SONARIS/Framework

Application 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

The **SONARIS/Framework** enables the rapid development of complex distributed Client/Server applications composed of structures of standard or custom produced *SONARIS Application Object* (SAO) components. This text will describe how to create and run a **SONARIS/Framework** application service composed of only standard SAO components using the **SONARIS Studio** program. The example presented here is a simple currency calculator service. You may follow the procedure described here to create the example for yourself (this will take about an hour) or, alternatively, you may refer to a preprepared version of the application service supplied with the SDK component of the **SONARIS/Framework** software package. As you follow the text you may wish to refer to the *SONARIS/Framework - Studio User Guide* for further details of how to use **SONARIS Studio**. Refer to the various **SONARIS/Framework** client programming guides for examples which describe how to create client programs which can access the calculated results produced by this application using a variety of programming language interfaces.

1.1 Prerequisites

Before continuing, make sure you have read the *SONARIS/Framework - Overview and Concepts* document. In order to repeat the examples presented in this text, please install the following subsets from the **SONARIS/Framework** kit:

- **Runtime Environment**
- **Server Environment**
- **Java Client Interface**
- **Studio User Interface**
- **Software Development Kit**

1.2 Architecture

A full description of the architecture of **SONARIS/Framework** applications appears in the *SONARIS/Framework - Overview and Concepts* document. You should be familiar with the following concepts covered by this document:

- **SONARIS/Framework** Client/Server architecture
- **SONARIS/Framework** Processes and Containers
- **SONARIS/Framework** SAO hierarchies and dependency relationships

Before starting to construct the currency calculator example, here is a brief outline of its structure.

For simplicity the currency calculator application will make use of prerecorded data rather than accessing external systems for foreign exchange data. It will use a single container managed by a single `saf` process to host the SAOs which form the calculator.

2 Creating A New Application

2.1 Overview

There are several methods by which new **SONARIS/Framework** application services may be created, but a common feature of these is the creation of a configuration database hosted by a relational database system such as **SQLServer** or **Microsoft Access** which stores details of the SAOs used by the service. This database must contain some basic system information to enable the **SONARIS/Framework** processes and containers to be created. For details of RDBMS systems supported by **SONARIS/Framework** please refer to the *SONARIS/Framework - Release Notes*. In this text we will use a simple procedure to create our example service. For full details of **SONARIS/Framework** application service design and creation and details of the utilities described here, please refer to the *SONARIS/Framework - Installation and Maintenance Guide*.

All **SONARIS/Framework** application services are controlled by an instance of a program called `safbase` which reads, publishes and maintains the system information from the configuration database. There must be a single instance of `safbase` running before the `saf` processes which comprise the application may start. All **SONARIS/Framework** containers are created and populated by running instances of the `saf` executable. Since application services are often composed of multiple containers run by several `saf` processes, a program called `safcontrol`, which runs in the background, is able to manage all `saf` processes which form the service by reading the system information for the application service from the corresponding `safbase` process. If a service is configured to use multiple `saf` processes on multiple machines then the instance of `safcontrol` running on each machine will manage the appropriate `saf` processes for that machine. Like `safbase` the `saf` processes read their configuration information (the details of the SAOs they are to host) from the configuration database. Details of new SAOs which are added to the application once it is running are recorded in this database.

2.2 Creating an Application

The method we will describe here to create the example application service uses the **SONARIS Studio** program. Refer to the *SONARIS/Framework - adminuserguide* for more information about this procedure.

1. Start **SONARIS Studio**. Depending on your operating system this may be available from a menu on your desktop. On all operating systems this program may be started from the command line by running `safstudio` from the **SONARIS/Framework** installation directory.

2. The first dialog you will see will prompt you for details of the application service to which you wish to connect, and also offers you the option of creating a new application. Select the New... option from the Application menu to bring up the Create S/AF Application dialog.

3. If you wish to store the configuration data for the new application in an **Microsoft Access** database, select the Access Application tab pane on the dialog. Alternatively, if you wish to store the configuration data in another database via an ODBC connection, select the ODBC Application tab pane.

4. In whichever tab pane you have selected, type `crossrate` as the Application name. The other fields in this dialog will be automatically completed with appropriate default values

