Getting Started with Protege-Frames

Author : Eliza Sachs
Current Contact : Jennifer Vendetti (vendetti at stanford dot edu)
Last Modified : June 6, 2006

Table Of Contents

Understanding ontologies ... 4
What is an ontology? ... 4
Why create an ontology? ... 4
How do I create an ontology? ... 4
How do I know it's right? ... 5
Where do I start? ... 5
Developing an example ... 6
Creating and saving a project ... 7
Creating the sample project ... 7
Naming and saving the sample project ... 9
Creating classes ... 14
Creating the Columnist class ... 14
Creating the Author class ... 17
Creating subclasses of Author ... 18
Modifying the hierarchy ... 19
Making a class Abstract ... 21
Creating the Employee class ... 22
Adding an additional superclass to an existing class ... 23
Adding a superclass using drag-and-drop ... 26
Creating slots ... 28
Creating a slot at the Slots Tab ... 28
Assigning a slot to a class ... 31
Creating a slot in the Classes Tab ... 34
Slots and inheritance ... 37
Creating slot facets ... 39
Defining slot facets for salary ... 39
Creating a relationship using slots ... 42
Entering instances ... 46
Setting the display slot ... 50
Creating a relationship between instances ... 51

Customizing a form 54
Resizing a widget 54
Moving a widget 56
Customizing widget buttons 58
Hiding a widget 61
Displaying a hidden widget 63
Using the default layout 64
Creating and saving a query 66
Creating a query 66
Running a query 68
Saving a query 69
Retrieving a query 70

Understanding ontologies

This tutorial gives an introduction to Protege-Frames, an extensible, platform-independent environment for creating and editing ontologies and knowledge bases which allows users to begin designing an ontology quickly and intuitively. You will learn how to create, modify, and save a Protege-Frames project. You will create a project called "tutorial," which contains a few of the classes and slots from the newspaper example that accompanies the Protege-Frames installation. Once you have read the tutorial, you will be ready to explore Protege-Frames on your own.

Not all Protege users will be familiar with ontology design and development. Therefore, in addition to introducing you to the Protege-Frames interface, this tutorial gives a quick overview of some of the basic concepts of ontology design and draws heavily on the concepts described in the Ontology Development 101 paper. If you are familiar with ontology design, you can skip directly to Creating and Saving a Project. For further documentation, tutorials, papers, and presentations, see our documentation page on the Protege Web site.

- What is an ontology?
- Why create an ontology?
- How do I create an ontology?
- How do I know it's right?
- Where do I start?

What is an ontology?

An ontology describes basic concepts in a domain and defines relations among them. Basic building blocks of ontology design include:

- classes or concepts
- properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties))
- restrictions on slots (facets (sometimes called role restrictions))

An ontology together with a set of individual instances of classes constitutes a *knowledge base*.

Why create an ontology?

An ontology provides a common vocabulary for researchers who need to share information in the domain. Some of the reasons to create an ontology are:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from operational knowledge
- To analyze domain knowledge

How do I create an ontology?

There is no one correct methodology for developing ontologies, nor is there a single correct result. Developing an ontology is usually an iterative process. You start with a rough first pass at the ontology. You then revise and refine the evolving ontology and fill in the details.

In practical terms, developing an ontology includes:

1. defining classes in the ontology
2. arranging the classes in a subclass-superclass hierarchy
3. defining slots and describing allowed values for these slots
4. filling in the values for slots for instances

How do I know it's right?

There are many possible ontologies for any given domain; any particular ontology is just one way of structuring the concepts and relations. There are a few simple principles which can help you make design decisions in many cases:

- There is no one correct way to model a domain - there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate.
- Ontology development is necessarily an iterative process.
- Concepts in the ontology should be close to objects (physical or logical) and relationships in your domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe your domain.

Where do I start?

You might start by determining what you are going to use the ontology for, and how detailed or general the ontology is going to be. Among several viable alternatives, you will want to determine which one would work better for the projected task, be more intuitive, more extensible, and more maintainable. Remember that an ontology is a model of a real domain in the world and the concepts in the ontology must reflect this reality.

After you define an initial version of the ontology, you can evaluate and debug it by using it in applications or problem-solving methods or by discussing it with experts in the field. As a result, you will almost certainly need to revise the initial ontology. This process of iterative design will likely continue through the entire lifecycle of the ontology.

Developing an Example

Suppose we want to develop a system that helps manage the costs and organization of a newspaper. The "examples/newspaper" subfolder of the Protege installation directory contains a completed Protege-Frames project, **newspaper**, which provides one possible ontology for this domain. Some of the questions we want to answer are:

- Who is responsible for each section of the newspaper?
- What is the content of each article in a section, and who is its author?
- Who does each author report to?
- What is the layout and cost of each section?

Once we have an idea of what we want to cover, we can list some of the important terms we need. These can include basic concepts, properties they might have, or relationships between them. To begin with we can just collect the terms without regard to the role they might play in the ontology.

In the newspaper example we have *sections*. Each section contains *content* such as *articles* and *advertisements* and has a *editor* who is *responsible for* the section. Each article has an *author*, and that author may or may not be an *employee*. For each employee, we want to know his or her *name* and *salary*, and who he or she *reports to*.

As we continue to generate terms, we are implicitly defining the scope of our ontology, by what we finally decide to include and what to exclude. For example, upon initial examination of the term *employee*, we might want to add *janitor* or *delivery truck driver*. However, on reflection, we might realize that we want our ontology to focus on the costs directly associated with the content generation and presentation of the newspaper. Therefore, we would decide not to include *janitor* as a term of interest.

When we have a fairly complete list, we can start to categorize the different terms according to their function in the ontology. Concepts that are objects, such as *article* or *author*, are likely to be best represented by **classes**. Properties of the classes, such as *content* or *salary*, can be represented by **slots**, and restrictions on properties or relationships between classes and or slots, are represented by **slot facets**.

We will now dive in and show how to use the Protege-Frames interface to create and structure these components of the ontology.

Creating and saving a project

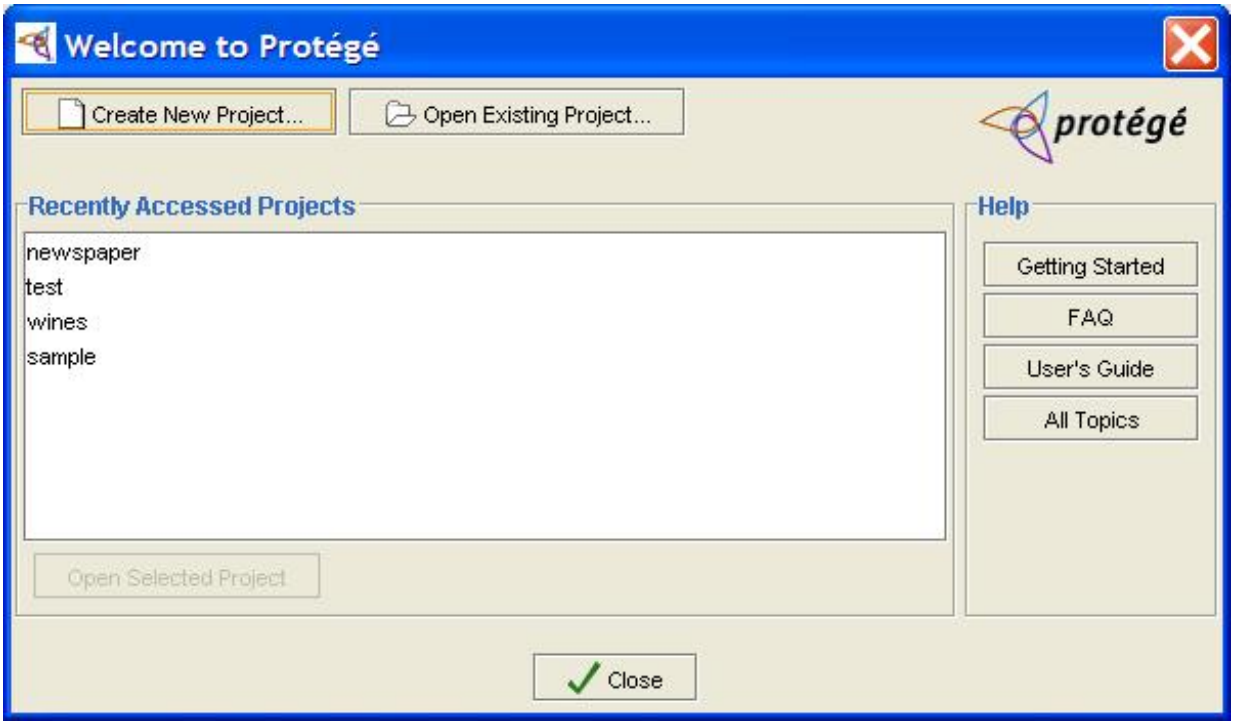
First you must create a new project in Protege-Frames. A more detailed explanation of the Protege-Frames knowledge model is available in the Overview section of the Protege Web site.

