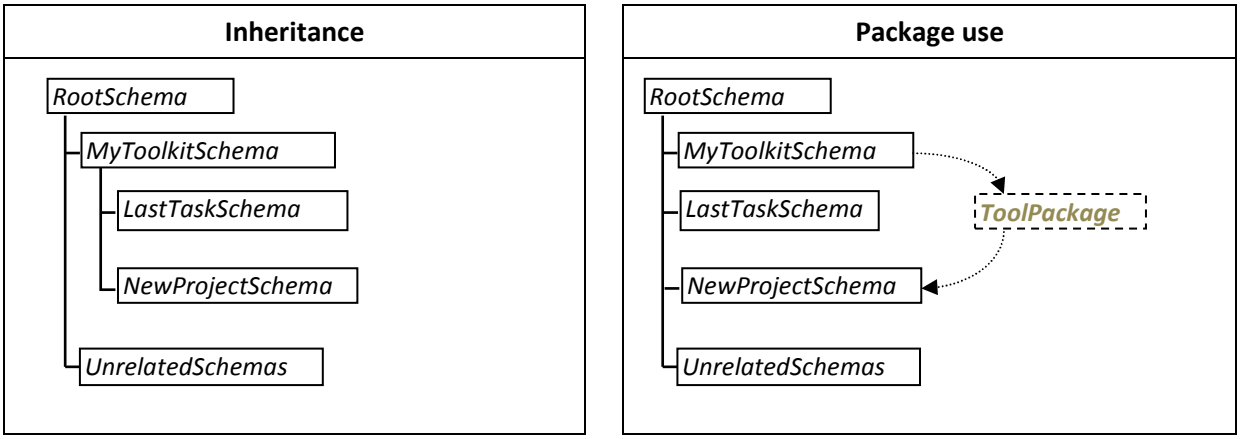
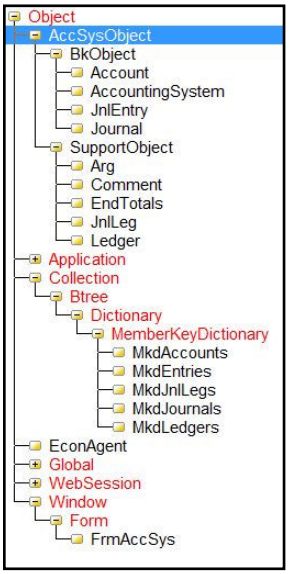
Designing and using packages in Jade

A package is a pre-written unit of software. Designed as a schema, it was then adapted (using the **Schema | Browse | Packages | Export Browser** tool) to be a package that can be imported into other schemas. Such a package delivers a cluster of classes and methods that a Jade developer would rather borrow than create. Without the package facility that goal could only be attained (a) by defining the classes and methods in a superschema[1] or (b) by using Selective extraction then a Schema Load.

- In the left diagram below we see *MyToolKitSchema*, in which we define the shareable clevernesses we have created. Because it is a superschema of *LastProjectSchema* and *NewProjectSchema* both inherit everything in it.
- In the right diagram we see that, as an alternative, a package called *ToolPackage* has been defined based upon *MyToolkitSchema* [2]. Thereafter any schema, anywhere in the repository, may import from *ToolPackage*.



The examples that spring to mind are packages that supply, say, an entire Accounting System, a Payroll package, a Taxation module, a Timetabling or Rostering system or a large tool for database or workflow management.



Designing a package

We shall illustrate the process with two schemas:

a) the schema *DemoAccSys* is the prototype—a complete, free-standing system for simple double entry accounting and reporting. *DemoAccSys* can be found on the CD in the \Demos\ folder. From this, a package called *AccountingEngine* will now be derived.

b) We shall then create the schema *CornerGrocer* . This new schema needs an accounting system as a 'back end' to do its accounting. So we shall import *AccountingEngine* into it.

DemoAccSys has the class structure shown at right. For tidiness and convenience in package-making, all the essential classes of an accounting system were defined under *AccSysObject* and *MemberKeyDictionary*. The small class called *EconAgent* represents the organisation which 'owns' the

[1] But some tedious editing may be needed to harmonise the new classes with classes of the target schema.