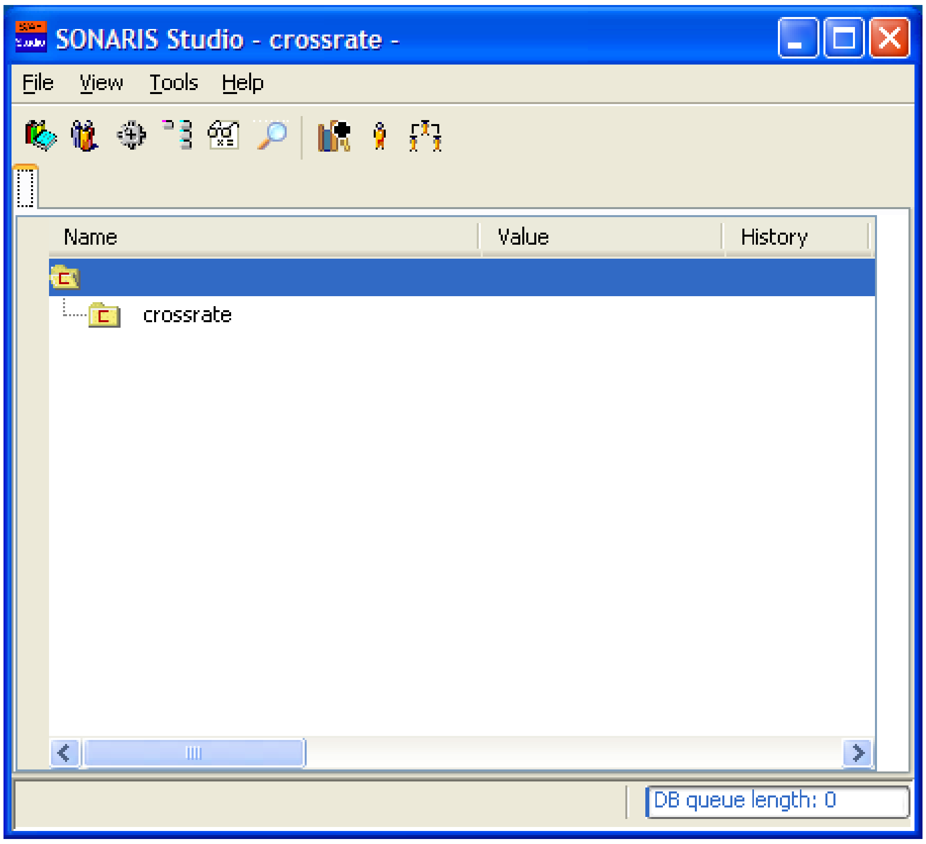


Figure 2.1: Newly Created **crossrate** Application

which correspond to the application name as you type it (N.B. If you are using the ODBC Application tab pane the value for Data source name will default to a value based on the Application name. You must either prepare a data source which matches this name or amend the name to refer to an existing data source).

5. Click the Create... button. **SONARIS Studio** will prepare a new configuration database for the **crossrate** application service before closing the dialog.
6. The appropriate values for the new application service will now appear in the connection dialog. Click the Connect... button.
7. The Start a **SONARIS/Framework** Application dialog will be displayed already completed with details of the application you just created. Click the Start button to start the application.

SONARIS Studio will connect to the service and provide you with a window by which you can examine and alter the SAOs which comprise the application. Your first view of the **crossrate** application should appear similar to that shown in Figure 2.1.

3 Creating SAO Hierarchies

In this chapter we will start to populate our **crossrate** application with SAOs to create a foreign exchange calculator. This application will take values for the current dollar exchange rate for various currencies, and calculate their direct exchange rates. We will use the **SONARIS Studio** application to create a suitable hierarchy to accomplish this. For full details of the features of **SONARIS Studio** used here refer to the *SONARIS/Framework - Studio User Guide*.

3.1 Folders

Our first task will be to create an outline structure for the application using folder SAOs to divide our application hierarchy into logical areas. Creating new Folder SAOs in **SONARIS Studio** is simply a matter of selecting the point in the hierarchy at which the new folder is to be added, and selecting one of the options from the New folder submenu accessed via a right mouse button click. Perform the following actions.

1. Create a folder called `TestData` as a child of the `crossrate` folder already prepared for us.
2. Also under `crossrate` create folders for `Calculations` and `Timers`.
3. Under `TestData` create a folder called `Forex`.

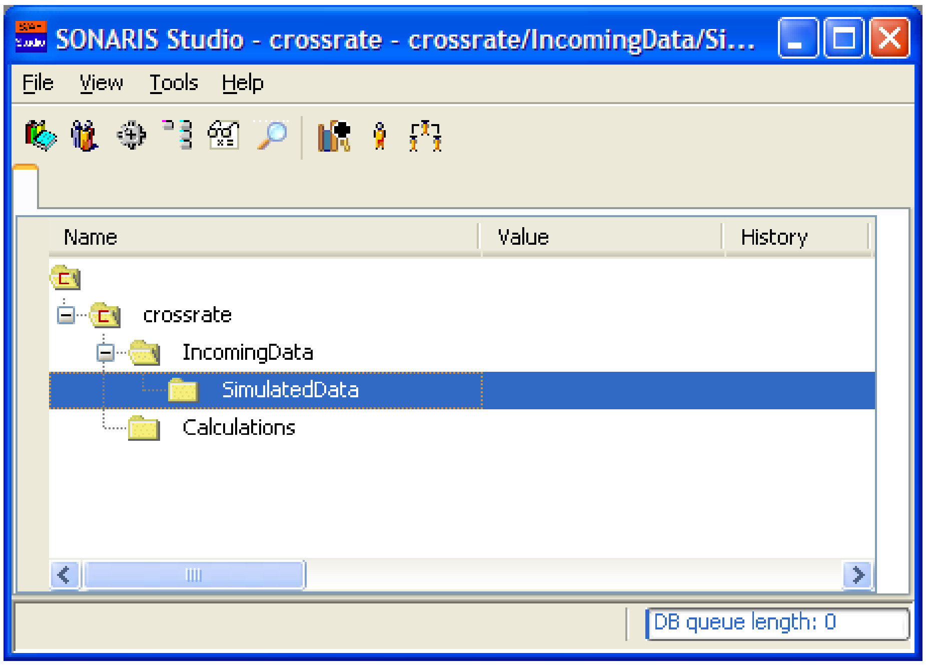
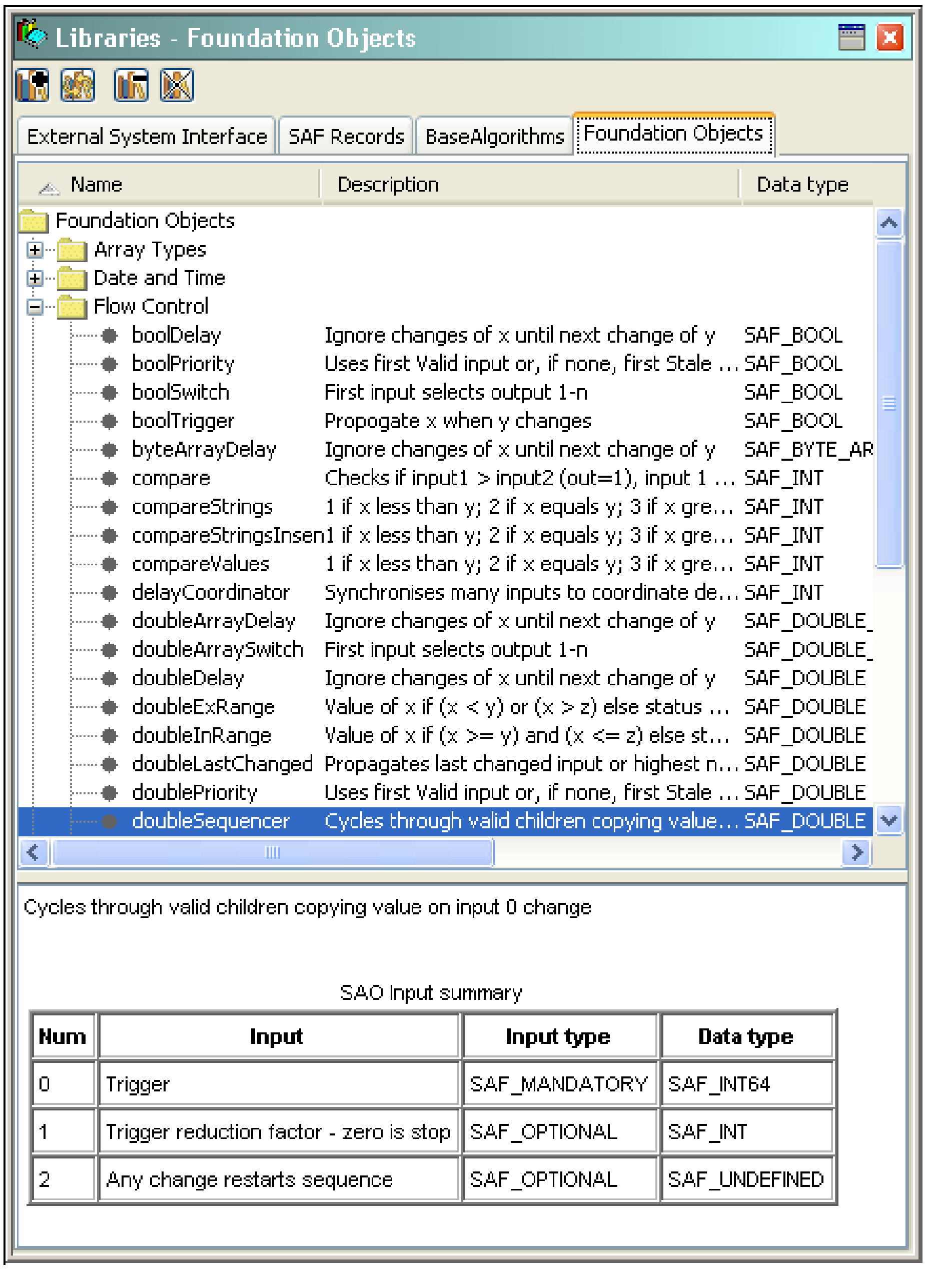


