ORIMOS

SONARIS/Framework

Custom Component Programming Guide

Version 1.4.3

ORIMOS S.A.

Innere Gueterstrasse 4
6304 Zug (Switzerland)

Phone +41-41-725-3570
Fax +41-41-725-3579

Web www.ORIMOS.com
Email info@ORIMOS.com

ORIMOS Group
Berlin · Frankfurt · London · Zurich

Contents

Preface

Product Conventions

SONARIS/Framework is case sensitive. For example, if an application is being run with the name *MyApp* then client programs must connect to it using *MyApp* as the application name; *myapp*, *MYAPP* or variations are not acceptable. Application database data source names may or may not be case sensitive, and this depends on the database system and operating system being used. Case sensitivity applies to anything that has a name.

Typographic Conventions

The following typesetting conventions apply throughout this guide:

- When referring to programs `this font` is used. Programs and libraries are referred to without any operating system specific file extension. For example, the program `safcontrol.exe` as supplied on a Windows system is referred to simply as `safcontrol`.
- When referring to files or directories `this font` is used. Note that the directory path separator is always **/.**
- Examples of the use of, and output from, command line programs are shown in `this font`. In cases where long lines have to be split over a number of document lines to keep them readable the \ character denotes continuation on a following line. For example:

```
>safbase -m FROG MyApp \
SAFSQLSERVER;UID=sa;DATABASE=saf_myapp safw32odbc
```

- When showing samples of code `this font` is used.
- When referring to items on dialogs, menu items or other named entities this font is used. Sub-menus and menu items are separated by the ▷ character (e.g. Programs ▷ SONARIS)

1 Introduction

Beyond providing the ability to create application hierarchies based on standard supplied components, **SONARIS/Framework** also provides an API with which it is possible to create your own custom *SONARIS Application Objects* (SAOs) for use in applications. By placing your own code in an SAO you can take advantage of the flexibility of the Framework to execute your code, supply information for your code to process, and relay the results to other components or **SONARIS/Framework** client applications. SAOs are C++ classes which implement certain **SONARIS/Framework** interfaces to allow them to interact with the Framework. However, your SAOs may call code written in any language which can be called from C++ (e.g. C, Fortran).

1.1 Prerequisites

Before continuing, make sure you have read the *SONARIS/Framework - Overview and Concepts* and the *SONARIS/Framework - Application Programming Guide*. In order to view and compile the example code presented in this text, please install the following **SONARIS/Framework** subsets (refer to the *SONARIS/Framework - Installation and Maintenance Guide* for details):

- **Runtime Environment**
- **Server Environment**
- **Software Development Kit**

1.2 The SONARIS/Framework Programming Environment

There are a number of techniques and concepts which are common to all areas of **SONARIS/Framework** programming whether you are creating **SONARIS/Framework** clients or your own custom SAOs.

1.2.1 Interfaces

The majority of the services made available by the **SONARIS/Framework** APIs are accessed through *interfaces*. In C++, an interface is a class which just contains pure virtual methods. Java has interfaces as part of the language. COM also has the concept of interfaces built in.

The interfaces used in the **SONARIS/Framework** APIs fall into two groups:

- *active* interfaces where **SONARIS/Framework** provides the implementation of the interface and methods to create and destroy instances of that implementation.
- *listener* interfaces for which the user must create an implementation on which **SONARIS/Framework** can then make calls to inform the user of asynchronous events. When such an event occurs **SONARIS/Framework** calls the appropriate method in the user's implementation of the interface; these methods are said to be "called back".

Object Management

In the C++ APIs, all the active interfaces have a create() method which the user should call to get a new instance of an object which implements the interface, and a corresponding destroy() method which should be called when the user no longer requires the instance.

SONARIS/Framework provides a means to automate the creation and deletion of these objects within a block of code.The SAF::AutoDestroy template creates an instance of the specified interface and then destroys it when it goes out of scope. This has the additional benefit of making the deletion of the object exception proof.

Backwards Compatibility

Interfaces are at the heart of **SONARIS/Framework**s backwards compatibility strategy as they allow the underlying `safsystem` DLL to be upgraded without the need to re-compile or re-link any programs that use the **SONARIS/Framework** APIs.

Active interfaces support backwards compatibility because **SONARIS/Framework** provides their implementation. This means that the implementation of the interface can be changed without affecting any already compiled code. The interface can also be safely extended as long as the new methods are added at the end of the interface definition.

Providing backwards compatibility for listener interfaces is more complicated. If a listener interface has a method added (it must be at the end) then **SONARIS/Framework** must guarantee not the call the method if the user's implementation of the interface does not contain that method. **SONARIS/Framework** is able to make this decision on the basis of the API version numbers automatically provided to it when the user initialises the API. **SONARIS/Framework** knows what "shape" the listener interfaces were for any previous version of the API and so knows whether any particular user's implementation of an interface contains the relevant method.

1.2.2 Data Types and Status

Individual items of data used by **SONARIS/Framework** are held in an implementation of the ISAFValue active interface. An ISAFValue can hold any of the data types supported by **SONARIS/Framework** as both single values and arrays.

An ISAFValue also has a status associated with the data it holds (if any) which indicates the quality of the data or the reason why it does not have any data. Only if the status is SAF_VALID or SAF_STALE does the ISAFValue have data available.

1.2.3 Compilation and Linking

Refer to Appendix E for details of compiling code for use with **SONARIS/Framework**.

2 Creating SAOs

Creating SAOs is a matter of following these steps:

1. Define the type and input details of your new SAO(s).
2. Provide an implementation of the IOperate interface for each new SAO.
3. Provide an implementation of the ISAOLibrary interface which will act as a factory object to create new instances of your SAO(s).
4. Compile and link the SAOs and the factory object to create a library (a DLL on windows, a shared library on Unix).
5. Generate a Library Description File for your new SAO library using the `saolib2xml` program.

The following text will describe the execution of these steps with reference to a simple example SAO library (MySAOLib) which implements the following SAOs:

- MyPlus - a simple arithmetic SAO which maintains its value as the sum of its two inputs demonstrating the fundamentals of creating SAOs.
- MyDelta - an SAO which maintains its value as the difference between the current and previous values of its single input.
- MySwitch - an SAO which sets its value to that of its *n*th input where *n* is the value of its first input demonstrating the definition and code for an SAO with an unlimited number of inputs.
- MyWrapper - an SAO that wraps an existing class which encapsulates multiple values and makes those values available to other SAOs by implementing the ISAOResource interface.
- MyReso - an SAO that takes its value from a resource owned by another SAO (e.g. MyWrapper).
- MyInputIndex - an SAO that uses a helper to make managing larger numbers of inputs easier and more efficient by reacting only to those which have changed.

All the interfaces whose implementation is required in order to create custom SAOs appear in the `isaflibrary.h` header file.

2.1 SAO Composition

Each SAO instance managed by a container within an application hierarchy is composed of two parts, a base object created by the container in which the SAO has been placed, and a user object created by an ISAOLibrary factory object supplied by a DLL. The base object is responsible for all interaction with the Framework, including -

- The name and type of the SAO.

- Storage of the SAO's current value and status.
- Management of links to the SAO's parent and children in a hierarchy.
- Management of input connections from other SAOs.

If any of the above properties of the base object is changed as a result of events within the Framework, the appropriate method of the accompanying user object is called to allow user supplied code to respond to the change. Within the implementation of these methods, the user object has access to a large number of methods made available to it by the public interfaces of the base object (ISAO) and the container (ISOC). These allow it to modify its own value and status, access other SAOs within the container, and use the container to provide services such as creating other SAOs, connecting and disconnecting SAO inputs, logging etc.

2.2 SAO Definition

Any SAO used by **SONARIS/Framework** must have a formal definition of its type, details of its inputs and some description of what it does and how it should be connected within an application, and an implementation of a minimum amount of code which determines its behaviour. The Framework will make use of this in determining -

- How the SAO may be connected to the inputs of other SAOs
- What other SAOs may serve as inputs for this SAO
- When calls should be made on the SAO to allow it to update its value or status, or to perform some other work within the Framework
- What code should be invoked by those calls

The information the Framework requires in order to use an SAO is supplied as instances of classes which implement the ISAODefinition and ISAOValidation interfaces, while the code which implements the specific behaviour of the SAO is supplied through an implementation of the IOperate interface.

ISAODefinition provides methods which are used by the Framework to get basic details about the SAO (e.g. ISAODefinition::getValueType() for the type of the underlying value stored by the SAO, ISAODefinition::getMaxInputs() for the number of inputs the SAO has etc.). One instance of an implementation of ISAODefinition corresponds to a single SAO type. Each instance of ISAODefinition refers (using a pointer) to an array of instances of ISAOValidation. These provide methods which describe each of the SAO's inputs (e.g. ISAOValidation::getValueType() for the type of an SAO expected to be connected to a given input).

The implementations of these two classes need only provide a constructor by which values may be stored which can then the supplied as results to calls on the interface methods. To assist with this, two such implementations are supplied in the `safhelper.h` header file. These are SAODef and SAOInputRule.

For convenience the instances of SAODef and SAOInputRule are often created within static methods of the class which implements IOperate. In addition, it is usually desirable to enumerate

the indexes of the inputs of the SAO to allow for their clear use in the code which implements the methods of IOperate.

Following this general approach, the MyPlus class definition (in `myplus.h`) is as follows:

```
class MyPlus : public IOperate {
    public:
        /// An enumeration of the input indexes of this SAO.
        typedef enum {
            FIRST_INPUT = 0,
            SECOND_INPUT = 1
        } Input_t;

        /// Static method for access to the definition of this SAO.
        static const ISAODefinition *getDefinition();

        /// Static method for access to the input rules of this SAO.
        static const ISAOValidation * const * getInputRules();

        /// Constructor - nothing special to do here.
        MyPlus() { }
```

By referring to the implementations of MyPlus::getDefinition() and MyPlus::getInputRules() in `myplus.cpp` it can be seen how instances of the SAODef and SAOInputRule helper classes can be used to provide the details of the SAO definition.

```
const ISAODefinition *MyPlus::getDefinition() {
    //  Supply a definition of this SAO which includes the input rules
    //  and other information which describes how to use it using
    //  an instance of SAODefinition from safhelper.h.
    static SAODef definition(
        "MyPlus",                       // name
        "Adds two numbers together.",   // what it does
        SAF_DOUBLE,                     // storage type for the result
        "Simple Examples",          // category information
        2,                          // must have 2 inputs
        2,                          // perform validation for both inputs
        MyPlus::getInputRules()     // validation rules
    );
    return &definition;
}

const ISAOValidation * const *MyPlus::getInputRules() {
    //  Supply an array of rules for the inputs to this SAO using
    //  instances of SAOInputRule from safhelper.h.
    static ISAOValidation *inputRules[] = {
        new SAOInputRule(SAF_DOUBLE, SAF_MANDATORY, "First number to add."),
        new SAOInputRule(SAF_DOUBLE, SAF_MANDATORY, "Second number to add.")
    };
    return inputRules;
}
```

Note that each input rule definition needs to be defined as SAF_MANDATORY or SAF_OPTIONAL depending on whether the SAO requires another SAO to be connected to the input and to have a usable value before the Framework will call its IOperate::operate() method. Inputs

defined using a third option, SAF_RESERVED, are reserved for private internal use by the SAO. The options available here are covered in more detail in the next section. The MyPlus SAO requires both inputs to be connected and available before a calculation can be performed and so declares both to be SAF_MANDATORY.

2.3 SAO Implementation

Having created the SAO description, the next step is to complete the implementation of the IOperate interface. The methods of this interface are called by the Framework when the value of the SAO may be required to change as a result of one of the following events:

2.3.1 Initialisation

Final initialisation of a newly created SAO instance is performed once the parent of the SAO instance has been set. At this point the Framework will make a call on the IOperate::initialise() method. This allows the SAO to perform any initialisation work it could not complete in its constructor method because it did not yet have access to the ISAO and ISOC interfaces via the base object.

2.3.2 Deletion

When some event within the Framework results in the intended deletion of an SAO instance from the application hierarchy the IOperate::destroy() method is called. This call is made before the base object of the SAO is removed from the hierarchy and deleted in order to allow the SAO to perform any housekeeping for which it is responsible while it still has access to the rest of the Framework via the ISAO and ISOC interfaces of the base object and the container. This may include arranging for its own deletion, possibly in conjunction with the ISAOLibrary factory object which created it.

2.3.3 Name Changes

When some event within the Framework has results in an attempt to change the name of the SAO instance the IOperate::rename() method is called. This call is made before the name change is performed and will supply the proposed new name. The code of the rename() method may take whatever action is appropriate to a name change and return true to allow the name change to proceed, or may disallow the change by returning false.

2.3.4 Input Changes

When the value or status of one of the inputs of the SAO has changed the IOperate::operate() method will be called. The code of the operate() method may take whatever action is appropriate in response to this by examining the SAOs which serve as its inputs, and possibly setting its own value or status. Typically this takes the form of some kind of calculation where a new result is yielded and stored based on the new value or status of the SAO's inputs. If the return value of this method is true the Framework will invoke the operate() methods of SAOs to which this SAO is connected as an input, and will pass the value of the SAO on to any client application which has a subscription to the SAO. You would probably want to do this if your SAO changes its value to allow other SAOs which use it to review their own values. If the return value is false the

Framework may still invoke the operate() method of other SAOs or pass on the value to clients if it detects a change is the SAOs status.

2.3.5 Incoming Values

When a new value is delivered direct to the SAO from a client the IOperate::update() method will be called and will supply the new value (or possibly a list of values). The new value may have come from an external interface, or from another SAO via a call to the ISOC::postValueToSAO() method. The code of the update() method may elect to not take any action for itself, but instead to allow the container to store the value within the SAO's base object for it, and to pass on calls to other affected SAO's operate() methods, subscribed clients etc. This can be done by setting the value of the supplied `handled` parameter to false (meaning it didn't process the supplied value) and returning false (meaning nothing has changed yet). Alternatively, the update() method may take all responsibility for the handling of the supplied values by setting the `handled` parameter to true. The update() method may then choose to use or discard the supplied value. As with the IOperate::operate() method, the return value indicates to the Framework whether any change has taken place to the SAO which should be passed on to other SAOs or clients.

2.3.6 Default Implemtations

In the majority of cases only some of these methods require anything other than a default implementation. In fact a perfectly valid (though not particularly useful) SAO can be created using simply the following:

```
class DoNothingSAO : public IOperate {
    protected:
        //  Destructor is protected - the Framework will indicate when
        //  it is ready to delete SAOs by calling destroy().
        //  Nothing special to do here anyway.
        virtual ~DoNothingSAO() {}

    public:
        //  Constructor - nothing special to do here.
        DoNothingSAO() { }

        //  No special initialisation to do
        virtual void initialise(ISAO& base) { }

        //  Called by the Framework when it wants to delete instances of
        //  this SAO - but we get final control of that, so we delete ourselves.
        virtual void destroy(ISAO& base) {
            delete this;
        }

        //  We will allow ourselves to be renamed, and we don't do anything
        //  as a consequence of the change.
        virtual bool rename(const ISAO& base, const char *name) {
            return true;
        }

        //  Discard any updates sent to us
        virtual bool update(ISAO& base, const IESRUpdate& update,
            bool& handled, bool& last) {
```

```
            handled = true;
            return false;
        }

        //  Don't do anything if our inputs change.
        virtual bool operate(ISAO& base, int count, ISAO **changed) {
            return false;
        }
};
```

For our implementation of the MyPlus SAO, most of the methods will indeed not be anything other than a default implementation. An addition calculation does not need any initialisation, doesn't depend on having a particular name so renames are acceptable and destroy will simply call our destructor. In particular, because we intend to calculate the value of MyPlus in the MyPlus::operate() method, we don't want to allow any other value sent to us to displace that calculated result. To ensure this we will use the implementation of the IOperate::update() method described above which handles values sent to the SAO (rather than allowing the container to handle them for us) but doesn't do anything with them.

As a result, the remainder of the MyPlus class appears as follows:

```
        //  *** Implementation of SAF::IOperate ***
        /// Called by the framework when it wants to delete instances of
        /// this SAO - but we get final control of that.
        virtual void destroy(ISAO& base) {
            delete this;
        }

        /// No special initialisation to do
        virtual void initialise(ISAO& base) {}

        /// Ignore any updates which are sent to us.
        virtual bool update(ISAO& base, const IESRUpdate& update,
            bool& handled, bool last) {
            handled = true;
            return false;
        }

        /// Allow instances of this SAO to be renamed.
        virtual bool rename(const ISAO& base, const char *name) {
            return true;
        }

        /// Take the values of our two inputs, add them together, and
        /// set our value accordingly.
        virtual bool operate(ISAO& base, int count, ISAO **changed);

    protected:
        /// Destructor is protected - the framework will indicate when
        /// it is ready to delete SAOs by calling destroy().
        /// Nothing special to do here anyway.
        virtual ~MyPlus() { }
};
```

The only element of MyPlus where significant code is required is the MyPlus::operate() method. The implementation of this is as follows (refer to the comments in the code for a description of

its behaviour) -

```
bool MyPlus::operate(ISAO& base, int count, ISAO **changed) {
    double op1, op2, result;
    bool valid;

    //  We can go straight ahead and use our inputs - our validation rules
    //  will ensure that this method will only be called if they are present
    //  and fit for use.
    op1 = base.getInput(FIRST_INPUT)->getDouble(valid);
    op2 = base.getInput(SECOND_INPUT)->getDouble(valid);
    result = op1 + op2;

    //  We don't need to worry about our status - it will always become
    //  SAF_VALID as a consequence of calling setValue().
    //  Our return value will be true to show that we have set our value.
    return base.setValue(result);
}
```

In fact we could make the code for such a simple SAO even more concise, relying on the fact that the presence and validity of type and value of the inputs is assured by their definition as SAF_MANDATORY in the SAO's input rules:

```
bool MyPlus::operate(ISAO& base, int count, ISAO **changed) {
    return base.setValue(base.getInput(FIRST_INPUT)->getDouble() +
        base.getInput(SECOND_INPUT)->getDouble());
}
```

Finally, although we have not chosen to use it here for the sake of presenting at least an initial review of the methods of IOperate, a default implementation of IOperate, SAOIOperate, is also provided in `safhelper.h`.

Code for two other SAOs, MyDelta and MySwitch is supplied along with that for MyPlus. The source for these SAOs appears in `mydelta.h`, `mydelta.cpp`, `myswitch.h` and `myswitch.cpp`. The MyDelta SAO shows how to handle inputs which are defined to be SAF_OPTIONAL, and MySwitch shows an implemention of an SAO with an unlimited number of inputs.

3 Packaging SAOs

3.1 Putting SAOs into the Library

Having created some SAOs, it still remains to create a class to implement the ISAOLibrary interface and act as a factory responsible for the creation of instances of the SAOs. The Framework will need access to an instance of this class in order to make calls on the methods of ISAOLibrary. As with the ISAODefinition described earlier, many of the methods of ISAOLibrary are simply accessor methods which supply some stored piece of information (e.g. ISAOLibrary::getSAFVersion() returns the major and minor numbers for the Framework against which the library was compiled). For convenience, those methods for which a default implementation is reasonable are implented by the SAOLib class supplied in `safhelper.h` leaving the minumum remaining implementation to be supplied.

In particular the principle outstanding method which must be created is ISAOLibrary::createOperate(). This will be called by the Framework when, in the process of adding an SAO to an application, it requires an instance of the user part of an SAO appropriate to a certain functional type, which it identifies by supplying a typeId. In effect this simply means that, if the library claims to offer three SAOs, the Framework will make calls to ISAOLibrary::createOperate() supplying numbers in the range 0 to 2. For clarity within the library class, it is generally desirable to create an enumeration which associates a name for each type of SAO offered with its number in the library. For example, in MySAOLib the SAO types are enumerated in `mysaolib.h` as follows:

The implementation of the ISAOLibrary::createOperate() method in `mysaolib.cpp` is a straightforward switch statement based on the supplied functional typeId:

```
IOperate *MySAOLib::createOperate(ISAO& base, typeId id) {
    switch (id) {
        case MYPLUS:
            return new MyPlus();
        case MYSWITCH:
            return new MySwitch();
        case MYDELTA:
            return new MyDelta();
        case MYWRAPPER:
            return new MyWrapper();
        case MYRESO:
            return new MyReso();
        case MYINPUTINDEX:
            return new MyInputIndex();
        default:
            return NULL;
    }
}
```

In addition to being able to supply instances of the user part of SAOs, the class implementing ISAOLibrary must also be able to supply the definitions of the SAOs in response to calls to the ISAOLibrary::getDefinitions() method. Since we have already created suitable definition classes for our SAOs, all we need to do to implement this method is to create an array of pointers to our definition classes, and return this when called -

```
}
```

The remaining methods of ISAOLibrary are all simple and are implemented in `mysaolib.h` -

```
    public:
        /// Constructor - nothing special to do here.
        MySAOLib() : SAOLib() {}

        /// Destructor - nothing special to do here.
        virtual ~MySAOLib() {}

        /// Return our name when asked.
        virtual const char *getName() const {
            return "MySAOLib";
        }

        /// Supply version information for this library when asked.
        virtual void getVersion(int &major, int &minor) const {
            major = 1;
            minor = 0;
        }

        /// Return the number of SAO types supplied by this library.
        virtual int typeCount() const;

        /// Return a description of this library.
        virtual const char*  getDescription () const {
            return "Some simple SAO examples.";
        }

        /// Return a pointer to an array of ISAODefinition objects
        /// which describe the SAO types supplied by this library.
        virtual const ISAODefinition * const *getDefinitions() const;

        /// Create a new instance of the SAO specified by the supplied typeId.
        virtual IOperate *createOperate(ISAO& base, typeId function);
};
```

Finally, since the SAO library will be supplied as a DLL, some function must be provided which may be exposed by the DLL to allow the Framework to gain access to an instance MySAOLib. All SAO libraries must implement and export such a function called getSAOLibrary() which takes no arguments and simply returns a pointer to an implementation of ISAOLibrary. Rather than continue to create new instances of MySAOLib each time this function is called, we can simply return pointers to a single instance:

```
extern "C" SAF_DLL_EXPORT ISAOLibrary *getSAOLibrary(void) {
    static MySAOLib instance;
    return &instance;
}
```

We now have all the code we need and the library can be compiled.

3.2 Generating a Library Description File

In order to start using an SAO library as part of an application `saolib2xml` is used to generate a xml file which contains the SAO descriptions which we created in SAO Definition like this -

```
saolib2xml mysaolib.dll
```

This will create a Library Desciption File called `mysaolib.xml` in the current directory. This file may then be used to load the library for use with an application by using either the `addsaolib` program, or via the **SONARIS Studio** program, `safstudio`.

4 Framework Services

The ISAO and ISOC interfaces presented to the user part of the SAO by the base part of the SAO and the container offer a very large number of methods which allow an individual SAO to interact with other parts of an application hierarchy. Almost any action which can be performed using the **SONARIS Studio** program or the various client APIs may be performed programmatically from within the various methods which implement the IOperate interface.

Each of these methods is fully documented in the class reference documentation. However, it is worth discussing some of the key methods here.

4.1 Logging and Trace

The ISOC interface presents a number of methods by which a message may be written to the log files of the process hosting the instance of the SAO. Selecting the appropriate method determines at which of the four log levels (defined by ISOC::MessageSeverity_t) the message is logged. These methods are essentially equivalent to a simple `printf` function. However, an extended form of these methods allows the caller to supply a reference to an SAO whose name and id will be used to prefix the message in the log. For example, here is an example of a call to the ISOC::logDebugMessage() -

```
//  Write some variable to the log, prefixed with my details.
//  N.B. no need to use a carriage return, the logger will do that for us.
ISOC *container = base.getISOC();
container->logDebugMessage(base, "value of i = %d", i);
```

Messages passed to these calls will be written to the log only if the logging level assigned to the instance of the SAO is the same or less than the level used to log the message. Given that some preparatory work may be required to assemble messages which may then not appear in the log due to the log level setting of the SAO, you may wish to test the log level using the ISAO::getLoggingLevel() method before performing such work and calling the logging method -

```
//  Check that I have debug logging enabled before writing my message
if (base->getLoggingLevel() == LOG_DEBUG) {
    ISOC *container = base.getISOC();
    container->logDebugMessage(base, "a debug message");
}

//  Log this message if our log level is set to LOG_INFO or lower
//  i.e. If our log level is set to LOG_DEBUG this message will
//  still be written
if (base->getLoggingLevel() <= LOG_INFO) {
    ISOC *container = base.getISOC();
    container->logInfoMessage(base, "an info message");
}
```

4.2 Hierarchy Monitoring

Some **SONARIS/Framework** applications might need to be notified of changes in their hierarchy; for example, if there are two different branches in the hierarchy whose children have to be kept in step i.e. if a child is created under branch X then a child with the same name has to be

created under branch Y. **SONARIS/Framework** provides a mechanism to support this; any SAO can ask to be notified of changes to another SAO in the hierarchy. The changes that are notified are:

- A child being added.
- A child being removed.
- The SAO being deleted.
- The SAO being renamed.
- A child being renamed.
- Its parent being renamed.
- An input being added.
- An input being removed.

If an SAO (a *watching* SAO) wishes to be notified of changes to any SAO (including itself) then it calls ISAOAdmin::watchSAOEvents() specifying the id of the SAO (the *watched* SAO) that it wants notifications for. A watching SAO can monitor any number of watched SAOs.

When a watched SAO changes then the watching SAO's operate() method is called. To receive the notifications of changes to the watched SAOs, the watching SAO then invokes ISAOAdmin::notifyEventListener() passing an implementation of IEventListener as a parameter. notifyEventListener() then calls back the relevant methods on IEventListener. If an SAO calls watchSAOEvents() then it must call notifyEventListener() in its operate() method otherwise the events will just accumulate.

5 SAO Programming Good Practice

There is almost nothing you cannot do within the code of the methods of a IOperate implementation. However, there are some techniques you can take advantage of to make your code safer and easier to write.

5.1 The AutoDestroy Template

Whenever the Framework calls a method on the IOperate interface of an SAO, it wraps the call in an exception block. This protects the Framework from otherwise fatal errors in the SAO's code over which the Framework has no control.

Within the code of, say, your IOperate::operate() method you may wish to create temporary instances of other **SONARIS/Framework** classes, such as ISAFValue which the Framework can create for you using its static ISAFValue::create() method. Having created such a class instance, your code would then be responsible for the corresponding call to ISAFValue::destroy() once you have finished with it before your method returns. However, should your code fail and result in an exception being caught by the Framework, any such objects created by your code would not be destroyed.

If you use the AutoDestroy template to create such objects you can be assured that they will be destroyed properly either when your method returns normally or when it fails and causes an exception. This also has the benefit that if your code has multiple return points you won't have to remember to destroy your temporary objects before each of them. Here is an example -

```
//  Here is a temporary SAFValue which isn't managed by an AutoDestroy...
ISAFValue *tempVal1 = ISAFValue::create();
//  and here is one that is.
AutoDestroy<ISAFValue>  tempVal2;

//  If some condition is reached we'd like to return, but we must remember
//  to destroy tempVal1, but tempVal2 will be handled for us.
if (...) {
    tempVal1->destroy();
    return;
}

//  Do something awful which will cause an exception in the Framework.
*((int *)0) = 0;

//  We never reach here, so tempVal1 is not destroyed, but tempVal2 will
//  be handled for us.
tempVal1->destroy();
return;
```

5.2 Respect const References and Pointers

Many methods provided by the ISAO and ISOC interfaces provide access to other SAOs within an application hierarchy. Some examples include ISAO::getParent() and ISOC::getISAO(). Both of these methods and many others return `const` pointers or references to another SAO. It is important to respect the `const`ness of this result. Having retrieved the pointer to another SAO it may be tempting to cast off the `const` to allow you to call methods on that SAO which would otherwise not be available to you such as ISAO::setValue() or ISAO::setStatus() etc. However,

you should be aware that your current execution thread may not be the only one active within the container Framework, and the SAO which you are affecting might be involved in another part of a ripple, possibly being managed by another thread. Even if the SAO in question is not otherwise involved, or is certain to be part of the same ripple as your own SAO, the effects of your calls on that SAO may not be properly interpreted by the container.

Try to restrict the interaction between your SAOs to the well defined relationships provided by the Framework; connections, virtual connections, post methods etc.

5.3 Synchronise Using the Container

In view of the remarks regarding `const` ISAO pointers described above, it is not advisable to leave ongoing activity running as a result of a call to one of the methods of IOperate using another thread which you have created. This thread might then call methods of the Framework after that method has returned and you may well encounter thread safety problems and unpredictable behaviour. If your SAO methods need to make use of threads of their own, any result which needs to be passed back to the SAO should be passed via the IESRUpdateHandler interface which is implemented by the container as part of ISOC. An appropriate implementation of IOperate::update() can be used to receive and process such results and make them visible to the rest of the Framework.

6 Integrating Existing Code

Chapter 2 demonstrated the fundamental elements of creating SAOs to perform simple operations on their inputs. However, clearly there are considerably more complex tasks which an SAO might be required to perform. It may be that a body of code already exists which is suitable for performing these tasks and which it is desirable to integrate into **SONARIS/Framework** applications rather than to reimplement from scratch as SAOs. The principle obstacle to easy integration of existing code is that SAOs can store only a single value or an array of similarly typed values where existing objects are likely to encapsulate a mixture of different types all of which must be made available within the Framework.

To make integration of such objects possible, **SONARIS/Framework** provides two APIs which allow groups of SAOs to be created which provide access to the underlying values of a single wrapped object: ISAOResource is a listener interface that must be implemented to provide access to the values of the wrapped object, while ISAOResourceManager is an active interface (implemented as part of ISOC) which allows implementations of ISAOResource to be shared safely between SAOs. This chapter uses the code of the MyWrapper and MyReso SAOs to demonstrate the use of these interfaces.

Consider the following class, Calculator, which stores three double values calculated from two supplied values:

```
class Calculator {
    public:
        /// Constructor
        Calculator() : _sum(0.0), _difference(0.0), _product(0.0) { }

        /// Destructor
        virtual ~Calculator() { }

        /// Recalculate the sum, difference and product of the two supplied
        /// values, setting the appropriate _resourceFlag for those which
        /// have changed.
        /// @param i First value.
        /// @param j Second value.
        virtual void calculateValues(double i, double j) {
            _sum = i + j;
            _difference = i - j;
            _product = i * j;
        }

        /// @return The sum of the values supplied in the last call to
        ///     calculateValues().
        virtual double getSum() const {
            return _sum;
        }
        /// @return The difference between the values supplied in the last
        ///     call to calculateValues().
        virtual double getDifference() const {
            return _difference;
        }
        /// @return The product of the values supplied in the last call to
        ///     calculateValues().
        virtual double getProduct() const {
            return _product;
        }
```

```
    protected:
        /// Calculated value derived from values supplied to calculateValues().
        double _sum, _difference, _product;
};
```

Although it would be easy enough to arrange for an SAO which invokes the Calculator::calculate-Values() method of this class by supplying values obtained from its inputs, it is not clear how the results of the calculation could be accessed by other SAOs. One possibility might be to define the storage type of the SAO as a SAF_DOUBLE_ARRAY and store each value as an array element. However, this is rather inflexible in that it would require any other SAO that takes this array as an input to extract the appropriate element from the array and then establish whether that element has changed since the last time it was called. Also, clearly this approach can only work where all the values available from the wrapped class are of the same type.

Another possibility might be to deliver the values produced by Calculator to other SAOs by means of posting them via the input queue of the container using the ISOC::postValueToSAO() method. Although useful in some circumstances, this solution also has several drawbacks. Firstly, it depends on the SAO to which the value is posted existing and being of the right type to receive the value. It also means that the value will not be received until the following ripple, which might result in an inconsistent set of values across SAOs between ripples.

An even more dubious solution might be to access the user part of the wrapper SAO or the Calculator object directly from another SAO via a pointer. This is clearly unsafe as it would be difficult for two independent SAOs to negotiate that the pointer should not be used if it goes out of scope.

A cleaner and more powerful solution is to create an SAO which implements the ISAOResource interface in addition to IOperate and which registers itself with the ISAOResourceManager implemented by ISOC in its IOperate::initialise() method, as illustrated in the MyWrapper class:

```
        virtual void initialise(ISAO& base) {
            base.getISOC()->addResource(base, *this);
        }
```

Once registered, other SAOs can then request access to the ISAOResource interface by specifying the identity (memory address) of the base part of the SAO which wraps the Calculator object. This can be supplied to them as an input connection. As such, provided the wrapping SAO undertakes to not delete the wrapped object or withdraw the ISAOResource interface from the ISAOResourceManager, the consuming SAOs can safely hold a pointer to the ISAOResource provided their input connections don't change. If their input connections do change they must go back to the ISAOResourceManager and re-acquire a pointer to the ISAOResource of the wrapper SAO. Figure 6.1 illustrates this relationship.

The use of inputs to coordinate access to the ISAOResource implementation of the wrapper SAO also ensures that during ripples the wrapper SAO will always be invoked before those SAOs which consume the resource values, and allows it to signal to those SAOs when the resource values have changed using the return value from its IOperate::operate() method.

