Jetspeed Cornerstone Sample Code

Jun Yang (junyang@cisco.com)

Emad Benjamin (ebenjami@cisco.com)

Service and Factory Samples

```
package org.apache.cornerstone.framework.demo.main;

import org.apache.log4j.PropertyConfigurator;
import org.apache.cornerstone.framework.api.context.IContext;
import org.apache.cornerstone.framework.api.service.IService;
import org.apache.cornerstone.framework.bean.visitor.BeanPrinter;
import org.apache.cornerstone.framework.context.BaseContext;
import org.apache.cornerstone.framework.init.Cornerstone;
import org.apache.cornerstone.framework.demo.bo.api.IA;
import org.apache.cornerstone.framework.demo.bo.api.IX;
import org.apache.cornerstone.framework.demo.bo.factory.api.IAFactory;
import org.apache.cornerstone.framework.demo.bo.factory.api.IXFactory;
import org.apache.cornerstone.framework.demo.service.DateService;

public class DemoMain
{
    public static final String REVISION = "$Revision: 1.13 $";

    public static void main(String[] args) throws Exception
    {
        // init log4j
        String log4jConfigFilePath = System.getProperty(
            "log4j.configuration",
            "log4j.properties"
        );
        PropertyConfigurator.configure(log4jConfigFilePath);

        // init Cornerstone
        Cornerstone.init();

        // ---------------------------------------------------------
        // Demo of calling services of the same class with different
        // configurations
        // ServiceManager looks into
        // ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
        //
cornerstone.service/cornerstone.demo.getDate.reg.properties
        // for the definition of this service
        String serviceName = "cornerstone.demo.getDate";
        IService service = Cornerstone.getServiceManager().
            createServiceByName(serviceName);
```

Cornerstone services reside under `implementation/cornerstone.service` in the registry and are just one particular type of service that Cornerstone framework supports. You can have your own services in a different place in the registry managed by a different service manager.

`cornerstone.service` is a short hand for `org.apache.cornerstone.framework.api.service.IService`. Short hands can have their own name spaces.

`cornerstone.demo.getDate` is the name of a Cornerstone service. Service names (which are regular implementation variant names) can also have name spaces.

`cornerstone.demo.getDate.reg.properties` has:

```
_.factory.className=
org.apache.cornerstone.framework.demo.
service.DateServiceFactory
```

which says this instance of `IService` is created by a factory whose class name is `DateServiceFactory` which creates an instance of `DateService`.

```
        // call passing no values in context
        // service will use its defaults
        IContext context = new BaseContext();
        String dateString = (String) service.invoke(context);
        printDate(serviceName, dateString, context);
```

`DateService` takes 2 arguments: *dateFormat* and *timeZone*. You can consider them both configuration and parameters (See *Service Configuration vs. Parameters* in the document *Jetspeed Cornerstone Concepts*). Configuration provides default values for parameters and parameters provide run-time overwrite of configuration. In this example we don't pass any parameters into the service. So the service will use their default values. Console shows:

```
cornerstone.demo.getDate (timeZone=null,
dateFormat=null):
    Monday, December 1, 2003
```

```
        // call passing value of one of invoke_direct_inputs
        context = new BaseContext();
        context.setValue(DateService.INPUT_TIME_ZONE, "GMT-0800");
        dateString = (String) service.invoke(context);
        printDate(serviceName, dateString, context);
```

This example provides one parameter. Console shows:

```
cornerstone.demo.getDate (timeZone=GMT-0800,
dateFormat=null):
    Sunday, November 30, 2003
```

```
        // call passing all values of invoke_direct_inputs
        context = new BaseContext();
        context.setValue(DateService.INPUT_TIME_ZONE, "GMT+0800");
        context.setValue(
            DateService.INPUT_DATE_FORMAT,
            DateService.DATE_FORMAT_SHORT
        );
        dateString = (String) service.invoke(context);
        printDate(serviceName, dateString, context);
```

