IrisSupportLib

Version 1.0

Reference Guide

Contents

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_HO↩ ME/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**
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.

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_M↩ AX)

    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.
- 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 &, ResourceReadResult &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultResourceReadDelegate ()

  Set default read 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.
- template<IrisErrorCode(∗)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  void iris::IrisInstanceBuilder::setDefaultResourceWriteDelegate ()

  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

rscId	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

rscId	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]

```
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.

7.2.2.12 setDefaultResourceReadDelegate() [2/3]

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

Set default read access function for all subsequently added resources.

Resources that do not explicitly override the access function using

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

will use this delegate.

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

```
class MyClass
{
    ...
    iris::IrisErrorCode myReadFunction(const iris::ResourceInfo &resourceInfo,
                                       iris::ResourceReadResult &result);
};

MyClass myInstanceOfMyClass;

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->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.

7.2.2.13 setDefaultResourceReadDelegate() [3/3]

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

Set default read access function for all subsequently added resources.

Resources that do not explicitly override the access function using

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

will use this delegate.

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

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

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

Template Parameters

FUNC	A function which is a resource read delegate.

7.2.2.14 setDefaultResourceWriteDelegate() [1/3]

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

Set default write access function for all subsequently added resources.

Resources that do not explicitly override the access function using

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

will use this delegate.

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

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

```
class MyClass
{
    ...
    iris::IrisErrorCode myWriteFunction(const iris::ResourceInfo &resourceInfo, const uint64_t *data);
};

MyClass myInstanceOfMyClass;

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
iris::ResourceWriteDelegate delegate =
    iris::ResourceWriteDelegate::make<MyClass, &MyClass::myWriteFunction>(myInstanceOfMyClass);
builder->setDefaultWriteDelegate(delegate);
builder->addRegister(...); // Uses myWriteFunction
```

Parameters

delegate	Delegate object which will be called to write resources.

7.2.2.15 setDefaultResourceWriteDelegate() [2/3]

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

```
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.

7.2.2.16 setDefaultResourceWriteDelegate() [3/3]

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

Set default write access function for all subsequently added resources.

Resources that do not explicitly override the access function using

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

will use this delegate.

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

```
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.

7.2.2.17 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

nextSub↩ RscId	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

rsc←↩ Id	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.
- void iris::IrisInstanceBuilder::setDefaultEsCreateDelegate (EventStreamCreateDelegate delegate)

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

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

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

  Add a new register read event source.
- EventSourceBuilder iris::IrisInstanceBuilder::setRegisterReadEvent (const std::string &name, IrisRegister↩
  EventEmitterBase &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, Iris↩
  RegisterEventEmitterBase &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]

```
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.

7.3.2.9 setDefaultEsCreateDelegate() [2/3]

```
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.

7.3.2.10 setDefaultEsCreateDelegate() [3/3]

```
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.

7.3.2.11 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, Memory←SpaceId pcSpaceId)

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

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

    Notify clients that a register breakpoint was hit (`IRIS_BREAKPOINT_HIT`).
- void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate (BreakpointDeleteDelegate delegate)

    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(∗)(const BreakpointInfo &) FUNC>
  void iris::IrisInstanceBuilder::setBreakpointDeleteDelegate ()

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

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

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

    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

bpt↩ Id	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

pc←→ SpaceId	Memory space id for the PC when the breakpoint was hit.
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.
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]

```
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.

7.4.2.6 setBreakpointDeleteDelegate() [2/3]

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

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

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

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

MyClass myInstanceOfMyClass;

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

Template Parameters

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.

7.4.2.7 setBreakpointDeleteDelegate() [3/3]

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

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

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

```
iris::IrisErrorCode deleteBreakpoint(iris::BreakpointInfo&);

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

Template Parameters

FUNC	Global function to call when a breakpoint is deleted.

7.4.2.8 setBreakpointSetDelegate() [1/3]

```
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.

7.4.2.9 setBreakpointSetDelegate() [2/3]

```
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.

7.4.2.10 setBreakpointSetDelegate() [3/3]

```
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.

7.5 IrisInstanceBuilder memory APIs

Set up address translation and memory space metadata and delegates.

Classes

- class iris::IrisInstanceBuilder::AddressTranslationBuilder

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

  Used to set metadata for a memory space.

Functions

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

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

  Add metadata for one memory space.
- void iris::IrisInstanceBuilder::setDefaultAddressTranslateDelegate (MemoryAddressTranslateDelegate delegate=MemoryAddressTranslateDelegate())

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

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

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

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

  Set the default read function for all subsequently added memory spaces.
- void iris::IrisInstanceBuilder::setDefaultMemoryWriteDelegate (MemoryWriteDelegate delegate=MemoryWriteDelegate())

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

- 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.
- 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::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.
out↵ SpaceId	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]

```
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.

7.5.2.4 setDefaultAddressTranslateDelegate() [2/3]

```
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.

7.5.2.5 setDefaultAddressTranslateDelegate() [3/3]

```
template<IrisErrorCode(*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FU↩
NC>
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.

7.5.2.6 setDefaultGetMemorySidebandInfoDelegate() [1/3]

```
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.

7.5.2.7 setDefaultGetMemorySidebandInfoDelegate() [2/3]

```
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.

7.5.2.8 setDefaultGetMemorySidebandInfoDelegate() [3/3]

```
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.

7.5.2.9 setDefaultMemoryReadDelegate() [1/3]

```
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.

7.5.2.10 setDefaultMemoryReadDelegate() [2/3]

```
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.

7.5.2.11 setDefaultMemoryReadDelegate() [3/3]

```
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.

7.5.2.12 setDefaultMemoryWriteDelegate() [1/3]

```
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.

7.5.2.13 setDefaultMemoryWriteDelegate() [2/3]

```
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.

7.5.2.14 setDefaultMemoryWriteDelegate() [3/3]

```
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.

7.5.2.15 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

- void iris::IrisInstanceBuilder::setLoadImageDataDelegate (ImageLoadDataDelegate delegate=ImageLoadDataDelegate())

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

  Set the delegate to load an image from the data provided.
- template<IrisErrorCode(∗)(const std::vector< uint64_t > &, uint64_t) FUNC>
  void iris::IrisInstanceBuilder::setLoadImageDataDelegate ()

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

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

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

  Set the delegate to load an image from a file.

7.6.1 Detailed Description

Set up image-loading delegates.