- Creating the sample project
- Naming and saving the sample project

Creating the sample project

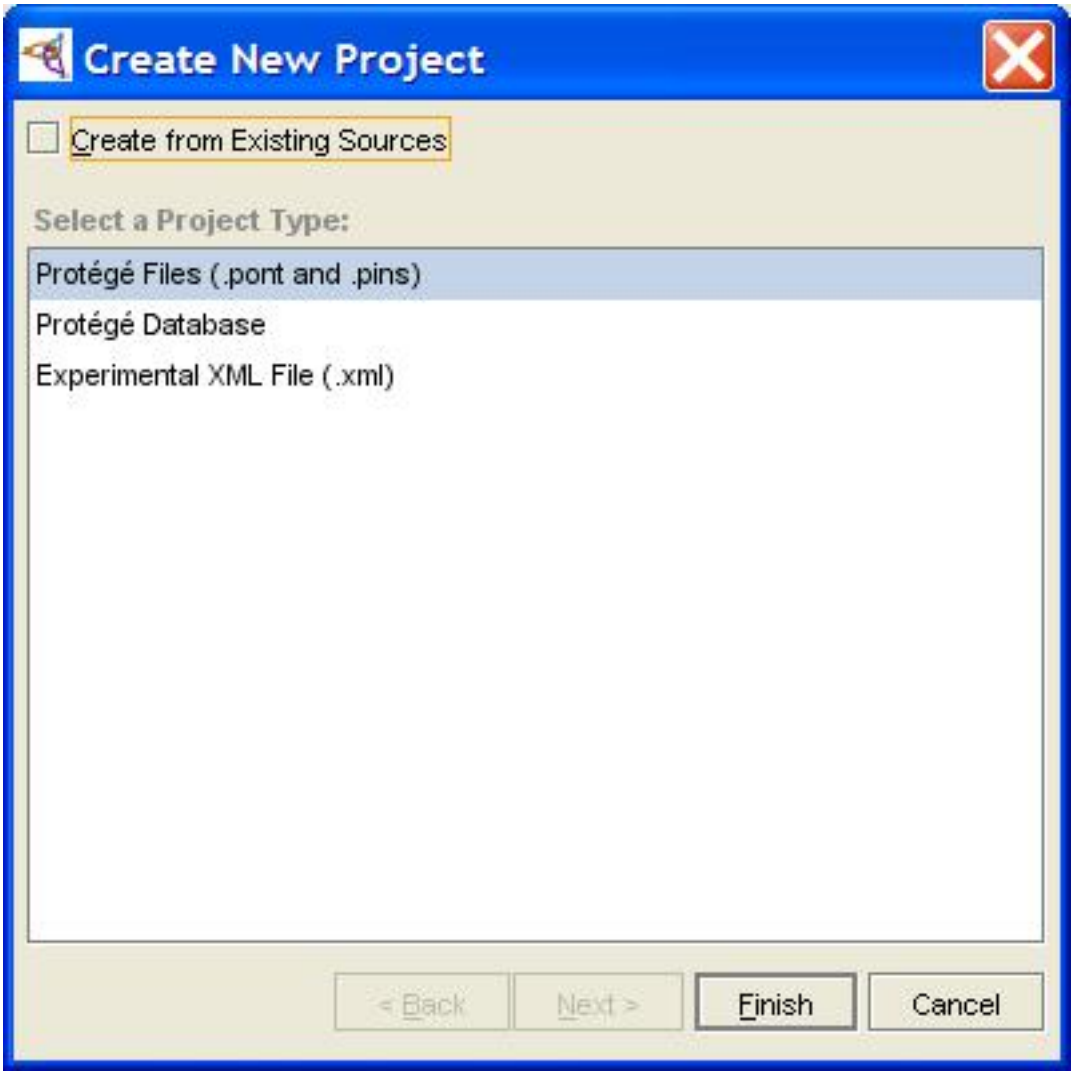
To create the sample project when you start Protege:

1. Start Protege. If you have a Protege project already open, simply exit and restart the program. When you start Protege, the Welcome to Protege dialog box prompts you to create a new project, open a recent project, or get help.

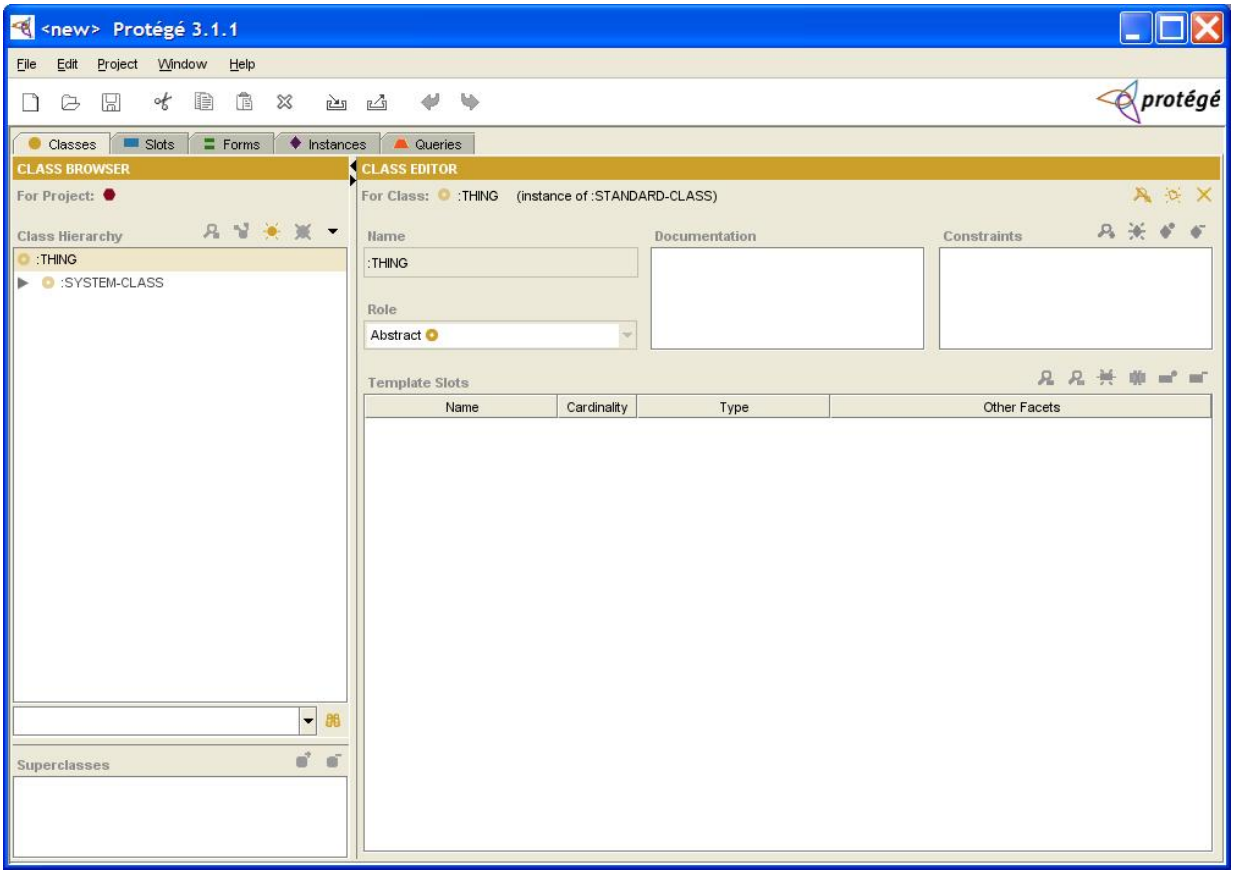


Note: To explore the examples included with Protege-Frames, click **Open Existing Project...** and select the project you wish to view. To create a new project later, select **File | New Project...**.

2. Click **Create New Project...**. A "Create New Project" dialog will open, allowing you to choose a Project Type. Unless you have a need for a special format for your files, just click **Finish** to accept the default: "Protege Files (.pont and pins)".



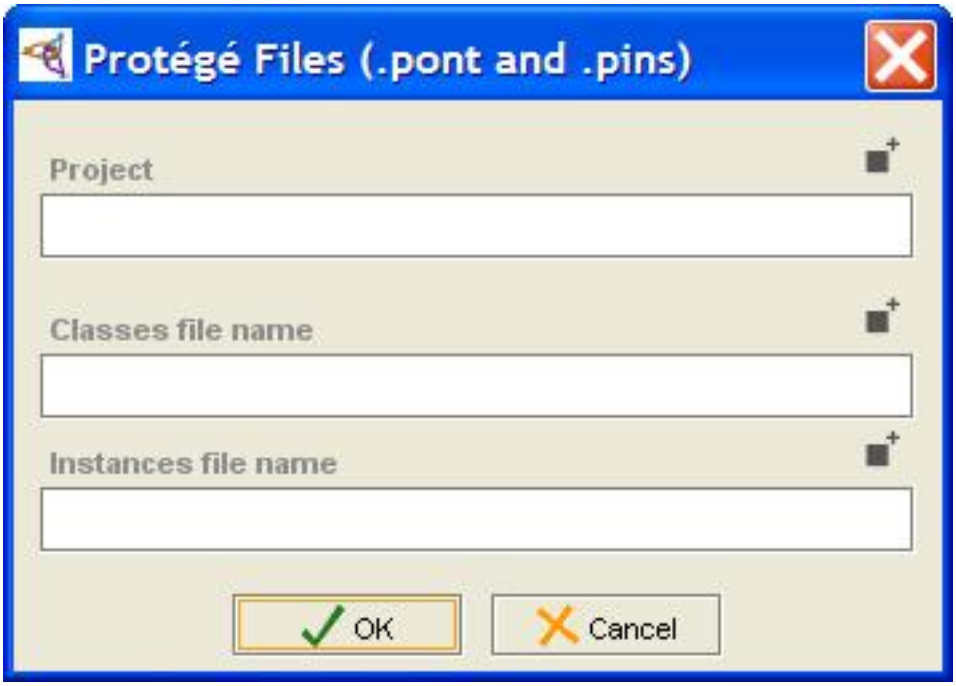
3. The Protege window opens and the standard tabs become visible. A new project always opens at the Classes view. The internal Protege system classes :THING and :SYSTEM-CLASS are all that is visible. No instances will be created.



