

# How to create SOFA 2 application in several steps

## Table of contents

1 Overview.....	2
2 Prerequisites.....	2
3 Steps.....	2

## 1. Overview

This "howto" presents a development of a very simple SOFA application. The application consists of three components, two primitive ones and single composite one. The architecture of the applications is following:

The `Logger` component provides the `Log` interface through which the `Tester` component (it requires the interfaces) calls the `log` method. Both components are then encapsulated in the `LogDemo` composite component.

## 2. Prerequisites

- Correctly installed SOFA 2 and Cushion.

## 3. Steps

1. Run the repository only or complete SOFAnode environment (see [SOFA 2 usage](#) documentation).
2. Create a directory in which you will develop the application and change to it.
3. First, you need to create the `Log` interface type. Call

```
cushion new interface initial foo.ILog
```

### Note:

To achieve uniqueness, names of all entities should follow the Java naming convention. Thus, in the example you should replace the "foo." prefix with something meaningful.

The command creates a new interface type in the repository (empty) and also generates an ADL file with the interface type definition, which you should fill in. The file is created in the `foo.ILog` directory and has name `adl.xml`. The completed ADL file should look as follows

```
<?xml version="1.0"?>
<itf-type name="foo.ILog" signature="foo.ILog" />
```

In the generated file, it is necessary to set the `signature` attribute. This attribute defines the name of the Java interface, which implements this SOFA interface type. The recommended practice is to use the same name as the interface type has.

4. Create a frame of the `Logger` component. Call

```
cushion new frame initial foo.FLogger
```

Similarly to creating a new interface type, the Cushion creates a new frame (empty) in the repository and generates ADL file, which should be filled in. The frame should provide an interface with the type you previously created. The filled in ADL file should look in the following way:

```
<?xml version="1.0"?>
<frame name="foo.FLogger">
  <provides name="log" itf-type="sofatype://foo.ILog"/>
</frame>
```

The bold line defines the provided interface with the name `log` and type `foo.ILog`.

**Note:**

Complete syntax of writing references to other elements is `sofatype://name?version=version`, where the version part can be omitted (if omitted, the Cushion tries to use a current working entity with the particular name).

5. Create a frame of the Tester component. Call

```
cushion new frame initial foo.FTester
```

The component should require the `log` interface. Fill in the corresponding ADL description:

```
<?xml version="1.0"?>
<frame name="foo.FTester">
  <requires name="log" itf-type="sofatype://foo.ILog"/>
</frame>
```

6. Create a frame for top-level composite components. Call

```
cushion new frame initial foo.FLogDemo
```

The top-level component does not provide or require anything. The ADL is following:

```
<?xml version="1.0"?>
<frame name="foo.FLogDemo">
</frame>
```

7. Create an architecture for the Logger component. Call

```
cushion new architecture initial foo.ALogger
```

This component is primitive, i.e. implemented directly in Java and therefore the architecture is empty. The ADL is following:

```
<?xml version="1.0"?>
<architecture name="foo.ALogger" frame="sofatype://foo.FLogger"
impl="foo.ALogger">
</architecture>
```

The **impl** attribute specifies a class, which implements the primitive architecture (it will be created in further steps but for the simplicity you can declare it now).

8. Similarly create an architecture for the Tester component. Call

```
cushion new architecture initial foo.ATester
```

This component is primitive, i.e. implemented directly in Java and therefore the architecture is empty. The ADL is following:

```
<?xml version="1.0"?>
<architecture name="foo.ATester" frame="sofatype://foo.FTester"
impl="foo.ATester">
</architecture>
```

The **impl** attribute has the same meaning as in the previous step.

9. Create an architecture for the LogDemo component. Call

```
cushion new architecture initial foo.ALogDemo
```

The ADL is following:

```
<?xml version="1.0"?>
<architecture name="foo.ALogDemo" frame="sofatype://foo.FLogDemo">
  <sub-comp name="tester" frame="sofatype://foo.FTester"
arch="sofatype://foo.ATester"/>>
  <sub-comp name="logger" frame="sofatype://foo.FLogger"
arch="sofatype://foo.ALogger"/>>
  <connection>
    <endpoint sub-comp="tester" itf="log" />
    <endpoint sub-comp="logger" itf="log" />
  </connection>
</architecture>
```

The architecture defines two subcomponents, i.e. instances of frames `foo.FTester` and `foo.FLogger`. The example uses the possibility to directly specify an architecture of the subcomponents (but is not necessary, the `arch` attribute can be omitted and architectures filled in the assembly descriptor). Then the architecture defines a single connection which connects log interfaces of the logger and tester subcomponents.