7.6.2 Function Documentation

7.6.2.1 setLoadImageDataDelegate() [1/3]

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

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

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

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode loadImageData(const std::vector<uint64_t> &data, uint64_t dataSizeInBytes);
};

MyClass myInstanceOfMyClass;

iris::MemoryAddressTranslateDelegate delegate =
    iris::MemoryAddressTranslateDelegate::make<MyClass, &MyClass::loadImageData>(&myInstanceOfMyClass);

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setLoadImageDataDelegate(delegate);
```

Parameters

delegate	Delegate object to call for image loading.

7.6.2.2 setLoadImageDataDelegate() [2/3]

```
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.

7.6.2.3 setLoadImageDataDelegate() [3/3]

```
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.

7.6.2.4 setLoadImageFileDelegate() [1/3]

```
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.

7.6.2.5 setLoadImageFileDelegate() [2/3]

```
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.

7.6.2.6 setLoadImageFileDelegate() [3/3]

```
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.

7.7 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

- 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::setRemainingStepGetDelegate ()

  Set the delegate to get 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::setRemainingStepSetDelegate ()

  Set the delegate to set the remaining steps for this instance.
- void iris::IrisInstanceBuilder::setStepCountGetDelegate (StepCountGetDelegate delegate=StepCountGetDelegate())

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

  Set the delegate to get the step count 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.

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]

```
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.

7.8.2.2 setRemainingStepGetDelegate() [2/3]

```
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.

7.8.2.3 setRemainingStepGetDelegate() [3/3]

```
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.

7.8.2.4 setRemainingStepSetDelegate() [1/3]

```
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.

7.8.2.5 setRemainingStepSetDelegate() [2/3]

```
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.

7.8.2.6 setRemainingStepSetDelegate() [3/3]

```
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.

7.8.2.7 setStepCountGetDelegate() [1/3]

```
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.

7.8.2.8 setStepCountGetDelegate() [2/3]

```
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.

7.8.2.9 setStepCountGetDelegate() [3/3]

```
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.

7.9 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&, Disassemble↩
Context&, 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_)
- AddressTranslationBuilder & setTranslateDelegate (MemoryAddressTranslateDelegate delegate)

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

  Set the delegate to perform an address translation.
- template<IrisErrorCode(∗)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FUNC>
  AddressTranslationBuilder & setTranslateDelegate ()

  Set the delegate to perform an address translation.

8.1.1 Detailed Description

Used to set metadata for an address translation.

8.1.2 Member Function Documentation

8.1.2.1 setTranslateDelegate() [1/3]

```
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.

8.1.2.2 setTranslateDelegate() [2/3]

```
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.

8.1.2.3 setTranslateDelegate() [3/3]

```
template<IrisErrorCode(*)(uint64_t, uint64_t, uint64_t, MemoryAddressTranslationResult &) FU↩
NC>
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.

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

*hasSide←Effects_*	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, uint64_t value)

  Add a uint field value.
- void addField (const IrisU64StringConstant &field, int64_t value)

  Add a sint field value.
- void addField (const IrisU64StringConstant &field, bool value)

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

  Add a field value.
- void addFieldSlow (const std::string &field, uint64_t value)

  Add a uint field value.
- void addFieldSlow (const std::string &field, int64_t value)

  Add a sint field value.
- void addFieldSlow (const std::string &field, bool value)

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

  Add a field value.
- bool checkRangePc (uint64_t pc) const

  Check the range for the PC.
- virtual IrisErrorCode disable ()=0

  Disable this event stream.
- void emitEventBegin (IrisRequest &req, uint64_t time, uint64_t pc=IRIS_UINT64_MAX)

  Start to emit an event callback.
- void emitEventBegin (uint64_t time, uint64_t pc=IRIS_UINT64_MAX)

  Start to emit an event callback.
- 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,
            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.

8.5.2.3 addField() [2/4]

```
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.
value	The value of the field.

8.5.2.4 addField() [3/4]

```
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.

8.5.2.5 addField() [4/4]

```
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.

8.5.2.6 addFieldSlow() [1/4]

```
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.

8.5.2.7 addFieldSlow() [2/4]

```
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.

8.5.2.8 addFieldSlow() [3/4]

```
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.

8.5.2.9 addFieldSlow() [4/4]

```
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.

8.5.2.10 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

request←→ Id	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

inst↵ Id	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.
- 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<IrisErrorCode(∗)(const ResourceInfo &, ResourceReadResult &) FUNC>
  FieldBuilder & setReadDelegate ()

  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, const IrisValue &value)

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

  Set the named boolean tag to true (e.g. isPc)
- FieldBuilder & setType (const std::string &type)

  Set the `type` field.
- template<typename T , IrisErrorCode(T::∗)(const ResourceInfo &, const ResourceWriteValue &) METHOD>
  FieldBuilder & setWriteDelegate (T ∗instance)

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

  Set the delegate to write the resource.
- template<IrisErrorCode(∗)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  FieldBuilder & setWriteDelegate ()

  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<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.

8.6.2.19 setReadDelegate() [2/3]

```
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.

8.6.2.20 setReadDelegate() [3/3]

```
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.

8.6.2.21 setResetData() [1/2]

```
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.

8.6.2.22 setResetData() [2/2]

```
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.

8.6.2.23 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

sub↩ RscId	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]

```
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.

8.6.2.31 setWriteDelegate() [2/3]

```
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.

8.6.2.32 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]

```
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.

8.6.2.34 setWriteMask() [2/2]

```
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.

8.6.2.35 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, 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 std::string &instName=std::string(), const std::string &connectionSpec=std::string())

  Client constructor.
- 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.
- 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 std::string &message) const
- int printErrorAndExit (const IrisErrorException &e) 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)> &message↩ Func)
- 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 'formalArgument↩ Name' 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=V↩ ALUE" 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 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.

8.9.2.14 printErrorAndExit() [2/2]

```
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.

8.9.2.15 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.

Parameters

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.
- const InstanceInfo & getInstanceInfo (InstanceId instId)

    Get InstanceInfo including properties for a specific instId.
- InstanceInfo getInstanceInfo (const std::string &instancePathFilter)

    Get instance info of a specific instance in the system.
- 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, ResourceId resourceId)

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

  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.
- template<class T >
  void registerEventCallback (T ∗instance, const std::string &name, const std::string &description, void(T↩ ::∗memberFunctionPtr)(IrisReceivedRequest &), const std::string &instanceTypeStr)

  Register a general event callback.
- void registerEventCallback (EventCallbackDelegate delegate, const std::string &name, const std::string &description, const std::string &dlgInstanceTypeStr)

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

  Register a general event callback using an EventCallbackDelegate.
- template<class T >
  void registerFunction (T ∗instance, const std::string &name, void(T::∗memberFunctionPtr)(IrisReceived↩ Request &), 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 instance←↩ 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_EVE↩
NT 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]

```
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.

8.14.3.8 getInstanceInfo() [2/2]

```
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.