Figure 3.1: Creating New Folders

3.2 Adding SAOs

Next we will create some SAOs which will simulate the arrival of foreign exchange data for a selection of currencies. The standard **SONARIS/Framework** library (the *Foundation Objects*) supplies some SAOs designed to help us with this called *Sequencers*. SAOs of this type take their



Name	Description	Data type
Foundation Objects		
Array Types		
Date and Time		
Flow Control		
boolDelay	Ignore changes of x until next change of y	SAF_BOOL
boolPriority	Uses first Valid input or, if none, first Stale ...	SAF_BOOL
boolSwitch	First input selects output 1-n	SAF_BOOL
boolTrigger	Propogate x when y changes	SAF_BOOL
byteArrayDelay	Ignore changes of x until next change of y	SAF_BYTE_AR
compare	Checks if input1 > input2 (out=1), input 1 ...	SAF_INT
compareStrings	1 if x less than y; 2 if x equals y; 3 if x gre...	SAF_INT
compareStringsInsen	1 if x less than y; 2 if x equals y; 3 if x gre...	SAF_INT
compareValues	1 if x less than y; 2 if x equals y; 3 if x gre...	SAF_INT
delayCoordinator	Synchronises many inputs to coordinate de...	SAF_INT
doubleArrayDelay	Ignore changes of x until next change of y	SAF_DOUBLE_
doubleArraySwitch	First input selects output 1-n	SAF_DOUBLE_
doubleDelay	Ignore changes of x until next change of y	SAF_DOUBLE
doubleExRange	Value of x if (x < y) or (x > z) else status ...	SAF_DOUBLE
doubleInRange	Value of x if (x >= y) and (x <= z) else st...	SAF_DOUBLE
doubleLastChanged	Propagates last changed input or highest n...	SAF_DOUBLE
doublePriority	Uses first Valid input or, if none, first Stale ...	SAF_DOUBLE
doubleSequencer	Cycles through valid children copying value...	SAF_DOUBLE

Cycles through valid children copying value on input 0 change

SAO Input summary

Num	Input	Input type	Data type
0	Trigger	SAF_MANDATORY	SAF_INT64
1	Trigger reduction factor - zero is stop	SAF_OPTIONAL	SAF_INT
2	Any change restarts sequence	SAF_OPTIONAL	SAF_UNDEFINED

Figure 3.2: The *doubleSequencer* SAO

value from that of their children, changing their value from one child to the next when triggered to do so by a value change on their first input. In this case we will use the *doubleSequencer* SAO since we wish to use decimal values as our test data.

3.2.1 Creating SAO Instances

To add a *doubleSequencer* to the application, perform the following actions.

1. Open the Libraries view by selecting Libraries from the View menu of the Hierarchy window or click on the Libraries icon on the tool bar. Select the *Foundation Objects* library tab pane, expand the *Flow Control* category and select the *doubleSequencer* SAO (see Figure 3.2).

2. Drag the *doubleSequencer* SAO across to the SAO hierarchy and drop it onto the `Forex` folder we just created. The new SAO will be created as a child of `Forex`, and the name will be highlighted ready for you to set.

3. Type the name of the SAO as `EURBid` and press Enter.

4. Next, return to the Libraries window, expand the *Simple Data Types* category, and select the *basicDouble* SAO.

5. Whilst pressing the CTRL key drag this SAO and drop it onto the `EURBid` sequencer SAO we just created. A new instance of the *basicDouble* SAO will be created as a child of the sequencer SAO, again with the name highlighted ready for you to set. N.B. Not using the CTRL key during the drag and drop will result in the new SAO being created as the next sibling of the *Sequencer* SAO rather than as a child - not what we want in this case.