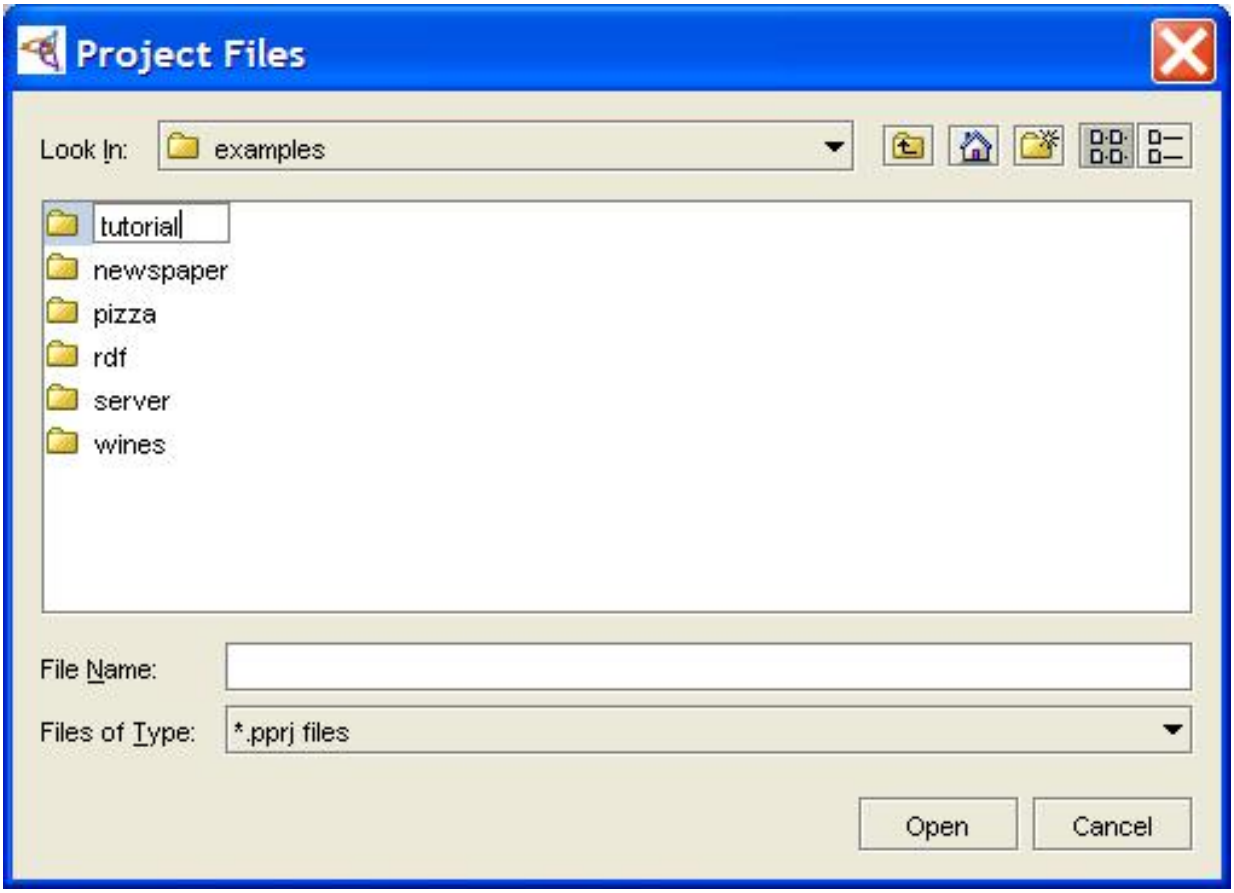
Naming and saving the sample project

Although the tutorial is not very long, you may find that you want to exit the project before you have finished. If you save and name it now, you will be ready to exit whenever is convenient. To save the project:

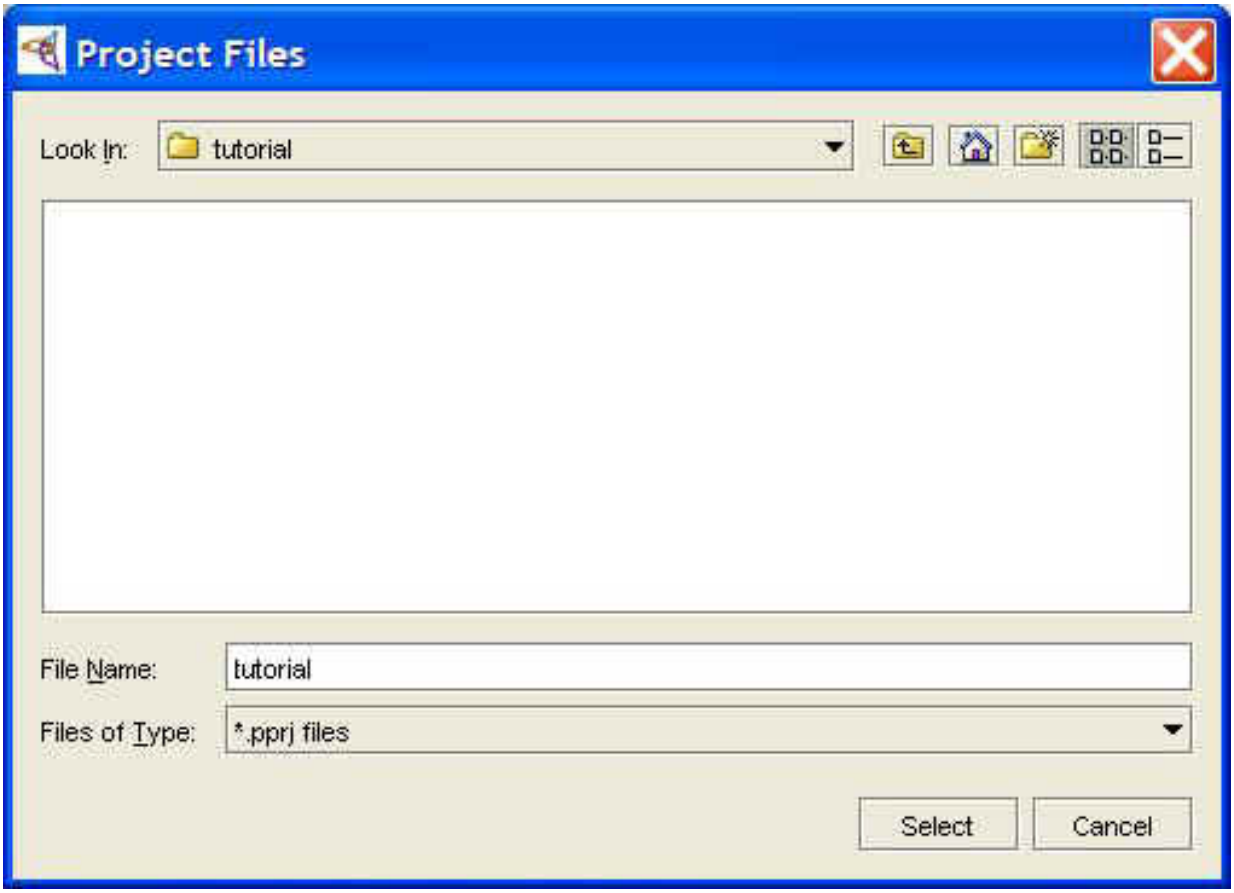
1. Click the Save Project button (third from the left at the top of the Protege window). You can also choose Save Project from the File menu. The **Protege Files** dialog box will open.



2. To choose the location where you want the project to be saved, click the top button at the right of the Project line. In the Project Files dialog, navigate to (and/or create) a directory.