8.14.3.9 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]

```
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.

8.14.3.24 registerEventCallback() [2/3]

```
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.

8.14.3.25 registerEventCallback() [3/3]

```
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.

8.14.3.26 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.
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 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.
- 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

bpt↩ Id	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.
pc← SpaceId	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.
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.
pc←↩ SpaceId	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.
- 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(∗)(const BreakpointInfo &) FUNC>
  void setBreakpointDeleteDelegate ()

    Set the delegate that is called when a breakpoint is deleted.
- void setBreakpointSetDelegate (BreakpointSetDelegate delegate)

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

    Set the delegate that is called when a breakpoint is set.
- template<IrisErrorCode(∗)(BreakpointInfo &) FUNC>
  void setBreakpointSetDelegate ()

    Set the delegate that is called when a breakpoint is set.
- void setDefaultAddressTranslateDelegate (MemoryAddressTranslateDelegate delegate=MemoryAddressTranslateDelegate())

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

    Set the default address translation function for all subsequently added memory spaces.
- void setDefaultEsCreateDelegate (EventStreamCreateDelegate delegate)

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

    Set the delegate that helps to create a new event stream for the simulation-specific event.
- template<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 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, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>
  void setDefaultGetMemorySidebandInfoDelegate ()

    Set the default sideband info function for all subsequently added memory spaces.
- void setDefaultMemoryReadDelegate (MemoryReadDelegate delegate=MemoryReadDelegate())

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

- template<typename T , IrisErrorCode(T::∗)(const MemorySpaceInfo &, 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 &, MemoryReadResult &) FUNC>
  void setDefaultMemoryReadDelegate ()

  Set the default read function for all subsequently added memory spaces.
- void setDefaultMemoryWriteDelegate (MemoryWriteDelegate delegate=MemoryWriteDelegate())

  Set the default write function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::∗)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t ∗, MemoryWriteResult &) METHOD>
  void setDefaultMemoryWriteDelegate (T ∗instance)

  Set the default write function for all subsequently added memory spaces.
- template<IrisErrorCode(∗)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t ∗, MemoryWriteResult &) FUNC>
  void setDefaultMemoryWriteDelegate ()

  Set default write function for all subsequently added memory spaces.
- template<typename T , IrisErrorCode(T::∗)(const ResourceInfo &, ResourceReadResult &) READER, IrisErrorCode(T::∗)(const ResourceInfo &, const ResourceWriteValue &) WRITER>
  void setDefaultResourceDelegates (T ∗instance)

  Set both read and write resource delegates if they are defined in the same class.
- void setDefaultResourceReadDelegate (ResourceReadDelegate delegate=ResourceReadDelegate())

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

  Set default read access function for all subsequently added resources.
- template<IrisErrorCode(∗)(const ResourceInfo &, ResourceReadResult &) FUNC>
  void setDefaultResourceReadDelegate ()

  Set default read access function for all subsequently added resources.
- void setDefaultResourceWriteDelegate (ResourceWriteDelegate delegate=ResourceWriteDelegate())

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

  Set default write access function for all subsequently added resources.
- template<IrisErrorCode(∗)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  void setDefaultResourceWriteDelegate ()

  Set default write access function for all subsequently added resources.
- void setDefaultTableReadDelegate (TableReadDelegate delegate=TableReadDelegate())

  Set the default table read function for all subsequently added tables.
- template<typename T , IrisErrorCode(T::∗)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) METHOD>
  void setDefaultTableReadDelegate (T ∗instance)

  Set the default table read function for all subsequently added tables.
- template<IrisErrorCode(∗)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
  void setDefaultTableReadDelegate ()

  Set the default table read function for all subsequently added tables.
- void setDefaultTableWriteDelegate (TableWriteDelegate delegate=TableWriteDelegate())

  Set the default table write function for all subsequently added tables.
- template<typename T , IrisErrorCode(T::∗)(const TableInfo &, const TableRecords &, TableWriteResult &) METHOD>
  void setDefaultTableWriteDelegate (T ∗instance)

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

- template<IrisErrorCode(∗)(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
  void setDefaultTableWriteDelegate ()

  Set the default table write function for all subsequently added tables.
- void setExecutionStateGetDelegate (PerInstanceExecutionStateGetDelegate delegate)

  Set the delegate to get the execution state for this instance.
- template<typename T , IrisErrorCode(T::∗)(bool &) METHOD>
  void setExecutionStateGetDelegate (T ∗instance)

  Set the delegate to get the execution state for this instance.
- template<IrisErrorCode(∗)(bool &) FUNC>
  void setExecutionStateGetDelegate ()

  Set the delegate to get the execution state for this instance.
- void setExecutionStateSetDelegate (PerInstanceExecutionStateSetDelegate delegate=PerInstanceExecutionStateSetDelegate

  Set the delegate to set the execution state for this instance.
- template<typename T , IrisErrorCode(T::∗)(bool) METHOD>
  void setExecutionStateSetDelegate (T ∗instance)

  Set the delegate to set the execution state for this instance.
- template<IrisErrorCode(∗)(bool) FUNC>
  void setExecutionStateSetDelegate ()

  Set the delegate to set the execution state for this instance.
- void setLoadImageDataDelegate (ImageLoadDataDelegate delegate=ImageLoadDataDelegate())

  Set the delegate to load an image from the data provided.
- template<typename T , IrisErrorCode(T::∗)(const std::vector< uint64_t > &, uint64_t) METHOD>
  void setLoadImageDataDelegate (T ∗instance)

  Set the delegate to load an image from the data provided.
- template<IrisErrorCode(∗)(const std::vector< uint64_t > &, uint64_t) FUNC>
  void setLoadImageDataDelegate ()

  Set the delegate to load an image from the data provided.
- void setLoadImageFileDelegate (ImageLoadFileDelegate delegate=ImageLoadFileDelegate())

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

  Set the delegate to load an image from a file.
- template<IrisErrorCode(∗)(const std::string &) FUNC>
  void setLoadImageFileDelegate ()

  Set the delegate to load an image from a file.
- void setNextSubRscId (uint64_t nextSubRscId)

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

- 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 setRemainingStepGetDelegate ()

  Set the delegate to get the remaining steps for this instance.
- void setRemainingStepSetDelegate (RemainingStepSetDelegate delegate=RemainingStepSetDelegate())

  Set the delegate to set the remaining steps for this instance.
- template<typename T , IrisErrorCode(T::∗)(uint64_t, 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 setRemainingStepSetDelegate ()

  Set the delegate to set the remaining steps 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.
- template<IrisErrorCode(∗)(uint64_t &, const std::string &) FUNC>
  void setStepCountGetDelegate ()

  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]

```
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.

