/IrisUtils.h"
13
14 #include <string>
15 #include <vector>
16
17 NAMESPACE_IRIS_START
18
22 class IrisInstantiationContext
23 {
24 private:
25     IrisConnectionInterface* connection_interface;
26
29     InstantiationResult& result;
30
33     IrisValueMap params;
34
39     std::string prefix;
40
42     std::string component_name;
43
44     uint64_t instance_flags;
45
47     std::vector<IrisInstantiationContext*> children;
48
49     void errorInternal(const std::string& severity,
50                        const std::string& code,
51                        const std::string& parameterName,
52                        const char*        format,
53                        va_list            args);
54
57     void processParameters(const std::vector<ResourceInfo>&                param_info_,
58                            const std::vector<InstantiationParameterValue>& param_values_);
59
61     IrisInstantiationContext(const IrisInstantiationContext* parent, const std::string& instance_name);
62
63 public:
64     IrisInstantiationContext(IrisConnectionInterface*                       connection_interface_,
65                              InstantiationResult&                           result_,
66                              const std::vector<ResourceInfo>&               param_info_,
67                              const std::vector<InstantiationParameterValue>& param_values_,
68                              const std::string&                             prefix_,
69                              const std::string&                             component_name_,
70                              uint64_t                                       instance_flags_);
71
72     ~IrisInstantiationContext();
73
85     IrisInstantiationContext* getSubcomponentContext(const std::string& child_name);
86
96     template <typename T>
97     void getParameter(const std::string& name, T& value)
98     {
99         IrisValueMap::const_iterator it = params.find(name);
100         if (it != params.end())
101         {
102             it->second.get(value);
103         }
104         else
105         {
106             throw IrisInternalError("Instance tried to read invalid parameter");
107         }
108     }
109
119     void getParameter(const std::string& name, std::vector<uint64_t>& value);
120
127     uint64_t getRecommendedInstanceFlags() const
128     {
129         return instance_flags;
130     }
131
138     std::string getInstanceName() const
139     {
140         return prefix + "." + component_name;
141     }
142
148     IrisConnectionInterface* getConnectionInterface() const
149     {
150         return connection_interface;
```

```
151     }
152
163     void warning(const std::string& code, const char* format, ...) INTERNAL_IRIS_PRINTF(3, 4);
164
176     void parameterWarning(const std::string& code, const std::string& parameterName, const char* format,
        ...) INTERNAL_IRIS_PRINTF(4, 5);
187     void error(const std::string& code, const char* format, ...) INTERNAL_IRIS_PRINTF(3, 4);
188
200     void parameterError(const std::string& code, const std::string& parameterName, const char* format,
        ...) INTERNAL_IRIS_PRINTF(4, 5);
201 };
202
203 NAMESPACE_IRIS_END
204
205 #endif // ARM_INCLUDE_IrisInstantiationContext_h
```

9.49 IrisParameterBuilder.h File Reference

Helper class to construct instantiation parameters.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisObjects.h"
#include <string>
#include <vector>
```

Classes

- class iris::IrisParameterBuilder

  Helper class to construct instantiation parameters.

9.49.1 Detailed Description

Helper class to construct instantiation parameters.

Copyright

9.50 IrisParameterBuilder.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisParameterBuilder_h
8 #define ARM_INCLUDE_IrisParameterBuilder_h
9
10 #include "iris/detail/IrisCommon.h"
11 #include "iris/detail/IrisObjects.h"
12
13 #include <string>
14 #include <vector>
15
16 NAMESPACE_IRIS_START
17
21 class IrisParameterBuilder
22 {
23 private:
24     ResourceInfo& info;
25
26     IrisParameterBuilder& setValueExtend(std::vector<uint64_t>& arr, uint64_t value, uint64_t extension)
27     {
28         arr.resize(info.getDataSizeInU64Chunks(), extension);
29         arr[0] = value;
30
31         return *this;
32     }
33
34     IrisParameterBuilder& setValueExtend(std::vector<uint64_t>& arr, const std::vector<uint64_t>& value,
        uint64_t extension)
35     {
36         size_t param_size = info.getDataSizeInU64Chunks();
37         if (param_size < value.size())
38         {
39             throw IrisInternalError("Invalid parameter configuration");
```