10. Commit all changes in the ADL files to the repository. Call

```
cushion commit
```

#### Note:

Without any further parameter, the `cushion commit` commits changes in all working elements.

11. Create code for the `foo.ILog` interface type. In the interface type directory, create code directory and inside it, create a file with the Java interface (Do not forget directories for java packages, in this case `foo`. The complete path to the file will be `foo.ILog/code/foo/ILog.java`).

File content:

```
package foo;
public interface ILog {
    void log(String message);
}
```

12. Create code for the `foo.ALogger` architecture (again in the code subdirectory of the architecture directory).

File content:

```
package foo;
public class ALogger implements ILog {
    public void log( String message ) {
        System.out.println("LOG: " + message);
    }
}
```

### 13. Create code for the foo.ATester architecture.

File content:

```
package foo;
import org.objectweb.dsrp.sofa.SOFAClient;
import org.objectweb.dsrp.sofa.SOFALifecycle;
import org.objectweb.dsrp.sofa.SOFAThreadHelper;

public class ATester implements SOFALifecycle, Runnable, SOFAClient {

    boolean end = false;
    ILog logger = null;

    // Implements method from the SOFALifecycle interface.
    // Called during instantiation of the component.
    public void start() {
        Thread t = new Thread(this);
        t.start();
    }

    // Implements method from the SOFALifecycle interface.
    // Called as a notification that component will be stopped
    public void stopping() {
    }

    // Implements method from the SOFALifecycle interface.
    // Called during stopping of the component
    public void stop() {
        end = true;
    }

    // Thread of the component (java.lang.Runnable interface)
    public void run() {
        while (!end) {
            logger.log("Hello world!");
            try {
                Thread.sleep(5000);
            } catch (Exception e) {}
        }
    }

    // Implements method from the SOFAClient interface
    // Called during initialization of the component
    public void setRequired(String name, Object iface) {
        if (name.equals("log")) {
            if (iface instanceof ILog) {
                logger = (ILog) iface;
            }
        }
    }
}
```

```
}

```

**Note:**

Current SOFA 2 implementation requires that code of components has to implement SOFA 2 related interfaces. In future releases, this will be replaced by annotation-style development.

14. Compile the code of the interface type and both architectures and upload code bundles with the code. Call

```
cushion compile
cushion upload

```

15. Create an assembly description for the application (i.e. for the `foo.ALogDemo` architecture). Call

```
cushion assembly initial foo.Assm foo.ALogDemo

```

As the `foo.ALogDemo` architecture defines subcomponents also via their architectures, the generated ADL file does not require any changes and can be directly committed. Call

```
cushion commit foo.Assm

```

16. Create a deployment plan for executing the application (based on the assembly descriptor). Call

```
cushion deplplan initial foo.DeplPlanA foo.Assm

```

The generated deployment plan is again stored in the `adl.xml`. In the plan, fill in the names of the deployment docks, where the components should be launched.

```
<?xml version="1.0" encoding="UTF-8"?>
<depl-plan
name="org.objectweb.dsrg.sofa.examples.logdemo.deplplan.Local"
node="nodeA" >
  <depl-subc name="logger" node="nodeA" />
  <depl-subc name="tester" node="nodeA" />
</depl-plan>

```

This example puts all components to a single deployment dock named `nodeA`.

17. Using the deployment plan, deploy the application (i.e. generate connectors). Call

```
cushion deploy foo.DeplPlanA

```

18. Now, the application is ready to be launched.

First, launch the deployment dock registry, global connector manager (unless you have already run them) and the deployment dock with name `nodeA` (see [SOFA 2 usage](#) documentation how to launch these entites).

Then, you can launch the application (to find out the version identifier (which is automatically assigned by the repository) of the deployment plan, you can use

cushion status).

19. Also you can create other deployment plans with different assignment of components to deployment docks.