8.16.3.5 setDbgStateGetAcknowledgeDelegate() [2/3]

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

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

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getAcknowledgeFlag(bool &debuggable_state_acknowledge);
};

MyClass myInstanceOfMyClass;

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

Template Parameters

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

Parameters

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

8.16.3.6 setDbgStateGetAcknowledgeDelegate() [3/3]

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

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

Usage:

```
    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.

8.16.3.7 setDbgStateSetRequestDelegate() [1/3]

```
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.

8.16.3.8 setDbgStateSetRequestDelegate() [2/3]

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

Set the delegate to set the debuggable state request flag for this instance.

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);
};

MyClass myInstanceOfMyClass;

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

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.

8.16.3.9 setDbgStateSetRequestDelegate() [3/3]

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

Set the delegate to set the debuggable state request flag for this instance.

Usage:

```
iris::IrisErrorCode setRequestFlag(bool request_debuggable_state);

iris::IrisInstanceBuilder *builder = myIrisInstance.getBuilder();
builder->setDbgStateSetRequestDelegate<&setRequestFlag>();
```

Template Parameters

FUNC	Global function to call to set the debuggable state request flag.

8.16.3.10 setDefaultTableReadDelegate() [1/3]

```
void iris::IrisInstanceBuilder::setDefaultTableReadDelegate (
              TableReadDelegate delegate = TableReadDelegate() )  [inline]
```

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

Tables that do not explicitly override the access function using

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

will use this delegate.

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

Usage:

```
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.

8.16.3.11 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]

```
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.

8.16.3.13 setDefaultTableWriteDelegate() [1/3]

```
void iris::IrisInstanceBuilder::setDefaultTableWriteDelegate (
             TableWriteDelegate delegate = TableWriteDelegate() )  [inline]
```

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

Tables that do not explicitly override the access function using

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

will use this delegate.

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

Usage:

```
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.

8.16.3.14 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.
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]

```
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.

8.16.3.16 setExecutionStateGetDelegate() [1/3]

```
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.

8.16.3.17 setExecutionStateGetDelegate() [2/3]

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

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

Usage:

```
class MyClass
{
    ...
    iris::IrisErrorCode getState(bool &execution_enabled);
};

MyClass myInstanceOfMyClass;

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

Template Parameters

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.

8.16.3.18 setExecutionStateGetDelegate() [3/3]

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

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

Usage:

```
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.

8.16.3.19 setExecutionStateSetDelegate() [1/3]

```
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.

8.16.3.20 setExecutionStateSetDelegate() [2/3]

```
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.

8.16.3.21 setExecutionStateSetDelegate() [3/3]

```
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.

8.16.3.22 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

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

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

  Add metadata for an event source.
- void attachTo (IrisInstance ∗irisInstance)

  Attach this IrisInstanceEvent add-on to a specific IrisInstance.
- 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]

```
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().

8.20.3.2 addEventSource() [2/2]

```
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.

8.20.3.3 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

ev↩ BufId	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().
request↩ Id	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.
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.
out←→ SpaceId	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=F↩ LAGS_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

rsc↵ Id	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_CA↩ LL_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.

- 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(∗)(std::vector< ResourceInfo > &) FUNC>
  void setGetParameterInfoDelegate (bool cache_result=true)

  Set the getParameterInfo() 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(∗)(InstantiationResult &) FUNC>
  void setInstantiateDelegate ()

  Set the instantiate() 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(∗)() FUNC>
  void setRequestShutdownDelegate ()

  Set the requestShutdown() 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 IrisSimulationResetContext &) FUNC>
  void setResetDelegate ()

  Set the reset() delegate.
- void setSetParameterValueDelegate (SimulationSetParameterValueDelegate delegate)

  Set the setParameterValue() delegate.
- template<typename T , IrisErrorCode(T::∗)(const InstantiationParameterValue &) METHOD>
  void setSetParameterValueDelegate (T ∗instance)

  Set the setParameterValue() delegate.
- template<IrisErrorCode(∗)(const InstantiationParameterValue &) FUNC>
  void setSetParameterValueDelegate ()

  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]

```
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.

8.28.3.10 setGetParameterInfoDelegate() [2/3]

```
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.

8.28.3.11 setGetParameterInfoDelegate() [3/3]

```
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.

8.28.3.12 setInstantiateDelegate() [1/3]

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

Set the instantiate() delegate.

Parameters

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

8.28.3.13 setInstantiateDelegate() [2/3]

```
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.

8.28.3.14 setInstantiateDelegate() [3/3]

```
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.

8.28.3.15 setRequestShutdownDelegate() [1/3]

```
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.

8.28.3.16 setRequestShutdownDelegate() [2/3]

```
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.
METHOD	A method of class *T* that is a requestShutdown delegate.

Parameters

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

8.28.3.17 setRequestShutdownDelegate() [3/3]

```
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.

8.28.3.18 setResetDelegate() [1/3]

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

Set the reset() delegate.

Parameters

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

8.28.3.19 setResetDelegate() [2/3]

```
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.

8.28.3.20 setResetDelegate() [3/3]

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

Set the reset() delegate.

Set the delegate to a global function.

Template Parameters

FUNC	A function that is a reset delegate.

8.28.3.21 setSetParameterValueDelegate() [1/3]

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

Set the setParameterValue() delegate.

Parameters

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

8.28.3.22 setSetParameterValueDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const InstantiationParameterValue &) METHOD>
void iris::IrisInstanceSimulation::setSetParameterValueDelegate (
             T * instance )  [inline]
```

Set the setParameterValue() delegate.

Set the delegate to call a method in class T.