This example provides both parameters. Console shows:

```
cornerstone.demo.getDate (timeZone=GMT+0800,
dateFormat=SHORT):
    12/1/03
```

```
        // call another instance of DateService which has different
        // configurations
        // ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
        //
cornerstone.service/cornerstone.demo.getDate2.reg.properties
        serviceName = "cornerstone.demo.getDate2";
        service = Cornerstone.getServiceManager().
            createServiceByName(serviceName);
        context = new BaseContext();
        dateString = (String) service.invoke(context);
        printDate(serviceName, dateString, context);
```

`cornerstone.demo.getDate2.reg.properties` has:

```
_.parent.name=
cornerstone.demo.getDate
dateFormatPattern=EEE, MMM d, yyyy
```

which says my parent is `getDate` (meaning I get all of its configuration) but I overwrite the configuration value `dateFormatPattern`. Console shows:

```
cornerstone.demo.getDate2 (timeZone=null,
dateFormat=null):
    Mon, Dec 1, 2003
```

```
        // call yet another instance of DateService which has different
        // configurations
        // ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
        //
cornerstone.service/cornerstone.demo.getDate3.reg.properties
        serviceName = "cornerstone.demo.getDate3";
        service = Cornerstone.getServiceManager().
            createServiceByName(serviceName);
        context = new BaseContext();
        dateString = (String) service.invoke(context);
        printDate(serviceName, dateString, context);
```

This example is similar to above but with a different value for `dateFormatPattern`. `cornerstone.demo.getDate2.reg.properties` has:

```
_.parent.name=
cornerstone.demo.getDate
dateFormatPattern=yyyy.MM.dd G
'at' HH:mm:ss z
```

Console shows:

```
cornerstone.demo.getDate3 (timeZone=null,
dateFormat=null):
    2003.12.01 AD at 04:07:46 GMT+00:00
```

```
// -------------------------------------
// Demo of calling services in a sequence

// ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
// cornerstone.service/cornerstone.demo.getDate1x1.reg.properties
// Notice getDate10? has overwriting _.invokeDirect.inputs
// and _.invokeDirect.output and "spread" the inputs and
// outputs around different names; otherwise the 3 getDate10?
// services will share the same inputs and output, whihc is
// not desirable.  This "spreading" is unnecessary if the
// services in the sequence are of different classes.

// name of service controller
serviceName = "cornerstone.demo.getDate1x1";
service = Cornerstone.getServiceManager().
    createServiceByName(serviceName);
context = new BaseContext();
context.setValue("tz102", "GMT-0800");
context.setValue("tz103", "GMT+0800");
context.setValue("df103", DateService.DATE_FORMAT_SHORT);
// s1 will use defaults for both dateFormat and timeZone
// s2 will use "tz102" passed in and default for dateFormat
// s3 will use "tz103" and "df103" passed in
String lastDateString = (String) service.invoke(context);
String date101 = (String) context.getValue("date101");
System.out.println("date101: '" + date101 + "'");
String date102 = (String) context.getValue("date102");
System.out.println("date102: '" + date102 + "'");
String date103 = (String) context.getValue("date103");
System.out.println("date103: '" + date103 + "'");
```

`cornerstone.demo.getDate1x1.reg.properties` has:

```
_.parent.name=
cornerstone.controller.sequence
sequence=s1,s2,s3
sequence.s1.parent.name=
cornerstone.demo.getDate101
sequence.s2.parent.name=
cornerstone.demo.getDate102
sequence.s3.parent.name=
cornerstone.demo.getDate103
```

which says my parent is `cornerstone.controller.sequence` which defines an instance of `SequenceServiceController`. It also says I call `s1`, `s2` and `s3` and in that sequence. Console shows:

```
date101: 'Monday, December 1, 2003'

date102: 'Mon, Dec 1, 2003'

date103: '2003.12.01 AD at 04:07:46 GMT'
```

```
// -------------------------------------------------
// Demo of indirect creation of implementations of an
// interface in various ways

// get the single implementation of an factory interface
// ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
//     ...IAFactory/.reg.properties
IAFactory aFactory = (IAFactory) Cornerstone.
    getImplementationManager().
        createImplementation(IAFactory.class);
IA a = (IA) aFactory.createInstance();
String aPrintString = BeanPrinter.getPrintString(a);
System.out.println("a=" + aPrintString);
```

Following are the demos of `ImplementationManager`.