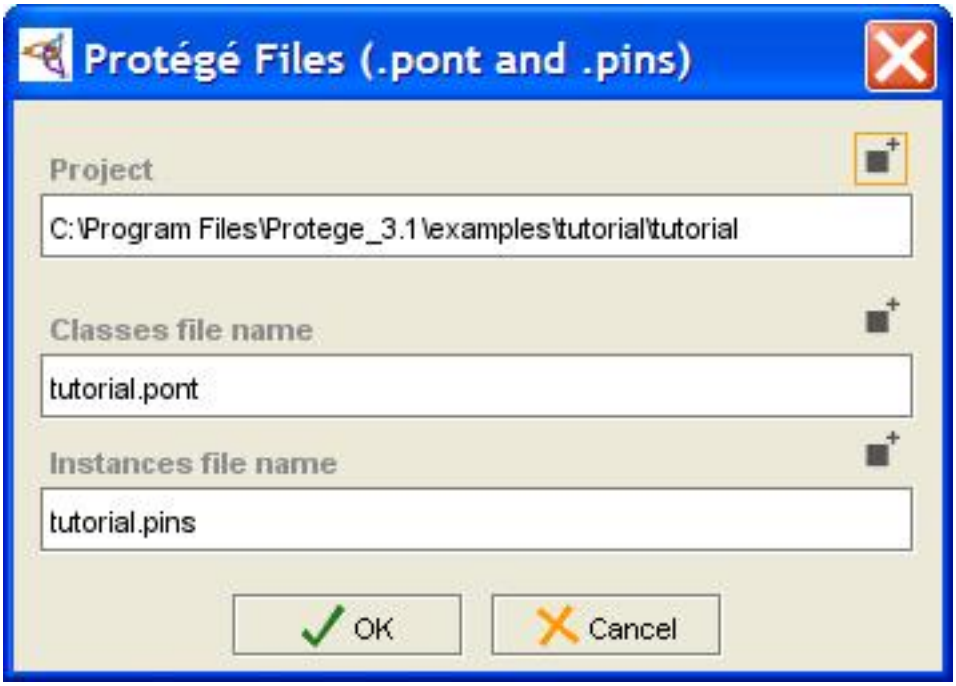


3. Enter a File Name (e.g., "tutorial").



4. Click on Select to proceed.

5. You will be returned to the Protege Files dialog box. Click OK to save the files and exit the dialog.



Note: You can also choose a location by typing the full path name in the **Project** line. The names of the other files will be filled in automatically.

Creating Classes

The Protege-Frames main window consists of tabs that display the knowledge model's various aspects. The most important tab when you start a project is the Classes tab. Usually classes will correspond to objects, or types of objects, in the domain. In the newspaper example, classes include people, such as editors, reporters, and salespeople, components of the newspaper layout, such as sections, and newspaper content, such as advertisements and articles.

Classes in Protege-Frames are shown in an inheritance hierarchy, displayed in the Class Browser in the left part of the Classes tab. The properties of the class currently selected in the tree are displayed in the Class Editor to the right.

In this section, you will learn how to create classes, subclasses, modify the class hierarchy, make classes abstract, and add additional superclasses to existing classes.

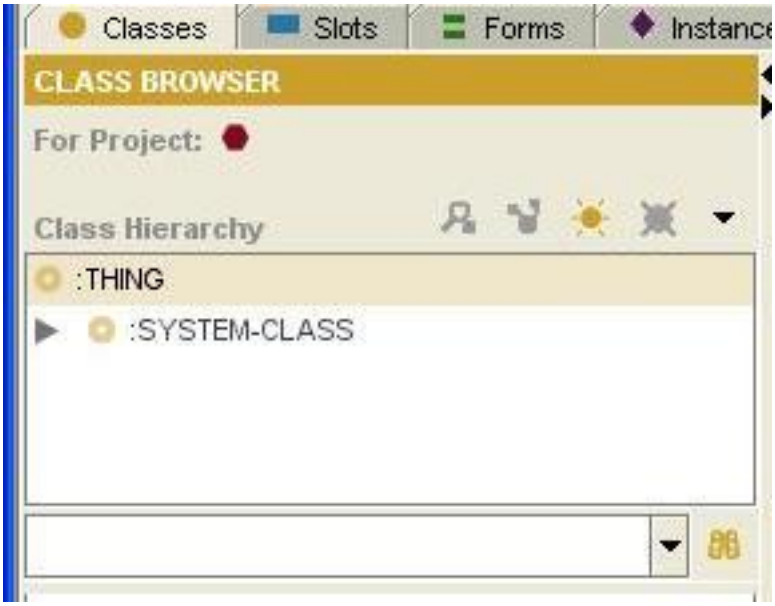
- Creating the Columnist class
- Creating the Author class
- Creating subclasses of Author
- Modifying the hierarchy
- Making a class Abstract
- Creating the Employee class
- Adding an additional superclass to an existing class
- Adding a superclass using drag-and-drop

Creating the Columnist class

We want to track the source of each article, so we will start by creating the different types of people and services that originate articles. First we will create a new class called **Columnist**:

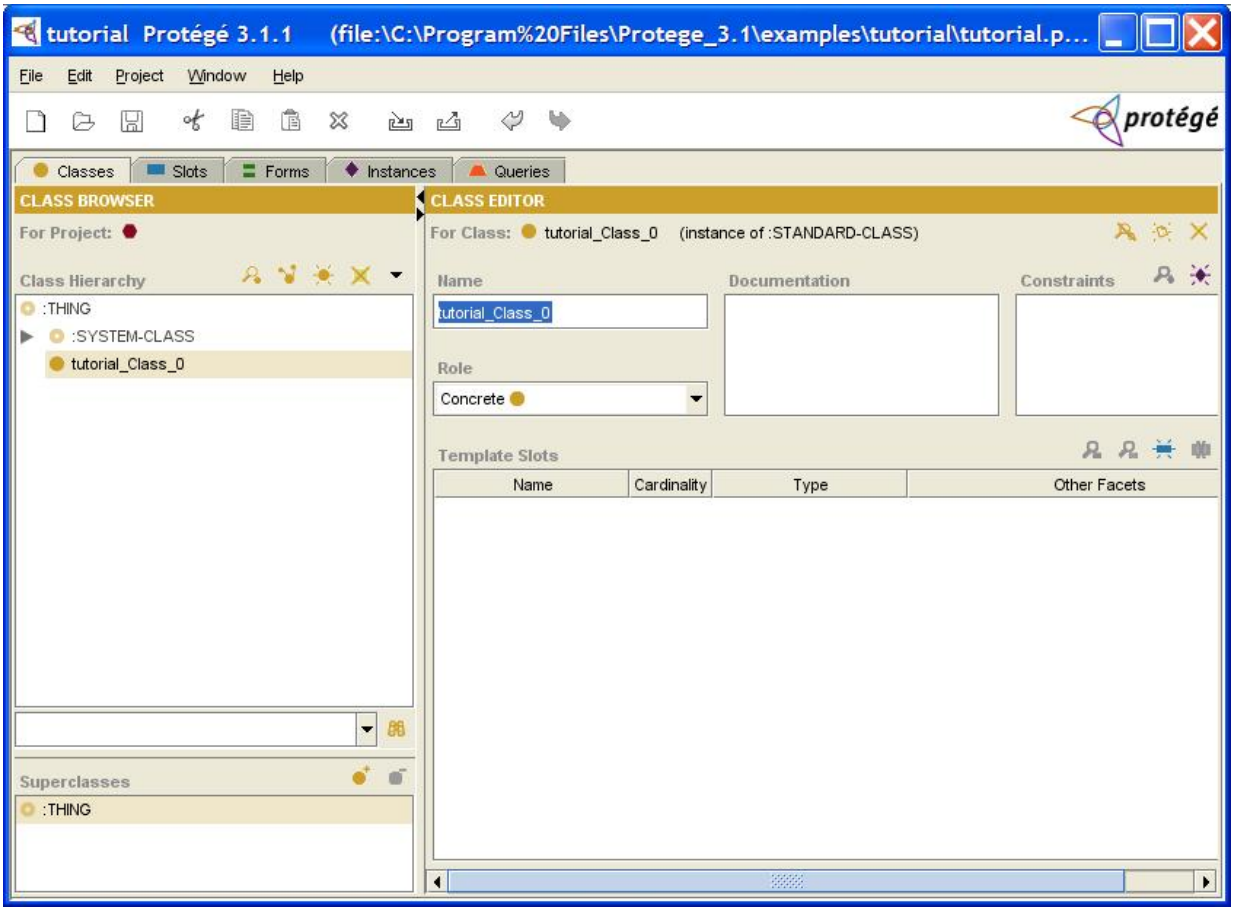
1. Select the **Classes Tab**.

2. Locate the Class Hierarchy area in the Class Browser pane at the left of the Protege window. This area displays the hierarchy of classes, with the current class highlighted.