Template Parameters

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

Parameters

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

8.28.3.23 setSetParameterValueDelegate() [3/3]

```
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.

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.
- 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(∗)(uint64_t &, uint64_t &, bool &) FUNC>
  void setSimTimeGetDelegate ()
  Set the getTime() delegate.
- void setSimTimeRunDelegate (SimulationTimeRunDelegate delegate)
  Set the run() delegate.
- template<typename T , IrisErrorCode(T::∗)() METHOD>
  void setSimTimeRunDelegate (T ∗instance)
  Set the run() delegate.
- template<IrisErrorCode(∗)() FUNC>
  void setSimTimeRunDelegate ()
  Set the run() delegate.
- void setSimTimeStopDelegate (SimulationTimeStopDelegate delegate)
  Set the stop() delegate.
- template<typename T , IrisErrorCode(T::∗)() METHOD>
  void setSimTimeStopDelegate (T ∗instance)
  Set the stop() delegate.
- template<IrisErrorCode(∗)() FUNC>
  void setSimTimeStopDelegate ()
  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]

```
void iris::IrisInstanceSimulationTime::setSimTimeGetDelegate (
            SimulationTimeGetDelegate delegate ) [inline]
```

Set the getTime() delegate.

Parameters

delegate	A delegate that is called to get the current simulation time.

8.29.3.5 setSimTimeGetDelegate() [2/3]

```
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.

8.29.3.6 setSimTimeGetDelegate() [3/3]

```
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.

8.29.3.7 setSimTimeRunDelegate() [1/3]

```
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.

8.29.3.8 setSimTimeRunDelegate() [2/3]

```
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.

8.29.3.9 setSimTimeRunDelegate() [3/3]

```
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.

8.29.3.10 setSimTimeStopDelegate() [1/3]

```
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.

8.29.3.11 setSimTimeStopDelegate() [2/3]

```
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.

8.29.3.12 setSimTimeStopDelegate() [3/3]

```
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.

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.
- template<typename T >
  void getParameter (const std::string &name, T &value)

  Get the value of an instantiation parameter.
- void getParameter (const std::string &name, std::vector< uint64_t > &value)

  Get the value of a large numeric instantiation parameter.
- uint64_t getRecommendedInstanceFlags () const

  Get the flags to use when registering the instance being instantiated.
- IrisInstantiationContext ∗ getSubcomponentContext (const std::string &child_name)

  Get an IrisInstanceContext pointer for a subcomponent instance.

- **IrisInstantiationContext** (IrisConnectionInterface ∗connection_interface_, InstantiationResult &result_↩ , const std::vector< ResourceInfo > &param_info_, const std::vector< InstantiationParameterValue > &param_values_, const std::string &prefix_, const std::string &component_name_, uint64_t instance_flags↩ _)
- void void void void parameterError (const std::string &code, const std::string &parameterName, const char ∗format,...) INTERNAL_IRIS_PRINTF(4

  Add an error to the InstantiationResult.
- void void parameterWarning (const std::string &code, const std::string &parameterName, const char ∗format,...) INTERNAL_IRIS_PRINTF(4

  Add a warning to the InstantiationResult.
- void warning (const std::string &code, const char ∗format,...) INTERNAL_IRIS_PRINTF(3

  Add a warning to the InstantiationResult.

8.32.1 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]

```
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.

8.32.2.5 getParameter() [2/2]

```
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.

8.32.2.6 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 (uint64_t value)

  Set the default value for a numeric parameter.
- IrisParameterBuilder & setDefault (const std::vector< uint64_t > &value)

  Set the default value for a numeric parameter.
- IrisParameterBuilder & setDefaultFloat (double value)

  Set the default value for a numericFp parameter.
- IrisParameterBuilder & setDefaultSigned (int64_t value)

  Set the default value for a numericSigned parameter.
- IrisParameterBuilder & setDefaultSigned (const std::vector< uint64_t > &value)

  Set the default value for a numericSigned parameter.
- IrisParameterBuilder & setDescr (const std::string &description)

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

  Set the `format` field.
- IrisParameterBuilder & setHidden (bool hidden)

  Set the resource to hidden !
- IrisParameterBuilder & setInitOnly (bool value=true)

  Set the `initOnly` field.
- IrisParameterBuilder & setMax (uint64_t max)

Set the `max` field.

- IrisParameterBuilder & setMax (const std::vector< uint64_t > &max)

  Set the `max` field.
- IrisParameterBuilder & setMaxFloat (double max)

  Set the `max` field for floating-point parameters.
- IrisParameterBuilder & setMaxSigned (int64_t max)

  Set the `max` field.
- IrisParameterBuilder & setMaxSigned (const std::vector< uint64_t > &max)

  Set the `max` field.
- IrisParameterBuilder & setMin (uint64_t min)

  Set the `min` field.
- IrisParameterBuilder & setMin (const std::vector< uint64_t > &min)

  Set the `min` field.
- IrisParameterBuilder & setMinFloat (double min)

  Set the `min` field for floating-point parameters.
- IrisParameterBuilder & setMinSigned (int64_t min)

  Set the `min` field.
- IrisParameterBuilder & setMinSigned (const std::vector< uint64_t > &min)

  Set the `min` field.
- IrisParameterBuilder & setName (const std::string &name)

  Set the `name` field.
- IrisParameterBuilder & setRange (uint64_t min, uint64_t max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRange (const std::vector< uint64_t > &min, const std::vector< uint64_t > &max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRangeFloat (double min, double max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRangeSigned (int64_t min, int64_t max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRangeSigned (const std::vector< uint64_t > &min, const std::vector< uint64_t > &max)

  Set both the `min` field and the `max` field.
- IrisParameterBuilder & setRwMode (const std::string &rwMode)

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

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

  Set a boolean tag for this parameter resource.
- IrisParameterBuilder & setTag (const std::string &tag, const IrisValue &value)

  Set a tag for this parameter resource.
- IrisParameterBuilder & setTopology (bool value=true)

  Set the `topology` field.
- IrisParameterBuilder & setType (const std::string &type)

  Set the type of this parameter.

8.33.1 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 (
            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.

8.33.3.6 setDefault() [3/3]

```
IrisParameterBuilder& iris::IrisParameterBuilder::setDefault (
            const std::vector< uint64_t > & value )  [inline]
```

Set the default value for a numeric parameter.

Use this variant for values that are >= 2∗∗64.

Parameters

value	The `defaultData` field of the ParameterInfo object.

Returns

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

8.33.3.7 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 (
             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.

8.33.3.9 setDefaultSigned() [2/2]

```
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.

8.33.3.10 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 (
            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.

8.33.3.15 setMax() [2/2]

```
IrisParameterBuilder& iris::IrisParameterBuilder::setMax (
            const std::vector< uint64_t > & max )  [inline]
```

Set the `max` field.

Use this variant to set values that are >= 2**64.

Parameters

max	The `max` field of the ParameterInfo object.

Returns

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

8.33.3.16 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 (
            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.