[2] *MyToolKitSchema* remains in the repository. *ToolPackage* is an *interface* to it, not a schema in its own right.

accounting system. *DemoAccSys* contains a demonstration user interface. It is entirely within *FrmAccSys*. To display accounting capabilities that form has four buttons, each of which runsan automatic demonstration for a separate business.

- Try it. Load and run *DemoAccSys*.

Test the source schema fully. It is a pain to create a package and find that the original was faulty.

Planning *DemoAccSys* to facilitate creating and using the *AccountingEngine* package.

Our *CornerGrocer* example will gain classes and methods by importing *AccountingEngine*, especially the class called *AccountingSystem.* Our *CornerGrocer* example needs only a single instance of an *Accounting System*. But we must be able to provide for a large host like *CharteredAcctgPractice*, which needs to manage a large collection of separate *AccountingSystems*.

We have designed DemoAccSys so that when we separate out the package from it the sole point of contact between the AccountingEngine package and a host (*CornerGrocer* or *CharteredAcctg-Practice*) will be an instance of the *EconAgent* class. Whatever the host schema may do—grocery; chartered accountancy, heavy engineering—the only point of contact with the package will be an instance of *EconAgent* called ce (for 'current EconAgent'). And ce.myAcs will be a reference to its *AccountingSystem*.[3]

Note that the host and the package will each have an *Application* subclass: *CornerGrocer*, say, for the host and *Accountant* for the package. Each may have an .initialize method, and the package's .initialize will run just after the host's. [4]

A schema does not need an *EconAgent* until it acquires an accounting package. So if the initialize method of the package finds that there is no instance of *EconAgent* extant it should create one. The constructor method of *EconAgent* will create a blank *AccountingSystem* and set EconAgent::myAcs to reference it.

Our plan is to

1) Export out of *DemoAccSys* a package we shall call *AccountingEngine*.
2) Create a new schema called *CornerGrocer*, which needs an accounting package
3) Import *AccountingEngine* into *CornerGrocer*.
4) Add a simple form class into *CornerGrocer* to communicate accounting facts with that package.

1) Exporting to a package

DemoAccSys has the functionality we want. We shall export that functionality as *AccountingEngine*.

1) Select *DemoAccSy*s in the Schema Browser.
2) Click **Browse | Packages | Export Browser**

---

[3] And though *CharteredAcctgPractice* might have a large collection, .allEconAgents, still each ce in allEconAgents has its own ce.acs, its own instance of AccountingSystem.

[4] This means that, within one complete schema, the classes sourced from *CornerGrocer* have an app variable referencing CornerGrocer:Application and the classes sourced from the package have an app variable referencing an Accountant: Application. We have not met this before. Take care if it is your practice to use app to hold relatively global properties or methods. It is for this reason that *DemoAccSys* uses not app but an instance of *EconAgent* as a common holder for such features.

3) Click **Packages | Add** and name the added package *AccountingEngine*.

4) Select an/the Application.

5) Describe the package in the text box. E.g. "A simple tool for double-entry accounting and reporting."[5] Click Next.

6) Select a working set of classes—*AccountingEngine* needs the *AccSysObject* class and all its subclasses, and the the *EconAgent* class.

   Select those two classes and click **>>**. Click **Next**.

7) In the sheet called Select Classes and Features for Package one is able to select individual properties of classes. We need them all—so select all classes and click **>>**.[6]
   Click **Next**.

8) Select interfaces for package. We have none to select. Click **Next**.

9) Change attributes? No, we wish to change none. Click **Finish**.

   Since the classes we have selected depend upon certain other classes (particularly collection classes) we are asked if we want to add them to the package.
   Click **Yes**. Then click **Finish** again.

   The export is complete. We see *AccountingEngine* listed in the **Browse Packages** screen. Note that *AccountingEngine* does not list in the Schema Browser. It is derived from a schema but is not itself a schema.

2) Create a new schema, *CornerGrocer*, and import the *AccountingEngine*

(Alternatively you might choose an existing schema to use the package.)