`registry/implementation/org.apache.cornerstone.framework.demo.bo.factory.api.IAFactory/.reg.properties` (this is the single implementation of `IAFactory` since the variant name is empty) has:

```
_.instance.className=
org.apache.cornerstone.framework.demo.
bo.factory.AFactory
```

which says my instance's class name is `AFactory`. Console shows:

```
a=
{
    b:
    {
        q:200
    }
    ,p:"p"
}
```

```
// get the "a1_viaInstanceClassName" implementation variant
// of interface IA
// This variant defines how an instance should be created
// by using "instance.className".
// ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
//     ...IA/a1_viaInstanceClassName.reg.properties
IA a1_viaInstanceClassName = (IA) Cornerstone.
    getImplementationManager().createImplementation(
        IA.class,
        "a1_viaInstanceClassName"
    );
String a1_viaInstanceClassNamePrintString =
    BeanPrinter.getPrintString(a1_viaInstanceClassName);
System.out.println(
    "a1_viaInstanceClassName=" +
    a1_viaInstanceClassNamePrintString
);
```

`registry/implementation/org.apache.cornerstone.framework.demo.bo.api.IA/a1_viaInstanceClassName.reg.properties` has:

```
_.instance.className=
org.apache.cornerstone.framework.
demo.bo.A1
```

Console shows:

```
a1_viaInstanceClassName=
{
    b:null
    ,p:"p"
}
```

Notice class `A1` doesn't know about the `B` side and thus property *b* is not populated.

```java
        // get the "a1_viaFactoryClassName" implementation variant
        // of interface IA
        // This variant defines how an instance should be created
        // by using "factory.className".
        // ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
        //      ...IA/a1_viaFactoryClassName.reg.properties
        IA a1_viaFactoryClassName = (IA) Cornerstone.
            getImplementationManager().createImplementation(
                IA.class,
                "a1_viaFactoryClassName"
            );
        String a1_viaFactoryClassNamePrintString =
            BeanPrinter.getPrintString(a1_viaFactoryClassName);
        System.out.println(
            "a1_viaFactoryClassName=" +
            a1_viaFactoryClassNamePrintString
        );
```

`registry/implementation/org.apache.cornerstone.framework.demo.bo.api.IA/a1_viaFactoryClassName.reg.properties` has:

```
_.factory.className=
org.apache.cornerstone.framework.
demo.bo.factory.AFactory
```

Console shows:

```
a1_viaFactoryClassName=
{
    b:
    {
        q:200
    }
    ,p:"p"
}
```

Notice class `AFactory` does know about the `B` side and thus property *b* is populated.

```java
        // get the "a1_viaParentName" implementation variant of
        // interface IA
        // This variant doesn't specify either instance.className
        // or factory.className but gets that
        // from its parent (another implementation for the same
        // interface) specified by "parent.name".
        // ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
        //      ...IA/a1_viaParentName.reg.properties
        IA a1_viaParentName = (IA) Cornerstone.
            getImplementationManager().createImplementation(
                IA.class,
                "a1_viaParentName"
            );
        String a1_viaParentNamePrintString =
            BeanPrinter.getPrintString(a1_viaParentName);
        System.out.println(
            "a1_viaParentName=" +
            a1_viaParentNamePrintString
        );
```

`registry/implementation/org.apache.cornerstone.framework.demo.bo.api.IA/a1_viaParentName.reg.properties` has:

```
_.parent.name=a1_viaInstanceClassName
```

which says I am just like `a1_viaInstanceClassName`. Console shows:

```
a1_viaParentName=
{
    b:null
    ,p:"p"
}
```

```
// ----------------------
// Demo of an IoC Factory

// First notice the demo.bo.api and demo.bo packages are
// completely independent of any framework

// get the implementation variant "x1y1" of factory interface
// IXFactory and create an instance
// the instance of X1 will be associated with an instance of Y1
// ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
//     ...IXFactory/x1y1.reg.properties
IXFactory xFactory = (IXFactory) Cornerstone.
    getImplementationManager().createImplementation(
        IXFactory.class,
        "x1y1"
    );
IX x1y1 = (IX) xFactory.createInstance();
String x1y1PrintString = BeanPrinter.getPrintString(x1y1);
System.out.println("x1y1=" + x1y1PrintString);
```