8.33.3.18 setMaxSigned() [2/2]

```
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.

8.33.3.19 setMin() [1/2]

```
IrisParameterBuilder& iris::IrisParameterBuilder::setMin (
            uint64_t min )  [inline]
```

Set the `min` field.

Parameters

min	The `min` field of the ParameterInfo object.

Returns

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

8.33.3.20 setMin() [2/2]

```
IrisParameterBuilder& iris::IrisParameterBuilder::setMin (
            const std::vector< uint64_t > & min )  [inline]
```

Set the `min` field.

Use this variant to set values that are >= 2**64.

Parameters

min	The `min` field of the ParameterInfo object.

Returns

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

8.33.3.21 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 (
            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.

8.33.3.23 setMinSigned() [2/2]

```
IrisParameterBuilder& iris::IrisParameterBuilder::setMinSigned (
            const std::vector< uint64_t > & min )  [inline]
```

Set the `min` field.

This implies that the parameter type is "numericSigned". Use this variant for signed values that are out of range for int64_t.

Parameters

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 (
             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.

8.33.3.26 setRange() [2/2]

```
IrisParameterBuilder& iris::IrisParameterBuilder::setRange (
            const std::vector< uint64_t > & min,
            const std::vector< uint64_t > & max )  [inline]
```

Set both the `min` field and the `max` field.

Use this variant to set values that are >= 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.

8.33.3.27 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 (
            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.

8.33.3.29 setRangeSigned() [2/2]

```
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.

8.33.3.30 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

sub← RscId	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.
- 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, uint64_t, uint64_t, const AttributeValueMap &, MemoryReadResult &) FUNC>
  MemorySpaceBuilder & setReadDelegate ()

  Set the delegate to read this memory space.
- 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, const IrisValueMap &, const std::vector< std::string > &, IrisValueMap &) FUNC>
  MemorySpaceBuilder & setSidebandDelegate ()

  Set the delegate to read sideband information.
- MemorySpaceBuilder & setWriteDelegate (MemoryWriteDelegate delegate)

  Set the delegate to write to this memory space.
- template<typename T , IrisErrorCode(T::∗)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t ∗, MemoryWriteResult &) METHOD>
  MemorySpaceBuilder & setWriteDelegate (T ∗instance)

  Set the delegate to write to this memory space.
- template<IrisErrorCode(∗)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64_t, const AttributeValueMap &, const uint64_t ∗, MemoryWriteResult &) FUNC>
  MemorySpaceBuilder & setWriteDelegate ()

  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]

```
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.

8.39.2.11 setReadDelegate() [2/3]

```
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.

8.39.2.12 setReadDelegate() [3/3]

```
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.

8.39.2.13 setSidebandDelegate() [1/3]

```
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.

8.39.2.14 setSidebandDelegate() [2/3]

```
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.

8.39.2.15 setSidebandDelegate() [3/3]

```
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.

8.39.2.16 setWriteDelegate() [1/3]

```
MemorySpaceBuilder& iris::IrisInstanceBuilder::MemorySpaceBuilder::setWriteDelegate (
            MemoryWriteDelegate delegate )  [inline]
```

Set the delegate to write to this memory space.

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

See also

IrisInstanceBuilder::setDefaultMemoryWriteDelegate

Parameters

delegate	MemoryWriteDelegate object.

Returns

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

8.39.2.17 setWriteDelegate() [2/3]

```
template<typename T , IrisErrorCode(T::*)(const MemorySpaceInfo &, uint64_t, uint64_t, uint64←
_t, const AttributeValueMap &, const uint64_t *, MemoryWriteResult &) METHOD>
```

```
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.

8.39.2.18 setWriteDelegate() [3/3]

```
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.

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.
- ParameterBuilder & setDefaultData (uint64_t value)

  Set the `default` value for numeric parameter to a value <= 64 bit.
- template< typename T >
  ParameterBuilder & setDefaultData (std::initializer_list< T > &&t)

  Set the `default` value for wide numeric parameters.
- template< typename Container >
  ParameterBuilder & setDefaultDataFromContainer (const Container &container)

  Set the `default` value for wide numeric parameters.
- ParameterBuilder & setDefaultString (const std::string &defaultString)

  Set the `defaultData` field for wide numeric parameters (bitWidth > 64 bit).
- ParameterBuilder & setDescr (const std::string &description)

  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.
- ParameterBuilder & setMax (uint64_t value)

  Set the `max` field to a value <= 64 bit.
- template< typename T >
  ParameterBuilder & setMax (std::initializer_list< T > &&t)

  Set the `max` field for wide numeric parameters.
- template< typename Container >
  ParameterBuilder & setMaxFromContainer (const Container &container)

  Set the `max` field for wide numeric parameters.
- ParameterBuilder & setMin (uint64_t value)

  Set the `min` field to a value <= 64 bit.
- template< typename T >
  ParameterBuilder & setMin (std::initializer_list< T > &&t)

  Set the `min` field for wide numeric parameters.
- 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.

- ParameterBuilder & setReadDelegate (ResourceReadDelegate readDelegate)

    Set the delegate to read the resource.

- template<IrisErrorCode(∗)(const ResourceInfo &, ResourceReadResult &) FUNC>
  ParameterBuilder & setReadDelegate ()

    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.

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

    Set the delegate to write the resource.

- ParameterBuilder & setWriteDelegate (ResourceWriteDelegate writeDelegate)

    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]

```
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.

8.41.2.8 setDefaultData() [2/2]

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

Set the `default` value for wide numeric parameters.

This function accepts a braced initializer-list and is otherwise 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.

8.41.2.9 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]

```
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.

8.41.2.16 setMax() [2/2]

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

Set the `max` field for wide numeric parameters.

This function accepts a braced initializer-list and is otherwise 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.

8.41.2.17 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]

```
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.

8.41.2.19 setMin() [2/2]

```
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.

8.41.2.20 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]

```
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.

8.41.2.24 setReadDelegate() [2/3]

```
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.

8.41.2.25 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

sub↩ RscId	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,
            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.

8.41.2.29 setTag() [2/2]

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

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

Parameters

tag	The name of the tag to set.

Returns

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

8.41.2.30 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<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.

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<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.

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.
- 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<IrisErrorCode(∗)(const ResourceInfo &, ResourceReadResult &) FUNC>
  RegisterBuilder & setReadDelegate ()

  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, const IrisValue &value)

  Set a tag to the specified value.
- RegisterBuilder & setTag (const std::string &tag)

  Set the named boolean tag to true (e.g. isPc)
- RegisterBuilder & setType (const std::string &type)

  Set the `type` field.
- template<typename T , IrisErrorCode(T::∗)(const ResourceInfo &, const ResourceWriteValue &) METHOD>
  RegisterBuilder & setWriteDelegate (T ∗instance)

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

  Set the delegate to write the resource.
- template<IrisErrorCode(∗)(const ResourceInfo &, const ResourceWriteValue &) FUNC>
  RegisterBuilder & setWriteDelegate ()

  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<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.