```
40          }
41          arr = value;
42          arr.resize(info.getDataSizeInU64Chunks(), extension);
43
44          return *this;
45      }
46
47      IrisParameterBuilder& setValueSignExtend(std::vector<uint64_t>& arr, int64_t value)
48      {
49          return setValueExtend(arr, static_cast<uint64_t>(value), (value < 0) ? IRIS_UINT64_MAX : 0);
50      }
51
52      IrisParameterBuilder& setValueZeroExtend(std::vector<uint64_t>& arr, uint64_t value)
53      {
54          return setValueExtend(arr, value, 0);
55      }
56
57      IrisParameterBuilder& setValueSignExtend(std::vector<uint64_t>& arr, const std::vector<uint64_t>&
        value)
58      {
59          return setValueExtend(arr, value, (static_cast<int64_t>(value.back()) < 0) ? IRIS_UINT64_MAX :
        0);
60      }
61
62      IrisParameterBuilder& setValueZeroExtend(std::vector<uint64_t>& arr, const std::vector<uint64_t>&
        value)
63      {
64          return setValueExtend(arr, value, 0);
65      }
66
67      IrisParameterBuilder& setValueDouble(std::vector<uint64_t>& arr, double value)
68      {
69          arr.resize(1);
70          *static_cast<double*>((void*) (&arr[0])) = value;
71
72          return *this;
73      }
74
75  public:
80      IrisParameterBuilder(ResourceInfo& info_)
81          : info(info_)
82      {
83          info.isParameter = true;
84      }
85
91      IrisParameterBuilder& setName(const std::string& name)
92      {
93          info.name = name;
94          return *this;
95      }
96
102      IrisParameterBuilder& setDescr(const std::string& description)
103      {
104          info.description = description;
105          return *this;
106      }
107
113      IrisParameterBuilder& setFormat(const std::string& format)
114      {
115          info.format = format;
116          return *this;
117      }
118
124      IrisParameterBuilder& setBitWidth(uint64_t bitWidth)
125      {
126          info.bitWidth = bitWidth;
127          return *this;
128      }
129
135      IrisParameterBuilder& setRwMode(const std::string& rwMode)
136      {
137          info.rwMode = rwMode;
138          return *this;
139      }
140
146      IrisParameterBuilder& setSubRscId(uint64_t subRscId)
147      {
148          info.subRscId = subRscId;
149          return *this;
150      }
151
157      IrisParameterBuilder& setTopology(bool value = true)
158      {
159          info.parameterInfo.topology = value;
160          return *this;
161      }
162
```

```
168     IrisParameterBuilder& setInitOnly(bool value = true)
169     {
170         info.parameterInfo.initOnly = value;
171         return *this;
172     }
173
179     IrisParameterBuilder& setMin(uint64_t min)
180     {
181         return setValueZeroExtend(info.parameterInfo.min, min);
182     }
183
189     IrisParameterBuilder& setMax(uint64_t max)
190     {
191         return setValueZeroExtend(info.parameterInfo.max, max);
192     }
193
200     IrisParameterBuilder& setRange(uint64_t min, uint64_t max)
201     {
202         return setMin(min).setMax(max);
203     }
204
213     IrisParameterBuilder& setMin(const std::vector<uint64_t>& min)
214     {
215         return setValueZeroExtend(info.parameterInfo.min, min);
216     }
217
226     IrisParameterBuilder& setMax(const std::vector<uint64_t>& max)
227     {
228         return setValueZeroExtend(info.parameterInfo.max, max);
229     }
230
240     IrisParameterBuilder& setRange(const std::vector<uint64_t>& min, const std::vector<uint64_t>& max)
241     {
242         return setMin(min).setMax(max);
243     }
244
253     IrisParameterBuilder& setMinSigned(int64_t min)
254     {
255         return setValueSignExtend(info.parameterInfo.min, min)
256             .setType("numericSigned");
257     }
258
267     IrisParameterBuilder& setMaxSigned(int64_t max)
268     {
269         return setValueSignExtend(info.parameterInfo.max, max)
270             .setType("numericSigned");
271     }
272
282     IrisParameterBuilder& setRangeSigned(int64_t min, int64_t max)
283     {
284         return setValueSignExtend(info.parameterInfo.min, min)
285             .setValueSignExtend(info.parameterInfo.max, max)
286             .setType("numericSigned");
287     }
288
298     IrisParameterBuilder& setMinSigned(const std::vector<uint64_t>& min)
299     {
300         return setValueSignExtend(info.parameterInfo.min, min)
301             .setType("numericSigned");
302     }
303
313     IrisParameterBuilder& setMaxSigned(const std::vector<uint64_t>& max)
314     {
315         return setValueSignExtend(info.parameterInfo.max, max)
316             .setType("numericSigned");
317     }
318
329     IrisParameterBuilder& setRangeSigned(const std::vector<uint64_t>& min, const std::vector<uint64_t>&
    max)
330     {
331         return setValueSignExtend(info.parameterInfo.min, min)
332             .setValueSignExtend(info.parameterInfo.max, max)
333             .setType("numericSigned");
334     }
335
344     IrisParameterBuilder& setMinFloat(double min)
345     {
346         return setValueDouble(info.parameterInfo.min, min)
347             .setType("numericFp");
348     }
349
358     IrisParameterBuilder& setMaxFloat(double max)
359     {
360         return setValueDouble(info.parameterInfo.max, max)
361             .setType("numericFp");
362     }
363
```