6. Type the name of the SAO as `Val1` and press Enter.

7. Drag and drop another *basicDouble* SAO (not using the CTRL key this time) onto the `EURBid` sequencer SAO. A new instance of the *basicDouble* SAO will be created as a child of the sequencer SAO, again with the name highlighted ready for you to set. Note that we *didn't* use the CTRL key during the drag and drop this time, so our new SAO is created as the last child of the `EURBid` sequencer SAO.

8. Type the name of the SAO as `Val2` and press Enter.

9. Repeat steps 7 and 8 six more times to create a set of eight children under the `EURBid` SAO (`Val1`, `Val2`, `Val3` etc.).

3.2.2 Initializing and Storing SAO Values

Having established the children of the sequencer, we need to give them some values and make sure that they are saved in the configuration database for the application. Perform the following actions.

1. Double click on the Value field of the `Val1` SAO and type in the value `1.234` and press Enter.

2. Click on the History field and change the setting from `No History` to `Current Value`.

3. Repeat the above step for the `Val6` SAO setting its value to `1.5678.` Leave the other child SAOs uninitialized.

3.3 Connecting SAOs

We now have a selection of values which can be used by the `EURBid` sequencer, but we don't yet have anything which will trigger it to start using those values. The simplest answer to this is to use an instance of an *IntervalTimer* whose value will update at a frequency determined by the name we give it. To create this SAO perform the following actions.

1. Expand the *Timers* category of the *Foundation Objects* library in the Library view and select the *IntervalTimer* SAO.

2. Drag the *IntervalTimer* SAO and drop it onto the `Timers` folder.

3. Type the name for this new SAO as `1000`. This will indicate to the timer that it should update its value every 1000 milliseconds.

The *IntervalTimer* should immediately start to show an updating value (the value supplied is actually the number of milliseconds since the January 1st 1970). To form a connection between the *IntervalTimer* SAO and the `EURBid` sequencer SAO, perform the following actions.

1. Open the View menu and select Show Dependencies. Two tables showing the connection condition of the SAO currently selected will be opened at the bottom of the hierarchy view.

2. Select the `EURBid` SAO in the hierarchy.

3. Drag the *IntervalTimer* which we added to the hierarchy as `1000` and drop it onto the first line of the Inputs table (Input No. 0).

As soon as the connection is formed the `EURBid` sequencer will adopt the value of its first child (`Val1` whose value we set to `1.234`). The next four updates to the *IntervalTimer* will not cause a change in the value of the sequencer since the children at positions 2, 3, 4 and 5 are uninitialized. However, the fifth update will cause the sequencer value to adopt the value of child `Val6` and so show a value of `1.5678`.

The SAO hierarchy should now be like that shown in Figure 3.3

3.4 Templates

We have now established a hierarchy which simulates updates for a single item of test data, `EURBid`. To simulate sufficient information for our foreign exchange calculator we are going to need updating values for at least another three items of data - a value for `EURAsk` and two more SAOs which represent bid and ask dollar exchange rates for some other currency. Fortunately this need not require any more tedious dragging and dropping because we can use the `EURBid` hierarchy as a template and simply make copies to which we can make minor alterations. This may be done as follows -

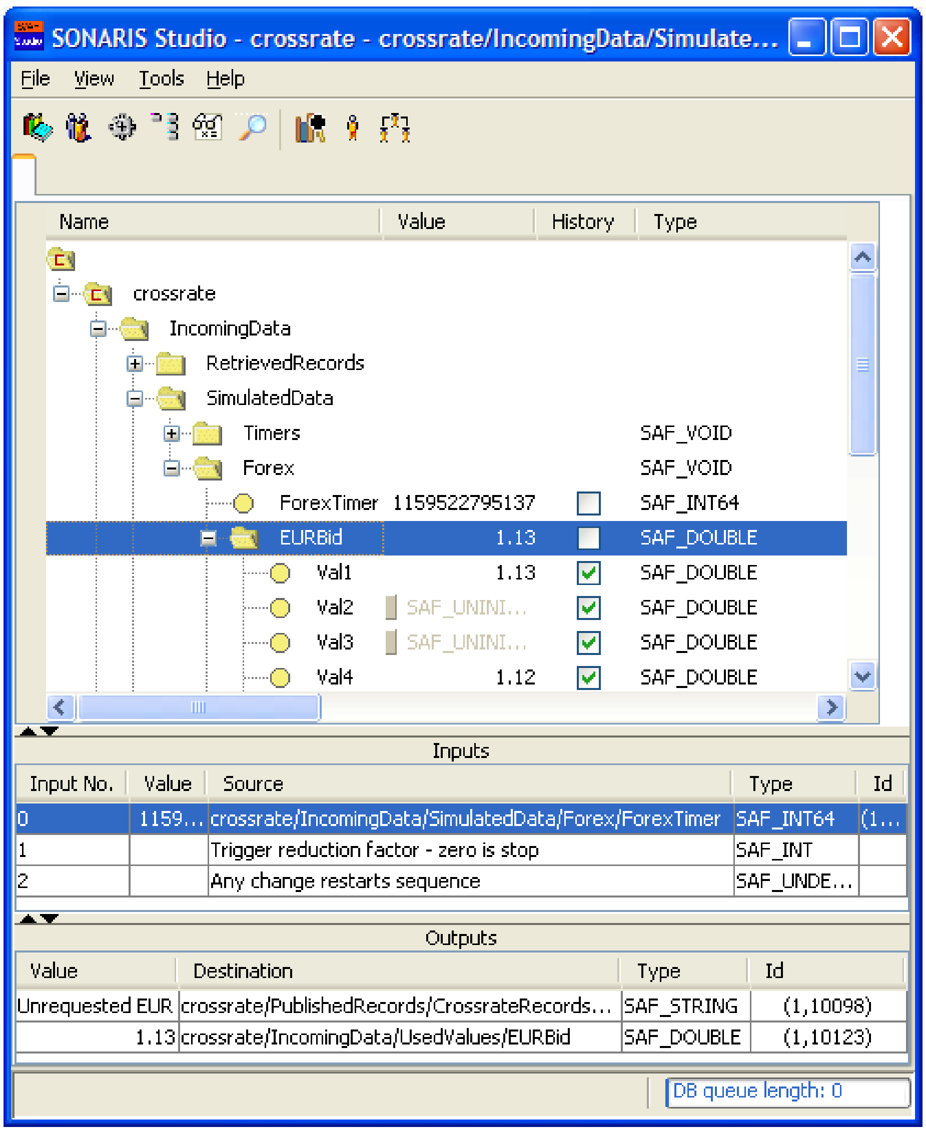
1. Select the `EURBid` SAO in the Hierarchy window and select Copy from the right mouse button menu.