Once they have retrieved the ISAOResource interface pointer from the ISAOResourceManager, the resource consuming SAOs first use it to retrieve resource numbers for the named resources

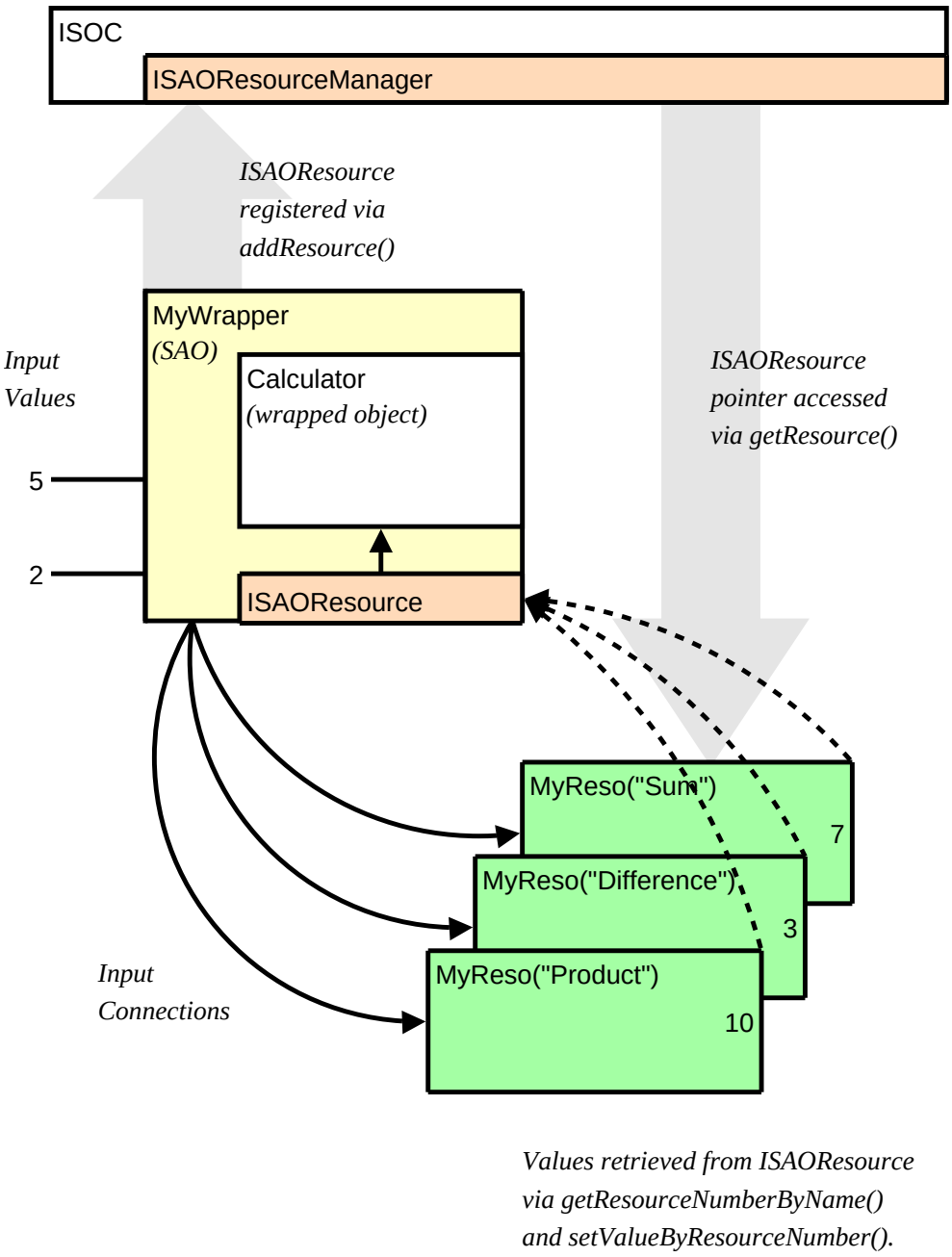


Figure 6.1: Sharing data between SAOs via ISAOResource

whose value they intend to adopt. These numbers are defined by the wrapping SAO. In My-Wrapper three resource numbers are defined. Note also that by using numbers with distinct binary bit patterns, these numbers can also be used to set flags by which changes in the resource values can be indicated.

```
//  An enumeration of all the values available from this class
//  via the ISAOResource implementation.
typedef enum {
    SUM = 0x0001,
    DIFF = 0x0002,
    PROD = 0x0004
} ResourceFlags_t;
```

```
        //  A record of flag settings for the resource values. If a resource
        //  value changes, the corresponding flag is set.
        unsigned int _resourceFlags;
```

The implementation of IOperate::operate() by the wrapper SAO invokes the appropriate methods of the wrapped object, and sets the MyWrapper::_resourceFlags only if the values have changed. The return value is true if either the SAO sets its own value (as for any simple SAO) or if any of its resource flags are set, thereby causing the SAOs which access its ISAOResource implementation to be visited during the ripple.

```
bool MyWrapper::operate(ISAO& base, int count, ISAO **changed) {

    //  Our value will be the number of resources whose values have changed.
    int noOfChanges = 0;

    //  Check the current values of the _calculator.
    double sum = _calculator.getSum();
    double diff = _calculator.getDifference();
    double prod = _calculator.getProduct();

    //  Get the calculator to update its values based on the inputs.
    _calculator.calculateValues(
        base.getInput(FIRST_INPUT)->getDouble(),
        base.getInput(SECOND_INPUT)->getDouble());

    //  Clear the _resourceFlags.
    _resourceFlags = 0;

    //  Reset flags as appropriate for those calculator values which have
    //  changed, and raise the count of the number of changes.
    if (sum != _calculator.getSum()) {
        _resourceFlags |= SUM;
        noOfChanges++;
    }
    if (diff != _calculator.getDifference()) {
        _resourceFlags |= DIFF;
        noOfChanges++;
    }
    if (prod != _calculator.getProduct()) {
        _resourceFlags |= PROD;
        noOfChanges++;
    }

    //  If noOfChanges is different from before, update our value.
    bool valueChanged = false;
    if (base.getInt() != noOfChanges)
        valueChanged = base.setValue(noOfChanges);

    //  Return true if either our value has changed or any of
    //  the _resourceFlags are set.
    return valueChanged || _resourceFlags;
}
```

The implementation of the methods of ISAOResource is straightforward. The

ISAOResource::getResourceNumberByName() method must simply return the appropriate resource number give the supplied name:

```
bool MyWrapper::getResourceNumberByName(const char *name,
    SAFValueType_t type, unsigned int& resourceNumber) {

    //  Only double values are available as resources, so fail if asked
    //  for any type other than SAF_DOUBLE or SAF_UNDEFINED (any basic type).
    if (! (type == SAF_DOUBLE || type == SAF_UNDEFINED))
        return false;

    //  Look up the resource number by name.
    if (! strcmp(name, "Sum"))
        resourceNumber = SUM;
    else if (! strcmp(name, "Difference"))
        resourceNumber = DIFF;
    else if (! strcmp(name, "Product"))
        resourceNumber = PROD;
    else
        return false;

    return true;
}
```

The ISAOResource::setValueByResourceNumber() method sets the value of supplied ISAFValue according to the supplied resource number provided that either the corresponding resource flag is set or the caller has indicated that they wish to be initialised even if the resource flag isn't set.

```
bool MyWrapper::setValueByResourceNumber(ISAFValue& value,
    unsigned int resourceNumber, bool init) {

    //  If not asked to initialise the supplied value and the specified
    //  resource hasn't changed, do nothing.
    if (! init && ! (_resourceFlags & resourceNumber))
        return false;

    //  Return true on setting the supplied value.
    switch (resourceNumber) {
        case SUM:
            value.setValue(_calculator.getSum());
            return true;
        case DIFF:
            value.setValue(_calculator.getDifference());
            return true;
        case PROD:
            value.setValue(_calculator.getProduct());
            return true;
        default:
            return false;
    }
}
```

MyReso provides an example of the code used to access resources (note that a standard set of SAOs capable of performing this action is supplied under the *resource* category of the *Foundation Objects* SAO library).

```
bool MyReso::operate(ISAO& base, int count, ISAO **changed) {

    //  Get the currentInput (its SAF_MANDATORY so we don't need to check it).
    const ISAO *currentInput = base.getInput(RESOURCE_INPUT);

    //  If its id is the same as the one we've recorded previously we can
    //  go ahead and use the existing _resource pointer to get our value
    //  updated.
    if (currentInput->getId() == _inputId)
        return _resource->setValueByResourceNumber(base, _resoNumber);

    //  Its a new input, so look up the SAOResource associated with it.
    _resource = base.getISOC()->getResource(*currentInput);

    //  If the input SAO has no resources, give up.
    if (! _resource) {
        base.setErrorText(SAF_BADINPUT, "Input has no resources.");
        return false;
    }

    //  If there is no resource which matches our name, give up.
    if (! _resource->getResourceNumberByName(base.getName(),
        base.getDefinition()->getValueType(), _resoNumber)) {

        base.setErrorText(SAF_BADINPUT,
            "Named resource not available from input.");
        return false;
    }

    //  Remember the _inputId, and ask the ISAOResource to set our value
    //  according to the _resoNumber which matched our name.
    _inputId = currentInput->getId();
    return _resource->setValueByResourceNumber(base, _resoNumber, true);
}
```

Appendix A Namespace Documentation

A.1 SAF Namespace Reference

A.1.1 Detailed Description

This file provides: Implementations or partial implementations of interfaces which the framework will expect a user supplied SAO and SAO Library to support.

- They are supplied for convenience and may be used as bases providing typical behaviour from which user defined libraries of SAOs may be created.
- Implementations of helper classes and fragments of code which are often useful when implementing SAO libraries.

Classes

- class SAF::SAODef

  An implementation of ISAODefinition which provides a description of an SAO.

- class SAF::SAOInputRule

  A basic implementation of ISAOValidation for creating input rules for SAOs.

- class SAF::SAOLib

  A basic partial implementation of ISAOLibrary which can serve as a base for user defined SAO libraries.

- class SAF::SAOIOperate

  A class to ease the provision of IOperate by providing a default implementation of everything.

- class SAF::InputIndex< T >

  A template helper class to keep track of which input pointers are in use at which input index for an SAO.

- struct **SAF::InputIndex< T >::Idx**

  An index number for an input pointer, with a pointer to the next (the same pointer can be connected to more than one input).

- struct **SAF::InputIndex< T >::Slot**

  A slot which matches an input pointer (_key) to an Idx chain.

- class SAF::Flags< T >

  An array of binary flags with methods to test and set - often useful when creating SAOs which implement ISAOResource in addition to IOperate.

- class SAF::Basename

  Finds the final element in a path (e.g.

- class SAF::ISAOValidation

  Interface used to obtain details of the validation of a single input to the operate method.

- class SAF::ISAODefinition

  Interface which supplies all the details needed to understand how an SAO provided by a library will operate at run time.

- class SAF::IOperate

  Interface to be implemented by the user part of an SAO in order to do its work when invoked as part of a ripple.

- class SAF::ISAOLibrary

  Interface implemented by SAO library providers to allow the Framework to determine the nature of the objects that this library can create.

- class SAF::ISOC

  This interface is visible to SAOs and defines the operations that are available to SAO 'operate' method code.

- class SAF::IBlackBox

  Implementations of the IBlackBox interface allow low level trace messages to be written to a rolling in memory log.

- class SAF::SAFTypeName

  Class to map SAF::SAFValueType_t to printable strings.

- class SAF::SAFStatusName

  Class to map SAF::SAFStatus_t to printable strings.

- class SAF::SAFDate

  A small class (taking up only 8 bytes convertible to a saf64_t) which allows either a date, a time of day or a single point in time to be represented to a resolution of milliseconds.

- class SAF::SAOFType

SAOFtype is a simple class combining a SAF::libraryId and SAF::typeId to make up the unique functional type identifier of an SAO.

- class SAF::SAOId

  SAOId is a simple class combining the objectId of an SAO with the containerId of the container which hosts it to provide a unique identifier for the SAO within the application.

- class SAF::SAOIdString

  Class to encapsulate a string representation of an SAOId (for debug, logging etc).

- class SAF::ISAFValue

  Interface to read and write SAF's basic item of data.

- class SAF::SAFValueString

  Class to assist conversion of a SAFValue to a printable string.

- class SAF::SAFValueDesc

  Class to assist conversion of a SAFValue to a printable string describing its type, status and current value.

- class SAF::IProperties

  Implementations of this interface handle an array of name value pairs where each value is a ISAFValue.

- class SAF::IESRUpdate

  An IESRUpdate is a data structure allowing chains of addressed values and properties to be built and delivered.

- class SAF::AutoDestroy< T >

  Template class removing the need to remember to destroy() one of the objects which are created via create() e.g.

- class SAF::AutoFree< T >

  Template class (similar to auto_ptr) removing the need to remember to free malloc'd memory for POD (plain old data) types.

- class SAF::IEventListener

  This interface is used to notify an SAO of changes to other SAOs.

- class SAF::ISAORead

  Interface to get the data contained within an SAO.

- class SAF::ISAOWrite

  Interface to set the data contained within an SAO.

- class SAF::ISAODependencies

  Interface describing the dynamic properties of the dependency relationship between SAOs.

- class SAF::ISAOSubInfo

  Interface used by the ClientAPI to hold details about a single other SAO, whether an input, output, virtual input or virtual output.

- class SAF::ISAOInfo

  Interface used by the ClientAPI to hold details about an SAO.

- class SAF::ISAOAdmin

  Interface to get hierarchy and database related information.

- class SAF::ISAOProxy

  Interface which is only realy relevant to Client code and Proxy SAOs.

- class SAF::ISAO

  The real working SAO.

- class SAF::ISAOResource

  Interface to be implemented by objects which hold multiple values to be shared by multiple SAOs.

- class SAF::ISAOResourceManager

  Objects which implement this interface arrange to store and retrieve objects which implement the ISAOResource interface.

Typedefs

- typedef int64_t **saf64_t**
- typedef saf64_t **saftime_t**
- typedef unsigned short libraryId

- typedef unsigned short typeId
- typedef unsigned short containerId
- typedef saf64_t objectId
- typedef enum SAF::SAOHist SAOHist_t

  Values to control whether on not the value of an SAO is persisted in the configuration database or not.

- typedef enum SAF::SAOLogging SAOLogging_t

  Logging can be controlled on a per SAO basis.

Enumerations

- enum SAOInputValidation_t { SAF_MANDATORY = 1, SAF_OPTIONAL = 2, SAF_RESERVED = 3 }

  An enumeration detailing the possible criteria for deciding whether a single input to an SAO is acceptable to it such that a call to that SAO's IOperate::operate() method will be made.

- enum SAFValueType_t {

  SAF_UNDEFINED = 0, SAF_VOID = 1, SAF_INT = 2, SAF_INT64 = 3,

  SAF_DOUBLE = 4, SAF_BOOL = 5, SAF_DATE = 6, SAF_STRING = 7,

  SAF_BYTE_ARRAY = 8, SAF_INT_ARRAY = 9, SAF_INT64_ARRAY = 10, SAF_DOUBLE_ARRAY = 11,

  SAF_STRING_ARRAY = 12, SAF_DATE_ARRAY = 13 }

- enum SAFStatus_t {

  SAF_VALID = 0, SAF_STALE = 1, SAF_INVALID = 7, SAF_UNINITIALIZED = 2,

  SAF_BADINPUT = 3, SAF_SUSPENDED = 6, SAF_NOLIBRARY = 10, SAF_NODEFINITION = 11,

  SAF_REMOTEDISCONNECT = 4, SAF_LOCALDISCONNECT = 5, SAF_DELETED = 8, SAF_NOSAO = 9,

  SAF_NEWPROXY = 12, SAF_ACCESS_DENIED = 15, SAF_CANCELLED = 13, SAF_DROPPED = 14,

  SAF_FAILOVER = 16, SAF_COMPLETED = 17 }

- enum ModifyTreeControl_t {

  MTC_TRANSIENT = 0x0001, MTC_CHILDREN = 0x0002, MTC_MERGE = 0x0004, MTC_HISTORY = 0x0008,

  MTC_LOGGING = 0x0010, MTC_MELDABLE = 0x0020 }

- enum SAOHist { **SAO_HIST_OFF** = 0, **SAO_HIST_ON** = 1 }

Values to control whether on not the value of an SAO is persisted in the configuration database or not.

- enum SAOLogging {

  SAO_LOGGING_OFF = 0, **SAO_LOGGING_INPUT** = 1,
  SAO_LOGGING_OUTPUT = 2, **SAO_LOGGING_RIPPLE** = 4,

  SAO_LOGGING_ONESHOT = 8, **SAO_LOGGING_FULLDUMP** = 0x10,
  SAO_LOGGING_USER = 0x20, **SAO_LOGGING_IMMUNE** = 0x40,

  SAO_LOGGING_NOMELD = 0x80, **SAO_LOGGING_USERMASK** = 0x3F,
  SAO_LOGGING_RESET = 0xBF, **SAO_LOGGING_KEEP** =
  ~SAO_LOGGING_USERMASK,

  SAO_LOGGING_DATABASE }

  Logging can be controlled on a per SAO basis.

Functions

- void BBNOOP ()

  Some macros to make using the BlackBox trace recorder easier and less obtrusive.

A.1.2 Typedef Documentation

typedef unsigned short SAF::libraryId

Description:

Index number of an SAO library (each library is assigned a unique number when its defintion is loaded for use within an application).

Definition at line 1069 of file isafvalue.h.

Referenced by SAF::SAOLib::getId(), and SAF::SAOFType::getLId().

typedef unsigned short SAF::typeId

Description:

Index number of an individual SAO type within an SAO library.

Definition at line 1073 of file isafvalue.h.

Referenced by SAF::SAOFType::getTId().

typedef unsigned short SAF::containerId

Description:

A number which uniquely identifies a container within an application.

Definition at line 1171 of file isafvalue.h.

Referenced by SAF::SAOId::getCId().

typedef saf64_t SAF::objectId

Description:

A number which uniquely identifies an SAO within a container. **Note** Although defined as a 64 bit value, only the bottom 48 bits are used.

Definition at line 1176 of file isafvalue.h.

Referenced by SAF::SAOId::getOId().

typedef enum SAF::SAOLogging SAF::SAOLogging_t

Logging can be controlled on a per SAO basis.

The default is for an SAO to have no logging enabled. The values in SAOLogging are bit flags and can be used in combination when calling setLogging. Logging is switched off by setting to zero (LOGGING_OFF). The other bits have the following meaning:- INPUT - Dump the value of all the inputs for the SAO being traced prior to operate being called. Details of the parameters to operate are also given. OUTPUT - Dump the value of the SAO after operate has returned. RIPPLE - Treat all remaining SAOs involved in the current ripple as if they had at least the level of trace of the current one (applies only to the INPUT, OUTPUT and FULLDUMP bits). ONESHOT - do this once only i.e. on the first call to operate after being set. FULLDUMP - if set will dump the whole set of values for array inputs and outputs. It will also dump the whole of a string. Use this bit with care because it can produce large amounts of output. USER - should be tested by the implementors of IOperate and ICommand methods to determine if they should produce trace events via the ISOC::log∗Message() methods.

Note that if any SAO throws an exception inside operate then the following action is taken:- Retrospective dump of its inputs (if not already done). State set to SAF_SUSPENDED (from which it can never emerge and which means it will no longer take any part in ripples). Of course this assumes that the base SAO itself has not been too badly corrupted but generally this technique will work.

Logging applies equally to any uses of the ICommand interface, with obvious analogies - inputs is the supplied properties, output is the return value and contents of reply.

A.1.3 Enumeration Type Documentation

enum SAOInputValidation_t

An enumeration detailing the possible criteria for deciding whether a single input to an SAO is acceptable to it such that a call to that SAO's IOperate::operate() method will be made.

The method will not be called if the input does not comply.

Enumeration values:

SAF_MANDATORY Any MANDATORY input must be connected and its status must be SAF::SAF_VALID or SAF::SAF_STALE.

before a call will be made on IOperate::operate().

SAF_OPTIONAL An OPTIONAL input may be absent, but if it is connected then its status is taken into account when.

SONARIS/Framework automatically determines the status of the output value (remembering that the IOperate::operate() method of an SAO is at liberty to change any automatically assigned status.

SAF_RESERVED A RESERVED input is one set up at run time by the SAO, that is not persisted in the configuration database, and that can only be disconnected by the SAO itself or through deletion of the source.

The input is treated as OPTIONAL when deciding whether to call IOperate::operate().

Definition at line 176 of file isaflibrary.h.

Referenced by SAF::SAOInputRule::getValidationCode().

enum SAFValueType_t

Description:

Enumeration which defines the type of data that an individual SAFValue and, therefore, SAO may hold.

Enumeration values:

SAF_UNDEFINED Default type, can be changed to any other.

SAF_VOID No value.

SAF_INT A 32 bit integer.

SAF_INT64 A 64 bit integer.

SAF_DOUBLE Extended precision (64 bit) floating point number.

SAF_BOOL A C++ bool.

SAF_DATE A 64 bit value containing a set of bit fields.

SAF_STRING A zero terminated byte array.

SAF_BYTE_ARRAY An array of bytes.

SAF_INT_ARRAY An array of SAF_INT.

SAF_INT64_ARRAY An array of SAF_INT64.

SAF_DOUBLE_ARRAY An array of SAF_DOUBLE.

SAF_STRING_ARRAY An array of SAF_STRING.

SAF_DATE_ARRAY An array of SAF_DATE.

Definition at line 458 of file isafvalue.h.

Referenced by SAF::SAOInputRule::getValueType(), SAF::SAODef::getValueType(), and SAF::SAFTypeName::noOfNames().

enum SAFStatus_t

Description:

Different statuses which describe the condition of an item of data (an SAO or SAFValue) or communication session within **SONARIS/Framework**. A useable value is only available from a SAFValue or SAO with a status of SAF_VALID or SAF_STALE. All other statuses

imply no usable data is available from the SAO or SAFValue and essentially are all equivalent to SAF_INVALID. Many of these apply only to SAOs or to communication sessions across the ISAFClient or IExternalSystem interfaces. Refer to those interfaces for more detail regarding the precise meaning of such statuses.

Enumeration values:

SAF_VALID A good value.

SAF_STALE value is stale (usable, but possibly out of date) or derived from stale data

SAF_INVALID The data is invalid.

SAF_UNINITIALIZED Value has never been set. Initial state.

SAF_BADINPUT The data held by an SAO is out of date as it has lost a mandatory input.

SAF_SUSPENDED The data held by an SAO is not updating due to an exception caught within that SAO's code.

SAF_NOLIBRARY The SAO is not complete because of missing functionality or missing library.

SAF_NODEFINITION The SAO is a dummy placeholder because its definition could not be found.

SAF_REMOTEDISCONNECT Failure detected in remote machine.

SAF_LOCALDISCONNECT Failure detected in this machine.

SAF_DELETED The associated SAO has been deleted.

SAF_NOSAO The SAO requested does not exist.

SAF_NEWPROXY Indicates that a proxy SAO has not had an initial response.

SAF_ACCESS_DENIED Indicates that a proxy SAO has had a response but that the client is not allowed to read the data in it.

SAF_CANCELLED Confirms cancellation of requests across the IExternalSystem interface.

SAF_DROPPED Indicates unexpected and unrecoverable termination of a request across the IExternalSystem interface.

SAF_FAILOVER Indicates unexpected termination of a request across the IExternalSystem interface and that recovery is being attempted.

SAF_COMPLETED Indicates expected termination of a request across the IExternalSystem interface at such times as the request comes to its natural end.

Definition at line 544 of file isafvalue.h.

Referenced by SAF::SAFStatusName::noOfNames().

enum ModifyTreeControl_t

Description:

Values used as bit flags in combinations when calling ISAFClient::modifyTree().

Enumeration values:

MTC_TRANSIENT If set then all SAOs which have to be created are made transient. Without this then all SAOs in the destination will end up matching (in terms of persistence) the corresponding SAO in the source.

MTC_CHILDREN If set then the destination SAO has the tranformation applied to all of its children.

MTC_MERGE If set then any extra SAOs already in the destination tree are left in place (although moved to the end of the child list).

MTC_HISTORY If set then SAOS already in the destination will have their history setting (getHistoryLevel()) amended to match that of the source.

MTC_LOGGING If set then SAOS already in the destination will have their logging setting (getLoggingLevel()) amended to match that of the source.

MTC_MELDABLE If set then SAOS already in the destination will have their meldability setting (isMeldable) amended to match that of the source.

Definition at line 1909 of file isafvalue.h.

enum SAOLogging

Logging can be controlled on a per SAO basis.

The default is for an SAO to have no logging enabled. The values in SAOLogging are bit flags and can be used in combination when calling setLogging. Logging is switched off by setting to zero (LOGGING_OFF). The other bits have the following meaning:- INPUT - Dump the value of all the inputs for the SAO being traced prior to operate being called. Details of the parameters to operate are also given. OUTPUT - Dump the value of the SAO after operate has returned. RIPPLE - Treat all remaining SAOs involved in the current ripple as if they had at least the level of trace of the current one (applies only to the INPUT, OUTPUT and FULLDUMP bits). ONESHOT - do this once only i.e. on the first call to operate after being set. FULLDUMP - if set will dump the whole set of values for array inputs and outputs. It will also dump the whole of a string. Use this bit with care because it can produce large amounts of output. USER - should be tested by the implementors of IOperate and ICommand methods to determine if they should produce trace events via the ISOC::log∗Message() methods.

Note that if any SAO throws an exception inside operate then the following action is taken:- Retrospective dump of its inputs (if not already done). State set to SAF_SUSPENDED (from which it can never emerge and which means it will no longer take any part in ripples). Of course this assumes that the base SAO itself has not been too badly corrupted but generally this technique will work.

Logging applies equally to any uses of the ICommand interface, with obvious analogies - inputs is the supplied properties, output is the return value and contents of reply.

Definition at line 333 of file isao.h.

A.1.4 Function Documentation

void BBNOOP () [inline, static]

Some macros to make using the BlackBox trace recorder easier and less obtrusive.

These can only be used from within SAO methods where the *ISAO& base* parameter is available. Set compiler flag -D_BBTRACE=1 for basic BlackBox logging, set -D_BBTRACE=2 for Black-Box logging and output to stderr. Example:

virtual bool operate(ISAO& base, int count, ISAO ∗∗changed) { BBFNC(”operate”); // Start of the method

BBCHK; // A checkpoint (just notes the filename and line no)

BBREM(”some text”); // A checkpoint with text

return BBRET(”operate(true)”), true; // End of the method }

Definition at line 605 of file safhelper.h.

References BBNOOP().

Referenced by BBNOOP().

Appendix B Class Documentation

B.1 SAF::AutoDestroy< T > Class Template Reference

`#include <isafvalue.h>`

B.1.1 Detailed Description

template<class T> class SAF::AutoDestroy< T >

Template class removing the need to remember to destroy() one of the objects which are created via create() e.g.

ISAOSpec, ISAFValue.

Definition at line 2141 of file isafvalue.h.

Public Member Functions

- **AutoDestroy** (T *input=NULL)
- T * **operator** → () const
- **operator T &** () const
- **operator T** * () const
- **operator const T** * () const
- T * get () const

  returns the stored pointer

- T & operator * () const

  dereferencing operator

- T * release ()

  Return the data and cede our ownership of it.

B.1.2 Member Function Documentation

template<class T> T* SAF::AutoDestroy< T >::release () `[inline]`

Return the data and cede our ownership of it.

This object will always return a NULL pointer after this call so it is unusable. You will need to destroy it yourself.

Definition at line 2172 of file isafvalue.h.

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

- isafvalue.h

B.2 SAF::AutoFree< T > Class Template Reference

`#include <isafvalue.h>`

B.2.1 Detailed Description

template<class T> class SAF::AutoFree< T >

Template class (similar to auto_ptr) removing the need to remember to free malloc'd memory for POD (plain old data) types.

Useful for memory which gets allocated by C functions (e.g. strdup), arrays etc.

Definition at line 2191 of file isafvalue.h.

Public Member Functions

- AutoFree (T *input=NULL)

  Constructor e.g.

- AutoFree (size_t size)

  Constructor - allocates buffer of size specified.

- ~AutoFree ()

  Destructor - tidies up.

- T * reset (T *input)

  Assignment, frees any previous data.

- T * release ()

  Removes data without freeing it.

- T * get () const

  returns the stored pointer

- void resize (size_t size)

  Alters size of allocated memory (0 frees memory, clears pointer).

- T * operator= (T *input)

  Assignment operator (calls reset()) e.g.

- operator T * () const

  Cast to underlying type.

B.2.2 Constructor & Destructor Documentation

template<class T> SAF::AutoFree< T >::AutoFree (T * *input* = NULL) `[inline]`

Constructor e.g.

AutoFree<char> s(strdup("foo"));

Definition at line 2195 of file isafvalue.h.

template<class T> SAF::AutoFree< T >::AutoFree (size_t *size*) `[inline]`

Constructor - allocates buffer of size specified.

e.g. AutoFree<char> s(64);

Definition at line 2199 of file isafvalue.h.

B.2.3 Member Function Documentation

template<class T> T* SAF::AutoFree< T >::reset (T * *input*) `[inline]`

Assignment, frees any previous data.

e.g. s.reset("bar");

Definition at line 2208 of file isafvalue.h.

template<class T> T* SAF::AutoFree< T >::operator= (T * *input*) `[inline]`

Assignment operator (calls reset()) e.g.

s = strdup("foo2"); e.g. s = NULL;

Definition at line 2236 of file isafvalue.h.

template<class T> SAF::AutoFree< T >::operator T * () const `[inline]`

Cast to underlying type.

e.g. *s = '';

Definition at line 2241 of file isafvalue.h.

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

- isafvalue.h

B.3 SAF::Basename Class Reference

`#include <safhelper.h>`

B.3.1 Detailed Description

Finds the final element in a path (e.g.

an SAO full name). Doesn't take a copy of the string, so can only be used while the original string still exists.

Definition at line 537 of file safhelper.h.

Public Member Functions

- Basename (const char *path, const char *seps="/\\")
- operator const char * () const

  The final element of the path.

B.3.2 Constructor & Destructor Documentation

SAF::Basename::Basename (const char * *path*, const char * *seps* = "/\\") [inline]

Parameters:

s The path.

seps A string of all possible path separator characters.

Definition at line 543 of file safhelper.h.

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

- safhelper.h

B.4 Example::Calculator Class Reference

`#include <mywrapper.h>`

B.4.1 Detailed Description

A simple class which stores the results of basic arithmetic operations (sum, difference and product) performed on two supplied values.

Definition at line 35 of file mywrapper.h.

Public Member Functions

- Calculator ()

  Constructor.

- virtual ~Calculator ()

  Destructor.

- virtual void calculateValues (double i, double j)

  Recalculate the sum, difference and product of the two supplied values, setting the appropriate _resourceFlag for those which have changed.

- virtual double getSum () const
- virtual double getDifference () const
- virtual double getProduct () const

Protected Attributes

- double _sum

  Calculated value derived from values supplied to calculateValues().

- double _difference

  Calculated value derived from values supplied to calculateValues().

- double _product

  Calculated value derived from values supplied to calculateValues().

B.4.2 Member Function Documentation

virtual void Example::Calculator::calculateValues (double *i*, double *j*) [inline, virtual]

Recalculate the sum, difference and product of the two supplied values, setting the appropriate _resourceFlag for those which have changed.

Parameters:

i First value.

j Second value.

Definition at line 48 of file mywrapper.h.

virtual double Example::Calculator::getSum () const [inline, virtual]

Returns:

The sum of the values supplied in the last call to calculateValues().

Definition at line 56 of file mywrapper.h.

virtual double Example::Calculator::getDifference () const [inline, virtual]

Returns:

The difference between the values supplied in the last call to calculateValues().

Definition at line 61 of file mywrapper.h.

virtual double Example::Calculator::getProduct () const [inline, virtual]

Returns:

The product of the values supplied in the last call to calculateValues().

Definition at line 66 of file mywrapper.h.

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

- mywrapper.h

B.5 SAF::Flags< T > Class Template Reference

`#include <safhelper.h>`

B.5.1 Detailed Description

template<class T> class SAF::Flags< T >

An array of binary flags with methods to test and set - often useful when creating SAOs which implement ISAOResource in addition to IOperate.

Definition at line 481 of file safhelper.h.

Public Member Functions

- Flags ()

  Simple initialisation.

- bool isSet (T mask=0) const
- bool set (T mask, bool setting)
- Flags & operator= (T flags)
- operator T () const

Protected Attributes

- T **_flags**

B.5.2 Member Function Documentation

template<class T> bool SAF::Flags< T >::isSet (T *mask* = `0`) const `[inline]`

Description:

Test the setting of flags in a supplied set.

Parameters:

mask The flags to be tested within the set. If no mask is specified (the default) then all flags are tested.

Returns:

True if any of the flags specified by the supplied mask are set (or any flag at all if no mask supplied), false otherwise.

Definition at line 496 of file safhelper.h.

template<class T> bool SAF::Flags< T >::set (T *mask*, bool *setting*) `[inline]`

Description:

Set or clear the specified flags in the supplied set.

Parameters:

mask The flags to be changed within the set.

setting The setting to be applied, either true or false.

Returns:

True all the supplied flags are set, false otherwise.

Definition at line 506 of file safhelper.h.

template<class T> Flags& SAF::Flags< T >::operator= (T *flags*) [inline]

Description:

Assignment taking new settings for all flags.