```
373     IrisParameterBuilder& setRangeFloat(double min, double max)
374     {
375         return setValueDouble(info.parameterInfo.min, min)
376             .setValueDouble(info.parameterInfo.max, max)
377             .setType("numericFp");
378     }
379
388     IrisParameterBuilder& addEnum(const std::string& symbol, const IrisValue& value, const std::string&
    description = std::string())
389     {
390         info.enums.push_back(EnumElementInfo(value, symbol, description));
391         return *this;
392     }
393
403     IrisParameterBuilder& addStringEnum(const std::string& value, const std::string& description =
    std::string())
404     {
405         info.enums.push_back(EnumElementInfo(IrisValue(value), std::string(), description));
406         return *this;
407     }
408
415     IrisParameterBuilder& setTag(const std::string& tag)
416     {
417         info.tags[tag] = IrisValue(true);
418         return *this;
419     }
420
427     IrisParameterBuilder& setHidden(bool hidden)
428     {
429         info.isHidden = hidden;
430         return *this;
431     }
432
440     IrisParameterBuilder& setTag(const std::string& tag, const IrisValue& value)
441     {
442         info.tags[tag] = value;
443         return *this;
444     }
445
452     IrisParameterBuilder& setDefault(const std::string& value)
453     {
454         info.parameterInfo.defaultString = value;
455         return *this;
456     }
457
464     IrisParameterBuilder& setDefault(uint64_t value)
465     {
466         return setValueZeroExtend(info.parameterInfo.defaultData, value);
467     }
468
477     IrisParameterBuilder& setDefault(const std::vector<uint64_t>& value)
478     {
479         return setValueZeroExtend(info.parameterInfo.defaultData, value);
480     }
481
488     IrisParameterBuilder& setDefaultSigned(int64_t value)
489     {
490         return setValueSignExtend(info.parameterInfo.defaultData, value);
491     }
492
501     IrisParameterBuilder& setDefaultSigned(const std::vector<uint64_t>& value)
502     {
503         return setValueSignExtend(info.parameterInfo.defaultData, value);
504     }
505
512     IrisParameterBuilder& setDefaultFloat(double value)
513     {
514         return setValueDouble(info.parameterInfo.defaultData, value);
515     }
516
525     IrisParameterBuilder& setType(const std::string& type)
526     {
527         if ((info.bitWidth != 32) && (info.bitWidth != 64) && (type == "numericFp"))
528         {
529             throw IrisInternalError(
530                 "Invalid parameter configuration."
531                 " NumericFp parameters must have a bitWidth of 32 or 64");
532         }
533
534         info.type = type;
535         return *this;
536     }
537 };
538
539 NAMESPACE_IRIS_END
540
541 #endif // ARM_INCLUDE_IrisParameterBuilder_h
```

9.51 IrisPluginFactory.h File Reference

A generic plug-in factory for instantiating plug-in instances.