2. Select the `Forex` folder which is the parent of the `EURBid` SAO and select Paste from the right mouse button menu. A new SAO will appear as the last child of the folder with its name highlighted and ready to be amended.

3. Type the name of the new SAO as `EURAsk`.

4. Repeat steps 2 and 3 six more times using the names `CHFBid`, `CHFAsk`, `CADBid`, `CADAsk`, `GBPBid` and `GBPAsk`.

5. Alter the values of the `Val` SAOs under each of the newly created *doubleSequencer* SAOs. Set values in a selection of these SAOs (remembering to set the History entry for each SAO to which you assign a value) to simulate updating values for each sequencer. If you want to clear any of these values, unset the History entry for the SAO and use Edit Status from the right mouse button menu to set the status of the SAO back to SAF_UNINITIALIZED.


SONARIS Studio - crossrate - crossrate/IncomingData/Simulate...
File View Tools Help
DB queue length: 0

Name	Value	History	Type
crossrate			
IncomingData			
RetrievedRecords			
SimulatedData			
Timers			SAF_VOID
Forex			SAF_VOID
ForexTimer	1159522795137	☐	SAF_INT64
EURBid	1.13	☐	SAF_DOUBLE
Val1	1.13	✓	SAF_DOUBLE
Val2	SAF_UNINI...	✓	SAF_DOUBLE
Val3	SAF_UNINI...	✓	SAF_DOUBLE
Val4	1.12	✓	SAF_DOUBLE

Inputs

Input No.	Value	Source	Type	Id
0	1159...	crossrate/IncomingData/SimulatedData/Forex/ForexTimer	SAF_INT64	(1...
1		Trigger reduction factor - zero is stop	SAF_INT	
2		Any change restarts sequence	SAF_UNDE...	

Outputs

Value	Destination	Type	Id
Unrequested EUR	crossrate/PublishedRecords/CrossrateRecords...	SAF_STRING	(1,10098)
1.13	crossrate/IncomingData/UsedValues/EURBid	SAF_DOUBLE	(1,10123)

Figure 3.3: The SAO Hierarchy with a Working *doubleSequencer* SAO

3.5 Loading SAO Libraries

So far all the SAOs we have used to prepare our simulated data have been available in the standard *Foundation Objects* library which is installed by default in any new **SONARIS/Framework** application. One of the steps involved in calculating the exchange rate between the currencies we have prepared is to establish the *mid price* value for each currency, this being the average of the bid and ask values for that currency. Although we could achieve this using simple mathematical operations (addition and division) available from SAOs in the *Foundation Objects* library, a more convenient *Mean* SAO is available. However, this SAO is supplied in a different SAO library called *BaseAlgorithms*. Before we can use the SAOs supplied by this library we must load it's description into our application.

1. Click on the Load library icon on the tool bar at the top of the Library view.
2. In the Load library definition dialog enter the `basealgorithms.xml` library description file. This is located in the `saolibs` directory under the **SONARIS/Framework** installation directory.

3. Click the Load button.

The new library description supplying a selection of new SAOs will be loaded into the Library window in a new tab page. One of these new SAOs will be the *Mean* SAO.