`registry/implementation/org.apache.`

`cornerstone.framework.demo.bo.`

`factory.api.IXFactory/x1y1.reg.properties` has:

```
_.instance.className=
org.apache.cornerstone.framework.demo.
bo.factory.XFactory
product.instance.className=
org.apache.cornerstone.framework.
demo.bo.X1
product.property.y.instance.className=
org.apache.cornerstone.framework.
demo.bo.Y1
```

which says I am an instance of `XFactory`; I create products of class `X1` whose property `y` is an instance of `Y1`. Console shows:

```
x1y1=
{
    y:
    {
        q:1000
    }
    ,p:"x1y1"
}
```

Notice `Y1`'s constructor sets property `q` to *1000*.

```
// get the implementation variant "x1y2" of factory interface
// IXFactory and create an instance
// the instance of X1 will be associated with an instance of Y2
// ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
//     ...IXFactory/x1y2.reg.properties
xFactory = (IXFactory) Cornerstone.
    getImplementationManager().createImplementation(
        IXFactory.class,
        "x1y2"
    );
IX x1y2 = (IX) xFactory.createInstance();
String x1y2PrintString = BeanPrinter.getPrintString(x1y2);
System.out.println("x1y2=" + x1y2PrintString);
```

`registry/implementation/org.apache.`

`cornerstone.framework.demo.bo.`

`factory.api.IXFactory/x1y2.reg.properties` has:

```
_.instance.className=
org.apache.cornerstone.framework.demo.
bo.factory.XFactory
product.instance.className=
org.apache.cornerstone.framework.
demo.bo.X1
product.property.y.factory.className=
org.apache.cornerstone.framework.
demo.bo.factory.Y2Factory
```

which says I am an instance of `XFactory`; I create products of class `X1` whose property `y` is created by a factory whose class is `Y2Factory`. Console shows:

```
x1y2=

{

    y:

    {

        q:2000

    }

    ,p:"x1y2"

}
```

Notice `Y2Factory`'s sets property `q` to *2000*.

```
// get the implementation variant "x1y3" of factory interface
// IXFactory and create an instance
// the instance of X1 will be associated with an instance of Y3
// ${CORNERSTONE_RUNTIME_HOME}/registry/implementation/
//     ...IXFactory/x1y3.reg.properties
xFactory = (IXFactory) Cornerstone.
    getImplementationManager().createImplementation(
        IXFactory.class,
        "x1y3"
    );
IX x1y3 = (IX) xFactory.createInstance();
String x1y3PrintString = BeanPrinter.getPrintString(x1y3);
System.out.println("x1y3=" + x1y3PrintString);
```

`registry/implementation/org.apache.`

`cornerstone.framework.demo.bo.`

`factory.api.IXFactory/x1y3.reg.properties` has:

```
_.instance.className=
org.apache.cornerstone.framework.demo.
bo.factory.XFactory
product.instance.className=
org.apache.cornerstone.framework.
demo.bo.X1
product.property.y.parent.name=y3
product.property.p.value=x1y3
```

which says I am an instance of `XFactory`; I create products of class `X1` whose property `y` is created by whatever is defined as `Y3` whose interface is obtained by Cornerstone via reflection on `IX`. The value of property `p` of `X1` is *x1y3*. Console shows:

```
x1y3=
{
    y:
    {
        q:3000
    }
    ,p:"x1y3"
}
```