```
#include "iris/IrisCConnection.h"
#include "iris/IrisInstance.h"
#include "iris/IrisInstanceFactoryBuilder.h"
#include "iris/IrisInstantiationContext.h"
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisFunctionInfo.h"
#include "iris/detail/IrisObjects.h"
#include "iris/detail/IrisU64JsonReader.h"
#include "iris/detail/IrisU64JsonWriter.h"
#include <mutex>
#include <string>
#include <vector>
```

Classes

- class iris::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.

9.51.1 Detailed Description

A generic plug-in factory for instantiating plug-in instances.

Copyright

9.51.2 Macro Definition Documentation

9.51.2.1 IRIS_PLUGIN_FACTORY

```
#define IRIS_PLUGIN_FACTORY(
            plugin_instance )
```

Value:

```
    extern "C" IRIS_EXPORT int64_t irisInitPlugin(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.

9.52 IrisPluginFactory.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisPluginFactory_h
8 #define ARM_INCLUDE_IrisPluginFactory_h
9
10 #include "iris/IrisCConnection.h"
11 #include "iris/IrisInstance.h"
12 #include "iris/IrisInstanceFactoryBuilder.h"
13 #include "iris/IrisInstantiationContext.h"
14 #include "iris/detail/IrisCommon.h"
15 #include "iris/detail/IrisFunctionInfo.h"
16 #include "iris/detail/IrisObjects.h"
17 #include "iris/detail/IrisU64JsonReader.h"
18 #include "iris/detail/IrisU64JsonWriter.h"
19
20 #include <mutex>
21 #include <string>
22 #include <vector>
23
24 NAMESPACE_IRIS_START
25
29 class IrisPluginFactoryBuilder : public IrisInstanceFactoryBuilder
30 {
31 private:
34     std::string plugin_name;
35
37     std::string instance_name_prefix;
38
40     std::string default_instance_name;
41
42 public:
46     IrisPluginFactoryBuilder(const std::string& name)
47         : IrisInstanceFactoryBuilder(/*parameter_prefix=*/"")
48         , plugin_name(name)
49         , instance_name_prefix("client.plugin")
50     {
51     }
52
61     void setPluginName(const std::string& name)
62     {
63         plugin_name = name;
64     }
65
70     const std::string& getPluginName() const
71     {
72         return plugin_name;
73     }
74
83     void setInstanceNamePrefix(const std::string& prefix)
84     {
85         instance_name_prefix = prefix;
86     }
87
92     const std::string& getInstanceNamePrefix() const
93     {
94         return instance_name_prefix;
95     }
96
105     void setDefaultInstanceName(const std::string& name)
106     {
107         default_instance_name = name;
108     }
109
115     const std::string& getDefaultInstanceName() const
116     {
117         if (default_instance_name.empty())
118         {
119             return getPluginName();
120         }
121         else
122         {
123             return default_instance_name;
124         }
125     }
126 };
127
128 template <typename PLUGIN_INSTANCE>
129 class IrisPluginFactory
130 {
131 private:
133     IrisCConnection connection_interface;
134
136     IrisInstance factory_instance;
137
139     std::vector<PLUGIN_INSTANCE*> plugin_instances;
140
141     std::mutex plugin_instances_mutex;
142
```