1) Select *RootSchema*. Use **Schema | Add**. We called ours *CornerGrocer*.

2) Load the *AccountingEngine* into it as follows:

   a) Click **Browse | Packages |Import Browser**

   b) **Packages | Add**. We see the Import Package dialog.
      Drop down the list and select *DemoAccSys:: AccountingEngine*. For information, check the Details checkBox. Click **OK**.

      The CornerGrocer Import Packages Browser confirms that *AccountingEngine* has been imported.

3) Return to *CornerGrocer's* Class Browser. The two classes in green, with their subclasses, are the imported ones.

3) Enhance *CornerGrocer*, to link to the classes and methods of the package

CornerGrocer needs to call upon the package. So we need to implement an interface. If you trace the working of the original, undivided *DemoAccSys* you will find that the .load method of its main form…

i. creates one instance, ac, of the *AccountingSystem* class,
ii. sets app.ac := ac; and app.fm := self;

---

[5] More than one package may be exported out of a given schema.

[6] In the right pane the labels C, M and P mean Class, Method and Property.

iii. And thereafter all the action is directed by the *FrmAccSys* form.

For simplicity of demonstration that form has four **Test** buttons. Each registers a fresh business—an *EconAgent*—with the accounting package and plays a script of accounting transactions and events.

The commands available are described in DemoAccSys.doc and you may create methods in your schemas which poke commands into the *AccountingEngine* in the same way as *DemoAccSys* does.

For speed and simplicity, if not realism, we will copy the *FrmAccSys* form across into *CornerGrocer*. We can do this by Selective Extract and Load, as described in the text.

a) Extracting the *FrmAccSys* form out of *DemoAccSys*

1) Select *DemoAccSys* again.
2) Click S**chema | Extract**.
3) Browse to a folder in which to put the extract and give it a memorable name, such as myForm.scm. **Save**. (But do not click **OK**.) Do the same for the .ddb file. **Save**.
4) Instead select ⊙ Selective extract.
5) On the Selective tab choose the *FrmAccSys* class then [>]. Choose the the *String* type[7] and then [>]. (The package will supply everything else.)
   Note: Of the options called **O** Classes and **O** Method the one you need is **O** Classes. This will extract both class and methods; **O** Method extracts methods only.
6) On the Applications tab choose **O** None. The target schema already has its own Application class.
7) On the Schema Options tab choose **O** None systematically.
8) Finally click **OK**. The extract is saved.

> Note: For a *single* class, like *FrmAccSys*, the even faster way to do a selective extract is to right-click on it in the Class Browser and click on Extract! It will be saved with a .cls extension.

b) Loading the *FrmAccSys* form into CornerGrocer

1) Return to the *CornerGrocer* schema.
2) Click **Schema | Load** and browse to where you stored the extract (e.g. MyForm.scm and .ddb.
3) Click on **Advanced** and set the Target schema to be *CornerGrocer*. **OK** and **OK**.
4) A **ReOrg** will almost certainly be required.
5) In the Class Browser you can see that *FrmAccSys* is now part of your schema.
6) Almost certainly a few method code errors are reported. Fix them.
   I. Enter the **Applications Browser**.
   II. Select **Change**. Set *FrmAccSys* as the Start-up form.
   III. To the *Application* sub-class, *CornerGrocer*, add two references
   fm : FrmAccSys and
   acs : properties.

---

[7] Because in *DemoAccSys* a method, rj( ), was added (in the Primitives Browser) to right-justify characters in a fild of specified width.

4) Test the system

1) Run the *CornerGrocer* application. The **Test** buttons should work, as in the original *DemoAccSys*. The difference is that you are now calling on the *AccountingEngine* package to perform the work.

2) Note: The package is really a path through to the original *DemoAccSys*. So that schema must remain in the same repository as any schemas which use the package.

Package Tip

The easiest way to maintain a package exported using the Export Package Class Browser is to open a Class Browser for the schema from which the package was exported and position the Export Package Class Browser next to it. Then simply drag and drop the classes and features you want to include in your exported package from the Class Browser to the Export Package Class Browser and change the class and property attributes as required.