Notice `y3` sets property `q` to *3000*.
`registry/implementation/org.apache.`

`cornerstone.framework.demo.bo.api.IY/`

`y3.reg.properties` has:

```
_.instance.className=
org.apache.cornerstone.framework.
demo.bo.Y3
```

```
    }

    protected static void printDate(
        String serviceName,
        String dateString,
        IContext context
    )
    {
        String timeZoneName = DateService.INPUT_TIME_ZONE;
        String timeZoneValue = (String) context.getValue(timeZoneName);
        String dateFormatName = DateService.INPUT_DATE_FORMAT;
        String dateFormatValue = (String) context.getValue
(dateFormatName);

        System.out.println(
            serviceName +
            " (" + timeZoneName + "=" + timeZoneValue + ", " +
            dateFormatName + "=" + dateFormatValue + "):\n" +
            "    " + dateString
        );
    }
}
```

MVC Samples

To be added.

Persistence Samples

Schema

```
create table test_user
(
        id int identity,
        login_name varchar,
        first_name varchar,
        last_name varchar
);
go

create table test_group
(
        id int identity,
        name varchar
);
go

create table test_user_group
(
        id int identity,
        user_id int,
        group_id int,
        foreign key (user_id) references test_user(id),
        foreign key (group_id) references test_group(id)
);
go
```

Data

```
ID      LOGIN_NAME      FIRST_NAME      LAST_NAME
101     dilbert         Dilbert         Funny
102     outm            Out             of Mind
201     pointy          Pointy          Hair
202     outt            Out             of Touch
301     userd           User            Dumb
302     userp           User            Picky

ID      NAME
100     engineers
200     managers
300     users

ID      USER_ID GROUP_ID
1       101     100
2       102     100
3       201     200
4       202     200
5       301     300
6       302     300
```

DemoPersistence

Why our own persistence layer? Our goal is to make the common cases as easy as it should be and push that common cases boundary as far as possible. For more complex cases, we will support other OR mapping technologies.

```java
package org.apache.cornerstone.framework.demo.main;

import java.util.List;

import org.apache.cornerstone.framework.api.factory.CreationException;
import org.apache.cornerstone.framework.api.factory.IFactory;
import org.apache.cornerstone.framework.api.implementation.
ImplementationException;
import org.apache.cornerstone.framework.bean.visitor.BeanPrinter;
import org.apache.cornerstone.framework.demo.bo.api.IGroup;
import org.apache.cornerstone.framework.demo.bo.api.IUser;
import org.apache.cornerstone.framework.init.Cornerstone;
import org.apache.cornerstone.framework.init.InitException;
import org.apache.log4j.PropertyConfigurator;

public class DemoPersistence
{
    public static final String REVISION = "$Revision: 1.6 $";

    public static void main(String[] args)
        throws InitException, ImplementationException, CreationException
    {
        // init log4j
        String log4jConfigFilePath = System.getProperty(
            "log4j.configuration",
            "log4j.properties"
        );
        PropertyConfigurator.configure(log4jConfigFilePath);

        Cornerstone.init();
```

`registry/implementation/cornerstone.dataSource/default.reg.properties` specifies the default system data source and has:

```
_.parent.name=hsqldb-standalone
```

which say I am actually the same as another data source named `hsqldb-standalone`. So if we change the parent name to say `oracle1` here in this one file, all default data source references are redirected to `oracle1`.

`registry/implementation/cornerstone.dataSource/hsqldb-standalone.reg.properties` has:

```
_.instance.className=
org.apache.cornerstone.framework.
persistence.datasource.BaseDataSource
_.instance.isSingleton=true
driver.className=org.hsqldb.jdbcDriver
connection.url=
jdbc:hsqldb:./hsqldb/data/test
connection.userName=sa
connection.password=
```

This shows an example of a singleton per virtual class (`hsqldb-standalone`). In the whole system, there is one instance of `hsqldb-standalone` but many instance of `BaseDataSource`.

```
IFactory groupFactory = (IFactory) Cornerstone.
    getImplementationManager().createImplementation(
        IFactory.class,
        "cornerstone.demo.groupFactory"
    );
IGroup group = (IGroup) groupFactory.createInstance(
    new Integer(100)
);
System.out.println("group=" + BeanPrinter.getPrintString(group));
```