```
144     IrisPluginFactoryBuilder builder;
145
147     void impl_plugin_getInstantiationParameterInfo(IrisReceivedRequest& req)
148     {
149         factory_instance.sendResponse(req.generateOkResponse(builder.getParameterInfo()));
150     }
151
153     void impl_plugin_instantiate(IrisReceivedRequest& req)
154     {
155         InstantiationResult result;
156         result.success = true; // Assume we will succeed until proven otherwise
157
158         uint64_t instance_flags = IrisInstance::DEFAULT_FLAGS;
159
160         std::string instName;
161
162         if (!req.getOptionalArg(ISTR("instName"), instName))
163         {
164             instName = builder.getDefaultInstanceName();
165             instance_flags |= IrisInstance::UNIQUIFY;
166         }
167
168         std::vector<InstantiationParameterValue> param_values;
169         req.getOptionalArg(ISTR("paramValues"), param_values);
170
171         // Build the full parameter info list
172         const std::vector<ResourceInfo>& param_info        = builder.getParameterInfo();
173         const std::vector<ResourceInfo>& hidden_param_info = builder.getHiddenParameterInfo();
174
175         std::vector<ResourceInfo> all_param_info;
176         all_param_info.insert(all_param_info.end(), param_info.begin(), param_info.end());
177         all_param_info.insert(all_param_info.end(), hidden_param_info.begin(), hidden_param_info.end());
178
179         IrisInstantiationContext init_context(&connection_interface, result,
180                                               all_param_info, param_values,
181                                               builder.getInstanceNamePrefix(),
182                                               instName, instance_flags);
183
184         // Parameters have been validated. If they all passed we can instantiate the plugin.
185
186         if (result.success)
187         {
188             try
189             {
190                 std::lock_guard<std::mutex> lock(plugin_instances_mutex);
191
192                 plugin_instances.push_back(new PLUGIN_INSTANCE(init_context));
193
194                 if (!result.success)
195                 {
196                     // The plugin instance set an error in its constructor so destroy it.
197                     delete plugin_instances.back();
198                     plugin_instances.pop_back();
199                 }
200             }
201             catch (IrisErrorException& e)
202             {
203                 result.success = false;
204                 result.errors.resize(result.errors.size() + 1);
205
206                 InstantiationError& error = result.errors.back();
207                 error.severity            = "error";
208                 error.code                = "error_general_error";
209                 error.message             = e.getMessage();
210             }
211             catch (...)
212             {
213                 result.success = false;
214                 result.errors.resize(result.errors.size() + 1);
215
216                 InstantiationError& error = result.errors.back();
217                 error.severity            = "error";
218                 error.code                = "error_general_error";
219                 error.message             = "Internal error while instantiating plugin";
220             }
221         }
222
223         factory_instance.sendResponse(req.generateOkResponse(result));
224     }
225
226 public:
227     IrisPluginFactory(IrisC_Functions* iris_c_functions, const std::string& plugin_name)
228         : connection_interface(iris_c_functions)
229         , factory_instance(&connection_interface)
230         , builder(plugin_name)
231     {
232         PLUGIN_INSTANCE::buildPluginFactory(builder);
```

```
233
234         typedef IrisPluginFactory<PLUGIN_INSTANCE> Self;
235
236         factory_instance.irisRegisterFunction(this, Self, plugin_getInstantiationParameterInfo,
237                                               function_info::plugin_getInstantiationParameterInfo);
238
239         factory_instance.irisRegisterFunction(this, Self, plugin_instantiate,
240                                               "{description:'Instantiate an instance of the " +
    builder.getPluginName() +
241                                                   " plugin',"
242                                                   "args:{"
243                                                   "  instName:{type:'String', description:'Used to
    construct the instance name for the new instance."
244                                                   " Instance name will be \""
245                                                   + builder.getInstanceNamePrefix() +
246                                                   "<instName>\"',"
247                                                   "defval:'"
248                                                   + builder.getDefaultInstanceName() +
249                                                   "', optional:true},"
250                                                   "  paramValues:{type:'Array',
    description:'Instantiation parameter values'}"
251                                                   "},"
252                                                   "retval:{type:'InstantiationResult',
    description:'Indicates success of and errors/warnings"
253                                                   " that occurred during plugin instantiation.'}}");
254
255         // Register factory instance
256         uint64_t flags = IrisInstance::DEFAULT_FLAGS
257             | IrisInstance::UNIQUIFY;
258
259         std::string factory_instName = "framework.plugin." + builder.getPluginName() + "Factory";
260         factory_instance.registerInstance(factory_instName, flags);
261         factory_instance.setProperty("componentType", "IrisPluginFactory");
262
263         IrisLogger log("IrisPluginFactory");
264     }
265
266     ~IrisPluginFactory()
267     {
268         {
269             std::lock_guard<std::mutex> lock(plugin_instances_mutex);
270
271             // Clean up plugin instances
272             typename std::vector<PLUGIN_INSTANCE*>::iterator it;
273             for (it = plugin_instances.begin(); it != plugin_instances.end(); ++it)
274             {
275                 delete *it;
276             }
277         }
278     }
279
280     // Unregister factory instance. Call this when unloading a plugin before simulation termination.
281     IrisErrorCode unregisterInstance()
282     {
283         return factory_instance.unregisterInstance();
284     }
285
286     // Implementation of the plugin entry point.
287     // This will initialize an IrisPluginFactory the first time it is called.
288     static int64_t initPlugin(IrisC_Functions* functions, const std::string& plugin_name)
289     {
290         static IrisPluginFactory<PLUGIN_INSTANCE>* factory = nullptr;
291
292         if (factory == nullptr)
293         {
294             factory = new IrisPluginFactory<PLUGIN_INSTANCE>(functions, plugin_name);
295             return E_ok;
296         }
297         else
298         {
299             return E_plugin_already_loaded;
300         }
301     }
302 };
303
309 #define IRIS_PLUGIN_FACTORY(plugin_instance)                                                    \
310     extern "C" IRIS_EXPORT int64_t irisInitPlugin(IrisC_Functions* functions)                   \
311     {                                                                                           \
312         return ::iris::IrisPluginFactory<plugin_instance>::initPlugin(functions, #plugin_instance); \
313     }
314
315 NAMESPACE_IRIS_END
316
317 #endif // ARM_INCLUDE_IrisPluginFactory_h
```

9.53 IrisRegisterEventEmitter.h File Reference

Utility classes for emitting register read and register update events.