Parameters:

flags A new set of flag settings.

Returns:

Reference to this Flags object.

Definition at line 517 of file safhelper.h.

template<class T> SAF::Flags< T >::operator T () const [inline]

Description:

Typecast underlying flags.

Returns:

Reference to this Flags object.

Definition at line 526 of file safhelper.h.

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

- safhelper.h

B.6 SAF::IBlackBox Class Reference

`#include <isoc.h>`

B.6.1 Detailed Description

Implementations of the IBlackBox interface allow low level trace messages to be written to a rolling in memory log.

These can then be read and dumped to a higher level log (e.g. using the logMessage() methods of ISOC). An implementation of IBlackBox is available to user code in implementations of IOperate via the ISOC::getIBlackBox() method. Any messages stored using this object will be written to the log only if an error occurs which results in the SAO being suspended.

Definition at line 803 of file isoc.h.

Public Member Functions

- virtual void log (const char *label, unsigned int lineno, const char *messageFormat=NULL,...)=0

B.6.2 Member Function Documentation

virtual void SAF::IBlackBox::log (const char * *label*, unsigned int *lineno*, const char * *messageFormat* = `NULL`, ...) `[pure virtual]`

Description:

Log the supplied label, lineno and optional message details.

Parameters:

label Pointer to a piece of text which describes the source code location (e.g. a filename, function name etc.). N.B. The implementation of IBlackBox supplied by ISOC will take a copy of this pointer rather than a copy of the string it points to in order to keep memory usage and execution time to a minimum. For this reason it is best to only pass pointers to static strings in this parameter (e.g. use the __FILE__ preprocessor macro).

lineno Line number (e.g. use the __LINE__ preprocessor macro).

messageFormat An optional printf style format string for use with the subsequently supplied parameters. This enables the logging of values only available at runtime, but should be used sparingly as it significantly increases the processing cost of calls to this method.

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

- isoc.h

B.7 SAF::IESRUpdate Class Reference

`#include <isafvalue.h>`

B.7.1 Detailed Description

An IESRUpdate is a data structure allowing chains of addressed values and properties to be built and delivered.

Each link in the chain has addressing information (addressee and Id, both 64 bit values), an ISAFValue and an IProperties. Typically they are used to pass updates between components. IOperate::update() delivers a value change to an SAO (in this instance the chaining ability is not relevant); IESRUpdateHandler::ESRUpdatesReady() takes a chain.

IESRUpdate provides methods for use by both the producer of the update to aid its construction, and its consumer by which its contents can be read. No restriction is placed on which of these methods may be called by the holder of an IESRUpdate object and this allows for changes to be made after the object has passed out of the control of the primary producer. This allows IESRUpdates to be read, modified and forwarded by their consumers.

Definition at line 1951 of file isafvalue.h.

Public Member Functions

- virtual void destroy () const =0
- virtual void setAddressee (saf64_t addressee)=0
- virtual saf64_t getAddressee () const =0
- virtual void setValue (const ISAFValue *value)=0
- virtual void setPair (saf64_t addressee, const ISAFValue *value)=0
- virtual void setTriplet (saf64_t addressee, saf64_t id, const ISAFValue *value)=0
- virtual void takeValue (ISAFValue *value)=0
- virtual void takePair (saf64_t addressee, ISAFValue *value)=0
- virtual void takeTriplet (saf64_t addressee, saf64_t id, ISAFValue *value)=0
- virtual const ISAFValue & getValue () const =0
- virtual saf64_t getId () const =0
- virtual void setNext (const IESRUpdate *update)=0
- virtual const IESRUpdate * getNext () const =0
- virtual void addProperty (const char *name, const ISAFValue *value)=0
- virtual void takeProperty (const char *name, ISAFValue *value)=0
- virtual int getNoOfProperties () const =0

- virtual const ISAFValue * getPropertyByNumber (int n) const =0
- virtual const char * getPropertyNameByNumber (int n) const =0
- virtual const ISAFValue * getPropertyByName (const char *name) const =0
- virtual bool removeProperty (size_t n)=0

  Remove the specified property.

- virtual bool removeProperty (const char *name)=0

  Remove the specified property.

- virtual void clear ()=0
- virtual void setId (saf64_t id)=0

  Set the id info of the subscriber for which this IESRUpdate is intended.

Static Public Member Functions

- IESRUpdate * create ()

Protected Member Functions

- virtual ~IESRUpdate ()

  Protected so instances can't be deleted through this base class.

B.7.2 Constructor & Destructor Documentation

virtual SAF::IESRUpdate::~IESRUpdate () [inline, protected, virtual]

Protected so instances can't be deleted through this base class.

Definition at line 2132 of file isafvalue.h.

B.7.3 Member Function Documentation

IESRUpdate* SAF::IESRUpdate::create () [static]

Description:

Create an instance of the standard implementation class. The caller is responsible for destroying the returned value.

virtual void SAF::IESRUpdate::destroy () const [pure virtual]

Description:

Get rid of the object. Calls the destructor which is then also reponsible for destroying all subsequent elements of the linked list.

virtual void SAF::IESRUpdate::setAddressee (saf64_t *addressee*) [pure virtual]

Description:

Set the address info of the subscriber for which this IESRUpdate is intended.

Parameters:

addressee The addressee to which this update is to be delivered.

virtual saf64_t SAF::IESRUpdate::getAddressee () const [pure virtual]

Returns:

The addressee to which this update is to be delivered.

virtual void SAF::IESRUpdate::setValue (const ISAFValue * *value*) [pure virtual]

Description:

Set the data provided by this update by taking a copy of the supplied ISAFValue.

Parameters:

value The value to be supplied.

virtual void SAF::IESRUpdate::setPair (saf64_t *addressee*, const ISAFValue * *value*) [pure virtual]

Description:

Set the addressee and the data provided by this update as a pair by taking a copy of the supplied ISAFValue.

Parameters:

addressee The addressee to which this update is to be delivered.

value The value to be supplied.

virtual void SAF::IESRUpdate::setTriplet (saf64_t *addressee*, saf64_t *id*, const ISAFValue * *value*) [pure virtual]

Description:

Set the addressee, id and the data provided by this update as a triplet by taking a copy of the supplied ISAFValue.

Parameters:

addressee The addressee to which this update is to be delivered.

id The id supplied when the request was made.

value The value to be supplied.

virtual void SAF::IESRUpdate::takeValue (ISAFValue * *value*) [pure virtual]

Description:

Set the data provided by this update by pointing to the supplied ISAFValue (hence ownership of the ISAFValue is transferred to the IESRUpdate).

Parameters:

value The value to be supplied.

virtual void SAF::IESRUpdate::takePair (saf64_t *addressee*, ISAFValue * *value*) [pure virtual]

Description:

Set the addressee and the data provided by this update by pointing to the supplied ISAFValue (hence ownership of the ISAFValue is transferred to the IESRUpdate).

Parameters:

addressee The addressee to which this update is to be delivered.

value The value to be supplied.

virtual void SAF::IESRUpdate::takeTriplet (saf64_t *addressee*, saf64_t *id*, ISAFValue * *value*) [pure virtual]

Description:

Set the addressee, id and the data provided by this update by pointing to the supplied ISAFValue (hence ownership of the ISAFValue is transferred to the IESRUpdate).

Parameters:

addressee The addressee to which this update is to be delivered.

id The id supplied when the request was made.

value The value to be supplied.

virtual const ISAFValue& SAF::IESRUpdate::getValue () const [pure virtual]

Returns:

A reference to the ISAFValue which holds the data supplied by this update.

virtual saf64_t SAF::IESRUpdate::getId () const [pure virtual]

Returns:

The id supplied by this update.

virtual void SAF::IESRUpdate::setNext (const IESRUpdate * *update*) [pure virtual]

Description:

Link the supplied IESRUpdate as the next element in the chain.

Parameters:

update The IESRUpdate to be added to the chain.

virtual const IESRUpdate* SAF::IESRUpdate::getNext () const [pure virtual]

Returns:

A pointer to the next IESRUpdate in this list of updates.

virtual void SAF::IESRUpdate::addProperty (const char * *name*, const ISAFValue * *value*) [pure virtual]

Description:

Adds a copy of the supplied ISAFValue as a property associated with the supplied name.

Parameters:

name The name to be given to the property being added.

value The value of the property being added.

virtual void SAF::IESRUpdate::takeProperty (const char * *name*, ISAFValue * *value*) [pure virtual]

Description:

Adds the supplied ISAFValue as a property associated with the supplied name (hence ownership of the ISAFValue is transferred to the IESRUpdate).

Parameters:

name The name to be given to the property being added.

value The value of the property being added.

virtual int SAF::IESRUpdate::getNoOfProperties () const [pure virtual]

Returns:

The number of name/value pairs supplied with this IESRUpdate.

virtual const ISAFValue* SAF::IESRUpdate::getPropertyByNumber (int *n*) const [pure virtual]

Parameters:

n The index number of a property of this IESRUpdate.

Returns:

The value of the indexed property or NULL if there is no such property.

virtual const char* SAF::IESRUpdate::getPropertyNameByNumber (int *n*) const [pure virtual]

Parameters:

n The index number of a property of this IESRUpdate.

Returns:

The name of the indexed property or NULL if there is no such property.

virtual const ISAFValue* SAF::IESRUpdate::getPropertyByName (const char * *name*) const [pure virtual]

Parameters:

name The index number of a property of this IESRUpdate.

Returns:
The value of the named property or NULL if there is no such property.

virtual bool SAF::IESRUpdate::removeProperty (size_t *n*) [pure virtual]

Remove the specified property.

Parameters:
n Index of a property held in the property array. Implicitly renumbers any properties higher than n.

Returns:
true of property found (and therefore removed) else false.

virtual bool SAF::IESRUpdate::removeProperty (const char * *name*) [pure virtual]

Remove the specified property.

Parameters:
name Name of a property held in the property array.

Returns:
true of property found (and therefore removed) else false.

virtual void SAF::IESRUpdate::clear () [pure virtual]

Description:
Empty the properties array of all its values (the values are deleted).

virtual void SAF::IESRUpdate::setId (saf64_t *id*) [pure virtual]

Set the id info of the subscriber for which this IESRUpdate is intended.

Parameters:
id The id.

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

- isafvalue.h

B.8 SAF::IEventListener Class Reference

`#include <isao.h>`

B.8.1 Detailed Description

This interface is used to notify an SAO of changes to other SAOs.

If an SAO (a ”watching” SAO) wishes to be notified of changes to any SAO (including itself) then it should call ISAOAdmin::watchSAOEvents specifying the id of the SAO (the ”watched” SAO) that it wants notifications for. A watching SAO can monitor any number of watched SAOs.

When a watched SAO changes then the watching SAO’s operate() method is called. To receive the notifications of changes to the watched SAOs, the watching SAO then invokes ISAOAdmin::notifyEventListener passing an implementation of IEventListener as a parameter. ISAOAdmin::notifyEventListener then calls back the relevant methods on IEventListener. If an SAO calls watchSAOEvents() then it must call notifyEventListener() in its operate() method otherwise the events will just accumulate.

There are some things which a programmer should be aware of:

- The notification is asynchronous. An extreme example of this would be that an SAO has been linked to input 2, the input has been removed and a different SAO has then been linked to input 2. Depending on timing factors, all these events could be delivered by a single call to notifyEventListener so the first event would refer to a different input SAO than was now actually connected to input 2.
- Using ISOC::getISAO() to convert a supplied SAO Id to an ISAO pointer may return NULL. This is again because of the asynchronocity i.e. the event was queued and then the SAO was deleted.
- If the SAO being watched is in a different container to the watching SAO then using ISOC::getISAO() will provide the address of an SAO proxy object; ISAOAdmin::isProxy() can be used to find out whether an ISAO object is a proxy. Many ISAO methods have empty implementations in an SAO proxy and so need to be used with caution.
- If the SAO being watched is in a different container to the watching SAO then some events are not notified. Each method’s description documents whether this restriction applies.

Definition at line 396 of file isao.h.

Public Member Functions

- virtual void childAdded (ISAO &base, const SAOId &watched, const SAOId &child, const char ∗name)=0

  This method is called when a child has been added to a watched of this SAO.

- virtual void childRemoved (ISAO &base, const SAOId &watched, const SAOId &child, const char ∗name, const SAOId &next)=0

  This method is called when a child has been removed from a watched of this SAO.

- virtual void deleted (ISAO &base, const SAOId &watched, const char ∗name)=0

  This method is called when a watched SAO is deleted.

- virtual void renamed (ISAO &base, const SAOId &watched, const char ∗newName)=0

  This method is called when a watched SAO has been renamed.

- virtual void childRenamed (ISAO &base, const SAOId &watched, const SAOId &child, const char ∗newName)=0

  This method is called when a child of a watched SAO has been renamed.

- virtual void parentRenamed (ISAO &base, const SAOId &watched, const SAOId &parent, const char ∗newName)=0

  This method is called when the parent of a watched SAO has been renamed.

- virtual void newParent (ISAO &base, const SAOId &watched, const SAOId &parent, const char ∗name)=0

  This method is called when the parent of a watched SAO is changed.

- virtual void inputAdded (ISAO &base, const SAOId &watched, const SAOId &input, const char ∗name, int index)=0

  This method is called when an input has been added to a watched SAO.

- virtual void inputRemoved (ISAO &base, const SAOId &watched, const SAOId &input, const char ∗name, int index)=0

  This method is called when an input has been removed from a watched SAO.

Protected Member Functions

- virtual ∼IEventListener ()

  Destructor.

B.8.2 Constructor & Destructor Documentation

virtual SAF::IEventListener::∼IEventListener () `[inline, protected, virtual]`

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 510 of file isao.h.

B.8.3 Member Function Documentation

virtual void SAF::IEventListener::childAdded (ISAO & *base*, const SAOId & *watched*, const SAOId & *child*, const char * *name*) [pure virtual]

This method is called when a child has been added to a watched of this SAO.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

child The id of the child which has been added.

name The name of the added child.

virtual void SAF::IEventListener::childRemoved (ISAO & *base*, const SAOId & *watched*, const SAOId & *child*, const char * *name*, const SAOId & *next*) [pure virtual]

This method is called when a child has been removed from a watched of this SAO.

Removal may be either because the child has been deleted or because it has been moved in the hierarchy.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

child The id of the removed child.

name The name of the removed child.

next The id of the SAO following the removed one; if ”isEmpty” then the removed one was the last child.

virtual void SAF::IEventListener::deleted (ISAO & *base*, const SAOId & *watched*, const char * *name*) [pure virtual]

This method is called when a watched SAO is deleted.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO; the SAO no longer exists.

name The name of the watched SAO that has been deleted

virtual void SAF::IEventListener::renamed (ISAO & *base*, const SAOId & *watched*, const char * *newName*) [pure virtual]

This method is called when a watched SAO has been renamed.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

newName The watched SAO’s new name.

virtual void SAF::IEventListener::childRenamed (ISAO & *base*, const SAOId & *watched*, const SAOId & *child*, const char * *newName*) [pure virtual]

This method is called when a child of a watched SAO has been renamed.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

child The id of the child which has been renamed.

newName The child's new name.

virtual void SAF::IEventListener::parentRenamed (ISAO & *base*, const SAOId & *watched*, const SAOId & *parent*, const char * *newName*) [pure virtual]

This method is called when the parent of a watched SAO has been renamed.

This event is NOT notified if the watched and watching SAOs are in different containers.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

parent The id of the parent.

newName The parent's new name.

virtual void SAF::IEventListener::newParent (ISAO & *base*, const SAOId & *watched*, const SAOId & *parent*, const char * *name*) [pure virtual]

This method is called when the parent of a watched SAO is changed.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

parent The id of the new parent.

name The new parent's name.

virtual void SAF::IEventListener::inputAdded (ISAO & *base*, const SAOId & *watched*, const SAOId & *input*, const char * *name*, int *index*) [pure virtual]

This method is called when an input has been added to a watched SAO.

This event is NOT notified if the watched and watching SAOs are in different containers.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

input The SAO that has been added as an input.

name The name of the added input.

index The index of the added input.

virtual void SAF::IEventListener::inputRemoved (ISAO & *base*, const SAOId & *watched*, const SAOId & *input*, const char * *name*, int *index*) [pure virtual]

This method is called when an input has been removed from a watched SAO.

Removal may be either because the input has been deleted or simply because the linkage has been broken. This event is NOT notified if the watched and watching SAOs are in different containers.

Parameters:

base The base part of the watching SAO.

watched The id of the watched SAO.

input The id of the removed input.

name The name of the removed input.

index The index of the removed input.

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

- isao.h

B.9 SAF::InputIndex< T > Class Template Reference

`#include <safhelper.h>`

B.9.1 Detailed Description

template<class T> class SAF::InputIndex< T >

A template helper class to keep track of which input pointers are in use at which input index for an SAO.

Can be used to quickly translate an array of changed input pointers supplied to IOperate::operate() into an array of indexes using the getIndexes() method. Understands that the same input pointer my be used by several inputs. If no array of changed SAO pointers is supplied, must be cleared (using clear()) or (if necessary) resized (using resize()) and repopulated with calls to insert().

Definition at line 349 of file safhelper.h.

Public Member Functions

- InputIndex (size_t n=0)
- ~InputIndex ()
- void clear ()
- void resize (size_t n)
- void insert (const T *p, unsigned int i)
- const unsigned int * getIndexes (unsigned int count, const T *const *p)

B.9.2 Constructor & Destructor Documentation

template<class T> SAF::InputIndex< T >::InputIndex (size_t *n* = `0`) `[inline]`

Description:

Initialize.

Definition at line 378 of file safhelper.h.

template<class T> SAF::InputIndex< T >::~InputIndex () `[inline]`

Description:

Tidy up and free allocated memory.

Definition at line 383 of file safhelper.h.

B.9.3 Member Function Documentation

template<class T> void SAF::InputIndex< T >::clear () [inline]

Description:

Clear the index ready to be repopulated with calls to insert().

Definition at line 389 of file safhelper.h.

template<class T> void SAF::InputIndex< T >::resize (size_t *n*) [inline]

Description:

Prepare the memory in which to store the indexes and allow for some working space.

Parameters:

n The maximum number of index entries.

Definition at line 397 of file safhelper.h.

template<class T> void SAF::InputIndex< T >::insert (const T * *p*, unsigned int *i*) [inline]

Description:

Insert a new pointer and index value into the index.

Parameters:

p The pointer to be indexed.

i The index value associated with the pointer.

Definition at line 414 of file safhelper.h.

template<class T> const unsigned int* SAF::InputIndex< T >::getIndexes (unsigned int *count*, const T *const * *p*) [inline]

Description:

Translates the supplied array of pointers into an array of indexes.

Parameters:

count The number of elements in the supplied pointer array.

p An array of pointers.

Returns:

A pointer to an array of indexes related to the supplied pointer array.

Definition at line 434 of file safhelper.h.

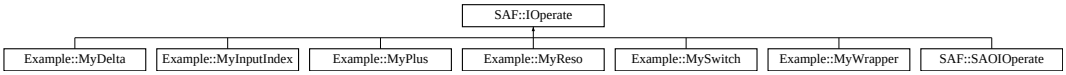
The documentation for this class was generated from the following file:

- safhelper.h

B.10 SAF::IOperate Class Reference

`#include <isaflibrary.h>`

Inheritance diagram for SAF::IOperate::



B.10.1 Detailed Description

Interface to be implemented by the user part of an SAO in order to do its work when invoked as part of a ripple.

An SAO is comprised of two parts, the base part (which implements the ISAO interface) and the user part. The base and user parts have identical lifetimes (the base is created first and deleted last).

Definition at line 324 of file isaflibrary.h.

Public Member Functions

- virtual void initialise (ISAO &base)=0
- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)=0
- virtual void destroy (ISAO &base)=0
- virtual bool operate (ISAO &base, int count, ISAO **changed)=0
- virtual bool rename (const ISAO &base, const char *name)=0

Protected Member Functions

- virtual ~IOperate ()

  Protected so instances can't be deleted through this base class.

B.10.2 Constructor & Destructor Documentation

virtual SAF::IOperate::~IOperate () `[inline, protected, virtual]`

Protected so instances can't be deleted through this base class.

Definition at line 451 of file isaflibrary.h.

B.10.3 Member Function Documentation

virtual void SAF::IOperate::initialise (ISAO & *base*) `[pure virtual]`

Description

Unless destroy() is called immediately this method is called once soon after creation. The hierarchy, inputs and outputs, name and value, where they are available from the database

are all set up in base before this call. It could be used, for instance, to generate a request to an external supplier or perform any other initialization.

Parameters:

base The base part of this SAO (the same value that was supplied to the call to ISAOLibrary::createOperate() which returned this object).

Implemented in Example::MyPlus, Example::MySwitch, Example::MyDelta, Example::MyWrapper, Example::MyReso, Example::MyInputIndex, and SAF::SAOIOperate.

virtual bool SAF::IOperate::update (ISAO & *base*, const IESRUpdate & *update*, bool & *handled*, bool *last*) [pure virtual]

Description:

This method is called when an update to the value of the SAO is supplied via the container. The return value indicates to the container what action it should take.

Parameters:

base The base part of this SAO (the same value that was supplied to the call to ISAOLibrary::create which returned this object).

update The update which supplies a new value or status.

handled True indicates to the container that the update has been handled and that the container should not do so. False indicates that the container should update the value of the SAO with that supplied by the update.

last A meldable SAO (see ISAO::isMeldable) may have several changes delivered to it prior to a single ripple. This parameter is true when this update call is the final one prior to the ripple; it could be used, for example, to quickly ignore all proposed changes prior to the last one.

Returns:

True if the value of the SAO has changed as a result of this call (e.g. through a call to ISAOWrite::setValue()) and hence this SAO should be included in the next ripple, false otherwise.

Implemented in Example::MyPlus, Example::MySwitch, Example::MyDelta, Example::MyWrapper, Example::MyReso, Example::MyInputIndex, and SAF::SAOIOperate.

virtual void SAF::IOperate::destroy (ISAO & *base*) [pure virtual]

Description:

Called by the container when this object is no longer required and will not be referenced again. The associated base part of the SAO is no longer available after this call. Normally you would call the destructor of the object but you are required to undo what was done in ISAOLibrary::createOperate(). This mathod may be called before initialise() in exceptional circumstances (e.g. the SAO is created and then immediately destroyed).

Parameters:

base The base part of this SAO (the same value that was supplied to the call to ISAOLibrary::createOperate() which returned this object). It is still inside the hierarchy and parent, children are still available. Note that when a tree of SAOs are being destroyed these calls will occur on all children before the parent.

Implemented in Example::MyPlus, Example::MySwitch, Example::MyDelta, Example::MyWrapper, Example::MyReso, Example::MyInputIndex, and SAF::SAOIOperate.

virtual bool SAF::IOperate::operate (ISAO & *base*, int *count*, ISAO ** *changed*)

`[pure virtual]`

Description:

The operate method is invoked when it may be appropriate for the value of the SAO to be changed due to a change of value or status on one or more of the inputs. This method will not be called unless:

- All the mandatory inputs are present, are of a compatible data type and have data available i.e. they have a status of SAF::SAF_VALID or SAF::SAF_STALE. "Compatible data type" means that the data type of the input is compatible with the data type specified in the relevant ISAOValidation.
- All the optional inputs that are present are of a compatible data type.

This method will be called when any of the following conditions occurs provided the conditions listed above are met:

- Any input is connected (this implies that it is an optional input with or without a value, or a mandatory input with an available value).
- Any input is disconnected (this implies that it is an optional input as disconnecting a mandatory input will prevent this call and cause the SAO's status to be set to SAF_BADINPUT).
- One or more inputs changes its status or its operate() method returns true.
- The SAO has registered to receive notification of changes to other SAOs (possibly including itself) by calling ISAO::watchSAOEvents() and some such event has occurred. ISAO::notifyEventListener() should be called to receive the notification of the events.

A true return value or change in status will cause the ripple to continue. Normally this method should return true if it changes the SAO's value (by using ISAOWrite::setValue()).

Note that the operate() method is *not* the only thing changing the value or status of the associated SAO. In particular the operate() method is only called when all the SAF_MANDATORY inputs defined have data available. This means that if one of these inputs fails then the status will change without a call to operate() and operate() will later be called when all such inputs are again SAF_VALID or SAF_STALE.

If one or more of the inputs (both mandatory and optional) has a status of SAF_STALE then the status of this SAO will be automatically set to SAF_STALE before operate() is

called. This means that for the majority of SAOs, the operate() method does not need to concern itself with the status. However, the operate() method is at liberty to change its status (including to SAF_VALID) if it thinks this is appropriate.

Parameters:

base The base part of this SAO (the same value that was supplied to the call to ISAOLibrary::createOperate() which returned this object).

count A count of the number of inputs that have changed. If this is zero then operate() has been invoked because the connection or disconnection or an input and should assume that all inputs have potentially changed. If this is non-zero then you may assume that only the inputs indicated by the `changed` paramter have changed.

changed The list of pointers to inputs which has changed.

Returns:

True if a change has been made to the value of the SAO which is to be acted upon by other SAOs through the continuation of the ripple, false otherwise.

Implemented in Example::MyPlus, Example::MySwitch, Example::MyDelta, Example::MyWrapper, Example::MyReso, Example::MyInputIndex, and SAF::SAOIOperate.

virtual bool SAF::IOperate::rename (const ISAO & *base*, const char * *name*) [pure virtual]

Description:

This method is called to inform of a suggested change of name. The old name is still present in base.

Parameters:

base The base part of this SAO (the same value that was supplied to the call to ISAOLibrary::createOperate() which returned this object).

name The proposed new name. Note that you do not have to change the name in base nor worry about any conflict with an identically named child in the parent.

Returns:

True to permit the name to be changed, false otherwise.

Implemented in Example::MyPlus, Example::MySwitch, Example::MyDelta, Example::MyWrapper, Example::MyReso, Example::MyInputIndex, and SAF::SAOIOperate.

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

- isaflibrary.h

B.11 SAF::IProperties Class Reference

`#include <isafvalue.h>`

B.11.1 Detailed Description

Implementations of this interface handle an array of name value pairs where each value is a ISAFValue.

Property lists are not expected to be very long, names are case sensitive.

Definition at line 1759 of file isafvalue.h.

Public Member Functions

- virtual void destroy () const =0
- virtual void addProperty (const char ∗name, const char ∗value)=0
- virtual void takeProperty (const char ∗name, ISAFValue ∗value)=0
- virtual void addProperty (const char ∗name, const ISAFValue ∗value)=0
- virtual void addProperty (const char ∗name, bool value)=0
- virtual void addProperty (const char ∗name, int value)=0
- virtual void addProperty (const char ∗name, saf64_t value)=0
- virtual void addProperty (const char ∗name, double value)=0
- virtual int getNoOfProperties () const =0
- virtual const ISAFValue ∗ getPropertyByNumber (int n) const =0
- virtual const char ∗ getPropertyNameByNumber (int n) const =0
- virtual const ISAFValue ∗ getPropertyByName (const char ∗name) const =0
- virtual const ISAFValue ∗ getPropertyByNoCaseName (const char ∗name) const =0
- virtual bool removeProperty (size_t n)=0

  Remove the specified property.

- virtual bool removeProperty (const char ∗name)=0

  Remove the specified property.

- virtual void clear ()=0

Static Public Member Functions

- IProperties ∗ create ()

B.11.2 Member Function Documentation

IProperties* SAF::IProperties::create () [static]

Returns:

A new instance of an implementation of this interface. The caller is responsible for destroying the returned value.

virtual void SAF::IProperties::destroy () const [pure virtual]

Description:

Release this interface instance - the implementation will be deleted.

virtual void SAF::IProperties::addProperty (const char * *name*, const char * *value*) [pure virtual]

Description:

Add a property with the supplied name to the end of the array and set its value to match the supplied value.

Parameters:

name The name of the property (property is not added if no name given).

value The value of the property (may be NULL).

virtual void SAF::IProperties::takeProperty (const char * *name*, ISAFValue * *value*) [pure virtual]

Description:

Add a property with the supplied name to the end of the array and transfer ownership of the supplied value so that it is used as the value part of the property.

Parameters:

name The name of the property (property is not added if no name given).

value The value of the property (ownership is transferred to the property).

virtual void SAF::IProperties::addProperty (const char * *name*, const ISAFValue * *value*) [pure virtual]

Description:

Add a property with the supplied name to the end of the array and set its value to match the supplied value.

Parameters:

name The name of the property (property is not added if no name given).

value The value of the property (may be NULL).

virtual void SAF::IProperties::addProperty (const char * *name*, bool *value*) [pure virtual]

Description:

Add a property with the supplied name to the end of the array and set its value to match the supplied value.

Parameters:

name The name of the property (property is not added if no name given).

value The value of the property (may be NULL).

virtual void SAF::IProperties::addProperty (const char * *name*, int *value*) [pure virtual]

Description:

Add a property with the supplied name to the end of the array and set its value to match the supplied value.

Parameters:

name The name of the property (property is not added if no name given).

value The value of the property (may be NULL).

virtual void SAF::IProperties::addProperty (const char * *name*, saf64_t *value*) [pure virtual]

Description:

Add a property with the supplied name to the end of the array and set its value to match the supplied value.

Parameters:

name The name of the property (property is not added if no name given).

value The value of the property (may be NULL).

virtual void SAF::IProperties::addProperty (const char * *name*, double *value*) [pure virtual]

Description:

Add a property with the supplied name to the end of the array and set its value to match the supplied value.

Parameters:

name The name of the property (property is not added if no name given).

value The value of the property (may be NULL).

virtual int SAF::IProperties::getNoOfProperties () const [pure virtual]

Returns:

The Number of name/value pairs.

virtual const ISAFValue* SAF::IProperties::getPropertyByNumber (int *n*) const [pure virtual]

Parameters:

n Index of a property held in the property array.

Returns:

The value of the property held in the array at the index number specified.

virtual const char* SAF::IProperties::getPropertyNameByNumber (int *n*) const [pure virtual]

Parameters:

n Index of a property held in the property array.

Returns:

The name of the property held in the array at the index number specified.

virtual const ISAFValue* SAF::IProperties::getPropertyByName (const char * *name*) const [pure virtual]

Parameters:

name Name of a property held in the property array.

Returns:

The value of the property held in the array with the name specified.

virtual const ISAFValue* SAF::IProperties::getPropertyByNoCaseName (const char * *name*) const [pure virtual]

Parameters:

name Name of a property held in the property array.

Returns:

The value of the first property held in the array with the name specified, regardless of differences in case within the name.

virtual bool SAF::IProperties::removeProperty (size_t *n*) [pure virtual]

Remove the specified property.

Parameters:

n Index of a property held in the property array. Implicitly renumbers any properties higher than n.

Returns:

true of property found (and therefore removed) else false.

virtual bool SAF::IProperties::removeProperty (const char * *name*) [pure virtual]

Remove the specified property.

Parameters:

name Name of a property held in the property array.

Returns:

true of property found (and therefore removed) else false.

virtual void SAF::IProperties::clear () [pure virtual]

Description:

Empty the properties array of all its values (the values are deleted).

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

- isafvalue.h

B.12 SAF::ISAFValue Class Reference

`#include <isafvalue.h>`

B.12.1 Detailed Description

Interface to read and write SAF's basic item of data.

ISAFValue objects are much lower level than an ISAO and, in general, will take the value or status that you given them.

Definition at line 1403 of file isafvalue.h.

Public Member Functions

- virtual void destroy () const =0

  Get rid of this instance.

- virtual void assign (const ISAFValue &from)=0

  Assign the contents of from to this.

- virtual SAFValueType_t getType () const =0

  Returns the current data type.

- virtual const char * getTypeName () const =0

  Returns the name of the current data type.

- virtual bool isArray () const =0

  Returns true for an array data type.

- virtual bool isVoid () const =0

  Returns true for an object taking no value.

- virtual void setValue (bool data)=0

  Unconditionally sets the data value and the data type.

- virtual void **setValue** (double data)=0
- virtual void **setValue** (int data)=0
- virtual void **setValue** (saf64_t data)=0
- virtual void **setValue** (const char *data)=0
- virtual void **setValue** (const SAFDate &data)=0

- virtual void **setValue** (const double *data, int length)=0
- virtual void **setValue** (const int *data, int length)=0
- virtual void **setValue** (const saf64_t *data, int length)=0
- virtual void **setValue** (const unsigned char *data, int length)=0
- virtual void **setValue** (const char *const *data, int length)=0
- virtual void **setValue** (const SAFDate *data, int length)=0
- virtual bool setValue (double data, int index)=0

  Sets a single data value for array data types.

- virtual bool **setValue** (const int data, int index)=0
- virtual bool **setValue** (const saf64_t data, int index)=0
- virtual bool **setValue** (const unsigned char data, int index)=0
- virtual bool **setValue** (const char *data, int index)=0
- virtual bool **setValue** (const SAFDate &data, int index)=0
- virtual double getDouble (bool &valid) const =0

  Accessors.

- virtual int **getInt** (bool &valid) const =0
- virtual saf64_t **getInt64** (bool &valid) const =0
- virtual const char * **getString** (bool &valid) const =0
- virtual bool **getBool** (bool &valid) const =0
- virtual SAFDate **getDate** (bool &valid) const =0
- virtual double getDouble () const =0

  Accessors Each returns the data in the requested format if the requested format is currently supported by the data or a null value if not.