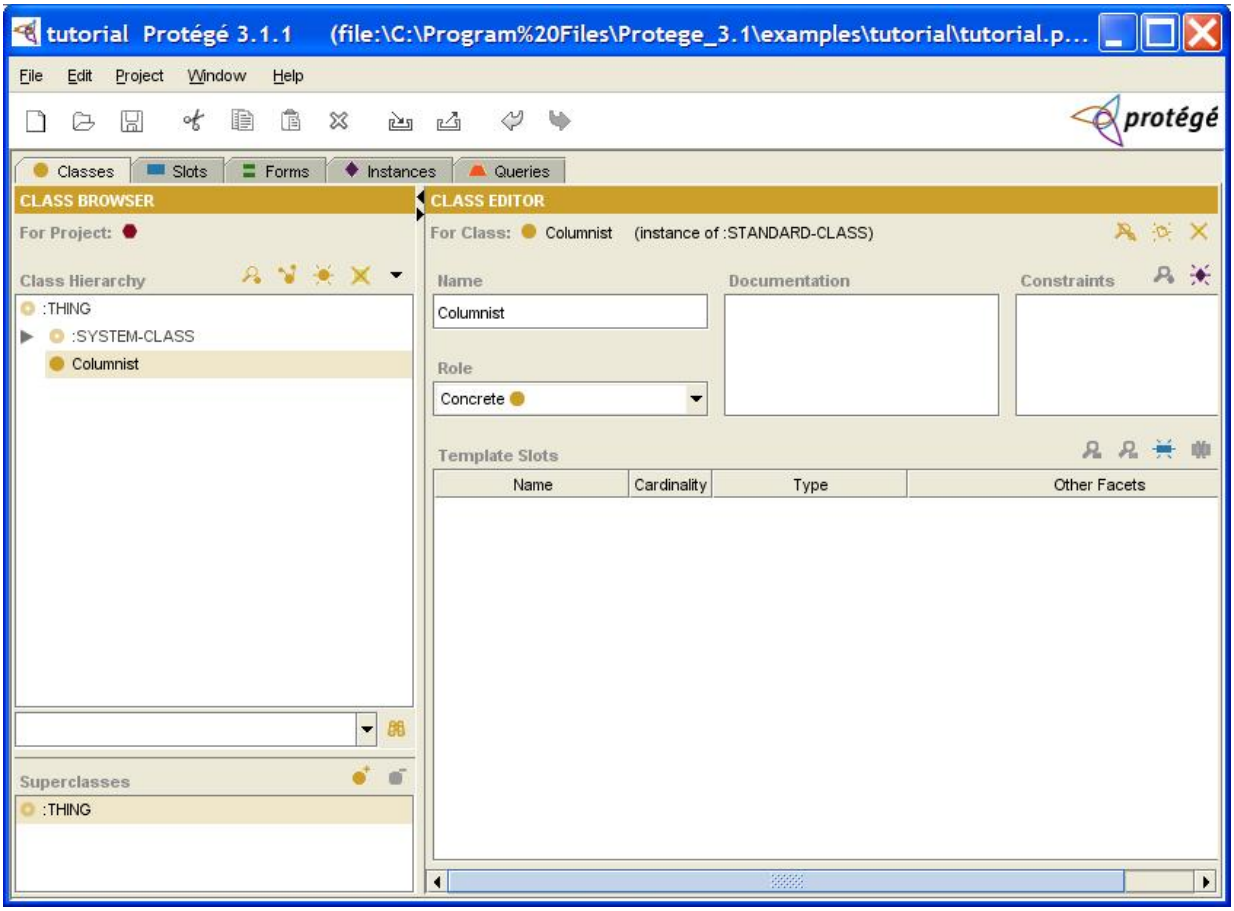
3. Make sure that class **:THING** is highlighted. Almost all the classes you create will be created subordinate to :THING. The class :SYSTEM_CLASS is used by Protege-Frames for defining the structure of various Protege forms.

4. Click the **Create Class** button at the top right of the Class Browser. A class is created with a generic name based on the name of the project, such as "tutorial_Class_0". You can see the name in the Class Browser and it is highlighted to show it is selected.



5. Type **Columnist**. A recommended Protege-Frames convention is to make the first character of each word in a class name uppercase and the rest lowercase, and to separate words with an underscore.

6. Hit **Enter** or click again on the highlighted class to complete and display your change.



If you have trouble renaming a class, look at the Class Editor pane at the right of the Protege-Frames window. The generic name of the class you just created should be displayed and highlighted in the Name field. If the correct generic name is displayed, but not highlighted, simply double-click in the Name field to edit the name. If the incorrect name is displayed, then the wrong class is selected in the Class Hierarchy area. Click on the class you want to rename.

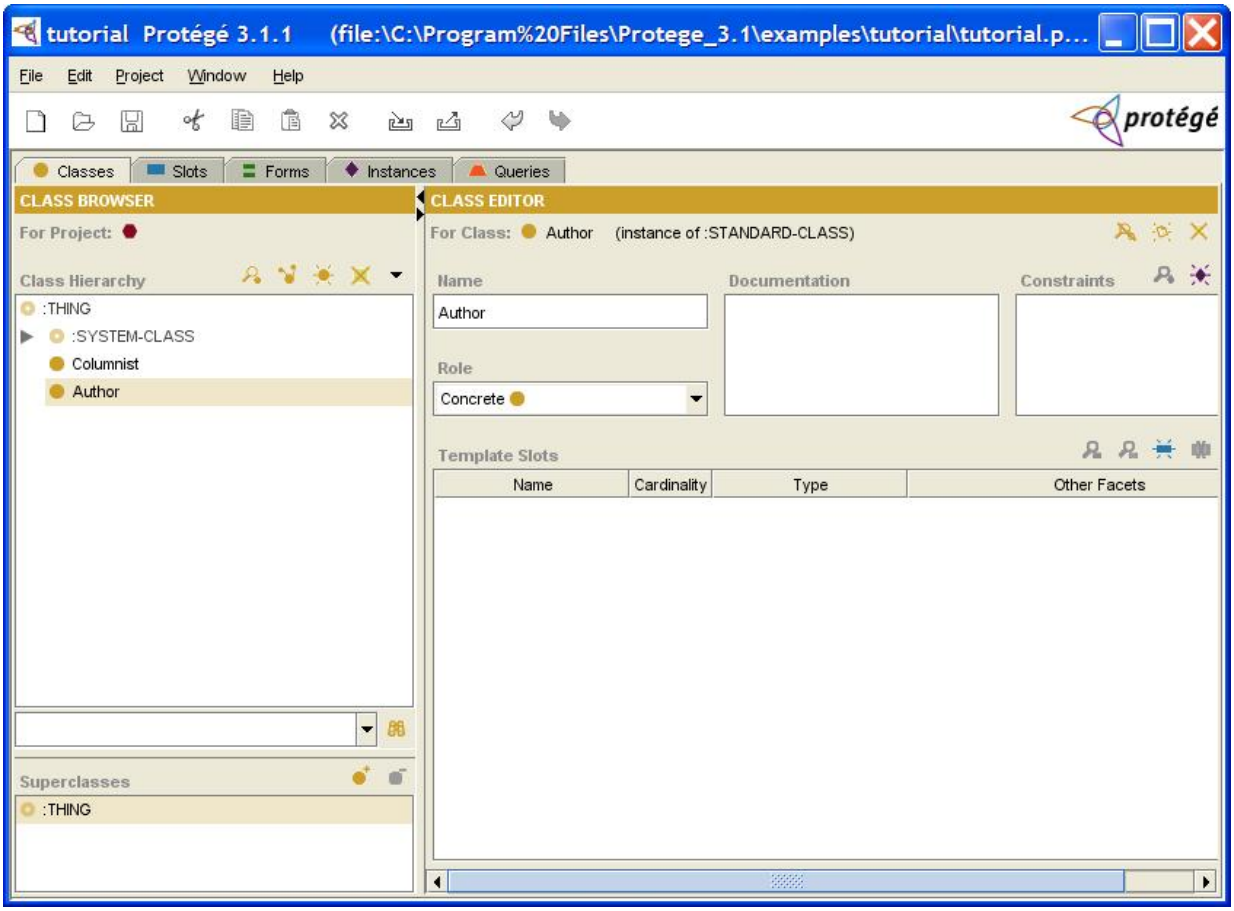
Creating the Author class

An **Author** is any possible source of an article, such as a news service or columnist. To create the **Author** class:

1. Highlight the class :THING. If you do not do this, you will create a class that is a subclass of **Columnist**.

2. Click the **Create Class** button and type **Author**.

3. Hit Enter to complete and display your change.



Note: If you accidentally create a subclass of Columnist, you forgot to select :THING before you created the class. You can delete and recreate the Author class by clicking the Delete Class button (fourth from the left, at the top of the Class Hierarchy area) to delete the class, and then clicking on :THING and creating another class. You can also rearrange the class hierarchy by dragging the class you created over :THING and then releasing the mouse button.

