

# Manual

## 1. Notice

**Warning:**

This user manual is based on Solar release 1.0.1.

## 2. Overview

Here are some quick facts about Solar system to get you started, more information can be found in various Solar [publications](#).

- In Solar, sources publish events and applications make subscriptions to receive events. So sources and applications are two types of Solar clients.
- You need to assign a static or context-sensitive name for the new source so it can be discovered by applications.
- Applications make subscriptions to Solar in a small subscription language. A subscription can be a simple source selection or a complicated operator graph.
- Solar consists of a set of Planets, and the sources and applications can connect to any of them.

## 3. Client Facade

Class `solar.Client` is a facade object which you can use to interact with Solar system. It lets you to advertise a (static or contexts-sensitive) name for a source, make subscription to receive events. Its constructor takes two arguments: the hostname and the port number of the Planet it should connect to. Its javadoc API can be found [here](#).

## 4. Source

To write a new source, you must inherit it from the abstract class `Source` in the `solar.service.dag` package. The new source class should implement following methods:

- `run()`: You should supply the logic of events producing here, typically an infinite loop. You need to call `publish` method to send the event to all the subscribers of this source.
- `handleQuery(Request query)`: If your source allows pull operation, you should implement

this method.

- `main(String args[])`: Although you can launch the new source from another class, it may be simpler to use the main method to start a source with a name using Client interface.

Your source class may look much like:

```
import java.util.List;
import java.util.Vector;
import solar.Client;
import solar.api.Event;
import solar.api.Request;
import solar.api.Attribute;
import solar.impl.EventFactory;
import solar.service.dag.Source;

public class MySource
    extends Source
{
    ...
    public void run()
        throws Exception
    {
        ...
        while(true)
        {
            ...
            List attrs = new Vector();
            Attribute attribute = new Attribute("dog_name", "skipper");
            attrs.add(attribute);
            ...
            Event new_event = EventFactory.getPlainEvent(attrs);
            ...
            publish(new_event);
            ...
        }
        ...
    }

    public Object handleQuery(Request query)
        throws Exception
    {
        // return answer to the query
    }

    public static void main(String args[])
        throws Exception
    {
        Client client = new Client("tahoe", 5470);
        String advname = "[service=dog]";
        client.advertise(advname, "mylobj", new MySource());
    }
}
```

You can put any serializable object as attribute value, but then you have to get `SerializedEvent` instead of `PlainEvent`. A dummy but complete example can be found as [ClockSource.java](#).

## 5. Operator

To write a new operator, you must inherit it from the abstract class `Operator` in the `solar.service.dag` package. The new operator class should implement following methods:

- `init(Options opts)`: Operator can be initialized with customized parameters (defined in the subscription). After Solar loaded the operator onto a particular Planet, it will call this method using the parameters you supplied to customize the operator. The javadoc for `Options` can be found [here](#).
- `handleEvent(Event evt)`: You should supply the logic of processing received events here. You need to call `publish` method to publish a new event to all the subscribers of this operator.
- `handleQuery(Request query)`: If your operator allows pull operation, you should implement this method.

Your operator class will look much like:

```
import serp.util.Options;
import java.util.List;
import solar.api.Event;
import solar.api.Request;
import solar.api.Attribute;
import solar.impl.PlainEvent;
import solar.service.dag.Operator;

public class MyOperator
    extends Operator
{
    ...
    public void init(Options opts)
    {
        //initialization using customized parameter...
    }

    public void handleEvent(Event evt)
        throws Exception
    {
        ...
        //process the event
        //publish a new event if necessary
        ...
    }

    public Object handleQuery(Request query)
```

```

        throws Exception
    {
        // return answer to the query
    }
}

```

A sample dummy operator that appends attributes to received events can be found as [MutateOperator.java](#).

## 6. Application

To write a new operator, you must inherit it from the abstract class `Application` in the `solar.service.dag` package. The new application class should implement following methods:

- `handleEvent(Event evt)`: You should supply the logic of processing received events here.

Your application segment may look much like:

```

import solar.api.Event;
import solar.service.dag.Application;

public class MyApp
    extends Application
{
    public void handleEvent(Event evt)
        throws Exception
    {
        ...
        //process the event
        ...
    }
}

```

A simple example can be found as [EventPrinter.java](#).

## 7. Composition Language

Solar uses a XML-based data-flow language for applications to compose FAP graph, which can then be used to make context-sensitive name advertisement or subscriptions. A sample graph can be found [here](#).

## 8. Running Solar

You must have Java 1.4.1 (or higher) to run Solar and its clients (run "java -version" to see the Java version you are using). For department Linux machines, it is located at `/usr/java/j2sdk1.4.1/bin`. There are two approaches to use Solar: 1) everyone connects your sources and applications to a public Solar system, or 2) runs your own copy of Solar to

service your sources and applications. You need to follow second approach in case the shared Solar crashed or does not function right.

- You need to download Solar package, and unpack it.
- If you are using department Linux machine, simply run "source script/env.rc" to setup the environment. Otherwise, you need to put all the jar files in lib/ and dist/lib on your Java classpath ([linux](#), [win32](#)).
- Start a Planet:
  - `java solar.Planet -cfg script/solar.cfg`
  - if you got an exception when starting Planet, mostly likely the default port for Planet (5460 and 5470) is used by another process on your host, either go to a different host or specify a different port in script/solar.cfg if you know what you are doing.
- Start your sources (if any) and applications:
  - If the host on which you run Planet is called "tahoe", then you need to initialize all the Client (in sources and applications) with two parameters: "tahoe" and 5470 (or your specified port).
  - Try to run "`java solar.test.ClockSource -host tahoe -port 5470 -name [sensor=clock]`"
  - Try to run "`java solar.test.EventPrinter -host tahoe -port 5470 -query [sensor=clock]`"
  - Your event printer shall print out the events published by clock source now.