- virtual int **getInt** () const =0
- virtual saf64_t **getInt64** () const =0
- virtual const char * **getString** () const =0
- virtual bool **getBool** () const =0
- virtual SAFDate **getDate** () const =0

- virtual const double * getDoubleArray (int &length) const =0

  Accessor.

- virtual const int * getIntArray (int &length) const =0

  Accessor.

- virtual const saf64_t * getLongArray (int &length) const =0

  Accessor.

- virtual const unsigned char * getByteArray (int &length) const =0

  Accessor.

- virtual const char *const * getStringArray (int &length) const =0

  Accessor.

- virtual const saf64_t * getDateArray (int &length) const =0

  Accessor.

- virtual void setRawString (const char *rawString)=0

  Set the value of the raw string.

- virtual const char * getRawString () const =0

  Get the raw string.

- virtual void setErrorText (const char *text)=0

  Set the error text associated with a bad status.

- virtual const char * getErrorText () const =0

  Obtain the error text associated with a bad status.

- virtual SAFStatus_t getStatus () const =0

  Returns the data status.

- virtual void setStatus (SAFStatus_t status)=0

  Sets the data status.

- virtual const char * getStatusName () const =0

Returns the name of the current status.

- virtual bool isValid () const =0

  Returns true when there is SAF_VALID data available.

- virtual bool notValid () const =0

  Returns true when there is not SAF_VALID data available (there may be SAF_STALE data).

- virtual bool isStale () const =0

  Returns true only if there is stale data available.

- virtual void setValid ()=0

  If the status is currently SAF_STALE then sets it to SAF_VALID otherwise the status is not changed.

- virtual void setStale ()=0

  If the status is currently SAF_VALID then sets it to SAF_STALE otherwise the status is not changed.

- virtual bool equals (const ISAFValue &op)=0

  Returns true if the status and value is the same as the one given.

Static Public Member Functions

- ISAFValue * create ()

  Create an instance of the implementation class.

Protected Member Functions

- virtual ~ISAFValue ()

  Destructor.

B.12.2 Constructor & Destructor Documentation

virtual SAF::ISAFValue::~ISAFValue () `[inline, protected, virtual]`

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 1638 of file isafvalue.h.

B.12.3 Member Function Documentation

ISAFValue* SAF::ISAFValue::create () [static]

Create an instance of the implementation class.

The contents are a type of SAF_UNDEFINED and a status of SAF_INITIALIZED. The caller is responsible for destroying the returned value.

virtual void SAF::ISAFValue::destroy () const [pure virtual]

Get rid of this instance.

virtual void SAF::ISAFValue::assign (const ISAFValue & *from*) [pure virtual]

Assign the contents of from to this.

Parameters:

from Instance to copy from.

virtual SAFValueType_t SAF::ISAFValue::getType () const [pure virtual]

Returns the current data type.

virtual const char* SAF::ISAFValue::getTypeName () const [pure virtual]

Returns the name of the current data type.

virtual bool SAF::ISAFValue::isArray () const [pure virtual]

Returns true for an array data type.

virtual bool SAF::ISAFValue::isVoid () const [pure virtual]

Returns true for an object taking no value.

virtual void SAF::ISAFValue::setValue (bool *data*) [pure virtual]

Unconditionally sets the data value and the data type.

The status is unconditionally set to SAF_VALID.

virtual bool SAF::ISAFValue::setValue (double *data*, int *index*) [pure virtual]

Sets a single data value for array data types.

If the SAO is not of the appropriate type then this method is ignored. These assignments will only succeed if the data type is an array of the correct type. If the array is not long enough then it is lengthened by inserting zero values. If the assignment succeeds then the status is set to SAF_VALID.

Returns:
true if the assignment succeeded.

virtual double SAF::ISAFValue::getDouble (bool & *valid*) const [pure virtual]

Accessors.

Each returns the data in the requested format if the requested format is currently supported by the data or a null value if not. It sets the given flag to true if a good value is returned and false otherwise. Simple type conversion will take place.

Parameters:
valid true if the data is returned

Returns:
a data value

virtual double SAF::ISAFValue::getDouble () const [pure virtual]

Accessors Each returns the data in the requested format if the requested format is currently supported by the data or a null value if not.

You cannot tell the difference between failure and a good null value if you have not checked the type.

Returns:
a data value

virtual const double* SAF::ISAFValue::getDoubleArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If data is not of the correct format then zero is returned. Conversion will not take place.

Parameters:
length Has the length of the array on return.

virtual const int* SAF::ISAFValue::getIntArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is not of the correct format then zero is returned. Conversion will not take place.

Parameters:
length Has the length of the array on return.

virtual const saf64_t* SAF::ISAFValue::getLongArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is not of the correct format then zero is returned. Conversion will not take place.

Parameters:

length Has the length of the array on return.

virtual const unsigned char* SAF::ISAFValue::getByteArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is not of the correct format then zero is returned. Conversion will not take place. If the data is a base field then it can be accessed this way and its length is 1.

Parameters:

length Has the length of the array on return.

virtual const char* const* SAF::ISAFValue::getStringArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If data is not of the correct format then zero is returned. Conversion will not take place.

Parameters:

length Has the length of the array on return.

virtual const saf64_t* SAF::ISAFValue::getDateArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is not of the correct format then zero is returned. Conversion will not take place. Note that date array are stored as an array of longs; NULL values indicate an empty array slot. A SAFDate can be obtained by construction from the array value.

Parameters:

length Has the length of the array on return.

virtual void SAF::ISAFValue::setRawString (const char * *rawString*) [pure virtual]

Set the value of the raw string.

This allows a seperate string to be associated with the value. Usually it will not be present; the main use is where an ESRSAO wishes the source from which it derived the value to be available.

virtual const char* SAF::ISAFValue::getRawString () const [pure virtual]

Get the raw string.

Returns:
the data or NULL if it has not been set.

virtual void SAF::ISAFValue::setErrorText (const char * *text*) [pure virtual]

Set the error text associated with a bad status.

Only applicable if the status is notValid().

Parameters:
text A simple string to explain the reson for the status.

virtual const char* SAF::ISAFValue::getErrorText () const [pure virtual]

Obtain the error text associated with a bad status.

Normally this will not be set and return NULL.

Returns:
the data or NULL if it has not been set.

virtual SAFStatus_t SAF::ISAFValue::getStatus () const [pure virtual]

Returns the data status.

A ISAFValue may have either a data value with a status of either SAF_VALID or SAF_STALE; any other status indicates that there is no data available and will help to indicate why this is so. isValid returns true if there is SAF_VALID data; isStale returns true if there is SAF_STALE data available. notValid is the opposite of isValid.

virtual void SAF::ISAFValue::setStatus (SAFStatus_t *status*) [pure virtual]

Sets the data status.

virtual const char* SAF::ISAFValue::getStatusName () const [pure virtual]

Returns the name of the current status.

virtual bool SAF::ISAFValue::isValid () const [pure virtual]

Returns true when there is SAF_VALID data available.

virtual bool SAF::ISAFValue::isStale () const [pure virtual]

Returns true only if there is stale data available.

virtual void SAF::ISAFValue::setValid () [pure virtual]

If the status is currently SAF_STALE then sets it to SAF_VALID otherwise the status is not changed.

virtual void SAF::ISAFValue::setStale () [pure virtual]

If the status is currently SAF_VALID then sets it to SAF_STALE otherwise the status is not changed.

virtual bool SAF::ISAFValue::equals (const ISAFValue & *op*) [pure virtual]

Returns true if the status and value is the same as the one given.

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

- isafvalue.h

B.13 SAF::ISAO Class Reference

`#include <isao.h>`

Inheritance diagram for SAF::ISAO::

B.13.1 Detailed Description

The real working SAO.

Definition at line 1472 of file isao.h.

Protected Member Functions

- virtual ~ISAO ()

  Destructor.

B.13.2 Constructor & Destructor Documentation

virtual SAF::ISAO::~ISAO () `[inline, protected, virtual]`

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 1481 of file isao.h.

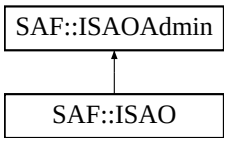
The documentation for this class was generated from the following file:

- isao.h

B.14 SAF::ISAOAdmin Class Reference

`#include <isao.h>`

Inheritance diagram for SAF::ISAOAdmin::



B.14.1 Detailed Description

Interface to get hierarchy and database related information.

Definition at line 1049 of file isao.h.

Public Member Functions

- virtual SAOId & getId (void)=0

  Accessor to the SAOId.

- virtual const SAOId & getId (void) const =0

  Const accessor to the SAOID.

- virtual objectId getOId () const =0

  Returns the SAOs objectId.

- virtual containerId getCId () const =0

  Returns the SAOs containerId.

- virtual const SAOFType & getFunction () const =0

  Return the functional type of this SAO.

- virtual const ISAODefinition * getDefinition () const =0

  Return the definition criteria associated with all SAOs of this functional type.

- virtual const ISAO * getParent () const =0

  Return the parent SAO of this object.

- virtual const ISAO * getChild (const char *name) const =0

  Return the first child with this name or NULL if none.

- virtual const ISAO * firstChild () const =0

  Obtains the first child of an SAO.

- virtual const ISAO * lastChild () const =0

  Obtains the last child of an SAO.

- virtual const ISAO * nextSibling () const =0

  Obtains the next child in the same parent.

- virtual const ISAO * previousSibling () const =0

  Obtains the previous child in the same parent.

- virtual int childCount () const =0

  Return the number of children.

- virtual const char * getName (void) const =0

  Returns the name of this SAO.

- virtual const char * getRealName (void) const =0

  Returns the "real" name of this SAO.

- virtual int getFullName (ISAFValue &target, char delimiter= '/', char escape= '\\') const =0

  Obtains the name of the SAO up through to the root.

- virtual ISOC * getISOC () const =0

  Returns the container.

- virtual const char * **toString** (char *s) const =0
- virtual SAOHist_t getHistoryLevel (void) const =0

  Returns the current History flag.

- virtual int getLoggingLevel (void) const =0

  Returns the current logging level.

- virtual bool isTraceEnabled (void) const =0

Returns true if trace is switched on.

- virtual bool isIndex () const =0

  An index SAO is a dummy one used to get to the mount point.

- virtual bool isProtected () const =0

  A protected SAO is one found at the client which is at or above a mount point or a proxy at a server.

- virtual bool isPersistent () const =0

  A persistent SAO is one which permanently uses its allocated id, usually because it is written to the database and is the opposite of a transient SAO.

- virtual bool isTransient () const =0

  A persistent SAO is one which permanently uses its allocated id, usually because it is written to the database and is the opposite of a transient SAO.

- virtual bool isProxy () const =0

  A proxy is an SAO standing in for the real SAO outside the container in which the real SAO lives.

- virtual bool isSystem () const =0

  A system SAO is one which is created as part of the base framework.

- virtual void setInputValidation (bool validation)=0

  An SAO may know that it is prepared to accept any inputs into its IOperate method and hint to the system that it would be a waste of time validating those inputs.

- virtual bool getInputValidation () const =0

  Returns true if validation is operative.

- virtual const ISAOInfo * getISAOInfo () const =0

  Returns the extended information available at the client.

- virtual UserId_t getOwner () const =0

  Returns the UserId_t of the owner of this SAO.

- virtual GroupId_t getGroup () const =0

  Returns the GroupId_t of the group containing this SAO.

- virtual const SAOPerms & getPermissions () const =0

  Returns a reference to the permission settings of this SAO.

- virtual const IOperate * getIOperate () const =0

  Returns a pointer to the associated IOperate which may be NULL.

- virtual const ICommand * getICommand () const =0

  Returns a pointer to the associated ICommand which may be NULL.

- virtual bool watchSAOEvents (const SAOId &sao)=0

  Request notification of events that affect an SAO.

- virtual bool endWatchSAOEvents (const SAOId &sao)=0

  Stop receiving notification of events that affect an SAO.

- virtual SAF_DEPRECATED bool watchParent ()=0

  No longer supported.

- virtual SAF_DEPRECATED bool endWatchParent ()=0

  No longer supported.

- virtual bool notifyEventListener (IEventListener &listener)=0

  Invoke callbacks matching the pending events.

- virtual UserId_t getLastChangeId () const =0

  Returns the UserId_t of the last person to cause a change to this SAO.

- virtual const SAFDate & getLastChangeTime () const =0

  Returns the time of the last change to this SAO.

- virtual bool isMeldable () const =0

  Returns true if the SAO can have input values merged.

- virtual void setMeldable (bool meldable)=0

  Allows the setting to be changed.

- virtual bool addReservedInput (int index, const ISAO *input)=0

  Adds an input.

- virtual bool removeReservedInput (int index)=0

  Remove an input.

Protected Member Functions

- virtual ~ISAOAdmin ()

  Destructor.

B.14.2 Constructor & Destructor Documentation

virtual SAF::ISAOAdmin::~ISAOAdmin () [inline, protected, virtual]

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 1056 of file isao.h.

B.14.3 Member Function Documentation

virtual SAOId& SAF::ISAOAdmin::getId (void) [pure virtual]

Accessor to the SAOId.

virtual const SAOId& SAF::ISAOAdmin::getId (void) const [pure virtual]

Const accessor to the SAOID.

virtual const SAOFType& SAF::ISAOAdmin::getFunction () const [pure virtual]

Return the functional type of this SAO.

virtual const ISAODefinition* SAF::ISAOAdmin::getDefinition () const [pure virtual]

Return the definition criteria associated with all SAOs of this functional type.

Referenced by SAF::SAOIOperate::update().

virtual const ISAO* SAF::ISAOAdmin::getParent () const [pure virtual]

Return the parent SAO of this object.

NULL means it is either not yet set or at the top of the hierarchy.

virtual const ISAO* SAF::ISAOAdmin::getChild (const char * *name*) const [pure virtual]

Return the first child with this name or NULL if none.

This call is not particularly efficient as it will iterate doing strcmp or similar. A NULL name cannot be searched for - NULL is returned.

virtual const ISAO* SAF::ISAOAdmin::firstChild () const [pure virtual]

Obtains the first child of an SAO.

Returns:
NULL if the SAO has no children.

virtual const ISAO* SAF::ISAOAdmin::lastChild () const [pure virtual]

Obtains the last child of an SAO.

Returns:
NULL if the SAO has no children.

virtual const ISAO* SAF::ISAOAdmin::nextSibling () const [pure virtual]

Obtains the next child in the same parent.

Returns:
NULL if the current object is the last child.

virtual const ISAO* SAF::ISAOAdmin::previousSibling () const [pure virtual]

Obtains the previous child in the same parent.

Returns:
NULL if the current object is the first child.

virtual const char* SAF::ISAOAdmin::getName (void) const [pure virtual]

Returns the name of this SAO.

The name that is returned is that used internally by SAF to identify the SAO. In the vast majority of cases this is the same name as assigned by the SAO's creator. The only situation where this differs is when the SAO implementation is a proxy in a container where getName() provides an asciified version of the SAO's id (a proxy in a SAF client does provide the assigned name). If the assigned name is needed then ISAOAdmin::getRealName() always returns the name given by the SAO's creator.

virtual const char* SAF::ISAOAdmin::getRealName (void) const [pure virtual]

Returns the "real" name of this SAO.

An SAO's real name is that assigned by the creator of the SAO.

virtual int SAF::ISAOAdmin::getFullName (ISAFValue & *target*, char *delimiter* = '/', char *escape* = '\\') const [pure virtual]

Obtains the name of the SAO up through to the root.

Simply uses the parent pointers. This is not guaranteed to give a name that can be used to request an SAO. It will include the name of the current object.

Parameters:

target This SAFValue will contain a string on return.

delimiter The character used to separate the components

escape The character used to escape the delimiter and itself so both character can be specified in a component name. These characters should be different.

Returns:

The number of parents found and included. -1 if their is no name for this object.

virtual ISOC* SAF::ISAOAdmin::getISOC () const [pure virtual]

Returns the container.

Referenced by Example::MyWrapper::destroy(), and Example::MyWrapper::initialise().

virtual SAOHist_t SAF::ISAOAdmin::getHistoryLevel (void) const [pure virtual]

Returns the current History flag.

virtual int SAF::ISAOAdmin::getLoggingLevel (void) const [pure virtual]

Returns the current logging level.

See SAOLogging_t

virtual bool SAF::ISAOAdmin::isTraceEnabled (void) const [pure virtual]

Returns true if trace is switched on.

Would normally be used to control calls to ISOC::logDebugMessage.

virtual bool SAF::ISAOAdmin::isIndex () const [pure virtual]

An index SAO is a dummy one used to get to the mount point.

virtual bool SAF::ISAOAdmin::isProtected () const [pure virtual]

A protected SAO is one found at the client which is at or above a mount point or a proxy at a server.

virtual bool SAF::ISAOAdmin::isPersistent () const [pure virtual]

A persistent SAO is one which permanently uses its allocated id, usually because it is written to the database and is the opposite of a transient SAO.

Transient SAOs can only have transient children.

virtual bool SAF::ISAOAdmin::isTransient () const [pure virtual]

A persistent SAO is one which permanently uses its allocated id, usually because it is written to the database and is the opposite of a transient SAO.

Transient SAOs can only have transient children.

virtual bool SAF::ISAOAdmin::isProxy () const [pure virtual]

A proxy is an SAO standing in for the real SAO outside the container in which the real SAO lives.

All SAOs at the client are proxies. A proxy will also be created in a container where one or more of it's SAOs depends on an SAO in another container (often in another process on another machine).

virtual bool SAF::ISAOAdmin::isSystem () const [pure virtual]

A system SAO is one which is created as part of the base framework.

Each container has the same set of system SAOs.

virtual void SAF::ISAOAdmin::setInputValidation (bool *validation*) [pure virtual]

An SAO may know that it is prepared to accept any inputs into its IOperate method and hint to the system that it would be a waste of time validating those inputs.

If this is the case any validation can be switched on and off by this method. The default is to always perform checking.

Parameters:

validation True to perform checking, false otherwise.

virtual bool SAF::ISAOAdmin::getInputValidation () const [pure virtual]

Returns true if validation is operative.

virtual const ISAOInfo* SAF::ISAOAdmin::getISAOInfo () const [pure virtual]

Returns the extended information available at the client.

This will only be non-NULL via the clientAPI and if this extended information is currently available.

virtual UserId_t SAF::ISAOAdmin::getOwner () const [pure virtual]

Returns the UserId_t of the owner of this SAO.

If the value returned is NO_USERID the owner is not known or unavailable.

virtual GroupId_t SAF::ISAOAdmin::getGroup () const [pure virtual]

Returns the GroupId_t of the group containing this SAO.

If the value returned is LIMITS_t::NO_GROUPID the owner is not known or unavailable.

virtual const SAOPerms& SAF::ISAOAdmin::getPermissions () const [pure virtual]

Returns a reference to the permission settings of this SAO.

If this information is not known or unavailable then the Pmask_t returned will indicate no access.

virtual const IOperate* SAF::ISAOAdmin::getIOperate () const [pure virtual]

Returns a pointer to the associated IOperate which may be NULL.

This method is primarily designed for library writers who wish to get at their own class implementing IOperate - normally the return will be cast to the class that it is known to be.

virtual const ICommand* SAF::ISAOAdmin::getICommand () const [pure virtual]

Returns a pointer to the associated ICommand which may be NULL.

This method is primarily designed for library writers who wish to get at their own class implementing ICommand - normally the return will be cast to the class that it is known to be.

virtual bool SAF::ISAOAdmin::watchSAOEvents (const SAOId & *sao*) [pure virtual]

Request notification of events that affect an SAO.

Whenever one of the events defined in IEventListener happens to the specified SAO then operate() is called. To actually receive the events then notifyEventListener() must be invoked from within operate(). This method may be called multiple times so that this SAO can watch many other SAOs. To receive notification of events that affect this SAO then just call with the ”sao” parameter set to this SAO’s id.

Parameters:

sao The id of the SAO that is to be watched.

Returns:

true if successful, false if not.

virtual bool SAF::ISAOAdmin::endWatchSAOEvents (const SAOId & *sao*)
`[pure virtual]`

Stop receiving notification of events that affect an SAO.

Parameters:
sao The id of the SAO that is not to be watched any longer.

Returns:
true if successful, false if not.

virtual SAF_DEPRECATED bool SAF::ISAOAdmin::watchParent ()
`[pure virtual]`

No longer supported.

Use addReservedInput(number, getParent()). Request notification of value or status changes to parent SAO. If this method is called, the operate() method of this SAO will be called whenever the value or status of its parent changes. A pointer to the parent will appear on the list of changed inputs supplied to the operate() method if a list is to be supplied at all. In effect this call is asking the container to link this SAO's parent to it as an implicit input. This method should only be called once (probably in initialise(), with a corresponding call to endWatchParent() in destroy()).

Note The parent must be available in the same container as this SAO. i.e. If getParent() for this SAO returns NULL, this call will fail.

Returns:
false if the SAO has no parent, true otherwise.

virtual SAF_DEPRECATED bool SAF::ISAOAdmin::endWatchParent ()
`[pure virtual]`

No longer supported.

Probably not needed but use removeReservedInput. Stop being notified of changes in value or status of parent.

Note It is important to cancel parent notifications prior to an SAO which receives them being destroyed.

Returns:
false if the SAO has no parent, true otherwise.

virtual bool SAF::ISAOAdmin::notifyEventListener (IEventListener & *listener*)
`[pure virtual]`

Invoke callbacks matching the pending events.

This method delivers any events that have happened to the SAOs being watched.

Parameters:
listener The listener object to make the callbacks on

Returns:

true if there were any events or false if there wasn't

virtual UserId_t SAF::ISAOAdmin::getLastChangeId () const [pure virtual]

Returns the UserId_t of the last person to cause a change to this SAO.

If the platform is responsible for the last change then 0 (SYSTEM Id) will be returned. If no data is available then NO_USERID is returned.

virtual const SAFDate& SAF::ISAOAdmin::getLastChangeTime () const [pure virtual]

Returns the time of the last change to this SAO.

This will be a null date (see SAFDate::isNull) unless the SAOs is one whose value is stored in the database.

virtual bool SAF::ISAOAdmin::isMeldable () const [pure virtual]

Returns true if the SAO can have input values merged.

SAOs default to being meldable, which means that if they are getting external updates via an ESR then two or more changes in value occurring close together can be merged into a single ripple i.e. only the latest change is used. Some SAOs, primarily those that know that they will receive sequences of updates, cannot afford to have these values melded, and they must use setMeldable(false) to prevent this from happenning. Setting an SAO with a very fast input rate not meldable might have serious impacts on the performance of the system because SAF must perform one ripple for each update.

virtual void SAF::ISAOAdmin::setMeldable (bool *meldable*) [pure virtual]

Allows the setting to be changed.

See isMeldable.

virtual bool SAF::ISAOAdmin::addReservedInput (int *index*, const ISAO * *input*) [pure virtual]

Adds an input.

This is the only way of connecting up a SAF_RESERVED input. If there is already an input present then it will be disconnected. The call to operate() which normally occurs to inform an SAO that it has had an input connected or disconnected will not occur as a result of a change caused by this method.

Parameters:

index The input number.

input The SAO being connected.

Returns:

true on success.

virtual bool SAF::ISAOAdmin::removeReservedInput (int *index*)

`[pure virtual]`

Remove an input.

This will disconnect the input, whether or not it is reserved, but is the only way of disconnecting a SAF_RESERVED input (except by deletion of one of the SAOs involved).

Parameters:

index The input number.

Returns:

true on success.

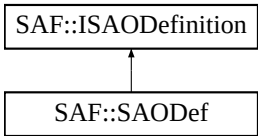
The documentation for this class was generated from the following file:

- isao.h

B.15 SAF::ISAODefinition Class Reference

`#include <isaflibrary.h>`

Inheritance diagram for SAF::ISAODefinition::



B.15.1 Detailed Description

Interface which supplies all the details needed to understand how an SAO provided by a library will operate at run time.

Definition at line 247 of file isaflibrary.h.

Public Member Functions

- virtual void destroy () const =0
- virtual const char * getName () const =0
- virtual const char * getDescription () const =0
- virtual const char * getCategory () const =0
- virtual SAFValueType_t getValueType () const =0
- virtual int getMaxInputs () const =0
- virtual int getValidatedInputs () const =0
- virtual const ISAOValidation *const * getValidation () const =0

Protected Member Functions

- virtual ~ISAODefinition ()

  Protected so instances can't be deleted through this base class.

B.15.2 Constructor & Destructor Documentation

virtual SAF::ISAODefinition::~ISAODefinition () `[inline, protected, virtual]`

Protected so instances can't be deleted through this base class.

Definition at line 314 of file isaflibrary.h.

B.15.3 Member Function Documentation

virtual void SAF::ISAODefinition::destroy () const [pure virtual]

Description:

Called when the object which implements this interface is no longer required and can be released.

Implemented in SAF::SAODef.

virtual const char* SAF::ISAODefinition::getName () const [pure virtual]

Returns:

The name of this SAO type which must be unique with the library.

Implemented in SAF::SAODef.

virtual const char* SAF::ISAODefinition::getDescription () const [pure virtual]

Returns:

The description of this SAO type (may be NULL).

Implemented in SAF::SAODef.

virtual const char* SAF::ISAODefinition::getCategory () const [pure virtual]

Description:

A category is simply a way of grouping together related SAOs in the library; for example, all the tigonometric SAOs in a maths library. The category is only used for display purposes. The format of the category string is ”category1/category2/category3/...”.

Returns:

The category of this SAO type (may be 0).

Implemented in SAF::SAODef.

virtual SAFValueType_t SAF::ISAODefinition::getValueType () const [pure virtual]

Returns:

The type of the single output produced by this SAO.

Implemented in SAF::SAODef.

virtual int SAF::ISAODefinition::getMaxInputs () const [pure virtual]

Returns:

The maximum number of valid inputs (May be SAF_UNLIMITED_INPUTS).

Implemented in SAF::SAODef.

Referenced by SAF::SAOIOperate::update().

virtual int SAF::ISAODefinition::getValidatedInputs () const [pure virtual]

Description:

The number of entries in the array returned by getValidation(). Cannot be SAF_-UNLIMITED_INPUTS. All inputs marked as MANDATORY must be present. Mandatory inputs need not be consecutive and need not be at the beginning of the array returned by getValidation(). The array returned by getValidation() may contain OPTIONAL inputs; optional inputs may be absent. If an optional input is present it is validated in the same way as a mandatory input. Any actual inputs beyond the number of (validated inputs - 1) are validated against the validation entry [validated inputs - 1] which allows the same validation to be applied to an unspecified (and limitless) number of inputs.

Returns:

The number of validated inputs.

Implemented in SAF::SAODef.

virtual const ISAOValidation* const* SAF::ISAODefinition::getValidation () const [pure virtual]

Description:

An array used to validate the inputs. Must have the same number of elements as the value returned by getValidatedInputs Return value may be NULL if the value returned by get-ValidatedInputs is zero.

Returns:

A pointer to an array of input validation criteria.

Implemented in SAF::SAODef.

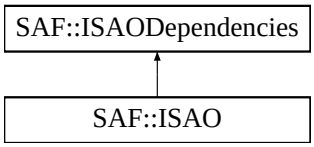
The documentation for this class was generated from the following file:

- isaflibrary.h

B.16 SAF::ISAODependencies Class Reference

`#include <isao.h>`

Inheritance diagram for SAF::ISAODependencies::



B.16.1 Detailed Description

Interface describing the dynamic properties of the dependency relationship between SAOs.

Note that setting or unsetting the input to one SAO will have a corresponding action on the outputs of another SAO.

Definition at line 932 of file isao.h.

Public Member Functions

- virtual int getInputCount () const =0

  Returns the number of inputs currently available.

- virtual const ISAO * getInput (int inputNumber) const =0

  Returns the current input on the given InputNumber or NULL if none.

- virtual const ISAO * nextInput (int &index) const =0

  Get the next non null input from the input number supplied.

- virtual int getOutputCount () const =0

  Returns the number of outputs currently available.

- virtual const ISAO * getOutput (int outputNumber) const =0

  Returns the const current output on the given OutputNumber or NULL if none.

Protected Member Functions

- virtual ~ISAODependencies ()

  Destructor.

B.16.2 Constructor & Destructor Documentation

virtual SAF::ISAODependencies::~ISAODependencies () [inline, protected, virtual]

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 939 of file isao.h.

B.16.3 Member Function Documentation

virtual int SAF::ISAODependencies::getInputCount () const [pure virtual]

Returns the number of inputs currently available.

Note that not all inputs may be connected so some calls to getInput below this number may still return NULL.

virtual const ISAO* SAF::ISAODependencies::getInput (int *inputNumber*) const [pure virtual]

Returns the current input on the given InputNumber or NULL if none.

virtual const ISAO* SAF::ISAODependencies::nextInput (int & *index*) const [pure virtual]

Get the next non null input from the input number supplied.

Allows all inputs to be selected in order after initializing the parameter to zero.

Parameters:

index First non-NULL input at or after this index number is returned and this parameter is set to one more than the index of the returned input.

Returns:

NULL if there are no inputs.

virtual int SAF::ISAODependencies::getOutputCount () const [pure virtual]

Returns the number of outputs currently available.

Note that not all below this number may be connected so some calls to getOutput below this number may still return NULL.

virtual const ISAO* SAF::ISAODependencies::getOutput (int *outputNumber*) const [pure virtual]

Returns the const current output on the given OutputNumber or NULL if none.

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

- isao.h

B.17 SAF::ISAOInfo Class Reference

`#include <isao.h>`

B.17.1 Detailed Description

Interface used by the ClientAPI to hold details about an SAO.

These details are not a key part of the SAO and can be separately requested.

Definition at line 1010 of file isao.h.

Public Member Functions

- virtual size_t getInputCount () const =0

  Returns the number of inputs currently available.

- virtual const ISAOSubInfo * getInput (size_t index) const =0

  Returns the current input on the given InputNumber or NULL if none.

- virtual size_t getOutputCount () const =0

  Returns the number of outputs currently available.

- virtual const ISAOSubInfo * getOutput (size_t index) const =0

  Returns the const current output on the given OutputNumber or NULL if none.

Protected Member Functions

- virtual ~ISAOInfo ()

  Destructor.

B.17.2 Constructor & Destructor Documentation

virtual SAF::ISAOInfo::~ISAOInfo () `[inline, protected, virtual]`

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 1016 of file isao.h.

B.17.3 Member Function Documentation

virtual size_t SAF::ISAOInfo::getInputCount () const `[pure virtual]`

Returns the number of inputs currently available.

Note that not all inputs may be connected so some calls to getInput below this number may still return NULL.

virtual const ISAOSubInfo* SAF::ISAOInfo::getInput (size_t *index*) const [pure virtual]

Returns the current input on the given InputNumber or NULL if none.

virtual size_t SAF::ISAOInfo::getOutputCount () const [pure virtual]

Returns the number of outputs currently available.

Note that not all outputs may be connected so some calls to getOutput below this number may still return NULL.

virtual const ISAOSubInfo* SAF::ISAOInfo::getOutput (size_t *index*) const [pure virtual]

Returns the const current output on the given OutputNumber or NULL if none.

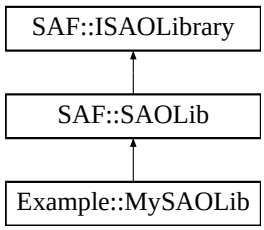
The documentation for this class was generated from the following file:

- isao.h

B.18 SAF::ISAOLibrary Class Reference

`#include <isaflibrary.h>`

Inheritance diagram for SAF::ISAOLibrary::



B.18.1 Detailed Description

Interface implemented by SAO library providers to allow the Framework to determine the nature of the objects that this library can create.

A library will be interrogated once at initialization time; it therefore provides a fixed range of objects. If it is re-implemented then it should maintain the original SAO type numbers (although if you create holes then previously existing objects cannot be created).

Definition at line 516 of file isaflibrary.h.

Public Member Functions

- virtual bool load (libraryId id, const char ∗const ∗config)=0

  The library has been loaded into the SAF process.

- virtual void unload ()=0

  The library will shortly be unloaded from the saf process.

- virtual libraryId getId () const =0

  Returns the Id provided by load.

- virtual void getVersion (int &major, int &minor) const =0

  Provides this library's version.

- virtual void getSAFVersion (int &major, int &minor) const =0

  Provides the Framework version against which this library was built.

- virtual int typeCount () const =0

  Returns the number of different types of SAO that this library can provide.

- virtual const char * getName () const =0

  Allows the library to provide a human readable name.

- virtual const char * getDescription () const =0

  Allows the library to provide a description of itself.

- virtual const ISAODefinition *const * getDefinitions () const =0

  Allows the library to provide details of the SAOs it supports.

- virtual IOperate * createOperate (ISAO &base, typeId function)=0

  Instruct the library to instantiate an IOperate object.

- virtual ICommand * createCommand (ISAO &base, typeId function)=0

  Instruct the library to instantiate an ICommand object.

Protected Member Functions

- virtual ~ISAOLibrary ()

  Protected so instances can't be deleted through this base class.

B.18.2 Constructor & Destructor Documentation

virtual SAF::ISAOLibrary::~ISAOLibrary () `[inline, protected, virtual]`

Protected so instances can't be deleted through this base class.

Definition at line 646 of file isaflibrary.h.