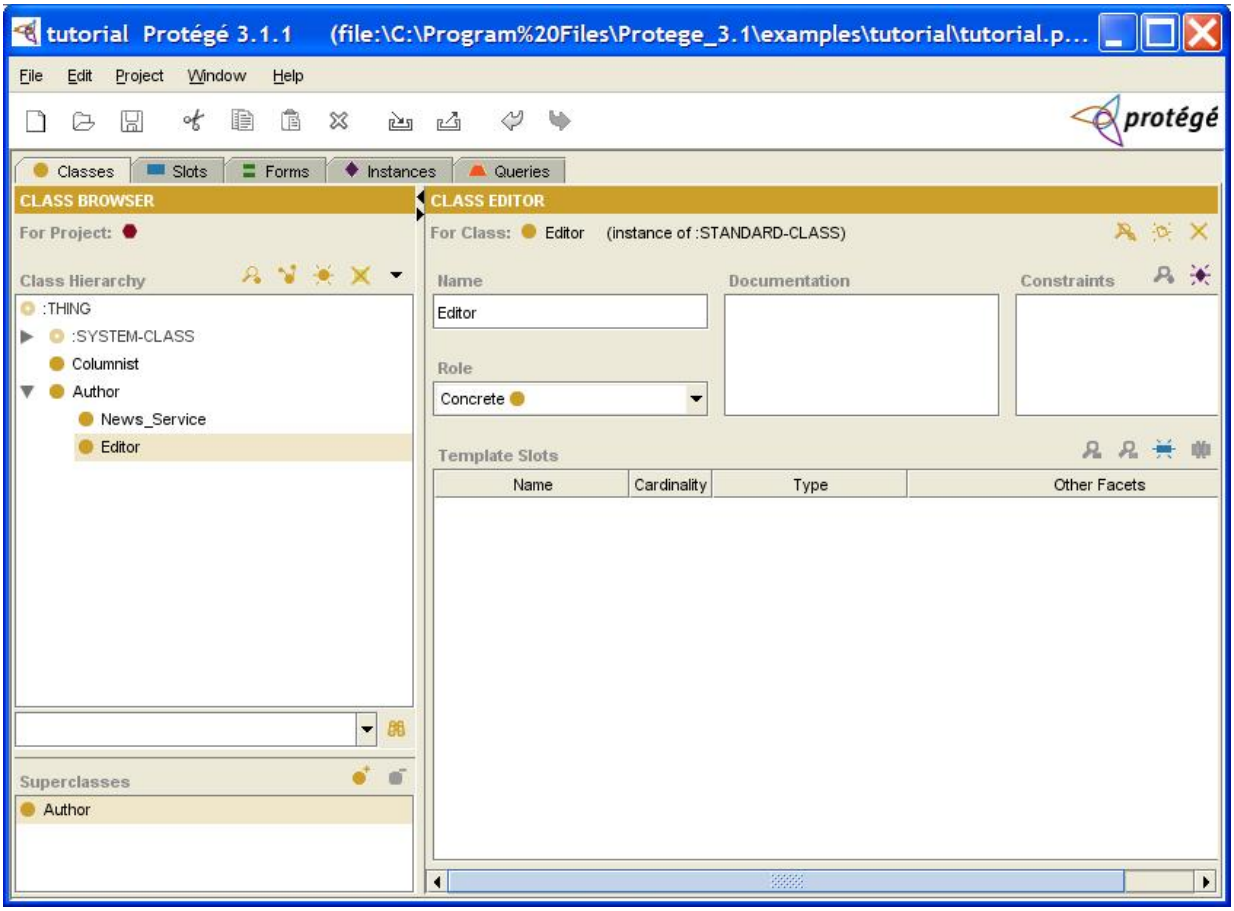
Creating subclasses of Author

Now we want to add two more sources of articles (**News_Service** and **Editor**), which we will create as subclasses of **Author**.

1. Select **Author** in the Class Hierarchy pane.

2. Click the **Create Class** button and rename the new class **News_Service**. Whenever you create a new class, it is created as a subclass of the currently highlighted class. Also notice that when you create the first subclass of a class, a or icon appears to its left. You can use this icon to display or hide the subclasses of a class. So that we can develop the ontology in later sections, we will create another subclass of **Author**.

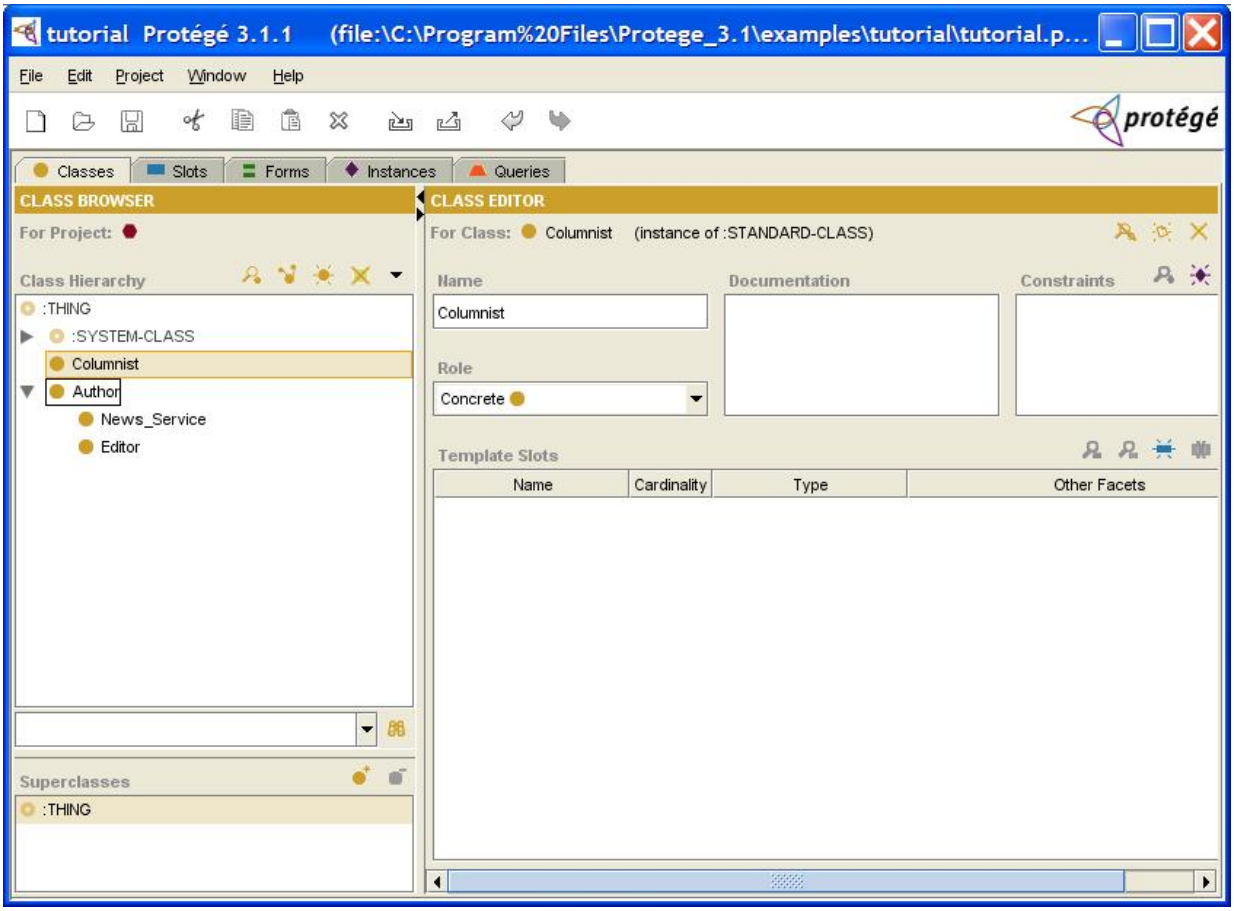
3. Select **Author** in the Class Browser.
4. Click the **Create Class** button and rename the new class **Editor**.