`registry/implementation/cornerstone.factory/cornerstone.demo.groupFactory` has:

```
_.instance.className=
org.apache.cornerstone.framework.
persistence.factory.
BasePersistentObjectFactory
dataSource.name=default
product.instance.className=
org.apache.cornerstone.framework.
demo.bo.BaseGroup
primaryKey.propertyName=id
primaryKey.columnName=id
query.byId=
select * from test_group where id = ?
db.columnToPropertyMap.id=id
db.columnToPropertyMap.name=name
```

which say I am an instance of `BasePersistentObjectFactory`; I use the data source called `default`; I create instances of `BaseGroup`; my product's primary key property name is `id`; my product's primary key column name is `id`; the query `byId` is so and so; the column `id` is mapped to property `id` and `name` to `name`. Console shows:

```
group=
{
    userList:null
    ,name:"engineers"
    ,id:100
}
```

```
IFactory userFactory = (IFactory) Cornerstone.
    getImplementationManager().createImplementation(
        IFactory.class,
        "cornerstone.demo.userFactory"
    );
IUser user = (IUser) userFactory.createInstance(
    new Integer(101)
);
System.out.println("user=" + BeanPrinter.getPrintString(user));
```

`registry/implementation/cornerstone.factory/cornerstone.demo.userFactory` has:

```
_.instance.className=
org.apache.cornerstone.framework.
persistence.factory.
BasePersistentObjectFactory
dataSource.name=default
product.instance.className=
org.apache.cornerstone.framework.
demo.bo.BaseUser
primaryKey.propertyName=id
primaryKey.columnName=id
query.byId=
select * from test_user where id = ?
db.columnToPropertyMap.id=id
db.columnToPropertyMap.login_name=
loginName
db.columnToPropertyMap.first_name=
firstName
db.columnToPropertyMap.last_name=
lastName
```

This shows an example of more column name to property name mappings. Console shows:

```
user=
{
    loginName:"dilbert"
    ,id:101
    ,firstName:"Dilbert"
    ,lastName:"Funny"
}
```

```
        IFactory userListFactory = (IFactory) Cornerstone.
            getImplementationManager().createImplementation(
                IFactory.class,
                "cornerstone.demo.userListFactory"
            );
        List userList = (List) userListFactory.createInstance();
        System.out.println(
            "userList=" + BeanPrinter.getPrintString(userList)
        );
    }
}
```

`registry/implementation/cornerstone.factory/cornerstone.demo.userListFactory` has:

```
_.instance.className=
org.apache.cornerstone.framework.
persistence.factory.
BasePersistentObjectListFactory
dataSource.name=default
query.all=select * from test_user
query.byGroup=
select tu.* from test_user tu,
test_user_group tug where tug.user_id =
tu.id and tug.group_id = ?
query.byGroup.parameters=groupId
query.default=all
element.parent.name=
cornerstone.demo.userFactory
```

which says I am an instance of `BasePersistentObjectListFactory`; I use the data source called `default`; I support 2 queries: `all` and `byGroup` (whose parameter list is *groupId*); my elements are created using the factory called `userFactory` (shown in previous example). Console shows:

```
userList=
[
{
    loginName:"dilbert"
    ,id:101
    ,firstName:"Dilbert"
    ,lastName:"Funny"
},
{
    loginName:"outm"
    ,id:102
    ,firstName:"Out"
    ,lastName:"of Mind"
},
{
    loginName:"pointy"
    ,id:201
```

Notice in the above examples, we didn't need to create any actual factory classes. We just created new configurations of Cornerstone factories `BasePersistentObjectFactory` and `BasePersistentObjectListFactory` which are subclasses of `InversionOfControlFactory`.

Support for auto-population of associations is not complete yet. The idea is group 100's user list will just be created by the `userListFactory` with the right parameter (group id *100*) passed into its `byGroup` query.

Change History

Revision	**Date**	**Changes**
0.2	12/02/2003	Added persistence sample.
0.1	11/30/2003	Created.