B.18.3 Member Function Documentation

virtual bool SAF::ISAOLibrary::load (libraryId *id*, const char *const * *config*) `[pure virtual]`

The library has been loaded into the SAF process.

This is the first method called on the library and is called immediately after the library has been loaded into the SAF process. Allows the library to do any internal initialisation.

Parameters:

id The libraryId which will be used to represent this library. The library should remember it. This is used, in conjunction with the function number used when creating an object in the library to specify the functional type (SAOFType) of an object.

config An array of NULL terminated strings terminated by a NULL pointer. If NULL then no data supplied. The meaning of these strings is opaque to the caller.

Returns:

true on success else the library will not be used.

Implemented in SAF::SAOLib.

virtual void SAF::ISAOLibrary::unload () [pure virtual]

The library will shortly be unloaded from the saf process.

This is the last method called on the library. Allows the library to perform any internal tidying up; for example, releasing any dynamic memory it might have allocated.

Implemented in SAF::SAOLib.

virtual void SAF::ISAOLibrary::getVersion (int & *major*, int & *minor*) const [pure virtual]

Provides this library's version.

The version number will have two parts; a major number and a minor number. When a library is changed, if the library is the same "shape" (i.e. new SAOs have only been added at the end and the order of the existing SAOs has not changed) then only the minor number increases and the library is guaranteed to work with all existing SAO instances. If the major number increases then the library is fundamentally incompatible with the existing SAO instances and the Admin Tool will refuse to load the description. Resolving this situation probably needs a shutdown of the SAF system and some offline tool that compares the current and new version of the library description and fixes the SAO instances in the database.

Parameters:

major On output, the library's major version number.

minor On output, the library's minor version number.

Implemented in Example::MySAOLib.

virtual void SAF::ISAOLibrary::getSAFVersion (int & *major*, int & *minor*) const [pure virtual]

Provides the Framework version against which this library was built.

The major and minor numbers are given by the #defines SAF_MAJOR_VER and SAF_MINOR_VER. These #defines are defined in the safversion.h header files and are changed as different versions of the Framework developer's kit are released.

Parameters:

major On output, the value of SAF_MAJOR_VER.

minor On output, the value of SAF_MINOR_VER.

Implemented in SAF::SAOLib.

virtual int SAF::ISAOLibrary::typeCount () const [pure virtual]

Returns the number of different types of SAO that this library can provide.

SAOs will be identified in the range zero to typeCount()-1. Each SAO is expected to be ultimately mapped to an SAOOperate::operate method but what goes on inside the library is completely private.

Implemented in Example::MySAOLib.

virtual const char* SAF::ISAOLibrary::getName () const [pure virtual]

Allows the library to provide a human readable name.

It is not a requirement that the name be unique.

Returns:

The name of the library; must not be NULL but can be "".

Implemented in Example::MySAOLib.

virtual const char* SAF::ISAOLibrary::getDescription () const [pure virtual]

Allows the library to provide a description of itself.

Returns:

A description for the library; must not be NULL but can be "".

Implemented in Example::MySAOLib.

virtual const ISAODefinition* const* SAF::ISAOLibrary::getDefinitions () const [pure virtual]

Allows the library to provide details of the SAOs it supports.

It is expected that the value returned is a pointer into static instances; the contents are not copied and should therefore not change. It should in any case have typeCount entries.

Implemented in Example::MySAOLib.

virtual IOperate* SAF::ISAOLibrary::createOperate (ISAO & *base*, typeId *function*) [pure virtual]

Instruct the library to instantiate an IOperate object.

You are responsible for ensuring that the object you return behaves in a way, with regard to its parameters for IOperate, consistent with the appropriate SAODefinition for the selected function. Note that there is no requirement to create an object for each call providing you also know what to do on the call to destroy.

Parameters:

base The base part of this SAO which will be provided with all the other calls on the returned object. You may assume that the base is not shared.

function Determines which of the library's SAOs is being created.

Returns:

NULL if the SAO does not have an Operate otherwise the Operate part of the SAO.

Implemented in Example::MySAOLib.

virtual ICommand* SAF::ISAOLibrary::createCommand (ISAO & *base*, typeId *function*) [pure virtual]

Instruct the library to instantiate an ICommand object.

You are responsible for ensuring that the object you return behaves in a way, with regard to its parameters for ICommand, consistent with the appropriate SAODefinition for the selected function. Note that there is no requirement to create an object for each call providing you also know what to do on the call to destroy.

Parameters:

base The base part of this SAO which will be provided with all the other calls on the returned object. You may assume that the base is not shared; you may also assume that createOperate() has already been called for the associated base and that this object is will be destroy()'d before the associated IOperate.

function Determines which of the library's SAOs is being created.

Returns:

NULL if the SAO does not have an Command otherwise the Command part of the SAO.

Implemented in SAF::SAOLib.

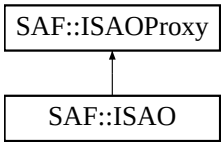
The documentation for this class was generated from the following file:

- isaflibrary.h

B.19 SAF::ISAOProxy Class Reference

`#include <isao.h>`

Inheritance diagram for SAF::ISAOProxy::



B.19.1 Detailed Description

Interface which is only realy relevant to Client code and Proxy SAOs.

Currently however the client side deals in ISAO so this will simply be part of that.

Definition at line 1420 of file isao.h.

Public Member Functions

- virtual bool isComplete () const =0

  Returns true if the SAO is on request and available.

Protected Member Functions

- virtual ~ISAOProxy ()

  Destructor.

B.19.2 Constructor & Destructor Documentation

virtual SAF::ISAOProxy::~ISAOProxy () `[inline, protected, virtual]`

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 1426 of file isao.h.

B.19.3 Member Function Documentation

virtual bool SAF::ISAOProxy::isComplete () const `[pure virtual]`

Returns true if the SAO is on request and available.

Unless this is true the names of the children for an SAO cannot be relied on and moving from child to child via next/previousSibling will not work. Non-client side SAOs always return true.

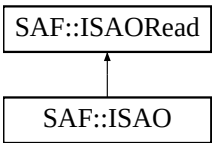
The documentation for this class was generated from the following file:

- isao.h

B.20 SAF::ISAORead Class Reference

`#include <isao.h>`

Inheritance diagram for SAF::ISAORead::



B.20.1 Detailed Description

Interface to get the data contained within an SAO.

Definition at line 517 of file isao.h.

Public Member Functions

- virtual SAFValueType_t getType (void) const =0

  Returns the current data type.

- virtual const char * getTypeName () const =0

  Returns the name of the current data type.

- virtual bool isArray (void) const =0

  Returns true for an array data type.

- virtual bool isVoid () const =0

  Returns true for an object taking no value.

- virtual bool isAmorphous (void) const =0

  Returns true for an amorphous data type.

- virtual double getDouble (bool &valid) const =0

  Accessor.

- virtual double getDouble () const =0

  Accessor.

- virtual int getInt (bool &valid) const =0

  Accessor.

- virtual int getInt () const =0

  Accessor.

- virtual saf64_t getInt64 (bool &valid) const =0

  Accessor.

- virtual saf64_t getInt64 () const =0

  Accessor.

- virtual const char * getString (bool &valid) const =0

  Accessor.

- virtual const char * getString () const =0

  Accessor.

- virtual bool isRawString (void) const =0

  Accessor.

- virtual const char * getRawString (bool &valid) const =0

  Accessor.

- virtual const char * getRawString () const =0

  Accessor.

- virtual bool getBool (bool &valid) const =0

  Accessor.

- virtual bool getBool () const =0

  Accessor.

- virtual SAFDate getDate (bool &valid) const =0

  Accessor.

- virtual SAFDate getDate () const =0

  Accessor.

- virtual const double * getDoubleArray (int &length) const =0

  Accessor.

- virtual const int * getIntArray (int &length) const =0

  Accessor.

- virtual const saf64_t * getLongArray (int &length) const =0

  Accessor.

- virtual const unsigned char * getByteArray (int &length) const =0

  Accessor.

- virtual const char *const * getStringArray (int &length) const =0

  Accessor.

- virtual const saf64_t * getDateArray (int &length) const =0

  Accessor.

- virtual SAFStatus_t getStatus (void) const =0

  Returns the data status.

- virtual bool isValid (void) const =0

  Returns true when the status is SAF_VALID.

- virtual bool notValid (void) const =0

  Returns true for any status except SAF_VALID.

- virtual bool isStale (void) const =0

  Returns true when the status is SAF_STALE.

- virtual bool isAvailable (void) const =0

  Returns true if there is data available.

- virtual bool notAvailable (void) const =0

  Returns the opposite of isAvailable.

- virtual bool isDisconnected (void) const =0

Returns true if the status means that there is not a complete path from the supplier of the data to the caller of this method.

- virtual bool isSuspended (void) const =0

  Returns true if the status is SAF_SUSPENDED.

- virtual bool isDenied (void) const =0

  Returns true if the status is SAF_ACCESS_DENIED.

- virtual const char * **getErrorText** () const =0
- virtual const ISAFValue & **getValue** () const =0

Protected Member Functions

- virtual ~ISAORead ()

  Destructor.

B.20.2 Constructor & Destructor Documentation

virtual SAF::ISAORead::~ISAORead () [inline, protected, virtual]

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 524 of file isao.h.

B.20.3 Member Function Documentation

virtual SAFValueType_t SAF::ISAORead::getType (void) const [pure virtual]

Returns the current data type.

For an amorphous object this may not be set and may change with use of setValue.

virtual const char* SAF::ISAORead::getTypeName () const [pure virtual]

Returns the name of the current data type.

Has same provisos as getType().

virtual bool SAF::ISAORead::isArray (void) const [pure virtual]

Returns true for an array data type.

virtual bool SAF::ISAORead::isVoid () const [pure virtual]

Returns true for an object taking no value.

virtual bool SAF::ISAORead::isAmorphous (void) const [pure virtual]

Returns true for an amorphous data type.

An amorphous data type will return the best value it can for all the base type accessors.

virtual double SAF::ISAORead::getDouble (bool & *valid*) const [pure virtual]

Accessor.

Obtains the data value and an indication of its validity. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then zero will be returned.

Parameters:

valid True on return if a real data value is provided else false.

Returns:

the data or a zero default.

virtual double SAF::ISAORead::getDouble () const [pure virtual]

Accessor.

Obtains the data value. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then a zero default will be returned. You cannot tell the difference between failure and a good zero value.

Returns:

the data or a zero default.

virtual int SAF::ISAORead::getInt (bool & *valid*) const [pure virtual]

Accessor.

Obtains the data value and an indication of its validity. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then zero will be returned.

Parameters:

valid True on return if a real data value is provided else false.

Returns:

the data or a zero default.

virtual int SAF::ISAORead::getInt () const [pure virtual]

Accessor.

Obtains the data value. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then a zero default will be returned. You cannot tell the difference between failure and a good zero value.

Returns:

the data or a zero default.

virtual saf64_t SAF::ISAORead::getInt64 (bool & *valid*) const [pure virtual]

Accessor.

Obtains the data value and an indication of its validity. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then zero will be returned.

Parameters:

valid True on return if a real data value is provided else false.

Returns:

the data or a zero default.

virtual saf64_t SAF::ISAORead::getInt64 () const [pure virtual]

Accessor.

Obtains the data value. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then a zero default will be returned. You cannot tell the difference between failure and a good zero value.

Returns:

the data or a zero default.

virtual const char* SAF::ISAORead::getString (bool & *valid*) const [pure virtual]

Accessor.

Obtains the data value and an indication of its validity. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then zero will be returned.

Parameters:

valid True on return if a real data value is provided else false.

Returns:

the data or a zero default.

virtual const char* SAF::ISAORead::getString () const [pure virtual]

Accessor.

Obtains the data value. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then a zero default will be returned. You cannot tell the difference between failure and a good zero value.

Returns:

the data or a zero default.

virtual bool SAF::ISAORead::isRawString (void) const [pure virtual]

Accessor.

Returns true if there is a separate raw string available.

virtual const char* SAF::ISAORead::getRawString (bool & *valid*) const [pure virtual]

Accessor.

An amorphous SAO might store a raw string as derived directly from an external data source. It is not suitable for converting into a number. Where there is no separate string available this method behaves exactly like getString.

Parameters:

valid True on return if a real data value is provided else false.

Returns:

the data or a zero default.

virtual const char* SAF::ISAORead::getRawString () const [pure virtual]

Accessor.

An amorphous SAO might store a raw string as derived directly from an external data source. It is not suitable for converting into a number. This method will only return a real raw string not one converted from another value.

Returns:

the data or a zero default.

virtual bool SAF::ISAORead::getBool (bool & *valid*) const [pure virtual]

Accessor.

Obtains the data value and an indication of its validity. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then false will be returned.

Parameters:

valid True on return if a real data value is provided else false.

Returns:

the data or a zero (false) default.

virtual bool SAF::ISAORead::getBool () const [pure virtual]

Accessor.

Obtains the data value. Data conversion will take place if necessary. If the data type is inappropriate or if the data is notAvailable() then a zero default will be returned. You cannot tell the difference between failure and a good zero value.

Returns:

the data or a zero default.

virtual SAFDate SAF::ISAORead::getDate (bool & *valid*) const [pure virtual]

Accessor.

Obtains the data value and an indication of its validity. Data conversion cannot take place; this method will only succeed if the type is SAF_DATE.

Parameters:

valid True on return if a real data value is provided else false.

Returns:

the data or a zero default.

virtual SAFDate SAF::ISAORead::getDate () const [pure virtual]

Accessor.

Obtains the data value. Data conversion cannot take place; this method will only succeed if the type is SAF_DATE. If the data type is inappropriate or if the data is notAvailable() then a zero default will be returned. You cannot tell the difference between failure and a good zero value.

Returns:

the data or a zero default.

virtual const double* SAF::ISAORead::getDoubleArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is notAvailable or not of the correct format then zero is returned. Conversion will not take place.

Parameters:

length Has the length of the array on return.

virtual const int* SAF::ISAORead::getIntArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is notAvailable or not of the correct format then zero is returned. Conversion will not take place.

Parameters:

length Has the length of the array on return.

virtual const saf64_t* SAF::ISAORead::getLongArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is notAvailable or not of the correct format then zero is returned. Conversion will not take place.

Parameters:

length Has the length of the array on return.

virtual const unsigned char* SAF::ISAORead::getByteArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is notAvailable or not of the correct format then zero is returned. Conversion will not take place.

Parameters:

length Has the length of the array on return.

virtual const char* const* SAF::ISAORead::getStringArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is notAvailable or not of the correct format then zero is returned. Conversion will not take place.

Parameters:

length Has the length of the array on return.

virtual const saf64_t* SAF::ISAORead::getDateArray (int & *length*) const [pure virtual]

Accessor.

Returns the data value. If status is not of the correct format then zero is returned. Conversion will not take place. Note that date array are stored as an array of longs; NULL values indicate an empty array slot. A SAFDate can be obtained by construction from the array value.

Parameters:

length Has the length of the array on return.

virtual SAFStatus_t SAF::ISAORead::getStatus (void) const [pure virtual]

Returns the data status.

Actual values of status are not intended for use and are subject to change. Normally you are only interested in the category of status as supplied by isValid() and isAvailable() and their opposites notValid() and notAvailable(). There is also isDisconnected().

virtual bool SAF::ISAORead::isAvailable (void) const [pure virtual]

Returns true if there is data available.

This means a good status (i.e. SAF_VALID or SAF_STALE) and a data type other than SAF_VOID.

virtual bool SAF::ISAORead::notAvailable (void) const [pure virtual]

Returns the opposite of isAvailable.

See ISAORead::isAvailable

virtual bool SAF::ISAORead::isDisconnected (void) const [pure virtual]

Returns true if the status means that there is not a complete path from the supplier of the data to the caller of this method.

virtual bool SAF::ISAORead::isSuspended (void) const [pure virtual]

Returns true if the status is SAF_SUSPENDED.

An SAO is suspended if either its IOperate or ICommand parts throw an exception. It would be unusual for one SAO to have both parts but the whole thing is suspended anyway.

virtual bool SAF::ISAORead::isDenied (void) const [pure virtual]

Returns true if the status is SAF_ACCESS_DENIED.

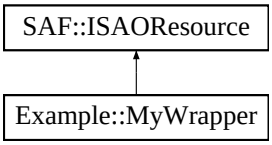
The documentation for this class was generated from the following file:

- isao.h

B.21 SAF::ISAOResource Class Reference

`#include <isaoresource.h>`

Inheritance diagram for SAF::ISAOResource::



B.21.1 Detailed Description

Interface to be implemented by objects which hold multiple values to be shared by multiple SAOs.

Objects which implement this interface are stored and accessed via an implementation of ISAOResourceManager.

Definition at line 69 of file isaoresource.h.

Public Member Functions

- virtual bool getResourceNumberByName (const char ∗name, SAFValueType_t type, unsigned int &resourceNumber)=0
- virtual bool setValueByResourceNumber (ISAO &value, unsigned int resourceNumber, bool init=false)=0
- virtual bool setValueByResourceNumber (ISAFValue &value, unsigned int resourceNumber, bool init=false)=0

B.21.2 Member Function Documentation

virtual bool SAF::ISAOResource::getResourceNumberByName (const char ∗ *name*, SAFValueType_t *type*, unsigned int & *resourceNumber*) [pure virtual]

Description:

Look up a resource number based on a supplied name.

Parameters:

name The name of the resource for which a number is required.

type The type of the values which will need to be supplied.

resourceNumber The resource number which is to be set according to the supplied name.

Returns:

False if the specified name does not correspond to a resource number with the correct type, true otherwise.

Implemented in Example::MyWrapper.

virtual bool SAF::ISAOResource::setValueByResourceNumber (ISAO & *value*, unsigned int *resourceNumber*, bool *init* = `false`) `[pure virtual]`

Description:

Set the value of the supplied ISAO according to the specified resource number if the resource is available (if the status of the supplied SAO is not set explicitely it will default).

Parameters:

value The ISAO whose value is to be set. Note that it must be set to a specific type, available as value.getDefinition()->getValueType() (the same as was provided in the registering call of getResourceNumberByName()).

resourceNumber The number of the resource to be used.

init If false (the default) the value will be set only if the value of the resource has changed since a previous request.

Returns:

True if the value was set, false otherwise.

Implemented in Example::MyWrapper.

virtual bool SAF::ISAOResource::setValueByResourceNumber (ISAFValue & *value*, unsigned int *resourceNumber*, bool *init* = `false`) `[pure virtual]`

Description:

Set the value of the supplied ISAFValue according to the specified resource number if the resource is available.

Parameters:

value The ISAFValue to be set.

resourceNumber The number of the resource to be used.

init If false (the default) the value will be set only if the value of the resource has changed since a previous request.

Returns:

True if the value was set, false otherwise.

Implemented in Example::MyWrapper.

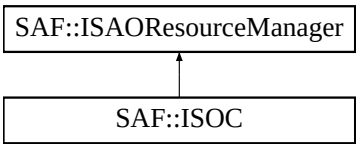
The documentation for this class was generated from the following file:

- isaoresource.h

B.22 SAF::ISAOResourceManager Class Reference

`#include <isaoresource.h>`

Inheritance diagram for SAF::ISAOResourceManager::



B.22.1 Detailed Description

Objects which implement this interface arrange to store and retrieve objects which implement the ISAOResource interface.

ISOC implements this interface.

Definition at line 122 of file isaoresource.h.

Public Member Functions

- virtual void addResource (const ISAO &base, ISAOResource &resource)=0
- virtual void removeResource (const ISAO &base)=0
- virtual ISAOResource * getResource (const ISAO &base) const =0

B.22.2 Member Function Documentation

virtual void SAF::ISAOResourceManager::addResource (const ISAO & *base*, ISAOResource & *resource*) [pure virtual]

Description:

Add an object which implements ISAOResource to an ISAOResourceManager

Parameters:

base The SAO which owns the resource.

resource The object which provides the resource.

Referenced by Example::MyWrapper::initialise().

virtual void SAF::ISAOResourceManager::removeResource (const ISAO & *base*) [pure virtual]

Description:

Remove an object which implements ISAOResource from an ISAOResourceManager

Parameters:

base The SAO which owns the resource.

Referenced by Example::MyWrapper::destroy().

virtual ISAOResource* SAF::ISAOResourceManager::getResource (const ISAO & *base*) const [pure virtual]

Parameters:

base The SAO which owns the resource.

Returns:

An object which implements ISAOResource from an ISAOResourceManager owned by the specified ISAO reference.

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

- isaoresource.h

B.23 SAF::ISAOSubInfo Class Reference

`#include <isao.h>`

B.23.1 Detailed Description

Interface used by the ClientAPI to hold details about a single other SAO, whether an input, output, virtual input or virtual output.

Definition at line 982 of file isao.h.

Public Member Functions

- virtual const SAOId & getId () const =0

  Get the SAO's id.

- virtual const ISAOSpec & getSpec () const =0

  Get the available details of this SAO.

Protected Member Functions

- virtual ~ISAOSubInfo ()

  Destructor.

B.23.2 Constructor & Destructor Documentation

virtual SAF::ISAOSubInfo::~ISAOSubInfo () `[inline, protected, virtual]`

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 988 of file isao.h.

B.23.3 Member Function Documentation

virtual const SAOId& SAF::ISAOSubInfo::getId () const `[pure virtual]`

Get the SAO's id.

virtual const ISAOSpec& SAF::ISAOSubInfo::getSpec () const `[pure virtual]`

Get the available details of this SAO.

If the spec is byId then it will contain no more information than the SAOId, normally it will have the path in it. The path will not be present when there is a proxy involved in the source container.

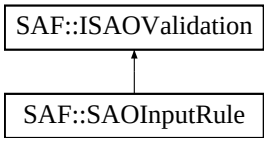
The documentation for this class was generated from the following file:

- isao.h

B.24 SAF::ISAOValidation Class Reference

`#include <isaflibrary.h>`

Inheritance diagram for SAF::ISAOValidation::



B.24.1 Detailed Description

Interface used to obtain details of the validation of a single input to the operate method.

Definition at line 200 of file isaflibrary.h.

Public Member Functions

- virtual void destroy () const =0
- virtual SAFValueType_t getValueType () const =0
- virtual SAOInputValidation_t getValidationCode () const =0
- virtual const char * getDescription () const =0

Protected Member Functions

- virtual ~ISAOValidation ()

  Protected so instances can't be deleted through this base class.

B.24.2 Constructor & Destructor Documentation

virtual SAF::ISAOValidation::~ISAOValidation () `[inline, protected, virtual]`

Protected so instances can't be deleted through this base class.

Definition at line 235 of file isaflibrary.h.

B.24.3 Member Function Documentation

virtual void SAF::ISAOValidation::destroy () const `[pure virtual]`

Description:

Called when the object which implements this interface is no longer required and can be released.

Implemented in SAF::SAOInputRule.

virtual SAFValueType_t SAF::ISAOValidation::getValueType () const
[pure virtual]

Description:

The type of input allowed. If SAF_UNDEFINED then any basic (i.e. non array) type will be acceptable and will be converted . If SAF_VOID then any input will be allowed. If the SAOInputValidation_t is SAF_OPTIONAL then the type will still be used if an input is present i.e. if it is present then it must be correct.

Returns:

The type of this input.

Implemented in SAF::SAOInputRule.

virtual SAOInputValidation_t SAF::ISAOValidation::getValidationCode () const
[pure virtual]

Description:

Allows the status of an input to be taken into account. An input may only be acceptable if it is valid, others may be acceptable if they are stale.

Returns:

The acceptable status level fo this input.

Implemented in SAF::SAOInputRule.

virtual const char* SAF::ISAOValidation::getDescription () const
[pure virtual]

Returns:

The description of this input or 0 if there isn't one.

Implemented in SAF::SAOInputRule.

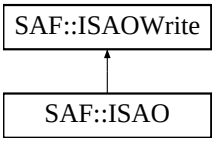
The documentation for this class was generated from the following file:

- isaflibrary.h

B.25 SAF::ISAOWrite Class Reference

`#include <isao.h>`

Inheritance diagram for SAF::ISAOWrite::



B.25.1 Detailed Description

Interface to set the data contained within an SAO.

Definition at line 799 of file isao.h.

Public Member Functions

- virtual bool setType (SAFValueType_t type)=0

  Sets the data type.

- virtual bool setValue (bool data)=0

  Sets the data value.

- virtual bool **setValue** (double data)=0
- virtual bool **setValue** (int data)=0
- virtual bool **setValue** (saf64_t data)=0
- virtual bool **setValue** (const char *data)=0
- virtual bool **setValue** (const SAFDate &data)=0
- virtual bool setValue (const ISAFValue &data)=0

  Sets the data value.

- virtual bool setValue (const double *data, int length)=0

  Sets the data value.

- virtual bool **setValue** (const int *data, int length)=0
- virtual bool **setValue** (const saf64_t *data, int length)=0
- virtual bool **setValue** (const unsigned char *data, int length)=0
- virtual bool **setValue** (const char *const *data, int length)=0

- virtual bool **setValue** (const SAFDate ∗data, int length)=0
- virtual bool setValue (double data, int index)=0

  Sets a single data value for array data types.

- virtual bool **setValue** (const int data, int index)=0
- virtual bool **setValue** (const saf64_t data, int index)=0
- virtual bool **setValue** (const unsigned char data, int index)=0
- virtual bool **setValue** (const char ∗data, int index)=0
- virtual bool **setValue** (const SAFDate &data, int index)=0
- virtual bool setStatus (SAFStatus_t status)=0

  Sets the data status.

- virtual bool setErrorText (SAFStatus_t status, const char ∗text)=0

  Sets the data status and provides some error text.

- virtual void setValid ()=0

  If the status is currently SAF_STALE then sets it to SAF_VALID otherwise the status is not changed.

- virtual void setStale ()=0

  If the status is currently SAF_VALID then sets it to SAF_STALE otherwise the status is not changed.

Protected Member Functions

- virtual ∼ISAOWrite ()

  Destructor.

B.25.2 Constructor & Destructor Documentation

virtual SAF::ISAOWrite::∼ISAOWrite () `[inline, protected, virtual]`

Destructor.

Protected so instances can't be deleted through this base class.

Definition at line 806 of file isao.h.

B.25.3 Member Function Documentation

virtual bool SAF::ISAOWrite::setType (SAFValueType_t *type*) [pure virtual]

Sets the data type.

If the SAO already has an incompatible type then this method is ignored. Note that the first use of setValue or setType will determine the data type irrevocably although the AMORPHOUS property may allow other types later.

Parameters:

type The desired type

Returns:

true if the type was allowed.

virtual bool SAF::ISAOWrite::setValue (bool *data*) [pure virtual]

Sets the data value.

If the data type is constrained by the definition of the SAO then the value given will be cast to the required type where appropriate - for example an integer may be placed into a SAF_INT64 or a SAF_BOOL. If the SAO is AMORPHOUS then the type of the SAO will be changed to match the input. More sophisticated conversion, for example allowing the string ”45.32” to be used to set a SAF_INT or a SAF_DOUBLE, will not take place. Assignments will succeed if the data type is compatible. If the assignment succeeds then the status is set to SAF_VALID unless the call takes place within the context of a ripple and there are one or more SAF_STALE inputs in which case it is set SAF_STALE. This is to support the management of stale inputs as described in IOperate::operate.

Returns:

true if the assignment succeeded.

Referenced by Example::MyWrapper::setValueByResourceNumber().

virtual bool SAF::ISAOWrite::setValue (const ISAFValue & *data*) [pure virtual]

Sets the data value.

This is not an assignment operation. It ignores a stale setting in the source, ignores the source completely if it is notAvailable(), won’t copy if the source is a SAF_VOID and doesn’t always copy arrays. Apparently this is useful functionality in some situations but I doubt that anybody will ever find such a situation.

virtual bool SAF::ISAOWrite::setValue (const double * *data*, int *length*) [pure virtual]

Sets the data value.

If the SAO is not of the appropriate type then this method is ignored. Conversion will not take place. Assignments will succeed if the data type is compatible. If the assignment succeeds then

the status is set to SAF_VALID unless the call takes place within the context of a ripple and there are one or more SAF_STALE inputs in which case it is set SAF_STALE. This is to support the management of stale inputs as described in IOperate::operate.

Returns:
true if the assignment succeeded.

virtual bool SAF::ISAOWrite::setValue (double *data*, int *index*) [pure virtual]

Sets a single data value for array data types.

If the SAO is not of the appropriate type then this method is ignored. Conversion will not take place. These assignments will only succeed if the data type is an array of the correct type. If the array is not long enough then it is lengthened by inserting zero values. If the assignment succeeds then the status is set to SAF_VALID unless it is currently SAF_STALE in which case it is left as SAF_STALE. This is to support the management of stale inputs as described in IOperate::operate.

Returns:
true if the assignment succeeded.

virtual bool SAF::ISAOWrite::setStatus (SAFStatus_t *status*) [pure virtual]

Sets the data status.

Normally this would not be necessary. The status can only be changed to a worse state (ie isValid can degrade to isAvailable can degrade to notAvailable) and can be raised by using setValue). If the new status is not worse than the old then this call is ignored and false returned.

Returns:
true if the assignment changes the value for rippling purposes.

virtual bool SAF::ISAOWrite::setErrorText (SAFStatus_t *status*, const char * *text*) [pure virtual]

Sets the data status and provides some error text.

If the status is SAF_STALE or SAF_VALID or the text is NULL or "" then this is the same as setStatus. Otherwise the status is set unconditionally to that given and the error text is copied. The error text will only last until the next call to the IOperate::Operate method. Setting an error text will make the SAO believe that it has changed for rippling purposes.

Returns:
false unless it was the same as setStatus.

virtual void SAF::ISAOWrite::setValid () [pure virtual]

If the status is currently SAF_STALE then sets it to SAF_VALID otherwise the status is not changed.

This method can be used inside an operate() method when it is not wished to accept the default SAO state supplied when setValue is used.

virtual void SAF::ISAOWrite::setStale () [pure virtual]

If the status is currently SAF_VALID then sets it to SAF_STALE otherwise the status is not changed.

This method can be used inside an operate() method when it is not wished to accept the default SAO state supplied when setValue is used.

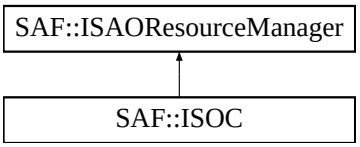
The documentation for this class was generated from the following file:

- isao.h

B.26 SAF::ISOC Class Reference

`#include <isoc.h>`

Inheritance diagram for SAF::ISOC::



B.26.1 Detailed Description

This interface is visible to SAOs and defines the operations that are available to SAO 'operate' method code.

Essentially the interface provides SAO discovery and linking facilities as well as support for logging and archiving through a database.

In general, the SAO pointer is obtained via getSAO(...) and then operations are performed on the SAO. It is imperative that SAO pointers are not retained between calls to operate() because the container may receive instructions to delete an SAO between calls.

Definition at line 253 of file isoc.h.

Public Types

- enum { **APPEND_SAO** = -1 }
- enum MessageSeverity_t { **LOG_DEBUG** = 0, **LOG_INFO** = 1, **LOG_WARNING** = 2, **LOG_ERROR** = 3 }

  The importance of messages logged by the container.

Public Member Functions

- virtual ISAO * createISAO (const SAOFType &ftype, const char *name, const ISAO *parent=NULL, const ISAO *next=NULL, bool save=true, ISAFValue *reply=NULL, SAOHist_t history=SAO_HIST_OFF)=0
- virtual ISAO * createISAO (const char *libName, const char *saoName, const char *name, const ISAO *parent=NULL, const ISAO *next=NULL, bool save=true, ISAFValue *reply=NULL, SAOHist_t history=SAO_HIST_OFF)=0
- virtual ISAO * copyHierarchy (const ISAO *source, const char *name, const ISAO *parent, const ISAO *next=NULL, bool all=true, bool transient=false, ISAFValue *reply=NULL)=0
- virtual bool addInputToSAO (const ISAO *source, const ISAO *dest, int inputNumber, bool overwrite=false, ISAFValue *reply=NULL, bool transient=false)=0

- virtual bool **addInputToSAO** (const ISAFValue ∗source, const ISAFValue ∗dest, int inputNumber, bool overwrite, ISAFValue &reply, bool transient=false)=0
- virtual bool addInputToSAO (const SAOId &source, const ISAO ∗dest, int inputNumber, bool overwrite=false, ISAFValue ∗reply=NULL, bool transient=false)=0
- virtual bool removeInputFromSAO (const ISAO ∗source, const ISAO ∗dest, int inputNumber, ISAFValue ∗reply=NULL)=0

  Remove an input from an SAO.

- virtual const ISAO ∗ getISAO (SAOId id)=0
- virtual const ISAO ∗ **getISAO** (const ISAOSpec ∗spec, bool fullPath=true)=0
- virtual const ISAO ∗ getISAO (const ISAOSpec &spec, bool fullPath=true)=0
- virtual bool moveSAO (const ISAO ∗sao, const ISAO ∗parent, const ISAO ∗next=NULL, ISAFValue ∗reply=NULL)=0

  Move an SAO to a new location in the hierarchy by specifying a new parent SAO.