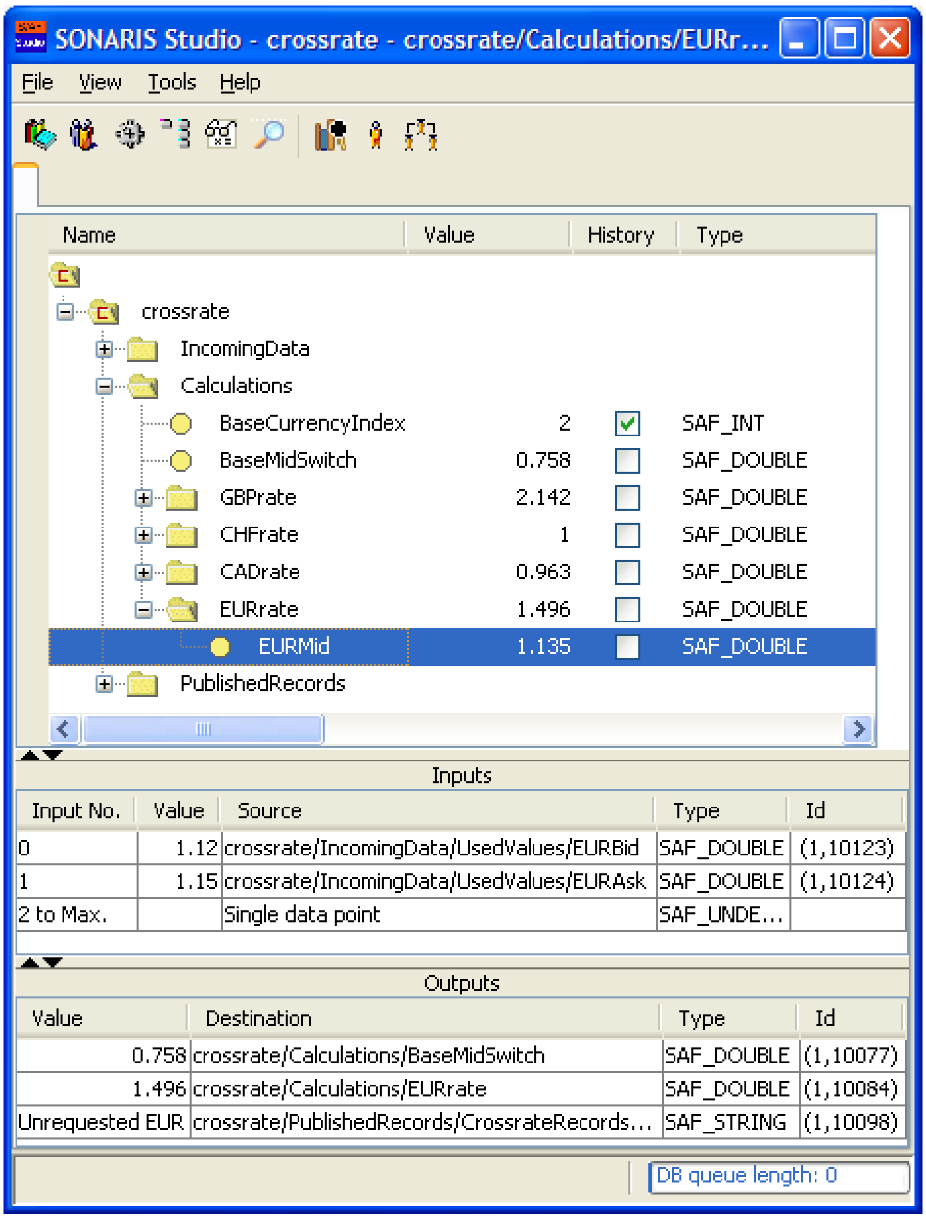
3.6 Performing Calculations

We now have the extra SAOs we need and a selection of simulated foreign exchange data. The next task is to start using this data to calculate the exchange rates. Again, we can start by assembling a group of SAOs by dragging and dropping from the Library view and connecting them to calculate a rate between two of the currencies. Let's start by calculating the number of Euros that one Swiss Franc buys.

3.6.1 Preparing the `CHF` SAOs

1. Open the *Simple Maths* category of the *Foundation Objects* in the Library view.
2. Drag the *doubleDivide* SAO and drop it on the `Calculations` folder in the SAO hierarchy. The name of the new SAO will be highlighted ready for you to set.
3. Enter the name of the SAO as `CHFRate`.
4. Select the *BaseAlgorithms* library tab page in the Library view.
5. Whilst pressing the CTRL key drag and drop the *Mean* SAO onto the `CHFRate` SAO we just created. Set the name of the new SAO to `CHFMid`.
6. Select the `CHFMid` SAO.
7. Locate and drag and drop the `CHFBid` *doubleSequencer* (under the `TestData/Forex` folder) SAO onto the first input (Input No. 0) of the `CHFMid` SAO in the other Hierarchy window.
8. Locate and drag and drop the `CHFAsk` *doubleSequencer*SAO (under the `TestData/Forex` folder) onto the second input (Input No. 1) of the `CHFMid` SAO. The `CHFMid` SAO will now start to show the mid value between the `CHFBid` and `CHFAsk` values.
9. Select the `CHFRate` SAO.
10. Drag and drop the `CHFMid` SAO onto the first input (Input No. 0) of the `CHFRate` SAO.

We now require a value for the mid price between the `EURBid` and `EURAsk` to use as the second input to the `CHFRate` SAO to give us the exchange rate between the two currencies. To calculate this mid price, and also to prepare an SAO with which we can calculate other exchange rates against the Euro, repeat the above steps but reading `EUR` at all points where you previously used `CHF`. Once you have finished, your completed SAO hierarchy should look like that shown in Figure 3.4



Name	Value	History	Type
crossrate			
IncomingData			
Calculations			
BaseCurrencyIndex	2	✓	SAF_INT
BaseMidSwitch	0.758		SAF_DOUBLE
GBPrate	2.142		SAF_DOUBLE
CHFrate	1		SAF_DOUBLE
CADrate	0.963		SAF_DOUBLE
EURrate	1.496		SAF_DOUBLE
EURMid	1.135		SAF_DOUBLE
PublishedRecords			

Inputs

Input No.	Value	Source	Type	Id
0	1.12	crossrate/IncomingData/UsedValues/EURBid	SAF_DOUBLE	(1,10123)
1	1.15	crossrate/IncomingData/UsedValues/EURAsk	SAF_DOUBLE	(1,10124)
2 to Max.		Single data point	SAF_UNDE...	

Outputs

Value	Destination	Type	Id
0.758	crossrate/Calculations/BaseMidSwitch	SAF_DOUBLE	(1,10077)
1.496	crossrate/Calculations/EURrate	SAF_DOUBLE	(1,10084)
Unrequested EUR	crossrate/PublishedRecords/CrossrateRecords...	SAF_STRING	(1,10098)