8.43.2.18 setReadDelegate() [2/3]

```
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.

8.43.2.19 setReadDelegate() [3/3]

```
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.

8.43.2.20 setResetData() [1/2]

```
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.

8.43.2.21 setResetData() [2/2]

```
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.

8.43.2.22 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

sub←↩ RscId	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<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.

8.43.2.30 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<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.

8.43.2.32 setWriteMask() [1/2]

```
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.

8.43.2.33 setWriteMask() [2/2]

```
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.

8.43.2.34 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_CA←LL_EXTENSION` event.
- std::vector< uint8_t > readData (uint64_t fDes, size_t max_size=0, uint64_t flags=semihost::DEFAULT)

  Read data for a given file descriptor.
- std::pair< bool, uint64_t > semihostedCall (uint64_t operation, uint64_t parameter)

  Allow a client to perform a semihosting extension defined by operation and parameter.
- **SemihostingManager** (IrisInstanceSemihosting ∗inst_semihost_)
- void **unblock** ()
- bool **writeData** (uint64_t fDes, const uint8_t ∗data, size_t size)
- bool **writeData** (uint64_t fDes, const std::vector< uint8_t > &data)

8.46.1 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.
- TableBuilder & setReadDelegate (TableReadDelegate delegate)

  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.
- template<IrisErrorCode(∗)(const TableInfo &, uint64_t, uint64_t, TableReadResult &) FUNC>
  TableBuilder & setReadDelegate ()

  Set the delegate to read the table.
- TableBuilder & setWriteDelegate (TableWriteDelegate delegate)

  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.
- template<IrisErrorCode(∗)(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
  TableBuilder & setWriteDelegate ()

  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]

```
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.

8.48.2.11 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]

```
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.

8.48.2.13 setWriteDelegate() [1/3]

```
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.

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]

```
template<IrisErrorCode(*)(const TableInfo &, const TableRecords &, TableWriteResult &) FUNC>
TableBuilder& iris::IrisInstanceBuilder::TableBuilder::setWriteDelegate ( )  [inline]
```

Set the delegate to write to the table.

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

See also

IrisInstanceBuilder::setDefaultTableWriteDelegate

Template Parameters

FUNC	A table write delegate function.

Returns

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

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

- IrisInstanceBuilder.h

8.49 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

9.2 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

9.2.1 Detailed Description

Iris client which supports multiple methods to connect to other Iris executables.

Date

Copyright ARM Limited 2015-2022 All Rights Reserved.

9.3 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.

9.3.1 Detailed Description

Generic command line parser.

Copyright

Copyright (C) 2020-2022 Arm Limited. All rights reserved.

9.4 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, **AR↩M_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, **A↩RM_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, **AAR↩ CH64_X6** = 0xb700000006,
AARCH64_X7 = 0xb700000007, **AARCH64_X8** = 0xb700000008, **AARCH64_X9** = 0xb700000009, **AAR↩ CH64_X10** = 0xb70000000a,
AARCH64_X11 = 0xb70000000b, **AARCH64_X12** = 0xb70000000c, **AARCH64_X13** = 0xb70000000d, **AARCH64_X14** = 0xb70000000e,
AARCH64_X15 = 0xb70000000f, **AARCH64_X16** = 0xb700000010, **AARCH64_X17** = 0xb700000011, **A↩ ARCH64_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, **AA↩ RCH64_V1** = 0xb700000041,
AARCH64_V2 = 0xb700000042, **AARCH64_V3** = 0xb700000043, **AARCH64_V4** = 0xb700000044, **AAR↩ CH64_V5** = 0xb700000045,
AARCH64_V6 = 0xb700000046, **AARCH64_V7** = 0xb700000047, **AARCH64_V8** = 0xb700000048, **AAR↩ CH64_V9** = 0xb700000049,
AARCH64_V10 = 0xb70000004a, **AARCH64_V11** = 0xb70000004b, **AARCH64_V12** = 0xb70000004c, **AARCH64_V13** = 0xb70000004d,
AARCH64_V14 = 0xb70000004e, **AARCH64_V15** = 0xb70000004f, **AARCH64_V16** = 0xb700000050, **A↩ ARCH64_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 }

9.4.1 Detailed Description

Constants for the register.canonicalRnScheme "ElfDwarf" for architecture Arm.

Date

Copyright ARM Limited 2019. All Rights Reserved.

9.5 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.

9.5.1 Detailed Description

A utility class for emitting Iris events.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

9.6 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

9.6.1 Detailed Description

Central instance which lives in the simulation engine and distributes all Iris messages.

Date

Copyright ARM Limited 2014-2019 All Rights Reserved.

The IrisGlobalInstance lives in the simulation engine. It contains all central data structures like the instance registry. It is responsible for distributing Iris messages to all in-process instances and to the IrisTcpServer.

9.7 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.

9.7.1 Detailed Description

Boilerplate code for an Iris instance, including clients and components.

Copyright

Copyright (C) 2015-2022 Arm Limited. All rights reserved.

The IrisInstance class provides infrastructure that is:

- Necessary for all Iris instances.
- Useful for Iris components.
- Useful for Iris clients.

Note

Using this class to implement a correct Iris interface is optional. This class does not form an interface between instances. It just forms an interface between itself and the code of an instance.

This class is useful for, and used by, both components and clients.

9.7.2 Typedef Documentation

9.7.2.1 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"
      );
```

9.8 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.

9.8.1 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.

9.8.2 Typedef Documentation

9.8.2.1 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.

9.8.2.2 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.

9.9 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.

9.9.1 Detailed Description

A high level interface to build up functionality on an IrisInstance.

Copyright

Copyright (C) 2016-2019 Arm Limited. All rights reserved.

9.10 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.

9.10.1 Detailed Description

Checkpoint add-on to IrisInstance.

Date

Copyright ARM Limited 2019 All Rights Reserved.

9.10.2 Typedef Documentation

9.10.2.1 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.

9.10.2.2 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.

9.11 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.

9.11.1 Detailed Description

IrisInstance add-on to implement debuggableState functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.11.2 Typedef Documentation

9.11.2.1 DebuggableStateGetAcknowledgeDelegate

```
typedef IrisDelegate<bool&> iris::DebuggableStateGetAcknowledgeDelegate
```

Interface to stop the simulation time progress.

```
IrisErrorCode getAcknowledge(bool &acknowledge_out);
```

9.11.2.2 DebuggableStateSetRequestDelegate

```
typedef IrisDelegate<bool> iris::DebuggableStateSetRequestDelegate
```

Delegate to set the debuggable-state-request flag.

```
IrisErrorCode setRequest(bool request);
```

9.12 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.