- virtual bool removeSAOTree (const ISAO ∗sao, ISAFValue ∗reply=NULL)=0
- virtual bool renameSAO (const ISAO ∗sao, const char ∗name, ISAFValue ∗reply=NULL)=0
- virtual bool transformHierarchy (const ISAO ∗source, const ISAO ∗dest, bool transient=false, ISAFValue ∗reply=NULL)=0
- virtual void logMessage (const MessageSeverity_t severity, const char ∗formatString,...)=0
- virtual void logDebugMessage (const char ∗formatString,...)=0
- virtual void logErrorMessage (const char ∗formatString,...)=0
- virtual void logWarnMessage (const char ∗formatString,...)=0
- virtual void logInfoMessage (const char ∗formatString,...)=0
- virtual void logDebugMessage (const ISAO ∗sao, const char ∗formatString,...)=0
- virtual void logInfoMessage (const ISAO ∗sao, const char ∗formatString,...)=0
- virtual void logWarnMessage (const ISAO ∗sao, const char ∗formatString,...)=0
- virtual void logErrorMessage (const ISAO ∗sao, const char ∗formatString,...)=0
- virtual bool postValueToSAO (const ISAO ∗sao, const ISAFValue ∗value)=0

  Post a value to an SAO.

- virtual bool setSAOProperties (const ISAO ∗sao, const IProperties &props, ISAFValue ∗reply=NULL)=0

  Set the SAO properties.

- virtual const ISAFUser * getUser () const =0

  Get access to current context.

- virtual bool reserveSAO (const ISAO &sao, const ISAO &base, ISAFValue *reply=NULL)=0
- virtual bool releaseSAO (const ISAO &sao, const ISAO &base, ISAFValue *reply=NULL)=0
- virtual bool exportHierarchy (const ISAO *source, bool inputs, bool values, bool transients, const ISAFValue *cids, const char *desc, ISAFValue &reply)=0

  Export part of the SAO hierarchy as XML.

- virtual bool importHierarchy (const ISAO *parent, const char *name, const ISAO *next, SAOId &importFrom, const ISAFValue *libraries, const char *import, bool inputs, bool values, bool transients, bool makeTransient, ISAFValue *reply)=0

  Called to import hierarchy from export file.

- virtual containerId getId () const =0
- virtual IBlackBox * getIBlackBox ()=0
- virtual void disableLockUpDetection ()=0

  Allows an SAO operation to specify that it is coded in such a way that it might block the container for a long time (in excess of 10 seconds) and that the normal mechanism which would detect what is assumed to be an errant SAO should be disabled until the end of the current call.

B.26.2 Member Function Documentation

virtual ISAO* SAF::ISOC::createISAO (const SAOFType & *ftype*, const char * *name*, const ISAO * *parent* = `NULL`, const ISAO * *next* = `NULL`, bool *save* = `true`, ISAFValue * *reply* = `NULL`, SAOHist_t *history* = `SAO_HIST_OFF`) `[pure virtual]`

Description:

Add a new SAO to the SAO hierarchy of this container according to the specified functional type. If successfully created, the SAO is added to the hierarchy as a child of the specified parent location.

Parameters:

ftype Functional type identifying the SAO to be created.

name The name of the new SAO (must not be NULL or an empty string).

parent The parent of the new SAO. If NULL (the default) the SAO will be added to the root set of SAOs with no parent.

next If specified, this SAO must be an existing child of the specified parent and will result in the new SAO being created immediately before this SAO in the parent's list of children. If NULL (the default), the new SAO is added as the last child of the specified parent.

save If true (the default) the new SAO will be persisted in the configuration database. In this case the parent (if specified) must be also be persistent or the call will fail.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

history For an SAO to be created persistent will also cause its value to be stored in the database.

Returns:

A pointer to the newly created SAO or NULL in case of failure.

virtual ISAO* SAF::ISOC::createISAO (const char * *libName*, const char * *saoName*, const char * *name*, const ISAO * *parent* = `NULL`, const ISAO * *next* = `NULL`, bool *save* = `true`, ISAFValue * *reply* = `NULL`, SAOHist_t *history* = `SAO_HIST_OFF`) `[pure virtual]`

Description

Add a new SAO to the SAO hierarchy of this container according to the specified SAO type name and library name. If successfully created, the SAO is added to the hierarchy as a child of the specified parent location.

Parameters:

libName The name of the library containing the SAO to be created.

saoName The name of the SAO definition within the library.

parent The parent of the new SAO. If NULL (the default) the SAO will be added to the root set of SAOs with no parent.

next If specified, this SAO must be an existing child of the specified parent and will result in the new SAO being created immediately before this SAO in the parent's list of children. If NULL (the default), the new SAO is added as the last child of the specified parent.

save If true (the default) the new SAO will be persisted in the configuration database. In this case the parent (if specified) must be also be persistent or the call will fail.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

history For an SAO to be created persistent will also cause its value to be stored in the database.

Returns:

A pointer to the newly created SAO or NULL in case of failure.

virtual ISAO* SAF::ISOC::copyHierarchy (const ISAO * *source*,
const char * *name*, const ISAO * *parent*, const ISAO * *next* = `NULL`,
bool *all* = `true`, bool *transient* = `false`, ISAFValue * *reply* = `NULL`)
`[pure virtual]`

Description:

Copy a hierarchy of SAOs and parent them beneath the supplied parent SAO. Where two SAOs in the hierarchy to be copied are connected to each other the equivalent connection will be formed between the corresponding SAOs of the copy. Where an SAO in the hierarchy to be copied uses some SAO external to that hierarchy as an input, the copy of that SAO will be connected to the same external SAO on its corresponding input. Where an SAO in the hierarchy to be copied is used by some SAO external to that hierarchy as an input, the copy of that SAO will not form any connection to the external SAO.

Parameters:

source The root of the hierarchy to be copied.

name The name to be given to the copy of the root SAO. The copy will fail if an SAO of the specified name already exists as a child of the specified parent.

parent The parent of the new SAO hierarchy. If NULL (the default) the hierarchy will be added to the root set of SAOs with no parent.

next If specified, this SAO must be an existing child of the specified parent and will result in the new hierarchy being created immediately before this SAO in the parent's list of children. If NULL (the default), the new hierarchy is added as the last child of the specified parent.

all If true (the default) all SAOs in the source hierarchy, both persistent and transient, will be copied. If false only persistent SAOs will be copied.

transient If false (the default) the new SAOs will be persisted in the configuration database. In this case the parent (if specified) must be also be persistent or the call will fail.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

Returns:

A pointer to the newly created hierarchy root SAO or NULL in case of failure.

virtual bool SAF::ISOC::addInputToSAO (const ISAO * *source*,
const ISAO * *dest*, int *inputNumber*, bool *overwrite* = `false`,
ISAFValue * *reply* = `NULL`, bool *transient* = `false`) `[pure virtual]`

Description:

Add an input to an SAO. The input SAO must be of a type which is appropriate to the input definition of the SAO to which it is to be connected.

Parameters:

source The SAO providing the input.

dest The SAO to receive the input.

inputNumber The input number of the destination SAO to which the source SAO is to be connected.

overwrite If true (the default is false) replace any existing input connection at the specified input number with the source SAO.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

transient Set true if you do not wish this connection to be persisted to the database.

Returns:

True on success, false otherwise.

virtual bool SAF::ISOC::addInputToSAO (const SAOId & *source*, const ISAO * *dest*, int *inputNumber*, bool *overwrite* = `false`, ISAFValue * *reply* = `NULL`, bool *transient* = `false`) `[pure virtual]`

Description:

Add an input to an SAO. The input SAO must be of a type which is appropriate to the input definition of the SAO to which it is to be connected.

Parameters:

source The SAOId of the SAO to provide the input. If this is not in the current container then a proxy will be created.

dest The SAO to receive the input.

inputNumber The input number of the destination SAO to which the source SAO is to be connected.

overwrite If true (the default is false) replace any existing input connection at the specified input number with the source SAO.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

transient Set true if you do not wish this connection to be persisted to the database.

Returns:

True on success, false otherwise.

virtual bool SAF::ISOC::removeInputFromSAO (const ISAO * *source*, const ISAO * *dest*, int *inputNumber*, ISAFValue * *reply* = `NULL`) `[pure virtual]`

Remove an input from an SAO.

Dest in the SAO with an input. If a number is given then it is used regardless of source. If not then source will be disconnected from dest however many times it is connected; if zero times then this does nothing successfully.

Parameters:

source The SAO providing the input (may be NULL)

dest The SAO with an input

inputNumber The input number of the SAO (-1 means find the source)

reply Holds information on why the command failed. Its status will be set to SAF_VALID if it has been used.

Returns:
true if it worked.

virtual const ISAO* SAF::ISOC::getISAO (SAOId *id*) [pure virtual]

Description:
Supplies a pointer to an SAO based on the specified id. Note that only pointers to SAOs in the current container can be returned.

Parameters:
id An SAOId which identifies an SAO in the current container.

Returns:
A pointer to the SAO with the id specified or NULL if it is not found in this container.

virtual const ISAO* SAF::ISOC::getISAO (const ISAOSpec & *spec*, bool *fullPath* = true) [pure virtual]

Description:
Supplies a pointer to an SAO based on the supplied path specified using an ISAOSpec. Note that only pointers to SAOs in the current container can be returned.

Parameters:
spec An ISAOSpec which identifies an SAO in the current container.

fullPath If true (the default) then it is assumed that the supplied path includes the path to the mount point of this container and therefore appears as a full path within the application hierarchy as the client would see it. If false then the path is assumed to be relative to the root of the container.

Returns:
A pointer to the SAO or NULL if it is not found in this container.

virtual bool SAF::ISOC::moveSAO (const ISAO * *sao*, const ISAO * *parent*, const ISAO * *next* = NULL, ISAFValue * *reply* = NULL) [pure virtual]

Move an SAO to a new location in the hierarchy by specifying a new parent SAO.

All the children of this SAO will move with it.

Parameters:
sao The SAO to move.

parent A new parent SAO for the SAO being moved. If NULL the SAO will be repositioned with the existing parent.

next If specified then this SAO becomes the next child in the new parent otherwise it becomes the last child.

reply Holds information on why the command failed. Its status will be set to SAF_VALID if it has been used.

Returns:

true if the call works. The call will fail if the parent has a child at the specified location or has a child of the same name as the SAO.

virtual bool SAF::ISOC::removeSAOTree (const ISAO * *sao*, ISAFValue * *reply* = NULL) [pure virtual]

Description:

Delete an SAO and all its children (if any) from the container. Where the SAO or its children are connected as inputs to other SAOs those inputs are disconnected.

Parameters:

sao The SAO to be deleted.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

Returns:

True if the command succeeds, false otherwise.

virtual bool SAF::ISOC::renameSAO (const ISAO * *sao*, const char * *name*, ISAFValue * *reply* = NULL) [pure virtual]

Description:

Rename an SAO.

Parameters:

sao The SAO whose name is to be changed.

name The new name for the SAO. The call will fail if the SAO's parent already has a child with the same name as that supplied.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

Returns:

True if the command succeeds, false otherwise.

virtual bool SAF::ISOC::transformHierarchy (const ISAO * *source*, const ISAO * *dest*, bool *transient* = false, ISAFValue * *reply* = NULL) [pure virtual]

Description:

Examine the hierarchies parented by the two supplied SAOs and modify the destination hierarchy so that it matches the source hierarchy using as few changes as possible by inserting or removing SAOs within the destination hierarchy. Connections to SAOs in the source hierarchy are replicated in the destination hierarchy according to the same rules as those applied by the copyHierarchy() method.

Parameters:

source The SAO at the root of the source hierarchy.

dest The SAO at the root of the destination hierarchy.

transient If false (the default) all newly created SAOs will be persisted in the configuration database, otherwise all new SAOs will be created as transient.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

Returns:

True if the command succeeds, false otherwise.

virtual void SAF::ISOC::logMessage (const MessageSeverity_t *severity*, const char * *formatString*, ...) [pure virtual]

Description:

Log a message with parameters.

Parameters:

severity The level of severity to be associated with this message.

formatString A printf format string.

virtual void SAF::ISOC::logDebugMessage (const char * *formatString*, ...) [pure virtual]

Description:

Log a debug message with parameters.

Parameters:

formatString A printf format string.

virtual void SAF::ISOC::logErrorMessage (const char * *formatString*, ...) [pure virtual]

Description:

Log an error message with parameters.

Parameters:

formatString A printf format string.

virtual void SAF::ISOC::logWarnMessage (const char * *formatString*, ...) [pure virtual]

Description:

Log a warning message with parameters.

Parameters:

formatString A printf format string.

virtual void SAF::ISOC::logInfoMessage (const char * *formatString*, ...) [pure virtual]

Description:

Log an information message with parameters.

Parameters:

formatString A printf format string.

virtual void SAF::ISOC::logDebugMessage (const ISAO * *sao*, const char * *formatString*, ...) [pure virtual]

Description:

Log a debug message that includes details of the specified SAO. The message is automatically prefixed with sao information using the result of the ISAO::toString() method.

Parameters:

sao The SAO whose details are to be included in the message.

formatString A printf format string.

virtual void SAF::ISOC::logInfoMessage (const ISAO * *sao*, const char * *formatString*, ...) [pure virtual]

Description:

Log an information message that includes details of the specified SAO. The message is automatically prefixed with sao information using the result of the ISAO::toString() method.

Parameters:

sao The SAO whose details are to be included in the message.

formatString A printf format string.

virtual void SAF::ISOC::logWarnMessage (const ISAO * *sao*, const char * *formatString*, ...) [pure virtual]

Description:

Log a warning message that includes details of the specified SAO. The message is automatically prefixed with sao information using the result of the ISAO::toString() method.

Parameters:

sao The SAO whose details are to be included in the message.

formatString A printf format string.

virtual void SAF::ISOC::logErrorMessage (const ISAO * *sao*, const char * *formatString*, ...) [pure virtual]

Description:

Log an error message that includes details of the specified SAO. The message is automatically prefixed with sao information using the result of the ISAO::toString() method.

Parameters:

sao The SAO whose details are to be included in the message.

formatString A printf format string.

virtual bool SAF::ISOC::postValueToSAO (const ISAO * *sao*, const ISAFValue * *value*) [pure virtual]

Post a value to an SAO.

This method copies the value and queues it on the containers input queue. The sao SAO must not have any inputs defined, and the types of the SAO and the value must match. The value is placed on the FRONT of the containers input queue and delivered to the SAO when it is removed from the queue. The new value will NEVER appear in the SAO prior to returning from this call.

Parameters:

sao The SAO to receive the value, which must be in this container.

value The value to post

virtual bool SAF::ISOC::setSAOProperties (const ISAO * *sao*, const IProperties & *props*, ISAFValue * *reply* = NULL) [pure virtual]

Set the SAO properties.

The properties object should contain properties of appropriate names and values. Properties that are not recognised are ignored.

Parameters:

sao The SAO to have its properties changed

props The properties object to be used to set the SAO properties.

reply Holds information on why the command failed. Its status will be set to SAF_VALID if it has been used.

virtual const ISAFUser* SAF::ISOC::getUser () const [pure virtual]

Get access to current context.

This allows you to see details of the user under which code is currently being run. The pointer returned has a very short lifetime - it is only valid within the current invocation of your code. For instance if in ISAO::operate() then the pointer is invalid as soon as you return from operate().

Returns:

The current user. May be NULL.

virtual bool SAF::ISOC::reserveSAO (const ISAO & *sao*, const ISAO & *base*, ISAFValue * *reply* = NULL) [pure virtual]

Description:

Reserve and SAO. Reserving an SAO has no effect other than that once an SAO is reserve any other call to reserveSAO() will fail. This allows applications to cooperatively control access to SAOs since an SAO can only be reserved once. As a failsafe mechanism if the SAO is not released within 5 minutes of being reserved then it is automatically released.

Parameters:

sao The SAO to be reserved.

base An SAO to be identified as having reserved the SAO (a subsequent releaseSAO must specify the same SAO to succeed).

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

Returns:

True if the SAO has been reserved, false otherwise.

virtual bool SAF::ISOC::releaseSAO (const ISAO & *sao*, const ISAO & *base*, ISAFValue * *reply* = NULL) [pure virtual]

Description:

Called to free a reserved SAO (see reserveSAO()).

Parameters:

sao The reserved SAO to be released.

base The reserving SAO specified in the call to reserveSAO() which reserved the SAO.

reply An optional pointer to an ISAFValue to act as a holder for diagnostic information should the command fail. The status will be set to SAF_VALID if it has been used.

Returns:

True if the call succeeds, false if it fails (perhaps because the specified base is not responsible for the reservation on the SAO).

virtual bool SAF::ISOC::exportHierarchy (const ISAO * *source*, bool *inputs*, bool *values*, bool *transients*, const ISAFValue * *cids*, const char * *desc*, ISAFValue & *reply*) [pure virtual]

Export part of the SAO hierarchy as XML.

The exported file can be imported into another **SONARIS/Framework** application (or the originating application) using importHierarchy().

Parameters:

source The SAO at the root of the hierarchy to be exported

inputs If true SAO input information is exported. Normally, only inputs from within the tree and for system SAOs will be retained. This behaviour can be modified by using the cids parameter.

values If true SAO values are exported, otherwise they are not.

transients If true transient SAOs are exported, otherwise they are not.

cids When the export is being performed across multiple containers this field contains an array of Ids containers being exported whose inputs should be retained where they would normally be discarded.

desc A description to be associated with the export. Can be NULL or an empty string. The actual content will be treated as HTML if it starts with '<html>' (my quotes).

reply Contains the export information as a string if the export works. If the export fails it will contain an appropriate error message.

Returns:

true if it worked and a value is in reply. false if it failed and an error message is in reply. The reply value (in the true return case) is a SAF_STRING_ARRAY. The array will have three elements: in [0] is the SAFVersion of the container <major>.<minor> in [1] is the SAFXMLVersion of the container <major>.<minor> in [2] is the Export XML

virtual bool SAF::ISOC::importHierarchy (const ISAO * *parent*, const char * *name*, const ISAO * *next*, SAOId & *importFrom*, const ISAFValue * *libraries*, const char * *import*, bool *inputs*, bool *values*, bool *transients*, bool *makeTransient*, ISAFValue * *reply*) [pure virtual]

Called to import hierarchy from export file.

The imported file is validated for the correct content, including the presence of the appropriate libraries in the application being imported into.

Parameters:

parent The root within which the imported tree will be created.

name The name of the SAO to be created as the root of the imported tree. If this name is empty or NULL, then the name of the root SAO in the import will be used.

next The SAO to be the next sibling of the SAO created. If NULL, the SAO will be the last sibling of the parent.

importFrom The ID of an SAO in the exported hierarchy which will form the root of the import (rather than the real root). Currently, this parameter is ignored.

libraries An array of integers that determines the mapping between library ids in the exported hierarchy and the application being imported into. Ids are arranged in pairs in the array. The export Id first, followed by the Id in the application. Where a library is to use the same Id as in the export file it can be omitted. Where SAOs from a library are not be imported, the application library Id should be -1.

import A valid export definition. Export definitions are in parsable XML. Currently validation of the XML is minimal, but later versions will be validated against an XML schema.

inputs Should input information be imported. Only valid for an export file where inputs were exported.

values Should values be imported. Only valid for an export file where values were exported.

transients Should transient SAOs be imported. Only valid if transient SAOs were exported.

makeTransient Should the imported SAOs be made transient?

reply Will contain an error message if the import fails, and the SAOId of the newly created root SAO if the import works.

Returns:

true if it worked .

virtual containerId SAF::ISOC::getId () const [pure virtual]

Returns:

the container Id of this container.

virtual IBlackBox* SAF::ISOC::getIBlackBox () [pure virtual]

Returns:

A pointer to an implementation of IBlackBox trace logger which will be checked in the event of an SAO being suspended after a call to any method on its IOperate interface as the result of an exception.

virtual void SAF::ISOC::disableLockUpDetection () [pure virtual]

Allows an SAO operation to specify that it is coded in such a way that it might block the container for a long time (in excess of 10 seconds) and that the normal mechanism which would detect what is assumed to be an errant SAO should be disabled until the end of the current call.

Note that any SAO taking this long will destroy the realtime nature of anything else that is supposed to be happenning in the container and such things would only be expected to be invoked when fast throughput in not important.

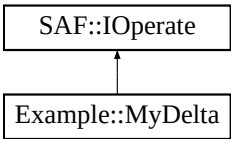
The documentation for this class was generated from the following file:

- isoc.h

B.27 Example::MyDelta Class Reference

`#include <mydelta.h>`

Inheritance diagram for Example::MyDelta::



B.27.1 Detailed Description

MyDelta is an SAO which sets its value to be the difference between the current value of its input, and its previous value.

Definition at line 15 of file mydelta.h.

Public Types

- enum **Input_t** { **DELTA_INPUT** = 0 }

Public Member Functions

- MyDelta ()

  Constructor - initialise member variables.

- virtual void destroy (ISAO &base)

  Called by the framework when it wants to delete instances of this SAO - but we get final control of that.

- virtual void initialise (ISAO &base)

  No special initialisation to do.

- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)

  Ignore any updates which are sent to us.

- virtual bool rename (const ISAO &base, const char *name)

  Allow instances of this SAO to be renamed.

- virtual bool operate (ISAO &base, int count, ISAO **changed)

  Takes the value of our input and calculates the difference between it and its previous value.

Static Public Member Functions

- const ISAODefinition * **getDefinition** ()
- const ISAOValidation *const * **getInputRules** ()

Protected Member Functions

- virtual ~MyDelta ()

  Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Protected Attributes

- bool _**set**
- double _previous

  > *Have we seen a previous value?*

B.27.2 Constructor & Destructor Documentation

virtual Example::MyDelta::~MyDelta () `[inline, protected, virtual]`

Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Nothing special to do here anyway.

Definition at line 60 of file mydelta.h.

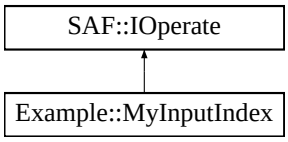
The documentation for this class was generated from the following file:

- mydelta.h

B.28 Example::MyInputIndex Class Reference

`#include <myinputindex.h>`

Inheritance diagram for Example::MyInputIndex::



B.28.1 Detailed Description

MyInputIndex.

Definition at line 73 of file myinputindex.h.

Public Types

- enum Input_t {

  INPUT_A = 0, **INPUT_B** = 1, **INPUT_C** = 2, **INPUT_D** = 3,

  INPUT_X = 4 }

  An enumeration of our input types.

Public Member Functions

- MyInputIndex ()

  Constructor - nothing special to do here.

- virtual void destroy (ISAO &base)

  Called by the framework when it wants to delete instances of this SAO - but we get final control of that.

- virtual void initialise (ISAO &base)

  No special initialisation to do.

- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)

  Ignore any updates which are sent to us.

- virtual bool rename (const ISAO &base, const char *name)

  Allow instances of this SAO to be renamed.

- virtual bool operate (ISAO &base, int count, ISAO **changed)

Use the _inputIndex to decide which inputs have changed.

Static Public Member Functions

- const ISAODefinition * getDefinition ()

  Static method for access to our SAOdefinition.

- const ISAOValidation *const * getInputRules ()

  Static method for access to our input rules.

Protected Member Functions

- virtual ~MyInputIndex ()

  Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Protected Attributes

- InputIndex< ISAO > _inputIndex

  Use an InputIndex to keep track of input changes.

- AutoFree< char > _tmpStr

  A character array in which to prepare the value of this SAO.

B.28.2 Constructor & Destructor Documentation

virtual Example::MyInputIndex::~MyInputIndex () `[inline, protected, virtual]`

Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Nothing special to do here anyway.

Definition at line 84 of file myinputindex.h.

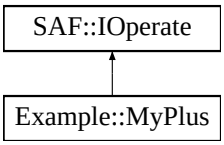
The documentation for this class was generated from the following file:

- myinputindex.h

B.29 Example::MyPlus Class Reference

`#include <myplus.h>`

Inheritance diagram for Example::MyPlus::



B.29.1 Detailed Description

MyPlus is an SAO which takes two numbers as inputs and adds them together.

As with any SAO, it must implement the IOperate interface to allow it to work as part of the Framework.

Definition at line 77 of file myplus.h.

Public Types

- enum Input_t { **FIRST_INPUT** = 0, **SECOND_INPUT** = 1 }

  An enumeration of the input indexes of this SAO.

Public Member Functions

- MyPlus ()

  Constructor - nothing special to do here.

- virtual void destroy (ISAO &base)

  Called by the framework when it wants to delete instances of this SAO - but we get final control of that.

- virtual void initialise (ISAO &base)

  No special initialisation to do.

- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)

  Ignore any updates which are sent to us.

- virtual bool rename (const ISAO &base, const char *name)

  Allow instances of this SAO to be renamed.

- virtual bool operate (ISAO &base, int count, ISAO **changed)

Take the values of our two inputs, add them together, and set our value accordingly.

Static Public Member Functions

- const ISAODefinition ∗ getDefinition ()

  Static method for access to the definition of this SAO.

- const ISAOValidation ∗const ∗ getInputRules ()

  Static method for access to the input rules of this SAO.

Protected Member Functions

- virtual ∼MyPlus ()

  Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

B.29.2 Constructor & Destructor Documentation

virtual Example::MyPlus::∼MyPlus () [inline, protected, virtual]

Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Nothing special to do here anyway.

Definition at line 124 of file myplus.h.

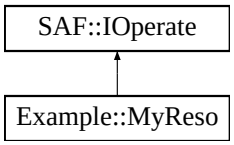
The documentation for this class was generated from the following file:

- myplus.h

B.30 Example::MyReso Class Reference

`#include <myreso.h>`

Inheritance diagram for Example::MyReso::



B.30.1 Detailed Description

SAO which sets its value from a resource available from its input SAO.

Definition at line 32 of file myreso.h.

Public Types

- enum Input_t { **RESOURCE_INPUT** = 0 }

  An enumeration of the input indexes of this SAO.

Public Member Functions

- MyReso ()

  Constructor - initialise member variables.

- virtual void destroy (ISAO &base)

  Called by the framework when it wants to delete instances of this SAO - but we get final control of that.

- virtual void initialise (ISAO &base)

  No special initialisation to do.

- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)

  Ignore any updates which are sent to us.

- virtual bool rename (const ISAO &base, const char *name)

  All instances of this SAO to be renamed.

- virtual bool operate (ISAO &base, int count, ISAO **changed)

  Takes the value of our input and calculates the difference between it and its previous value.

Static Public Member Functions

- const ISAODefinition * getDefinition ()

  Static method for access to the definition of this SAO.

- const ISAOValidation *const * getInputRules ()

  Static method for access to the input rules of this SAO.

Protected Member Functions

- virtual ~MyReso ()

  Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Protected Attributes

- SAOId _inputId

  Keep track of the input SAO in case it is replaced with another one.

- ISAOResource * _resource

  Keep a pointer to the resource available from our input SAO.

- unsigned int _resoNumber

  Keep track of the resource number which corresponds to our name.

B.30.2 Constructor & Destructor Documentation

virtual Example::MyReso::~MyReso () [inline, protected, virtual]

Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Nothing special to do here anyway.

Definition at line 86 of file myreso.h.

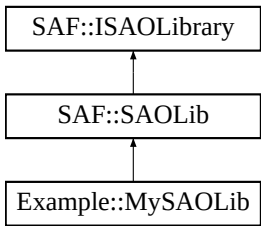
The documentation for this class was generated from the following file:

- myreso.h

B.31 Example::MySAOLib Class Reference

`#include <mysaolib.h>`

Inheritance diagram for Example::MySAOLib::



B.31.1 Detailed Description

MySAOLib is based on SAOLib from safhelper.h which will manage the supply of instances of the MyPlus SAO.

Definition at line 16 of file mysaolib.h.

Public Member Functions

- MySAOLib ()

  Constructor - nothing special to do here.

- virtual ~MySAOLib ()

  Destructor - nothing special to do here.

- virtual const char * getName () const

  Return our name when asked.

- virtual void getVersion (int &major, int &minor) const

  Supply version information for this library when asked.

- virtual int typeCount () const

  Return the number of SAO types supplied by this library.

- virtual const char * getDescription () const

  Return a description of this library.

- virtual const ISAODefinition *const * getDefinitions () const

  Return a pointer to an array of ISAODefinition objects which describe the SAO types supplied by this library.

- virtual IOperate ∗ createOperate (ISAO &base, typeId function)

  Create a new instance of the SAO specified by the supplied typeId.

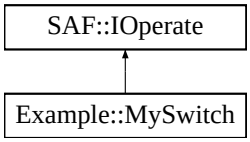
The documentation for this class was generated from the following file:

- mysaolib.h

B.32 Example::MySwitch Class Reference

`#include <myswitch.h>`

Inheritance diagram for Example::MySwitch::



B.32.1 Detailed Description

MySwitch is an SAO which sets its value to be that of its nth input where n is specified on its first input.

Definition at line 15 of file myswitch.h.

Public Types

- enum Input_t { **INDEX_INPUT** = 0, **FIRST_DATA_INPUT** = 1 }

  An enumeration of the input indexes of this SAO.

Public Member Functions

- MySwitch ()

  Constructor - nothing special to do here.

- virtual void destroy (ISAO &base)

  Called by the framework when it wants to delete instances of this SAO - but we get final control of that.

- virtual void initialise (ISAO &base)

  No special initialisation to do.

- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)

  Ignore any updates which are sent to us.

- virtual bool rename (const ISAO &base, const char *name)
- virtual bool operate (ISAO &base, int count, ISAO **changed)

  Take the value n of our first input and use it to set our value to our nth input.

Static Public Member Functions

- const ISAODefinition * getDefinition ()

  Static method for access to the definition of this SAO.

- const ISAOValidation *const * getInputRules ()

  Static method for access to the input rules of this SAO.

Protected Member Functions

- virtual ~MySwitch ()

  Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

B.32.2 Constructor & Destructor Documentation

virtual Example::MySwitch::~MySwitch () [inline, protected, virtual]

Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Nothing special to do here anyway.

Definition at line 60 of file myswitch.h.

B.32.3 Member Function Documentation

virtual bool Example::MySwitch::rename (const ISAO & *base*, const char * *name*) [inline, virtual]

Description:

This method is called to inform of a suggested change of name. The old name is still present in base.

Parameters:

base The base part of this SAO (the same value that was supplied to the call to ISAOLibrary::createOperate() which returned this object).

name The proposed new name. Note that you do not have to change the name in base nor worry about any conflict with an identically named child in the parent.

Returns:

True to permit the name to be changed, false otherwise.

Implements SAF::IOperate.

Definition at line 48 of file myswitch.h.

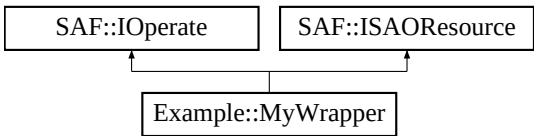
The documentation for this class was generated from the following file:

- myswitch.h

B.33 Example::MyWrapper Class Reference

`#include <mywrapper.h>`

Inheritance diagram for Example::MyWrapper::



B.33.1 Detailed Description

MyWrapper is an SAO which takes two numbers as inputs and passes them to an instance of Calculator to produce results for three separate arithmetic operations.

The results of these are made available to other SAOs via the ISAOResource interface.

Definition at line 80 of file mywrapper.h.

Public Types

- enum Input_t { **FIRST_INPUT** = 0, **SECOND_INPUT** = 1 }

  An enumeration of the input indexes of this SAO.

Public Member Functions

- MyWrapper ()

  Constructor - nothing special to do here.

- virtual void initialise (ISAO &base)

  Ask the ISAOResourceManager (implemented by ISOC) to make resources available to other SAOs.

- virtual void destroy (ISAO &base)

  Called by the framework when it wants to delete instances of this SAO - but we get final control of that.

- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)

  Ignore any updates which are sent to us.

- virtual bool rename (const ISAO &base, const char *name)

  Allow instances of this SAO to be renamed.

- virtual bool operate (ISAO &base, int count, ISAO **changed)

  Take the values of our two inputs, add them together, and set our value accordingly.

- virtual bool getResourceNumberByName (const char *name, SAFValueType_t type, unsigned int &resourceNumber)
- virtual bool setValueByResourceNumber (ISAFValue &value, unsigned int resourceNumber, bool init)
- virtual bool setValueByResourceNumber (ISAO &value, unsigned int resourceNumber, bool init)

Static Public Member Functions

- const ISAODefinition * getDefinition ()

  Static method for access to the definition of this SAO.

- const ISAOValidation *const * getInputRules ()

  Static method for access to the input rules of this SAO.

Protected Types

- enum **ResourceFlags_t** { **SUM** = 0x0001, **DIFF** = 0x0002, **PROD** = 0x0004 }

Protected Member Functions

- virtual ~MyWrapper ()

  Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Protected Attributes

- unsigned int **_resourceFlags**
- Calculator _calculator

  Use an instance of Calculator to do the work for us.

B.33.2 Constructor & Destructor Documentation

virtual Example::MyWrapper::~MyWrapper () [inline, protected, virtual]

Destructor is protected - the framework will indicate when it is ready to delete SAOs by calling destroy().

Nothing special to do here anyway.

Definition at line 165 of file mywrapper.h.

B.33.3 Member Function Documentation

virtual void Example::MyWrapper::destroy (ISAO & *base*) [inline, virtual]

Called by the framework when it wants to delete instances of this SAO - but we get final control of that.

Tell the ISAOResourceManager to withdraw our resources.

Implements SAF::IOperate.

Definition at line 107 of file mywrapper.h.