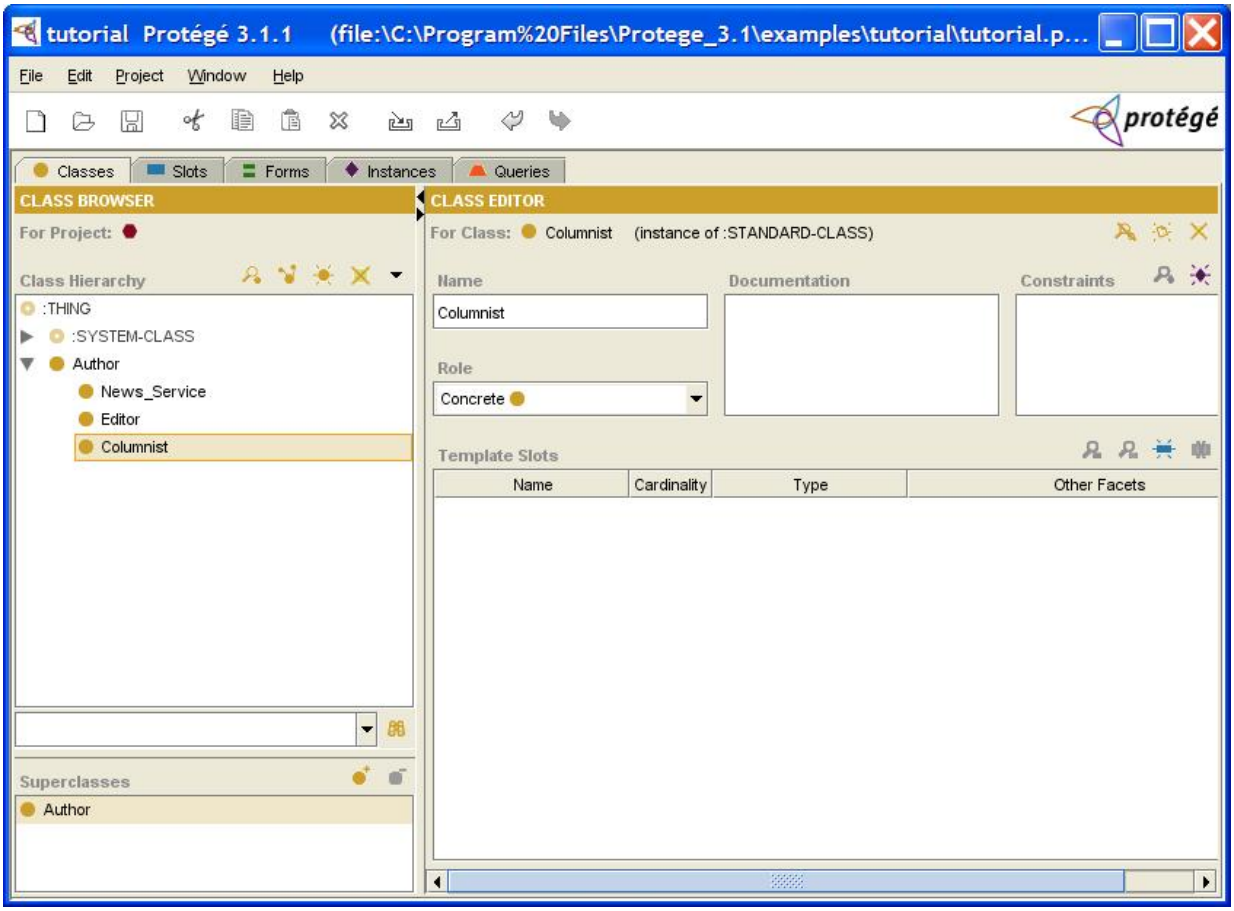
Modifying the hierarchy

At this point, you will see that **Author** and **Columnist** are at the same level (*siblings*) in the hierarchy, while **News_Service** and **Editor** are subclasses of **Author**. Conceptually, however, **News_Service**, **Editor**, and **Columnist** are all at the same level of generality, while **Author** is a more general concept. This goes against a principle of good ontology design: *all the siblings in a hierarchy (except for the ones at the root) should be at the same level of generality*. Therefore, we want to modify our hierarchy to make **Columnist** a subclass of **Author**, which will more accurately reflect the underlying structure. To modify the hierarchy by making **Columnist** a subclass of **Author**:

1. Click on **Columnist** and drag it over the **Author** class (**Author** will be outlined), then release.



2. **Columnist** is moved and now appears as a subclass of **Author**.

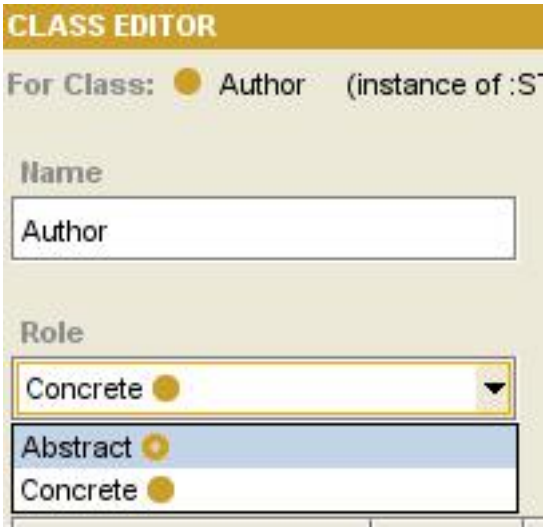


In this case, the mismatch may have been obvious. However, when you create your own ontologies, the development process itself can reveal differences or parallels that were not clear at the beginning. Therefore it is likely as you go along that you will rearrange the hierarchy, as well as create and delete classes, to better model the concepts you are trying to capture.

Making a class Abstract

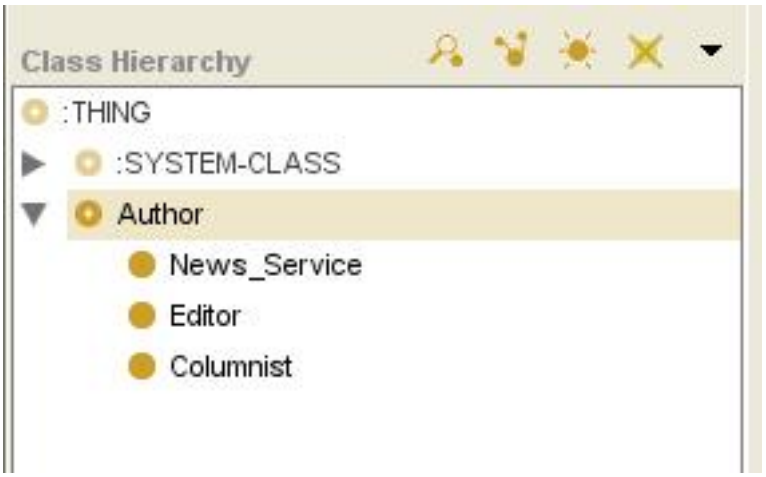
A class in Protege-Frames can be Concrete, meaning it can have direct instances, or it can be Abstract, which means that while it appears in the class hierarchy, it has no direct instances. When you first create a class, by default it is Concrete. Now that we know a little more about the **Author** class, we realize any instance of **Author** will actually be an instance of a particular type of author, such as a news service or columnist. Therefore, we will change the **Author** class to be Abstract. To change the **Author** class to be an Abstract class:

1. Select **Author** in the Class Hierarchy. In the Class Editor to the right of the Class Browser, locate the **Role** menu, just below the name of the class.



2. Click on the **Role** menu and select **Abstract**.

3. Notice that when you change the class role, the icon in front of Author changes. This new icon indicates an abstract class in the hierarchy. A solid icon indicates a concrete class.



Creating the Employee class

Now we will to create another class, **Employee**. An **Employee** is any employee of the newspaper, whether or not they are an **Author**. Note, however, that we are only interested in employees who are somehow involved in generating and managing newspaper content. One of the choices you make in designing an ontology is what to leave out. While you want to make sure that you include all the concepts that are relevant, you don't want to complicate the hierarchy by adding classes that are superfluous. Therefore, while janitors may be technically employees of the newspaper, we would *not* be interested in creating a subclass called Janitor.

1. Select :THING in the Class Browser. Although some authors are employees, we do not want **Employee** to be a subclass of **Author**. *In most cases, you should select :THING to create a*

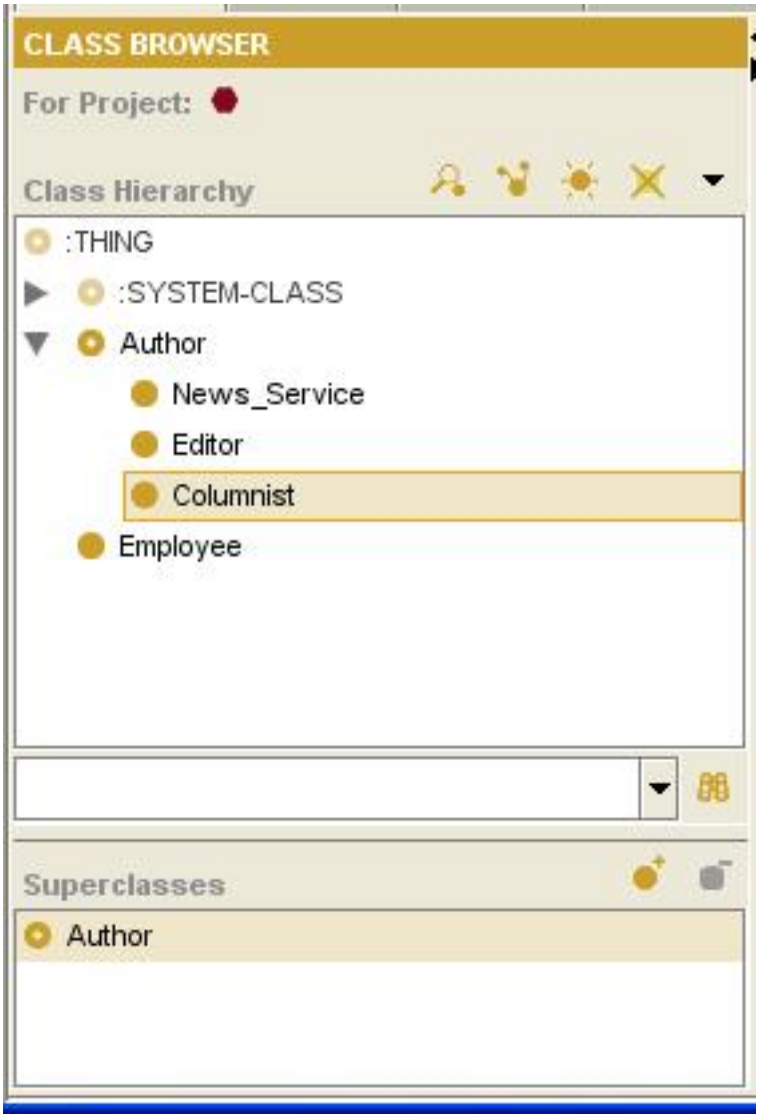
top-level class in the hierarchy.