```
#include "iris/detail/IrisCommon.h"
#include "iris/detail/IrisRegisterEventEmitterBase.h"
```

Classes

- class iris::IrisRegisterReadEventEmitter< REG_T, ARGS >

  An EventEmitter class for register read events.
- class iris::IrisRegisterUpdateEventEmitter< REG_T, ARGS >

  An EventEmitter class for register update events.

9.53.1 Detailed Description

Utility classes for emitting register read and register update events.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

9.54 IrisRegisterEventEmitter.h

Go to the documentation of this file.

```
1
8 #ifndef ARM_INCLUDE_IrisRegisterEventEmitter_h
9 #define ARM_INCLUDE_IrisRegisterEventEmitter_h
10
11 #include "iris/detail/IrisCommon.h"
12 #include "iris/detail/IrisRegisterEventEmitterBase.h"
13
14 NAMESPACE_IRIS_START
15
57 template <typename REG_T, typename... ARGS>
58 class IrisRegisterReadEventEmitter : public IrisRegisterEventEmitterBase
59 {
60 public:
61     IrisRegisterReadEventEmitter()
62         : IrisRegisterEventEmitterBase(sizeof...(ARGS) + 3)
63     {
64     }
65
74     void operator()(ResourceId rscId, bool debug, REG_T value, ARGS... args)
75     {
76         // Emit event
77         emitEvent(rscId, debug, value, args...);
78
79         // Check if this event indicates a breakpoint was hit
80         if (!debug)
81         {
82             checkBreakpointHit(rscId, value, /*is_read=*/true);
83         }
84     }
85 };
86
126 template <typename REG_T, typename... ARGS>
127 class IrisRegisterUpdateEventEmitter : public IrisRegisterEventEmitterBase
128 {
129 public:
130     IrisRegisterUpdateEventEmitter()
131         : IrisRegisterEventEmitterBase(sizeof...(ARGS) + 4)
132     {
133     }
134
144     void operator()(ResourceId rscId, bool debug, REG_T old_value, REG_T new_value, ARGS... args)
145     {
146         // Emit event
147         emitEvent(rscId, debug, old_value, new_value, args...);
148
149         // Check if this event indicates a breakpoint was hit
150         if (!debug)
151         {
152             checkBreakpointHit(rscId, new_value, /*is_read=*/false);
153         }
```

```
154     }
155 };
156
157 NAMESPACE_IRIS_END
158
159 #endif // ARM_INCLUDE_IrisRegisterEventEmitter_h
```

9.55 IrisTcpClient.h File Reference

IrisTcpClient Type alias for IrisClient.
`#include "iris/IrisClient.h"`

Typedefs

- using **iris::IrisTcpClient** = IrisClient

  Alias for backward compatibility.

9.55.1 Detailed Description

IrisTcpClient Type alias for IrisClient.

Date

9.56 IrisTcpClient.h

Go to the documentation of this file.

```
1
7 #ifndef ARM_INCLUDE_IrisTcpClient_h
8 #define ARM_INCLUDE_IrisTcpClient_h
9
10 #include "iris/IrisClient.h"
11
12 NAMESPACE_IRIS_START
13
15 using IrisTcpClient = IrisClient;
16
17 NAMESPACE_IRIS_END
18
19 #endif // #ifndef ARM_INCLUDE_IrisTcpClient_h
```