References SAF::ISAOAdmin::getISOC(), and SAF::ISAOResourceManager::removeResource().

virtual bool Example::MyWrapper::getResourceNumberByName (const char * *name*, SAFValueType_t *type*, unsigned int & *resourceNumber*) [virtual]

Description:

Look up a resource number based on a supplied name.

Parameters:

name The name of the resource for which a number is required.

type The type of the values which will need to be supplied.

resourceNumber The resource number which is to be set according to the supplied name.

Returns:

False if the specified name does not correspond to a resource number with the correct type, true otherwise.

Implements SAF::ISAOResource.

virtual bool Example::MyWrapper::setValueByResourceNumber (ISAFValue & *value*, unsigned int *resourceNumber*, bool *init*) [virtual]

Description:

Set the value of the supplied ISAFValue according to the specified resource number if the resource is available.

Parameters:

value The ISAFValue to be set.

resourceNumber The number of the resource to be used.

init If false (the default) the value will be set only if the value of the resource has changed since a previous request.

Returns:

True if the value was set, false otherwise.

Implements SAF::ISAOResource.

virtual bool Example::MyWrapper::setValueByResourceNumber (ISAO & *value*, unsigned int *resourceNumber*, bool *init*) [inline, virtual]

Description:

Set the value of the supplied ISAO according to the specified resource number if the resource is available (if the status of the supplied SAO is not set explicitely it will default).

Parameters:

value The ISAO whose value is to be set. Note that it must be set to a specific type, available as value.getDefinition()->getValueType() (the same as was provided in the registering call of getResourceNumberByName()).

resourceNumber The number of the resource to be used.

init If false (the default) the value will be set only if the value of the resource has changed since a previous request.

Returns:

True if the value was set, false otherwise.

Implements SAF::ISAOResource.

Definition at line 135 of file mywrapper.h.

References SAF::ISAOWrite::setValue().

B.33.4 Member Data Documentation

Calculator Example::MyWrapper::_calculator [protected]

Use an instance of Calculator to do the work for us.

Its values will be available to other SAOs through our implementation of the ISAOResource interface.

Definition at line 160 of file mywrapper.h.

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

- mywrapper.h

B.34 SAF::SAFDate Class Reference

`#include <isafvalue.h>`

B.34.1 Detailed Description

A small class (taking up only 8 bytes convertible to a saf64_t) which allows either a date, a time of day or a single point in time to be represented to a resolution of milliseconds.

SAFDate is built on top of the common system utilities time(), mktime(), gmtime() and localtime() but with additions to allow dates from 1900-2999 to be dealt with. Note however that not all of these dates can be converted to a system time. If it has only a date then no conversions will be attempted for local time. If it has only a time then conversions for local time will wrap and the ”date” will always be Jan 1 1970; such conversions will never fail. If it has both a date and time the local time conversions will take place. A < comparison operation is provided; it is valid to make comparisons between the saf64_t which represents this object but, of course, you need to ensure that they represent similar things (e.g two times, or two dates). No validation is performed for consistency or correctness of inputs, although in general they will be passed unchecked to the underlying utilities (mktime, localtime etc). Code which sets things like tm_mon or tm_mday way outside the normal range is not guaranteed to work because of the manipulation which occurs to support the extended range. **Note** The overrides of operator new() and operator delete() are to prevent attempts to allocate in one heap and free in another. Use create() and destroy() to get a dynamically allocated instance of this class.

Definition at line 672 of file isafvalue.h.

Public Member Functions

- SAFDate (saftime_t t, int msec=0)
- SAFDate (const struct tm *date, int msec=0)
- SAFDate (int msec, const struct tm *date)
- SAF_DEPRECATED SAFDate (const struct tm &date)
- SAFDate (int year, int month, int day, int hour, int min=0, int sec=0, int msec=0)
- SAFDate (int year, int month, int day)
- SAFDate (bool dummy, int hour=0, int min=0, int sec=0, int msec=0)
- SAFDate (const char *date)
- SAFDate ()
- SAFDate (saf64_t value, bool dummy)
- SAFDate (const SAFDate &src)
- SAFDate & operator= (const SAFDate &src)
- bool operator< (const SAFDate &other) const

- bool operator> (const SAFDate &other) const
- bool operator== (const SAFDate &other) const
- SAFDate & operator= (saf64_t val)
- operator saf64_t () const
- ~SAFDate ()
- void setTimeOnly ()
- void setDateOnly ()
- bool hasDate () const
- bool hasTime () const
- bool isNull () const
- bool hasSystemTime () const
- saftime_t systemTime () const
- tm * utc () const
- tm * localtime () const
- bool getDate (int &year, int &month, int &day) const
- bool getTime (int &hour, int &min, int &sec, int &msec) const
- int getMsec () const
- bool setMsec (int msec)
- saf64_t seconds () const
- saf64_t milliseconds () const
- SAFDate & setNow ()
- SAF_DEPRECATED SAFDate & setNowLocal ()
- int codeSQL (char *buf) const
- void decodeSQL (int codeVal)
- char * iso (char *buf, bool localtime=true) const
- void dump (char *buf) const
- bool parseDate (const char *dateString)
- void destroy () const

Static Public Member Functions

- bool parseDate (const char *dateString, saf64_t &date)
- SAFDate * create ()

Protected Member Functions

- void * **operator new** (size_t size)
- void **operator delete** (void *mem)

B.34.2 Constructor & Destructor Documentation

SAF::SAFDate::SAFDate (saftime_t *t*, int *msec* = 0)

Description:

Constructor taking system time.

Parameters:

t Number of seconds in the epoch (implicitly UTC); often this will be the result of a call to time() (or _time64 on Windows)

msec The milliseconds in the range 0-999

SAF::SAFDate::SAFDate (const struct tm * *date*, int *msec* = 0)

Description:

Constructor from a tm structure.

Parameters:

date This method is similar in its behaviour to mktime() except for the added ability to copy with years from 1900 to 1969.

msec The milliseconds in the range 0-999

SAF::SAFDate::SAFDate (int *msec*, const struct tm * *date*)

Description:

Constructor from a tm structure assumed to contain UTC.

Parameters:

msec The milliseconds in the range 0-999

date This method uses mktime() internally, but nullifies the effect of local time adjustment.

SAF_DEPRECATED SAF::SAFDate::SAFDate (const struct tm & *date*)

Description:

Deprecated constructor from a tm structure containing localtime.

Parameters:

date The tm_year, tm_mon, tm_mday, tm_hour, tm_min and tm_sec ONLY will be used - no conversion will take place and the values are assumed to be in the correct ranges e.g. tm_mday is 0-30.

SAF::SAFDate::SAFDate (int *year*, int *month*, int *day*, int *hour*, int *min* = 0, int *sec* = 0, int *msec* = 0)

Description:

Constructor taking a date and time assumed to be in localtime.

Parameters:

year The year from 1900 upwards.

month The month in the range 1-12

day The day in the range 1-31

hour The hour in the range 0-23

min The minutes in the range 0-59

sec The seconds in the range 0-59

msec The milliseconds in the range 0-999

SAF::SAFDate::SAFDate (int *year*, int *month*, int *day*)

Description:

Constructor taking a date only. Implicitly setDateOnly().

Parameters:

year The year from 1900 upwards.

month The month in the range 1-12

day The day in the range 1-31

SAF::SAFDate::SAFDate (bool *dummy*, int *hour* = 0, int *min* = 0, int *sec* = 0, int *msec* = 0)

Description:

Constructor taking a local time only. Implicitly setTimeOnly().

Parameters:

dummy This is present just to distinguish the constructors and is ignored

hour The hour in the range 0-23

min The minutes in the range 0-59

sec The seconds in the range 0-59

msec The milliseconds in the range 0-999

SAF::SAFDate::SAFDate (const char * *date*)

Description:

Constructor from a localtime string. The string is parsed using `SAFDate::parseDate(const char *, saf64_t &)`

Parameters:

date The date string. If all parses fail the SAFDate is empty.

SAF::SAFDate::SAFDate ()

Description:

Default constructor. Creates a date with neither date nor time.

SAF::SAFDate::SAFDate (saf64_t *value*, bool *dummy*)

Description:

Constructor taking a saf64_t

Parameters:

value This is not validated and had better have come from another SAFDate if it is going to make any sense.

dummy. Used only do distinguish this constructor from the more normal saftime_t one.

SAF::SAFDate::SAFDate (const SAFDate & *src*)

Description:

Copy Constructor.

SAF::SAFDate::~SAFDate () [inline]

Description:

Destructor - does nothing. Deliberately not virtual to allow SAFDate and saf64_t pointers to be castable to each other.

Definition at line 801 of file isafvalue.h.

B.34.3 Member Function Documentation

SAFDate& SAF::SAFDate::operator= (const SAFDate & *src*)

Description:

Assignment.

bool SAF::SAFDate::operator< (const SAFDate & *other*) const

Description:

Comparison. Makes meaningful comparisons between values of the same type.

bool SAF::SAFDate::operator> (const SAFDate & *other*) const

Description:

Comparison. Makes meaningful comparisons between values of the same type.

bool SAF::SAFDate::operator== (const SAFDate & *other*) const

Description:

Comparison. Makes meaningful comparisons between values of the same type.

SAFDate& SAF::SAFDate::operator= (saf64_t *val*)

Description:

Assignment taking a saf64_t (which had better have come from another SAFDate if it is going to make any sense).

SAF::SAFDate::operator saf64_t () const

Description:

Typecast operator to supply a date representation as a saf64_t

void SAF::SAFDate::setTimeOnly ()

Description:

Forces the date part to be removed. If no date part is present then does nothing. A SAFDate with only a time will still be stored in UTC and may be fetched in localTime or UTC. Note however that without a date it will be always converted as for the DST applying at 1/1/1970.

void SAF::SAFDate::setDateOnly ()

Description:

Forces the time part to be removed. If no time part is present then does nothing. Note that the UTC and Localtime date might be different prior to the removal of the time; the Localtime date is chosen.

bool SAF::SAFDate::hasDate () const

Returns:

true if the SAFDate has a date within it; it might also contain a time.

bool SAF::SAFDate::hasTime () const

Returns:

true if the SAFDate has a time within it; it might also contain a date.

bool SAF::SAFDate::isNull () const

Returns:

true if the SAFDate has neither date nor time.

bool SAF::SAFDate::hasSystemTime () const

Returns:

true if the date is convertible to a system time i.e. not prior to 1970.

saftime_t SAF::SAFDate::systemTime () const

Description:

Returns the UTC time represented or -1 if the date is prior to 1970.

struct tm* SAF::SAFDate::utc () const

Description:

Uses gmtime() to return the date in UTC.

struct tm* SAF::SAFDate::localtime () const

Description:

Uses localtime() to return the date in localtime.

bool SAF::SAFDate::getDate (int & *year*, int & *month*, int & *day*) const

Description:

Obtains the date in localtime. This method functions via a call of localtime().

Parameters:

year The year from 1900 upwards.

month The month in the range 1-12

day The day in the range 1-31

Returns:

False if the SAFDate represents only a time (in which case the date is set to 1st Jan 0 AD), otherwise true.

bool SAF::SAFDate::getTime (int & *hour*, int & *min*, int & *sec*, int & *msec*) const

Description:

Obtains the time in localtime. This method functions via a call of localtime().

Parameters:

hour The hour in the range 0-23

min The minutes in the range 0-59

sec The seconds in the range 0-59

msec The milliseconds in the range 0-999

Returns:

False if the SAFDate represents only a date (in which case the time is set to 00:00:00.000), otherwise true.

int SAF::SAFDate::getMsec () const

Returns:

The number of milliseconds stored in the date

bool SAF::SAFDate::setMsec (int *msec*)

Description:

Sets the number of msec stored.

Parameters:

msec The milliseconds in the range 0-999

Returns:

True unless hasTime() is false.

saf64_t SAF::SAFDate::seconds () const

Returns:

The number of seconds since the epoch represented by the date if it is in the range supported by the operating system, -1 otherwise. If the SAFDate represents only a time then the number of seconds since midnight is returned. Same as systemTime().

saf64_t SAF::SAFDate::milliseconds () const

Returns:

The number of milliseconds since the epoch represented by the date if it is in the range supported by the operating system, -1 otherwise. If the SAFDate represents only a time then the number of milliseconds since midnight is returned.

SAFDate& SAF::SAFDate::setNow ()

Description:

Sets the SAFDate to be the current system time.

Returns:

This object.

SAF_DEPRECATED SAFDate& SAF::SAFDate::setNowLocal ()

Description:

No longer a valid operation. Same as setNow.

Returns:

This object

int SAF::SAFDate::codeSQL (char * *buf*) const

Description:

Used to produce the information required to store this date in an SQL database.

Parameters:

buf This area needs to be SAFDATESQLSTRING bytes in size and has the following put into it. "'yyyy-mm-dd hh:mm:ss'". Note that the single quotes are part of the output string. If the date cannopt be converted then supplies "'9999-12-31 00:00:00'" and returns zero.

Returns:

An int which must be stored and used in decodeDate() on retrieval from the database.

void SAF::SAFDate::decodeSQL (int *codeVal*)

Description:

Used to apply the int returned by codeSQL to amend the date after restoration from a database.

Parameters:

codeVal The associated return from codeSQL.

char* SAF::SAFDate::iso (char * *buf*, bool *localtime* = `true`) const

Description:

Simple way to get an ISO printable version of the date. The format will be one of ”YYYY-MM-DD” (date but no time), ”HH:MM:SS.sss” (time but no date) or ”YYYY-MM-DD HH:MM:SS.sss”. The ”.sss” will not be present if the number of milliseconfs is zero.

Parameters:

buf Address of the output string, it should be at least SAFDATEPRINTSIZE bytes.

localtime Determines if the string is output in localtime (if true) or utc (if false).

Returns:

buf.

void SAF::SAFDate::dump (char * *buf*) const

Description:

Simple way to get a printable version of the date in localtime. Either ”dd/mm/yyyy hh:mm:ss.mmmm” or ”hh:mm:ss.mmmm” is put into the supplied buffer which must be at least of size SAFDATEPRINTSIZE. This is for diagnostic purposes only and is not internationalized in any way.

bool SAF::SAFDate::parseDate (const char * *dateString*, saf64_t & *date*) `[static]`

Description:

Parses the supplied localtime date string into the supplied saf64_t. The date string is assumed to have either come from a call to the dump() method or to be a string in one of the following formats:

- dd/mm/yyyy hh:mm:ss.msec
- dd/mm/yyyy hh:mm:ss
- dd/mm/yyyy
- yyyy-mm-dd hh:mm:ss.msec
- yyyy-mm-dd hh:mm:ss
- yyyy-mm-dd
- hh:mm:ss.msec
- hh:mm:ss

The formats are parsed in the above order. The values supplied are not validated to be within suitable ranges, merely that the number and position of the digits is correct.

Parameters:

dateString A string containing the date in a suitable format.

date A saf64_t which can be converted to a SAFDate via the SAFDate(saf64_t, bool) constructor. If the parse fails this value is not modified.

Returns:

True if the date parses and the `date` parameter is set, false otherwise.

bool SAF::SAFDate::parseDate (const char * *dateString*)

Description:

Parses the supplied localtime date to set the date. The date string is assumed to have either come from a call to the dump() method or to be a string in one of the following formats:

- dd/mm/yyyy hh:mm:ss.msec
- dd/mm/yyyy hh:mm:ss
- dd/mm/yyyy
- yyyy-mm-dd hh:mm:ss.msec
- yyyy-mm-dd hh:mm:ss
- yyyy-mm-dd
- hh:mm:ss.msec
- hh:mm:ss

The formats are parsed in the above order. The values supplied are not validated to be within suitable ranges, merely that the number and position of the digits is correct.

Parameters:

dateString A string containing the date in a suitable format.

Returns:

True if the date parses and the date is set, false otherwise.

SAFDate* SAF::SAFDate::create () `[static]`

Returns:

A new instance of a SAFDate.

void SAF::SAFDate::destroy () const

Description:

Delete this object.

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

- isafvalue.h

B.35 SAF::SAFStatusName Class Reference

`#include <isafvalue.h>`

B.35.1 Detailed Description

Class to map SAF::SAFStatus_t to printable strings.

If an instance of this class is constructed it holds a string representation of the status supplied to its constructor. Alternatively, that string can be accessed via the getStatusName() static method.

Definition at line 596 of file isafvalue.h.

Public Member Functions

- SAFStatusName (SAFStatus_t i)
- const char * set (SAFStatus_t i)
- operator const char * () const

Static Public Member Functions

- const char * getStatusName (SAF::SAFStatus_t i)
- int noOfNames ()

B.35.2 Constructor & Destructor Documentation

SAF::SAFStatusName::SAFStatusName (SAFStatus_t *i*) [inline]

Parameters:

i Status whose name is required.

Definition at line 607 of file isafvalue.h.

B.35.3 Member Function Documentation

const char* SAF::SAFStatusName::set (SAFStatus_t *i*) [inline]

Description:

Reset this name to that of the status supplied.

Parameters:

i Status whose name is required.

Returns:

A pointer to a string representation of the supplied status.

Definition at line 616 of file isafvalue.h.

const char* SAF::SAFStatusName::getStatusName (SAF::SAFStatus_t *i*) [inline, static]

Returns:

A pointer to a string representation of the supplied status.

Definition at line 623 of file isafvalue.h.

int SAF::SAFStatusName::noOfNames () [inline, static]

Returns:

The number of SAF::SAFStatus_t values for which there are names.

Definition at line 632 of file isafvalue.h.

References SAF::SAFStatus_t.

SAF::SAFStatusName::operator const char * () const [inline]

Description:

Operator to allow casting instances of SAFStatusName to strings.

Definition at line 641 of file isafvalue.h.

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

- isafvalue.h

B.36 SAF::SAFTypeName Class Reference

`#include <isafvalue.h>`

B.36.1 Detailed Description

Class to map SAF::SAFValueType_t to printable strings.

If an instance of this class is constructed it holds a string representation of the status supplied to its constructor. Alternatively, that string can be accessed via the getStatusName() static method.

Definition at line 482 of file isafvalue.h.

Public Member Functions

- SAFTypeName (SAFValueType_t i)
- const char * set (SAFValueType_t i)
- operator const char * () const
- const char * getName () const

Static Public Member Functions

- const char * getTypeName (SAFValueType_t i)
- int noOfNames ()

B.36.2 Constructor & Destructor Documentation

SAF::SAFTypeName::SAFTypeName (SAFValueType_t *i*) [inline]

Parameters:

i Status whose name is required.

Definition at line 492 of file isafvalue.h.

B.36.3 Member Function Documentation

const char* SAF::SAFTypeName::set (SAFValueType_t *i*) [inline]

Description:

Reset this name to that of the type supplied.

Parameters:

i Type whose name is required.

Returns:

A pointer to a string representation of the supplied type.

Definition at line 501 of file isafvalue.h.

const char* SAF::SAFTypeName::getTypeName (SAFValueType_t *i*) [inline, static]

Returns:

A pointer to a string representation of the supplied type.

Definition at line 508 of file isafvalue.h.

int SAF::SAFTypeName::noOfNames () [inline, static]

Returns:

The number of SAF::SAFType_t values for which there are names.

Definition at line 516 of file isafvalue.h.

References SAF::SAFValueType_t.

SAF::SAFTypeName::operator const char * () const [inline]

Description:

Operator to allow casting instances of SAFTypeName to strings.

Definition at line 525 of file isafvalue.h.

const char* SAF::SAFTypeName::getName () const [inline]

Returns:

A pointer to a string representation of the type used to construct this instance of SAFTypeName.

Definition at line 531 of file isafvalue.h.

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

- isafvalue.h

B.37 SAF::SAFValueDesc Class Reference

`#include <isafvalue.h>`

B.37.1 Detailed Description

Class to assist conversion of a SAFValue to a printable string describing its type, status and current value.

Definition at line 1694 of file isafvalue.h.

Public Member Functions

- SAFValueDesc (const ISAFValue *v, bool full=true)
- SAFValueDesc (const ISAFValue &v, bool full=true)
- ~SAFValueDesc ()
- operator const char * () const
- const char * getValueDesc () const

B.37.2 Constructor & Destructor Documentation

SAF::SAFValueDesc::SAFValueDesc (const ISAFValue * *v*, bool *full* = `true`) `[inline]`

Parameters:

v A const pointer to an SAF::ISAFValue.

full If true (the default) then the whole value of any string or array array type are dumped, otherwise a short version is produced which will fit into about 120 bytes.

Definition at line 1709 of file isafvalue.h.

SAF::SAFValueDesc::SAFValueDesc (const ISAFValue & *v*, bool *full* = `true`) `[inline]`

Parameters:

v A const reference to an SAF::ISAFValue.

full If true (the default) then the whole value of any string or array array type are dumped, otherwise a short version is produced which will fit into about 120 bytes.

Definition at line 1718 of file isafvalue.h.

SAF::SAFValueDesc::~SAFValueDesc ()

Description:

Delete this object.

B.37.3 Member Function Documentation

SAF::SAFValueDesc::operator const char * () const [inline]

Description:

Cast operator for access to the string value.

Definition at line 1730 of file isafvalue.h.

const char* SAF::SAFValueDesc::getValueDesc () const [inline]

Returns:

The string value.

Definition at line 1735 of file isafvalue.h.

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

- isafvalue.h

B.38 SAF::SAFValueString Class Reference

`#include <isafvalue.h>`

B.38.1 Detailed Description

Class to assist conversion of a SAFValue to a printable string.

Definition at line 1650 of file isafvalue.h.

Public Member Functions

- SAFValueString (const ISAFValue *v)
- SAFValueString (const ISAFValue &v)
- ~SAFValueString ()
- operator const char * () const
- const char * getValueString () const

B.38.2 Constructor & Destructor Documentation

SAF::SAFValueString::SAFValueString (const ISAFValue * *v*) [inline]

Parameters:

v A const pointer to an SAF::ISAFValue. If NULL is supplied then the string will be empty, not NULL.

Definition at line 1663 of file isafvalue.h.

SAF::SAFValueString::SAFValueString (const ISAFValue & *v*) [inline]

Parameters:

v A const reference to an SAF::ISAFValue.

Definition at line 1670 of file isafvalue.h.

SAF::SAFValueString::~SAFValueString ()

Description:

Delete this object.

B.38.3 Member Function Documentation

SAF::SAFValueString::operator const char * () const [inline]

Description:

Cast operator for access to the string value.

Definition at line 1682 of file isafvalue.h.

const char* SAF::SAFValueString::getValueString () const [inline]

Returns:

The string value.

Definition at line 1687 of file isafvalue.h.

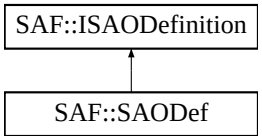
The documentation for this class was generated from the following file:

- isafvalue.h

B.39 SAF::SAODef Class Reference

`#include <safhelper.h>`

Inheritance diagram for SAF::SAODef::



B.39.1 Detailed Description

An implementation of ISAODefinition which provides a description of an SAO.

Definition at line 148 of file safhelper.h.

Public Member Functions

- SAODef (const char *name, const char *description, SAFValueType_t type, const char *category, int maxInputs, int validatedInputs, const ISAOValidation *const *validation)

  Constructor - just take all the supplied parameters and save them.

- virtual ~SAODef ()

  Destructor - nothing special to do.

- virtual void destroy () const

  Called when this object is finished with.

- virtual const char * getName () const

  Return the name supplied to the constructor.

- virtual const char * getDescription () const

  Return the description supplied to the constructor.

- virtual const char * getCategory () const

  Return the category supplied to the constructor.

- virtual SAFValueType_t getValueType () const

  Return the type supplied to the constructor.

- virtual int getMaxInputs () const

Return the maximum input number supplied to the constructor.

- virtual int getValidatedInputs () const

  Return the number of inputs to be validated as supplied to the constructor.

- virtual const ISAOValidation *const * getValidation () const

  Return the input validation rules supplied to the constructor.

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

- safhelper.h

B.40 SAF::SAOFType Class Reference

`#include <isafvalue.h>`

B.40.1 Detailed Description

SAOFtype is a simple class combining a SAF::libraryId and SAF::typeId to make up the unique functional type identifier of an SAO.

Note The overrides of operator new() and operator delete() are to prevent attempts to allocate in one heap and free in another. Use create() and destroy() to get a dynamically allocated instance of this class.

Definition at line 1083 of file isafvalue.h.

Public Member Functions

- SAOFType ()
- SAOFType (libraryId lid, typeId tid)
- void setId (libraryId lid, typeId tid)
- libraryId getLId () const
- typeId getTId () const
- void setLId (libraryId lid)
- void setTId (typeId tid)
- bool operator< (const SAOFType &other) const
- void destroy () const

Static Public Member Functions

- SAOFType * create ()

Protected Member Functions

- void * **operator new** (size_t size)
- void **operator delete** (void *mem)

B.40.2 Constructor & Destructor Documentation

SAF::SAOFType::SAOFType () `[inline]`

Description:

Default constructor (inserts a zero in all fields).

Definition at line 1088 of file isafvalue.h.

SAF::SAOFType::SAOFType (libraryId *lid*, typeId *tid*) [inline]

Description:

Construct the Id from the supplied libraryId and typeId values.

Parameters:

lid LibraryId.

tid TypeId.

Definition at line 1095 of file isafvalue.h.

B.40.3 Member Function Documentation

void SAF::SAOFType::setId (libraryId *lid*, typeId *tid*) [inline]

Description:

Set the Id useing the supplied libraryId and typeId.

Definition at line 1103 of file isafvalue.h.

libraryId SAF::SAOFType::getLId () const [inline]

Returns:

The libraryId component of the SAO functional type identifier.

Definition at line 1112 of file isafvalue.h.

References SAF::libraryId.

typeId SAF::SAOFType::getTId () const [inline]

Returns:

The typeId component of the SAO functional type identifier.

Definition at line 1119 of file isafvalue.h.

References SAF::typeId.

void SAF::SAOFType::setLId (libraryId *lid*) [inline]

Description:

Sets the libraryId component of the SAO functional type identifier.

Parameters:

lid LibraryId.

Definition at line 1127 of file isafvalue.h.

void SAF::SAOFType::setTId (typeId *tid*) [inline]

Description:

Sets the typeId component of the SAO functional type identifier.

Parameters:

tid TypeId.

Definition at line 1135 of file isafvalue.h.

bool SAF::SAOFType::operator< (const SAOFType & *other*) const

Description:

Operator to allow comparision of two SAO functional type identifiers (allows instances of this class to be used in conjunction with STL containers).

SAOFType* SAF::SAOFType::create () [static]

Returns:

An instance of an SAO functional type identifier.

void SAF::SAOFType::destroy () const

Description:

Delete this object.

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

- isafvalue.h

B.41 SAF::SAOId Class Reference

`#include <isafvalue.h>`

B.41.1 Detailed Description

SAOId is a simple class combining the objectId of an SAO with the containerId of the container which hosts it to provide a unique identifier for the SAO within the application.

Since the containerId requires 16 bits and the objectId uses 48 bits, the unique SAOId of any SAO can be represented as a single 64 bit value.

Definition at line 1189 of file isafvalue.h.

Public Member Functions

- SAOId ()
- SAOId (containerId cid, objectId oid)
- SAOId (saf64_t id)
- operator saf64_t () const
- SAOId & setId (saf64_t id)
- SAOId & setId (containerId cid, objectId oid)
- containerId getCId () const
- objectId getOId () const
- void setCId (containerId id)
- void setOId (objectId id)

  Set the objectId.

- bool operator< (const SAOId &other) const
- bool isEmpty () const
- bool isPopulated () const
- char * toString (char *target) const
- const char * fromString (const char *buf)
- bool setId (const char *s)
- void destroy () const

Static Public Member Functions

- SAOId * create ()

Protected Member Functions

- void * **operator new** (size_t size)
- void **operator delete** (void *mem)

B.41.2 Constructor & Destructor Documentation

SAF::SAOId::SAOId () [inline]

Description:

Default constructor - inserts a zero in all fields.

Definition at line 1194 of file isafvalue.h.

SAF::SAOId::SAOId (containerId *cid*, objectId *oid*) [inline]

Description:

Constructs the SAOId from the supplied containerId and objectId.

Parameters:

cid A containerId.

oid An objectId.

Definition at line 1201 of file isafvalue.h.

SAF::SAOId::SAOId (saf64_t *id*) [inline]

Description:

Construct the SAOId from a 64 bit value.

Parameters:

id An id for an SAO supplied as a 64 bit value.

Definition at line 1209 of file isafvalue.h.

B.41.3 Member Function Documentation

SAF::SAOId::operator saf64_t () const [inline]

Description:

Operator to allow type conversion to a 64 bit value.

Definition at line 1218 of file isafvalue.h.

SAOId& SAF::SAOId::setId (saf64_t *id*) [inline]

Description:

Assign the SAOId from a supplied 64 bit value.

Parameters:

id An id for an SAO supplied as a 64 bit value.

Returns:

A reference to the SAOId.

Definition at line 1227 of file isafvalue.h.

SAOId& SAF::SAOId::setId (containerId *cid*, objectId *oid*) [inline]

Description:

Assign the SAOId from a supplied containerId and objectId.

Parameters:

cid A containerId.

oid An objectId.

Returns:

A reference to the SAOId.

Definition at line 1238 of file isafvalue.h.

containerId SAF::SAOId::getCId () const [inline]

Returns:

The containerId component of this SAOId.

Definition at line 1247 of file isafvalue.h.

References SAF::containerId.

objectId SAF::SAOId::getOId () const [inline]

Returns:

The objectId component of this SAOId.

Definition at line 1254 of file isafvalue.h.

References SAF::objectId.

void SAF::SAOId::setCId (containerId *id*) [inline]

Description:

Set the containerId.

Parameters:

cid A containerId.

Definition at line 1262 of file isafvalue.h.

void SAF::SAOId::setOId (objectId *id*) [inline]

Set the objectId.

Parameters:

oid An objectId.

Definition at line 1270 of file isafvalue.h.

bool SAF::SAOId::operator< (const SAOId & *other*) const [inline]

Description:

Operator to allow comparision of two SAOIds (allows instances of this class to be used in conjunction with STL containers).

Definition at line 1278 of file isafvalue.h.

References _id.

bool SAF::SAOId::isEmpty () const [inline]

Returns:

True if both the containerId and objectId have NOT been set, false otherwise.

Definition at line 1286 of file isafvalue.h.

bool SAF::SAOId::isPopulated () const [inline]

Returns:

True if either the containerId or objectId are set, false otherwise.

Definition at line 1294 of file isafvalue.h.

char* SAF::SAOId::toString (char * *target*) const

Description:

Convert an SAOId to a string in the form ”(CId,OId)”.

Parameters:

target A buffer of at least SAOIDSTRINGLEN bytes.

Returns:

The target string supplied.

const char* SAF::SAOId::fromString (const char * *buf*)

Description:

Set the SAOId by reading the supplied string in the form ”(CId,OId)”.

Parameters:

buf Assumed to be a string produced by toString().

Returns:

NULL on failure (string not in correct format), otherwise the address of the character after the closing ')'.

bool SAF::SAOId::setId (const char * s)

Description:

Set the SAOId from a value derived from toString().

Parameters:

s A string in the form CId,OId.

Returns:

True on success, false if the separating comma is not present (in which case the value is left unchanged).

SAOId* SAF::SAOId::create () [static]

Returns:

An instance of this class.

void SAF::SAOId::destroy () const

Description:

Delete this instance.

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

- isafvalue.h

B.42 SAF::SAOIdString Class Reference

`#include <isafvalue.h>`

B.42.1 Detailed Description

Class to encapsulate a string representation of an SAOId (for debug, logging etc).

Definition at line 1361 of file isafvalue.h.

Public Member Functions

- SAOIdString (const SAOId n)

  Constructor.

- ~SAOIdString ()

  Destructor.

- operator const char * () const

  Cast operator to retrieve string.

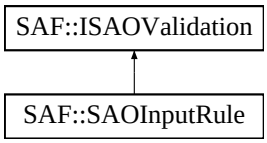
The documentation for this class was generated from the following file:

- isafvalue.h

B.43 SAF::SAOInputRule Class Reference

`#include <safhelper.h>`

Inheritance diagram for SAF::SAOInputRule::



B.43.1 Detailed Description

A basic implementation of ISAOValidation for creating input rules for SAOs.

Definition at line 226 of file safhelper.h.

Public Member Functions

- SAOInputRule (SAFValueType_t type, SAOInputValidation_t validation, const char *description)

  Constructor - just take all the supplied parameters and save them.

- void destroy () const

  Called when this object is finished with.

- SAFValueType_t getValueType () const

  Return the type supplied to the constructor.

- SAOInputValidation_t getValidationCode () const

  Return the type of the input validation rule (MANDATORY or OPTIONAL).

- const char * getDescription () const

  Return a short description of the validation rule.

Public Attributes

- const char * _description

  The description of this input rule.

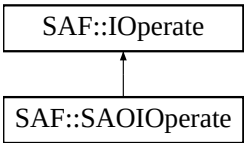
The documentation for this class was generated from the following file:

- safhelper.h

B.44 SAF::SAOIOperate Class Reference

`#include <safhelper.h>`

Inheritance diagram for SAF::SAOIOperate::



B.44.1 Detailed Description

A class to ease the provision of IOperate by providing a default implementation of everything.

Definition at line 311 of file safhelper.h.