DB queue length: 0

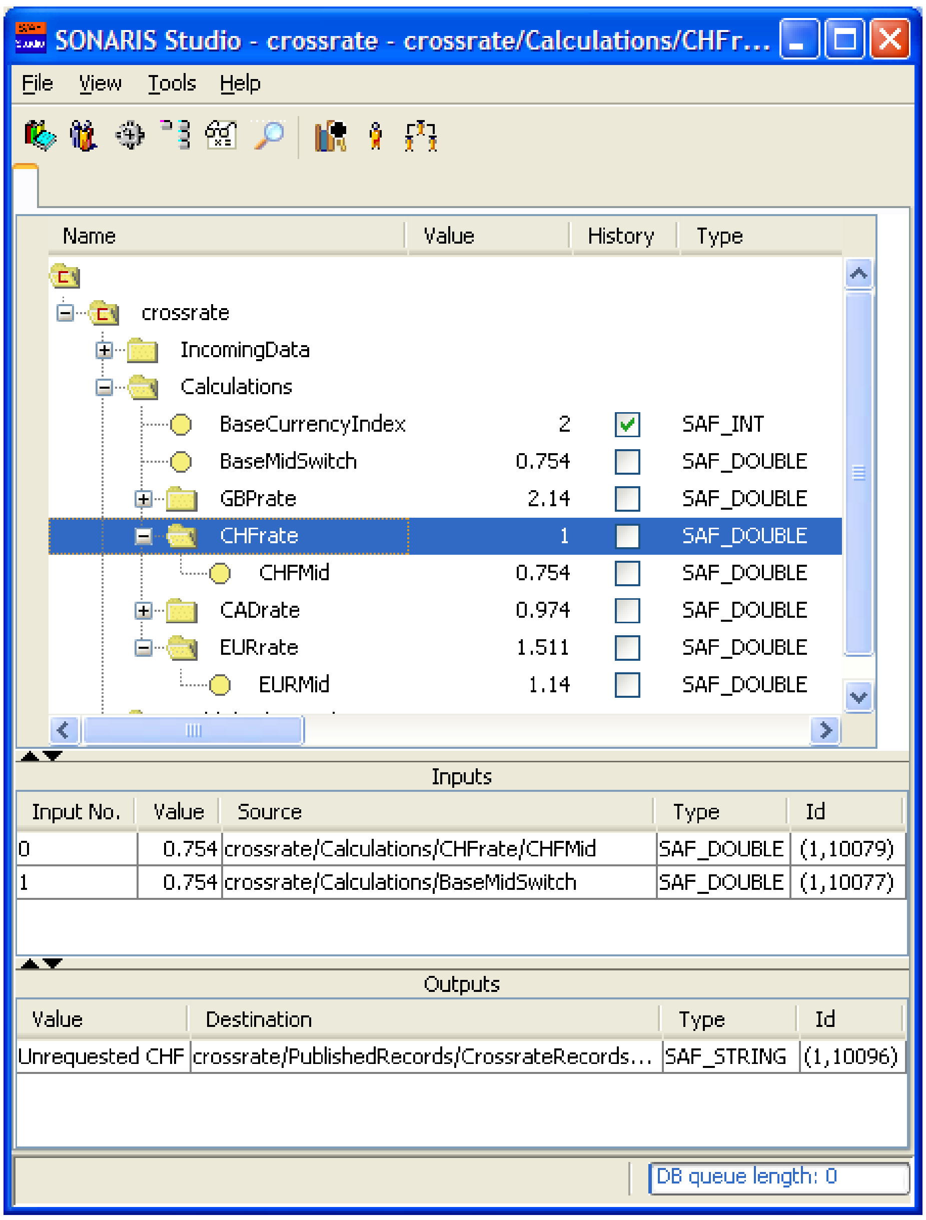
Figure 3.4: Mid Prices for Euro and Swiss Franc

3.6.2 Connecting the Rate Calculator

All that now remains to be done to calculate the exchange rate between Euros and Swiss Francs is to connect the `EURMid` SAO as the second input of the `CHFRate` SAO.

1. Select the `CHFRate` SAO.

2. Locate the `EURMid` SAO (the child of `EURRate`) and drag and drop it onto the second input (Input No. 1) of the `CHFRate` SAO. The exchange rate between the two currencies will immediately be calculated and kept up to date as the simulated exchange rate data changes.



Name	Value	History	Type
crossrate			
IncomingData			
Calculations			
BaseCurrencyIndex	2	✓	SAF_INT
BaseMidSwitch	0.754	☐	SAF_DOUBLE
GBPrate	2.14	☐	SAF_DOUBLE
CHFrate	1	☐	SAF_DOUBLE
CHFMid	0.754	☐	SAF_DOUBLE
CADrate	0.974	☐	SAF_DOUBLE
EURrate	1.511	☐	SAF_DOUBLE
EURMid	1.14	☐	SAF_DOUBLE

Inputs

Input No.	Value	Source	Type	Id
0	0.754	crossrate/Calculations/CHFrate/CHFMid	SAF_DOUBLE	(1,10079)
1	0.754	crossrate/Calculations/BaseMidSwitch	SAF_DOUBLE	(1,10077)

Outputs

Value	Destination	Type	Id
Unrequested CHF	crossrate/PublishedRecords/CrossrateRecords...	SAF_STRING	(1,10096)

DB queue length: 0

Figure 3.5: Euro Rates for Swiss Franc, Canadian Dollar and British Pound

3.6.3 Completing the `Calculations` Hierarchy

Now that we have created the hierarchies for `EUR` and `CHF`, all that remains is to repeat the steps in section Section 3.6.1 for the other currencies, `GBP` and `CAD`. Once SAOs for rate calculation for these additional currencies have been created, their Euro exchange rates may be calculated by connecting the `EURMid` SAO to the second input of their `Rate` SAO in a similar manner to the connections made to the `CHFRate` SAO. The resulting hierarchy will look like that shown in Figure 3.5. For completeness, also connect the `EURMid` SAO to the second input of the `EURRate` SAO, thus showing the exchange rate from Euros to Euros - of course the result of this calculation will always be `1`.