9.12.1 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.

9.13 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.

9.13.1 Detailed Description

Event add-on to IrisInstance.

Copyright

Copyright (C) 2016-2021 Arm Limited. All rights reserved.

The IrisInstanceEvent class:

- Implements all event-related Iris functions.
- Maintains and provides event source metadata.
- Converts between Iris event functions (event∗()) and various C++ access functions.

9.13.2 Typedef Documentation

9.13.2.1 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.

9.14 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.

9.14.1 Detailed Description

A helper class to build instantiation parameter metadata.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.15 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.

9.15.1 Detailed Description

Image-loading add-on to IrisInstance and image-loading callback add-on to the caller.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

The IrisInstanceImage class:

- Implements all image-loading Iris functions.
- Maintains and provides image metadata, for example path, instanceSideFile, rawAddr.
- Converts between Iris image-loading functions (image_load∗()) and various C++ access functions.

9.15.2 Typedef Documentation

9.15.2.1 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.

9.15.2.2 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.

9.16 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.

9.16.1 Detailed Description

Memory add-on to IrisInstance.

Copyright

Copyright (C) 2015 Arm Limited. All rights reserved.

The IrisInstanceMemory class:

- Implements all memory-related Iris functions.
- Feeds memory-related properties (memory.∗) to instance_getProperties() of the associated IrisInstance.
- Provides infrastructure that is useful for Iris clients.
- Maintains and provides memory meta information (memory spaces, address translations, sideband information).
- Converts between Iris memory access functions (memory_read()) and various C++ access functions.

9.16.2 Typedef Documentation

9.16.2.1 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.

9.16.2.2 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.

9.16.2.3 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.

9.16.2.4 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.

9.17 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.

9.17.1 Detailed Description

Per-instance execution control add-on to IrisInstance.

Copyright

Copyright (C) 2016 Arm Limited. All rights reserved.

Implements all per-instance execution control-related Iris functions.

9.17.2 Typedef Documentation

9.17.2.1 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.

9.17.2.2 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.

9.18 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)

9.18.1 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.

9.18.2 Typedef Documentation

9.18.2.1 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.

9.18.2.2 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.

9.18.3 Function Documentation

9.18.3.1 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.

9.18.3.2 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.

9.19 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

9.19.1 Detailed Description

IrisInstance add-on to implement semihosting functionality.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.20 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_ENT←
ER**, **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_PHA←
SE_TERMINATE**, **IRIS_SIM_PHASE_TERMINATE_LEAVE**,
IRIS_SIM_PHASE_NUM }

  List of IRIS_SIMULATION_PHASE events.

9.20.1 Detailed Description

IrisInstance add-on to implement simulation_* functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.20.2 Typedef Documentation

9.20.2.1 SimulationGetParameterInfoDelegate

```
typedef IrisDelegate<std::vector<ResourceInfo>&> iris::SimulationGetParameterInfoDelegate
```

Delegate to get a list of parameter information.

```
IrisErrorCode getInstantiationParameterInfo(std::vector<ResourceInfo> &parameters_out)
```

9.20.2.2 SimulationInstantiateDelegate

```
typedef IrisDelegate<InstantiationResult&> iris::SimulationInstantiateDelegate
```

Delegate to instantiate the simulation.

```
IrisErrorCode instantiate(InstantiationResult &result_out)
```

9.20.2.3 SimulationRequestShutdownDelegate

```
typedef IrisDelegate iris::SimulationRequestShutdownDelegate
```

Delegate to request that the simulation be shut down.

```
IrisErrorCode requestShutdown()
```

9.20.2.4 SimulationResetDelegate

```
typedef IrisDelegate<const IrisSimulationResetContext&> iris::SimulationResetDelegate
```

Delegate to reset the simulation.

```
IrisErrorCode reset(const IrisSimulationResetContext &)
```

9.20.2.5 SimulationSetParameterValueDelegate

```
typedef IrisDelegate<const InstantiationParameterValue&> iris::SimulationSetParameterValueDelegate
```

Delegate to set the value of an instantiation parameter.

```
IrisErrorCode setInstantiationParameterValue(const InstantiationParameterValue &value)
```

9.21 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_BREAKP
iris::TIME_EVENT_TRACE_COUNTER_OVERFLOW }

    The reasons why the simulation time stopped.

9.21.1 Detailed Description

IrisInstance add-on to implement simulationTime functions.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.21.2 Typedef Documentation

9.21.2.1 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);
```

9.21.2.2 SimulationTimeRunDelegate

```
typedef IrisDelegate iris::SimulationTimeRunDelegate
```

Delegate to resume the simulation time progress.

```
IrisErrorCode run();
```

9.21.2.3 SimulationTimeStopDelegate

```
typedef IrisDelegate iris::SimulationTimeStopDelegate
```

Delegate to stop the simulation time progress.

```
IrisErrorCode stop();
```

9.21.3 Enumeration Type Documentation

9.21.3.1 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.

9.22 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.

9.22.1 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.

9.22.2 Typedef Documentation

9.22.2.1 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.

9.22.2.2 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.

9.22.2.3 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.

9.23 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.

9.23.1 Detailed Description

Table add-on to IrisInstance.

Copyright

The IrisInstanceTable class implements all table-related Iris functions.

9.23.2 Typedef Documentation

9.23.2.1 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.

9.23.2.2 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.

9.24 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.

9.24.1 Detailed Description

Helper class used to instantiate Iris instances from generic factories.

Copyright

9.25 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.

9.25.1 Detailed Description

Helper class to construct instantiation parameters.

Copyright

9.26 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.

9.26.1 Detailed Description

A generic plug-in factory for instantiating plug-in instances.

Copyright

Copyright (C) 2017 Arm Limited. All rights reserved.

9.26.2 Macro Definition Documentation

9.26.2.1 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.

9.27 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.

9.27.1 Detailed Description

Utility classes for emitting register read and register update events.

Copyright

9.28 IrisTcpClient.h File Reference

IrisTcpClient Type alias for IrisClient.

```
#include "iris/IrisClient.h"
```

Typedefs

- using iris::IrisTcpClient = IrisClient
  Alias for backward compatibility.

9.28.1 Detailed Description

IrisTcpClient Type alias for IrisClient.

Date