Public Member Functions

- virtual void initialise (ISAO &base)

  No initialisation to do.

- virtual void destroy (ISAO &base)

  Delete ourselves.

- virtual bool update (ISAO &base, const IESRUpdate &update, bool &handled, bool last)

  If we are defined to have inputs ignore updates otherwise accept any last value and let the container set our value.

- virtual bool rename (const ISAO &base, const char *name)

  We allow renames.

- virtual bool operate (ISAO &base, int count, ISAO **changed)

  Don't do anything if our inputs change.

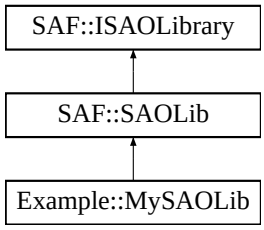
The documentation for this class was generated from the following file:

- safhelper.h

B.45 SAF::SAOLib Class Reference

`#include <safhelper.h>`

Inheritance diagram for SAF::SAOLib::



B.45.1 Detailed Description

A basic partial implementation of ISAOLibrary which can serve as a base for user defined SAO libraries.

Definition at line 268 of file safhelper.h.

Public Member Functions

- SAOLib ()

  Constructor.

- virtual ~SAOLib ()

  Destructor.

- virtual bool load (libraryId id, const char *const *config)

  Record the id supplied to us.

- virtual void unload ()

  No additional tidying up to do here.

- virtual void getSAFVersion (int &major, int &minor) const

  Supply version information for the version of SAF against which this library was compiled (values come from safuser.h).

- virtual libraryId getId () const

  Return our id when asked.

- virtual ICommand * createCommand (ISAO &base, typeId function)

  Most SAO libraries don't contain command SAOs, so a default implementation of createCommand is supplied which does nothing.

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

- safhelper.h

Appendix C File Documentation

C.1 isaflibrary.h File Reference

C.1.1 Detailed Description

SAO library interface.

This header contains definitions of interfaces which must be implemented in order to extend **SONARIS/Framework** through the addition of custom SAOs.

Definition in file isaflibrary.h.

Namespaces

- namespace SAF

Classes

- class SAF::ISAOValidation

  Interface used to obtain details of the validation of a single input to the operate method.

- class SAF::ISAODefinition

  Interface which supplies all the details needed to understand how an SAO provided by a library will operate at run time.

- class SAF::IOperate

  Interface to be implemented by the user part of an SAO in order to do its work when invoked as part of a ripple.

- class SAF::ISAOLibrary

  Interface implemented by SAO library providers to allow the Framework to determine the nature of the objects that this library can create.

Defines

- #define SAF_UNLIMITED_INPUTS 0x7FFFFFFF

  Definition of the maximum number of inputs possible for an SAO to be used when an effectively unlimited number of inputs is desired.

Enumerations

- enum SAOInputValidation_t { SAF_MANDATORY = 1, SAF_OPTIONAL = 2, SAF_RESERVED = 3 }

  An enumeration detailing the possible criteria for deciding whether a single input to an SAO is acceptable to it such that a call to that SAO's IOperate::operate() method will be made.

C.1.2 Enumeration Type Documentation

enum SAOInputValidation_t

An enumeration detailing the possible criteria for deciding whether a single input to an SAO is acceptable to it such that a call to that SAO's IOperate::operate() method will be made.

The method will not be called if the input does not comply.

Enumeration values:

SAF_MANDATORY Any MANDATORY input must be connected and its status must be SAF::SAF_VALID or SAF::SAF_STALE.

before a call will be made on IOperate::operate().

SAF_OPTIONAL An OPTIONAL input may be absent, but if it is connected then its status is taken into account when.

SONARIS/Framework automatically determines the status of the output value (remembering that the IOperate::operate() method of an SAO is at liberty to change any automatically assigned status.

SAF_RESERVED A RESERVED input is one set up at run time by the SAO, that is not persisted in the configuration database, and that can only be disconnected by the SAO itself or through deletion of the source.

The input is treated as OPTIONAL when deciding whether to call IOperate::operate().

Definition at line 176 of file isaflibrary.h.

Referenced by SAF::SAOInputRule::getValidationCode().

C.2 isafvalue.h File Reference

C.2.1 Detailed Description

Fundamental data objects.

This header contains definitions of fundamental objects which identify and encapsulate data within **SONARIS/Framework**, along with enumerations for the type and status of such data and helper classes to make working with them easier.

Definition in file isafvalue.h.

```
#include <sys/types.h>
#include <limits.h>
#include <time.h>
```

Namespaces

- namespace SAF
- namespace **SAF::XML**

Classes

- class SAF::SAFTypeName

  Class to map SAF::SAFValueType_t to printable strings.

- class SAF::SAFStatusName

  Class to map SAF::SAFStatus_t to printable strings.

- class SAF::SAFDate

  A small class (taking up only 8 bytes convertible to a saf64_t) which allows either a date, a time of day or a single point in time to be represented to a resolution of milliseconds.

- class SAF::SAOFType

  SAOFtype is a simple class combining a SAF::libraryId and SAF::typeId to make up the unique functional type identifier of an SAO.

- class SAF::SAOId

  SAOId is a simple class combining the objectId of an SAO with the containerId of the container which hosts it to provide a unique identifier for the SAO within the application.

- class SAF::SAOIdString

  Class to encapsulate a string representation of an SAOId (for debug, logging etc).

- class SAF::ISAFValue

  Interface to read and write SAF's basic item of data.

- class SAF::SAFValueString

  Class to assist conversion of a SAFValue to a printable string.

- class SAF::SAFValueDesc

  Class to assist conversion of a SAFValue to a printable string describing its type, status and current value.

- class SAF::IProperties

  Implementations of this interface handle an array of name value pairs where each value is a ISAFValue.

- class SAF::IESRUpdate

  An IESRUpdate is a data structure allowing chains of addressed values and properties to be built and delivered.

- class SAF::AutoDestroy< T >

  Template class removing the need to remember to destroy() one of the objects which are created via create() e.g.

- class SAF::AutoFree< T >

  Template class (similar to auto_ptr) removing the need to remember to free malloc'd memory for POD (plain old data) types.

Defines

- #define **SAF64MASK**(h) h ## LL
- #define **SAF64_MAX** LONG_LONG_MAX;
- #define **PRINTFSAF64** "%lld"
- #define **PRINTFSAF64X** "%#llx"
- #define **ATOSAF64** atoll
- #define **SAFDATESQLSTRING** 24
- #define SAFDATEPRINTSIZE 32

  Minimum size of charater buffer which should be allocated in order to hold a string representation of a SAF::SAFDate populated during a call to the SAF::SAFDate::dump() method.

- #define SAF_NOCONTAINER 0xFFFF

  Flag for an unused container number.

- #define SAF_DUMMYCONTAINER 0xFFFE

  Container hosting dummy SAOs.

- #define SAFBASECONTAINER 0

  Container which is SAFBase.

- #define SAOIDSTRINGLEN 30

  Minimum size of charater buffer which should be allocated in order to hold a string representation of an SAF::SAOId populated during a call to the SAF::SAOId::toString() method.

- #define PRINTSAOID "(%d," PRINTFSAF64 ")"

  Format which can be used in conjunction with printf style functions to print an SAOId provided the CId and OId values are supplied.

- #define **getPropertyNameNumber** getPropertyNameByNumber

Typedefs

- typedef int64_t **saf64_t**
- typedef saf64_t **saftime_t**
- typedef unsigned short libraryId
- typedef unsigned short typeId
- typedef unsigned short containerId
- typedef saf64_t objectId

Enumerations

- enum SAFValueType_t {

  SAF_UNDEFINED = 0, SAF_VOID = 1, SAF_INT = 2, SAF_INT64 = 3,

  SAF_DOUBLE = 4, SAF_BOOL = 5, SAF_DATE = 6, SAF_STRING = 7,

  SAF_BYTE_ARRAY = 8, SAF_INT_ARRAY = 9, SAF_INT64_ARRAY = 10, SAF_DOUBLE_ARRAY = 11,

  SAF_STRING_ARRAY = 12, SAF_DATE_ARRAY = 13 }

- enum SAFStatus_t {

  SAF_VALID = 0, SAF_STALE = 1, SAF_INVALID = 7, SAF_UNINITIALIZED = 2,

  SAF_BADINPUT = 3, SAF_SUSPENDED = 6, SAF_NOLIBRARY = 10, SAF_NODEFINITION = 11,

  SAF_REMOTEDISCONNECT = 4, SAF_LOCALDISCONNECT = 5, SAF_DELETED = 8, SAF_NOSAO = 9,

  SAF_NEWPROXY = 12, SAF_ACCESS_DENIED = 15, SAF_CANCELLED = 13, SAF_DROPPED = 14,

  SAF_FAILOVER = 16, SAF_COMPLETED = 17 }

- enum ModifyTreeControl_t {

  MTC_TRANSIENT = 0x0001, MTC_CHILDREN = 0x0002, MTC_MERGE = 0x0004, MTC_HISTORY = 0x0008,

  MTC_LOGGING = 0x0010, MTC_MELDABLE = 0x0020 }

C.2.2 Typedef Documentation

typedef unsigned short SAF::libraryId

Description:

Index number of an SAO library (each library is assigned a unique number when its defintion is loaded for use within an application).

Definition at line 1069 of file isafvalue.h.

Referenced by SAF::SAOLib::getId(), and SAF::SAOFType::getLId().

typedef unsigned short SAF::typeId

Description:

Index number of an individual SAO type within an SAO library.

Definition at line 1073 of file isafvalue.h.

Referenced by SAF::SAOFType::getTId().

typedef unsigned short SAF::containerId

Description:

A number which uniquely identifies a container within an application.

Definition at line 1171 of file isafvalue.h.

Referenced by SAF::SAOId::getCId().

typedef saf64_t SAF::objectId

Description:

A number which uniquely identifies an SAO within a container. **Note** Although defined as a 64 bit value, only the bottom 48 bits are used.

Definition at line 1176 of file isafvalue.h.

Referenced by SAF::SAOId::getOId().

C.2.3 Enumeration Type Documentation

enum SAFValueType_t

Description:

Enumeration which defines the type of data that an individual SAFValue and, therefore, SAO may hold.

Enumeration values:

SAF_UNDEFINED Default type, can be changed to any other.

SAF_VOID No value.

SAF_INT A 32 bit integer.

SAF_INT64 A 64 bit integer.

SAF_DOUBLE Extended precision (64 bit) floating point number.

SAF_BOOL A C++ bool.

SAF_DATE A 64 bit value containing a set of bit fields.

SAF_STRING A zero terminated byte array.

SAF_BYTE_ARRAY An array of bytes.

SAF_INT_ARRAY An array of SAF_INT.

SAF_INT64_ARRAY An array of SAF_INT64.

SAF_DOUBLE_ARRAY An array of SAF_DOUBLE.

SAF_STRING_ARRAY An array of SAF_STRING.

SAF_DATE_ARRAY An array of SAF_DATE.

Definition at line 458 of file isafvalue.h.

Referenced by SAF::SAOInputRule::getValueType(), SAF::SAODef::getValueType(), and SAF::SAFTypeName::noOfNames().

enum SAFStatus_t

Description:

Different statuses which describe the condition of an item of data (an SAO or SAFValue) or communication session within **SONARIS/Framework**. A useable value is only available from a SAFValue or SAO with a status of SAF_VALID or SAF_STALE. All other statuses imply no usable data is available from the SAO or SAFValue and essentially are all equivalent to SAF_INVALID. Many of these apply only to SAOs or to communication sessions across the ISAFClient or IExternalSystem interfaces. Refer to those interfaces for more detail regarding the precise meaning of such statuses.

Enumeration values:

SAF_VALID A good value.

SAF_STALE value is stale (usable, but possibly out of date) or derived from stale data

SAF_INVALID The data is invalid.

SAF_UNINITIALIZED Value has never been set. Initial state.

SAF_BADINPUT The data held by an SAO is out of date as it has lost a mandatory input.

SAF_SUSPENDED The data held by an SAO is not updating due to an exception caught within that SAO's code.

SAF_NOLIBRARY The SAO is not complete because of missing functionality or missing library.

SAF_NODEFINITION The SAO is a dummy placeholder because its definition could not be found.

SAF_REMOTEDISCONNECT Failure detected in remote machine.

SAF_LOCALDISCONNECT Failure detected in this machine.

SAF_DELETED The associated SAO has been deleted.

SAF_NOSAO The SAO requested does not exist.

SAF_NEWPROXY Indicates that a proxy SAO has not had an initial response.

SAF_ACCESS_DENIED Indicates that a proxy SAO has had a response but that the client is not allowed to read the data in it.

SAF_CANCELLED Confirms cancellation of requests across the IExternalSystem interface.

SAF_DROPPED Indicates unexpected and unrecoverable termination of a request across the IExternalSystem interface.

SAF_FAILOVER Indicates unexpected termination of a request across the IExternalSystem interface and that recovery is being attempted.

SAF_COMPLETED Indicates expected termination of a request across the IExternalSystem interface at such times as the request comes to its natural end.

Definition at line 544 of file isafvalue.h.

Referenced by SAF::SAFStatusName::noOfNames().

enum ModifyTreeControl_t

Description:

Values used as bit flags in combinations when calling ISAFClient::modifyTree().

Enumeration values:

MTC_TRANSIENT If set then all SAOs which have to be created are made transient. Without this then all SAOs in the destination will end up matching (in terms of persistence) the corresponding SAO in the source.

MTC_CHILDREN If set then the destination SAO has the tranformation applied to all of its children.

MTC_MERGE If set then any extra SAOs already in the destination tree are left in place (although moved to the end of the child list).

MTC_HISTORY If set then SAOS already in the destination will have their history setting (getHistoryLevel()) amended to match that of the source.

MTC_LOGGING If set then SAOS already in the destination will have their logging setting (getLoggingLevel()) amended to match that of the source.

MTC_MELDABLE If set then SAOS already in the destination will have their meldability setting (isMeldable) amended to match that of the source.

Definition at line 1909 of file isafvalue.h.

C.3 isao.h File Reference

C.3.1 Detailed Description

SONARIS Application Object interface definitions.

This header contains definitions of the interface by which SAOs may be manipulated.

Definition in file isao.h.

```
#include <isafvalue.h>
#include <isaospec.h>
#include <ipermissions.h>
```

Namespaces

- namespace SAF

Classes

- class SAF::IEventListener

  This interface is used to notify an SAO of changes to other SAOs.

- class SAF::ISAORead

  Interface to get the data contained within an SAO.

- class SAF::ISAOWrite

  Interface to set the data contained within an SAO.

- class SAF::ISAODependencies

  Interface describing the dynamic properties of the dependency relationship between SAOs.

- class SAF::ISAOSubInfo

  Interface used by the ClientAPI to hold details about a single other SAO, whether an input, output, virtual input or virtual output.

- class SAF::ISAOInfo

  Interface used by the ClientAPI to hold details about an SAO.

- class SAF::ISAOAdmin

  Interface to get hierarchy and database related information.

- class SAF::ISAOProxy

  Interface which is only realy relevant to Client code and Proxy SAOs.

- class SAF::ISAO

  The real working SAO.

Defines

- #define SAOTOSTRINGSIZE 64

  A basic string representation of an SAO.

Typedefs

- typedef enum SAF::SAOHist SAOHist_t

  Values to control whether on not the value of an SAO is persisted in the configuration database or not.

- typedef enum SAF::SAOLogging SAOLogging_t

  Logging can be controlled on a per SAO basis.

Enumerations

- enum SAOHist { **SAO_HIST_OFF** = 0, **SAO_HIST_ON** = 1 }

  Values to control whether on not the value of an SAO is persisted in the configuration database or not.

- enum SAOLogging {

  SAO_LOGGING_OFF = 0, **SAO_LOGGING_INPUT** = 1,
  SAO_LOGGING_OUTPUT = 2, **SAO_LOGGING_RIPPLE** = 4,

  SAO_LOGGING_ONESHOT = 8, **SAO_LOGGING_FULLDUMP** = 0x10,
  SAO_LOGGING_USER = 0x20, **SAO_LOGGING_IMMUNE** = 0x40,

  SAO_LOGGING_NOMELD = 0x80, **SAO_LOGGING_USERMASK** = 0x3F,
  SAO_LOGGING_RESET = 0xBF, **SAO_LOGGING_KEEP** =
  ∼SAO_LOGGING_USERMASK,

  SAO_LOGGING_DATABASE }

  Logging can be controlled on a per SAO basis.

C.3.2 Define Documentation

#define SAOTOSTRINGSIZE 64

A basic string representation of an SAO.

Currently formats a string as <name>(<container Id>:<object Id>) of course this involves an extra string copy (or two), but is neater in the code and gives a consistent format. The name of the SAO is truncated to 16 chars

Parameters:

s the string to format into. This must be long enough to hold the full formatted string and a terminating null. The current recommendation is at least 64 characters (use SAOTOSTRINGSIZE).

Returns:

The string representation (in the string you supplied)

Definition at line 1181 of file isao.h.

C.3.3 Typedef Documentation

typedef enum SAF::SAOLogging SAF::SAOLogging_t

Logging can be controlled on a per SAO basis.

The default is for an SAO to have no logging enabled. The values in SAOLogging are bit flags and can be used in combination when calling setLogging. Logging is switched off by setting to zero (LOGGING_OFF). The other bits have the following meaning:- INPUT - Dump the value of all the inputs for the SAO being traced prior to operate being called. Details of the parameters to operate are also given. OUTPUT - Dump the value of the SAO after operate has returned. RIPPLE - Treat all remaining SAOs involved in the current ripple as if they had at least the level of trace of the current one (applies only to the INPUT, OUTPUT and FULLDUMP bits). ONESHOT - do this once only i.e. on the first call to operate after being set. FULLDUMP - if set will dump the whole set of values for array inputs and outputs. It will also dump the whole of a string. Use this bit with care because it can produce large amounts of output. USER - should be tested by the implementors of IOperate and ICommand methods to determine if they should produce trace events via the ISOC::log∗Message() methods.

Note that if any SAO throws an exception inside operate then the following action is taken:- Retrospective dump of its inputs (if not already done). State set to SAF_SUSPENDED (from which it can never emerge and which means it will no longer take any part in ripples). Of course this assumes that the base SAO itself has not been too badly corrupted but generally this technique will work.

Logging applies equally to any uses of the ICommand interface, with obvious analogies - inputs is the supplied properties, output is the return value and contents of reply.

C.3.4 Enumeration Type Documentation

enum SAOLogging

Logging can be controlled on a per SAO basis.

The default is for an SAO to have no logging enabled. The values in SAOLogging are bit flags and can be used in combination when calling setLogging. Logging is switched off by setting to zero (LOGGING_OFF). The other bits have the following meaning:- INPUT - Dump the value of all the inputs for the SAO being traced prior to operate being called. Details of the parameters

to operate are also given. OUTPUT - Dump the value of the SAO after operate has returned. RIPPLE - Treat all remaining SAOs involved in the current ripple as if they had at least the level of trace of the current one (applies only to the INPUT, OUTPUT and FULLDUMP bits). ONESHOT - do this once only i.e. on the first call to operate after being set. FULLDUMP - if set will dump the whole set of values for array inputs and outputs. It will also dump the whole of a string. Use this bit with care because it can produce large amounts of output. USER - should be tested by the implementors of IOperate and ICommand methods to determine if they should produce trace events via the ISOC::log∗Message() methods.

Note that if any SAO throws an exception inside operate then the following action is taken:- Retrospective dump of its inputs (if not already done). State set to SAF_SUSPENDED (from which it can never emerge and which means it will no longer take any part in ripples). Of course this assumes that the base SAO itself has not been too badly corrupted but generally this technique will work.

Logging applies equally to any uses of the ICommand interface, with obvious analogies - inputs is the supplied properties, output is the return value and contents of reply.

Definition at line 333 of file isao.h.

C.4 isaoresource.h File Reference

C.4.1 Detailed Description

SAO resource interface.

This header contains definitions of interfaces which enable the safe encapsulation of complex data structures by SAOs.

Definition in file isaoresource.h.

Namespaces

- namespace SAF

Classes

- class SAF::ISAOResource

  Interface to be implemented by objects which hold multiple values to be shared by multiple SAOs.

- class SAF::ISAOResourceManager

  Objects which implement this interface arrange to store and retrieve objects which implement the ISAOResource interface.

C.5 isoc.h File Reference

C.5.1 Detailed Description

SAO container interface.

This header contains definitions of the interface to the SAO container through which SAOs and the connections between them may be managed.

Definition in file isoc.h.

```
#include <isaoresource.h>
```

Namespaces

- namespace SAF

Classes

- class SAF::ISOC

  This interface is visible to SAOs and defines the operations that are available to SAO 'operate' method code.

- class SAF::IBlackBox

  Implementations of the IBlackBox interface allow low level trace messages to be written to a rolling in memory log.

Appendix D Example SAO Catalogue

File:	`mysaolib`
Library Version:	1.0
Framework Version:	1.4
No. SAO types:	6

Description:

Some simple SAO examples.

D.1 Input Handling Examples

The Input Handling Examples category contains the following SAO:

- MyInputIndex

D.1.1 MyInputIndex

Functional Type:	5
Data Type:	SAF_STRING
Maximum Inputs:	SAF_UNLIMITED_INPUTS
Validated Inputs:	5
Input 0:	*Input A*
	Type: SAF_VOID
	Code: SAF_OPTIONAL
Input 1:	*Input B*
	Type: SAF_VOID
	Code: SAF_OPTIONAL
Input 2:	*Input C*
	Type: SAF_VOID
	Code: SAF_OPTIONAL
Input 3:	*Input D*
	Type: SAF_VOID
	Code: SAF_OPTIONAL
Input 4:	*Other Input*
	Type: SAF_VOID
	Code: SAF_OPTIONAL

Description:

Quickly establishes which input has changed. Shows the state of inputs and input changes by means of a string value. Changed inputs which are connected are shown with uppercase letters, those which are not connected are shown lowercase, and those which have not changed are shown using the underscore character.

D.2 Resource Examples

The Resource Examples category contains the following SAOs:

- MyWrapper
- MyReso

D.2.1 MyWrapper

Functional Type:	3
Data Type:	SAF_INT
Maximum Inputs:	2
Validated Inputs:	2
Input 0:	*First number*
	Type: SAF_DOUBLE
	Code: SAF_MANDATORY
Input 1:	*Second number*
	Type: SAF_DOUBLE
	Code: SAF_MANDATORY

Description:

Calculates the sum, difference and product of its inputs and makes them available as resources `Sum` , `Difference` and `Product` respectively. These can be accessed via instances of either the *Resources/double* SAO in the *Foundation Objects* SAO library, or the *MyReso* SAO in this library. Displays the number of calculated values which have changed (1, 2 or 3) as a result of input changes.

D.2.2 MyReso

Functional Type:	4
Data Type:	SAF_UNDEFINED
Maximum Inputs:	1
Validated Inputs:	1
Input 0:	*Resource owner*
	Type: SAF_VOID
	Code: SAF_MANDATORY

Description:

Adopts the value based on a resource available from the SAO connected where the resource name matches that of this SAO.

D.3 Simple Examples

The Simple Examples category contains the following SAOs:

• MyPlus • MySwitch • MyDelta

D.3.1 MyPlus

Functional Type:	0
Data Type:	SAF_DOUBLE
Maximum Inputs:	2
Validated Inputs:	2
Input 0:	*First number to add*
	Type: SAF_DOUBLE
	Code: SAF_MANDATORY
Input 1:	*Second number to add*
	Type: SAF_DOUBLE
	Code: SAF_MANDATORY

Description:

Adds two numbers together.

D.3.2 MySwitch

Functional Type:	1
Data Type:	SAF_DOUBLE
Maximum Inputs:	SAF_UNLIMITED_INPUTS
Validated Inputs:	2
Input 0:	*Selector in the range 1-n*
	Type: SAF_INT
	Code: SAF_MANDATORY
Input 1:	*Selectable data*
	Type: SAF_DOUBLE
	Code: SAF_OPTIONAL

Description:

Switches between numerical input values.

D.3.3 MyDelta

Functional Type:	2
Data Type:	SAF_DOUBLE
Maximum Inputs:	1
Validated Inputs:	1
Input 0:	*Difference from previous value*
	Type: SAF_DOUBLE
	Code: SAF_OPTIONAL

Description:

Calculates the difference between the current value of our input and its previous value.

Appendix E Compiling C++ Programs and Components

The creation of **SONARIS/Framework** client programs or custom SAOs is achieved using the various **SONARIS/Framework** interfaces. All the class definitions required for this are provided by the `saf.h` header file. This file acts as a top level header which itself includes the various other headers supplied by the SDK each of which relates to a specific area of **SONARIS/Framework** (e.g. client programming, external interfaces, SAO development etc.). Newly created user code should simply include `saf.h` and be linked against the `safsystem` runtime library appropriate to your operating system.

For convenience, implementations of some of the relatively simple and frequently used interfaces are supplied in the `safhelper.h` header file.

Windows

SONARIS/Framework clients and components may be created using Microsoft Visual Studio v6 or higher. All the programming examples supplied with **SONARIS/Framework** written in C++ can be compiled using the project files which accompany them. These project files are created with Visual Studio v6 but can be converted for use with later versions of Visual Studio.

The `safsystem.lib` stub library should be used to link newly created **SONARIS/Framework** client programs or SAO libraries.

Glossary

Active Interface
An interface to a *SONARIS/Framework* programming object for which a standard implementation is available. Instances of objects which provide this implementation are supplied by the *SONARIS/Framework Runtime Environment* and can be queried and manipulated via their interface definition. See also *Listener Interface*.

Application
In *SONARIS/Framework* terms, a complete solution composed of an *application service* which processes and publishes data using *SAOs*, and *client* programs capable of accessing the data held by those SAOs.

Application Service
In *SONARIS/Framework* terms, a group of *containers* hosted by *processes* using *SAOs* to implement a service capable of supplying the data held by those SAOs to *client* programs.

Child
In terms of *SAO hierarchies*, an *SAO* which is parented by another SAO within a hierarchy is said to be a *child* of that SAO.

Client
In *SONARIS/Framework* terms, a program capable of subscribing to the data held by *SAOs* which form part of an *application service*.

Configuration Database
A database (usually hosted by some RDBMS system) which acts as a store for configuration information that describes *hierarchies* of *SAOs*, the properties of those SAOs (value, history, log level etc.), the *connections* between them, and by which *container* they are hosted as part of an *application service* (refer to the *SONARIS/Framework - Release Notes* for details of the RDBMS systems supported).

Connection
A *dependency* relationship between two *SAOs* which specifies that the value of one of the SAOs is dependant on the value of the other, and that changes in the value of the former will be reflected in the value of the later, or that the later will take some action based on the change of the former.

Container
An object created by a `saf` *process* responsible for hosting a *hierarchy* of *SAOs* and servicing them with data supplied from other containers, *clients* or *External System Retrievers*. A container uses a single execution thread to deliver *updates* to SAOs and then, by means of a *ripple*, propagate resultant changes to other *dependant* SAOs and supply updates to other containers or clients which hold subscriptions to the affected SAOs.

Cycle
A pattern in the *dependency graph* of *connections* between *SAOs* which revisits an SAO during a single *ripple*. The most direct form of a cycle occurs when an SAO uses its own *output* value as one of its own inputs. Cycles in the dependency graph can be used to give controlled *feedback* effects, but should be used with care, as they can yield results which are difficult to interpret.

Data Source
In *SONARIS/Framework* terms, a source of configuration data which can be read in order to initialise a SONARIS/Framework *application service* (usually a relational database). See *Configuration Database*.

Dependant
In *SONARIS/Framework* terms, an *SAO* whose value is established and maintained based on a *connection* which uses the *output* value of another SAO as one of its *inputs* is said to be dependant on that other SAO.

Dependency Graph
The set of *dependency* relationships established as a result of forming *connections* between *SAOs*.

ESR
See *External System Retriever*.

ESS
See *External System Sender*.

Export
The process of creating an XML description of a *hierarchy* of *SAOs* which can then be used to create a copy of that hierarchy in another location in, possibly, another *application service*. See also *Import File*.

External System Retriever
An implementation of the **IExternalSystemRetriever** interface (defined in `isafesr.h`) which wraps third party software provided by an external information supplier allowing data to be passed to individual SAOs within a container. An implementation of an ESR will typically map the information available from the external supplier into a hierarchical namespace which corresponds to hierarchies of SAOs that can be created within application containers. If the supplier being wrapped also supports the return of data for external publishing or contribution a combined IExternalSystemSender interface may also be supplied.

External System Sender
An implementation of the **IExternalSystemSender** interface (defined in `isafess.h`) which wraps third party software provided by an external information publisher allowing data to be sent from individual SAOs within a container for publication by the external system. Like an *ESR*, an implementation of an ESS will typically map the third party structures via which data can be published into a hierarchical namespace which corresponds to hierarchies of SAOs that can be created within application containers.

Feedback
A condition arising from a *cycle* in the *dependency graph* where the value of an *SAO* is set during a *ripple* but not used by one of its *dependants* until a later ripple.

Hierarchy
A complete tree, or branch of a greater tree, of *SAOs* hosted by one or more *containers*. The complete hierarchy representing an *application* is usually composed of several individual hierarchies hosted by separate containers distributed across numerous *machines* and *processes* and joined together at *mount points*.

Import File
An XML file containing a description of a *hierarchy* of *SAOs*. Generally, such files also contain information about *connections* between SAOs in the file as well as their values. Import files can be used to create complex hierarchies of connected SAOs from a single command.

Input
Part of the *definition* of an *SAO* which specifies the end point of a *connection* formed from some other SAO to this in order that this SAO be able to use the *output* value of that SAO in establishing and maintaining its own value.

LCS
See *Licence/Configuration Server*.

Licence/Configuration Server
A *node* within a *partition* configured to manage configuration, licencing and connection topology for the other nodes within the partition.

Listener Interface
An interface to a *SONARIS/Framework* programming object which must be implemented by a program or component wishing to communicate with *SONARIS/Framework Runtime Environment*. See also *Active Interface*.

Machine
In *SONARIS/Framework* terms, a host on which *processes* hosting *containers* are run to provide all or part of an *application service.*

Mount Point
A defined location within a *hierarchy* of *SAOs* hosted by a *container* at which another hierarchy, hosted by another container is rooted.

Node
A machine on which *SONARIS/Messaging* is installed and which forms messaging connections to other nodes within the same *partition*.

Output
The current value of an *SAO* available to any *client* which holds a subscription to it, or to any other SAO which uses it as one of its *inputs*.

Parent In terms of *SONARIS/Framework hierarchies*, an *SAO* which has one or more *child* SAOs is said to act as the *parent* for those SAOs.

Partition
A group of *nodes* one of which is configured to act as *LCS* for the others.

Persistent SAO
An instance of an *SAO* whose details (including Id, name, *hierarchical* location, ownership details, *input* details etc.) and (optionally) value, are stored in the *configuration database* for the *application* such that the SAO will be available every time the *container* which hosts it is run.

Process
In *SONARIS/Framework* terms, an individual invocation of the `saf` program which creates and manages one or more *containers* to provide part or all of an *application* service.

Proxy SAO
A subset of the full functionality of an *SAO* used to remotely represent an SAO in either a *client* program or a *container* other than the one which hosts it.

Ripple
A sequence of calls by a *container* to methods of a subset of the *SAOs* managed by that container to ensure that the values held by those SAOs is brought up to date with respect to one another, and to pass on any value changes which have occurred to *client* programs which hold a subscription to those SAOs. A ripple is usually performed as a result of new data being supplied as *updates* to one or more SAOs from some external source (another container, a client, an *ESR* etc.) to allow any *dependant* SAOs to modify their values accordingly.

SAO
See *SONARIS Application Object*.

SAO Definition
A formal description of an *SAO* that provides a general description of the SAO, defines the type of data it is capable of storing, and specifies the number, types, usage conditions, and general descriptions of its *inputs*. Such definitions are supplied with every *SAO library* as XML text, and must be loaded by every *application* which makes use of that library to verify appropriate usage of instances of the SAOs within the application.

SAO Library
A library of *SAOs* supplied in two parts; a dynamic linked library binary file specific to a particular computer architecture, and an architecture and operating system neutral *SAO library definition* file which formally describes the SAOs provided by the library using XML.

Sibling
In terms of *SONARIS/Framework hierarchies*, an *SAO* whose *parent* also acts as a parent for one or more other SAOs is said to be a *sibling* of those SAOs.

SONARIS/Framework Runtime Environment The core features and services of the *SONARIS/Framework* system available to client programs and SAOs.

SONARIS/Framework
Component based, distributed development framework for implementing real-time calculation and data processing solutions.

SONARIS Application Object
The most basic component of a *SONARIS/Framework application*. SAO instances are hosted by *containers* and arranged in *hierarchies*. The implementations of SAOs are supplied by *SAO libraries*.

SONARIS/Messaging
The underlying messaging system used by *SONARIS/Framework* for the exchange of data between SONARIS/Framework *containers* and *clients*.

Template

A branch of a *hierarchy* of *SAOs* which serves as a master copy to be replicated at other points in the hierarchy.

Transient SAO

An instance of an *SAO* which, unlike a *persistent SAO*, is only available for the life time of the of the *container* which hosts it. The Id assigned to a transient SAO will not be used again by any other SAO.

Update

A delivery of a new value to an *SAO* (usually from outside of the current *container*) or from a container to a *client*.