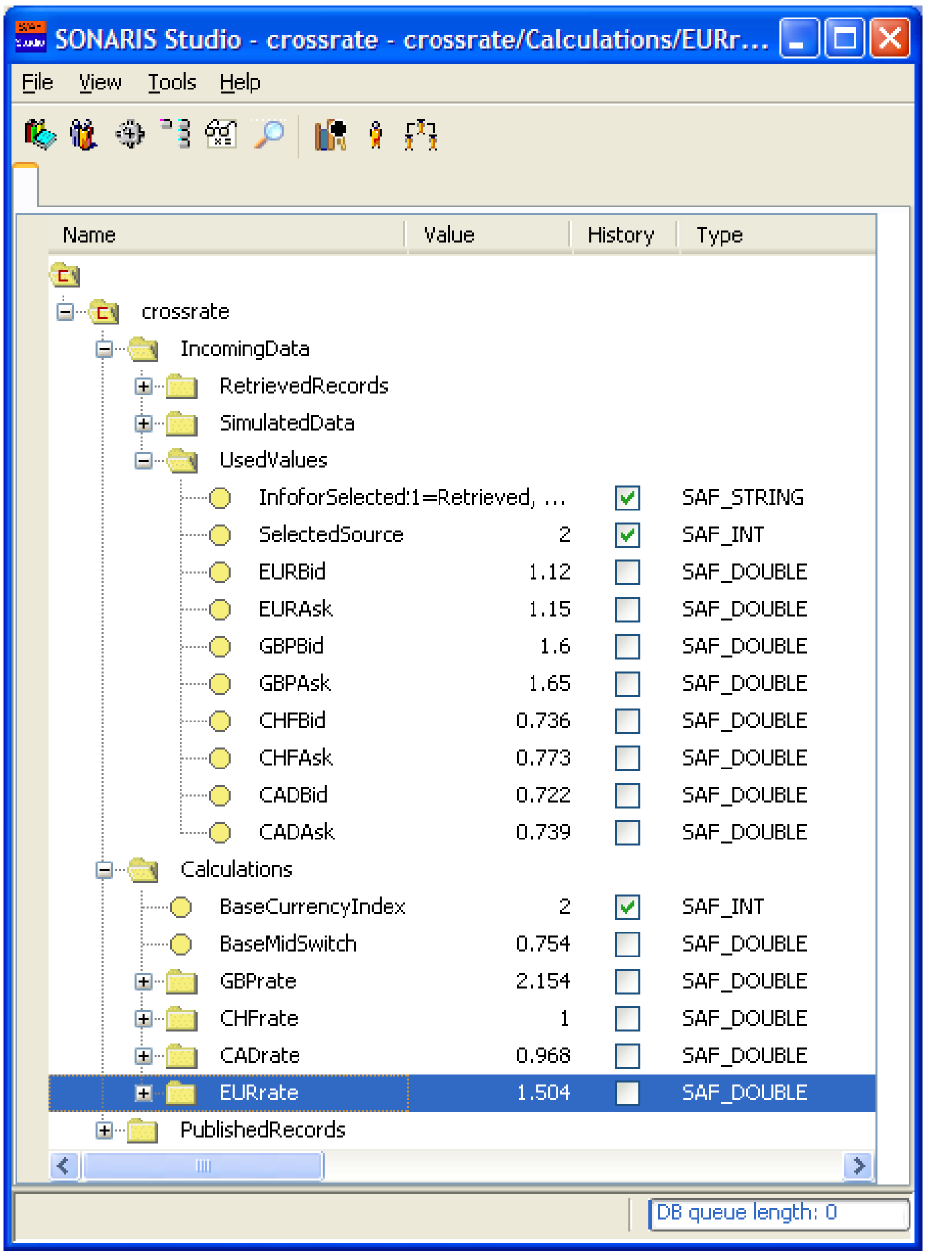
3.6.4 Extending Behaviour by Modifying Connections

Our hierarchy is currently calculating the value in Euros of the other three currencies. With the addition of two new SAOs and some alteration to the connections we can enhance the application to allow it to use any of the four currencies as the base rate against which the other three will be valued.

1. Expand the *Simple Data Types* category and select the *basicInteger* SAO from the Libraries view.
2. Hold the Shift key down and Drag the *basicInteger* SAO and drop it onto the `Calculations` folder in the SAO hierarchy (the effect of the Shift key being to cause the new SAO to be created as the first child of the `Calculations` SAO).
3. Type the name of the SAO as `BaseCurrencyIndex` and press Enter.
4. Next, return to the Libraries view, expand the *Flow Control* category, and select the *doubleSwitch* SAO.
5. Drag the *doubleSwitch* SAO across to the SAO hierarchy and drop it onto the new `BaseCurrencyIndex` SAO. The new SAO will be created as the next sibling of the `BaseCurrencyIndex`.
6. Type the name of the SAO as `BaseMidSwitch` and press Enter.
7. Select the new `BaseMidSwitch` SAO.
8. Drag and drop the `BaseCurrencyIndex` SAO onto the first input (Input No. 0) of the `BaseMidSwitch` SAO.
9. Assign the value `1` to the `BaseCurrencyIndex` SAO, and set the History field to `Current Value`.

The behaviour of a *switch* SAO is to adopt the value of its *n*th input where *n* is an integer value supplied by the SAO connected to its first input. Our intention is to connect the `Mid` value SAOs for each of our currencies to the remaining inputs of the `BaseMidSwitch` SAO, and to use this SAO as the second input to the `Rate` SAOs for each of the currency calculations. By altering the value of the `BaseCurrencyIndex` SAO, we can dynamically switch the base currency against which the exchange rates are calculated.

1. Drag and drop the `EURMid` SAO onto the second input (Input No. 1) of the `BaseMidSwitch` SAO.
2. Repeat the previous step three more times to connect the `CHFMid`, `CADMid` and `GBPMid` SAOs to the inputs 2, 3 and 4 of the `BaseMidSwitch` SAO respectively.
3. Select the `EURRate` SAO.
4. Drag and drop the `BaseMidSwitch` SAO onto the second input (Input No. 1) of the `EURRate` SAO. Confirm that you wish to replace the existing input (`EURMid`) with the new one.



Name	Value	History	Type
crossrate			
IncomingData			
RetrievedRecords			
SimulatedData			
UsedValues			
InfoforSelected:1=Retrieved, ...		✓	SAF_STRING
SelectedSource	2	✓	SAF_INT
EURBid	1.12	☐	SAF_DOUBLE
EURAsk	1.15	☐	SAF_DOUBLE
GBPBid	1.6	☐	SAF_DOUBLE
GBPAsk	1.65	☐	SAF_DOUBLE
CHFBid	0.736	☐	SAF_DOUBLE
CHFAsk	0.773	☐	SAF_DOUBLE
CADBid	0.722	☐	SAF_DOUBLE
CADAsk	0.739	☐	SAF_DOUBLE
Calculations			
BaseCurrencyIndex	2	✓	SAF_INT
BaseMidSwitch	0.754	☐	SAF_DOUBLE
GBPrate	2.154	☐	SAF_DOUBLE
CHFrate	1	☐	SAF_DOUBLE
CADrate	0.968	☐	SAF_DOUBLE
EURrate	1.504	☐	SAF_DOUBLE
PublishedRecords			

Figure 3.6: The Complete **crossrate** Application Hierarchy

5. Repeat steps 3 and 4 above to replace the existing second inputs on the `CHFRate`, `CADRate` and `GBPRate` SAOs with the `BaseMidSwitch` SAO.

Your complete application hierarchy will now look like that shown in Figure 3.6. Altering the value of the `BaseCurrencyIndex` SAO will alter the calculations for the exchange rates to use the currency specified.

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*.