2. Click the **Create Class** button and rename the class **Employee**.

Adding an additional superclass to an existing class

As mentioned above, we want a **Columnist** to be an **Employee**. Since we have already created the **Columnist** class, we do not create want to create it again. Instead, we make the existing **Columnist** class a subclass of **Employee** as follows:

1. Select **Columnist** in the Class Browser.

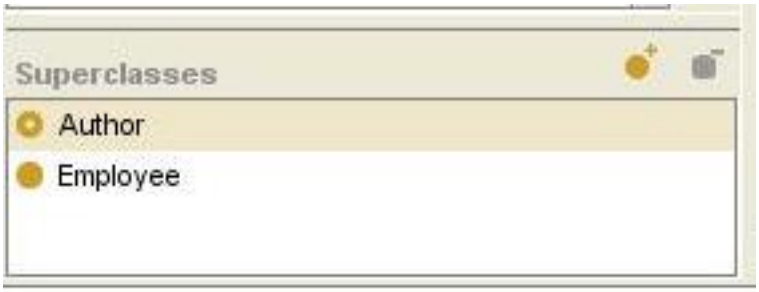


2. Find the Superclasses pane at the bottom left of the Protege window, under the Class Browser. Notice that when **Columnist** is selected, the current superclass of **Columnist**, **Author**, is displayed in the Superclasses pane.

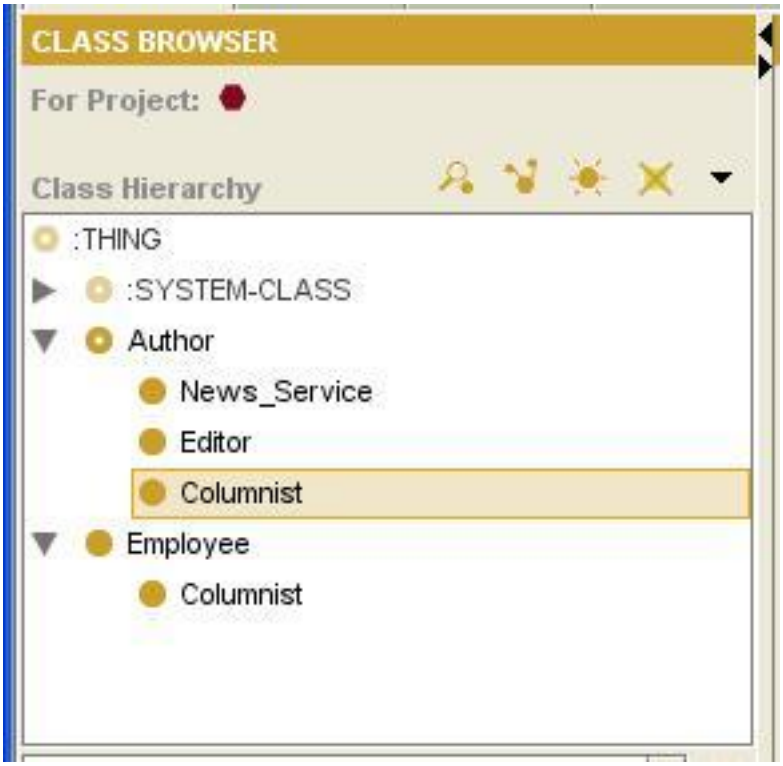
3. Click the **Add Superclass** button at the top right of the Superclasses pane. A Select

Classes dialog box displays all the classes you have created so far, organized in the class hierarchy.

4. Highlight **Employee** and click OK. **Columnist** now has two superclasses: **Author** and **Employee**. Both superclasses are shown in the Superclasses pane.



5. Note that an icon ▶ has appeared to the left of **Employee**. Click this icon to expand the children of **Employee**. **Columnist** now appears in two places in the Class Browser: once under **Author** and again under **Employee**.



Adding a superclass using drag-and-drop

You can also add a superclass using drag-and-drop:

1. Select **Editor** in the Class Browser.
2. Hold down the mouse button and drag the Editor class over **Employee**. **Employee** is outlined.
3. Before releasing the mouse button, hold down the **Ctrl** key, then release the mouse button to drop the subclass.

To remove a superclass from a class, highlight the superclass you want to remove in the Superclasses pane and click the **Remove Superclass** button. You are now ready to assign some attributes to the classes you have created, by creating *slots*, which are covered in the next section.

Creating slots

In Protege-Frames, classes can be thought of as concrete concepts from the domain, such as **Editor** and **Columnist**. Classes are more than simple objects arranged in a hierarchy. They can also have attributes, such as a name, phone number, or salary, and relations between them, such as the **Author** of an **Article**.

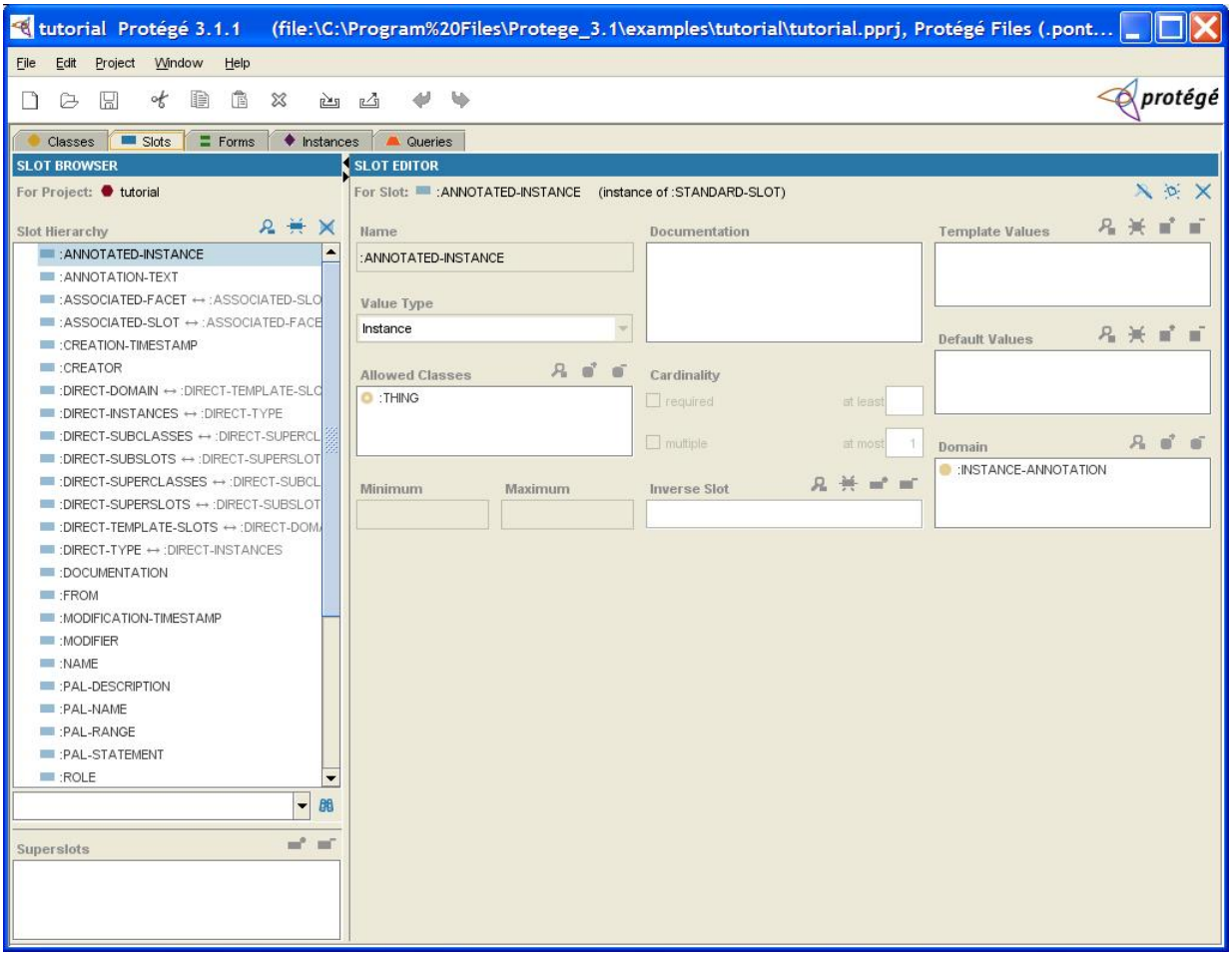
Class attributes and relations are described using *slots*. In the following sections you will learn how to create slots, assign slots to classes, describe relationships between classes using slot, and how slot inheritance works.

- Creating a slot at the Slots Tab
- Assigning a slot to a class
- Creating a slot in the Classes Tab
- Slots and inheritance

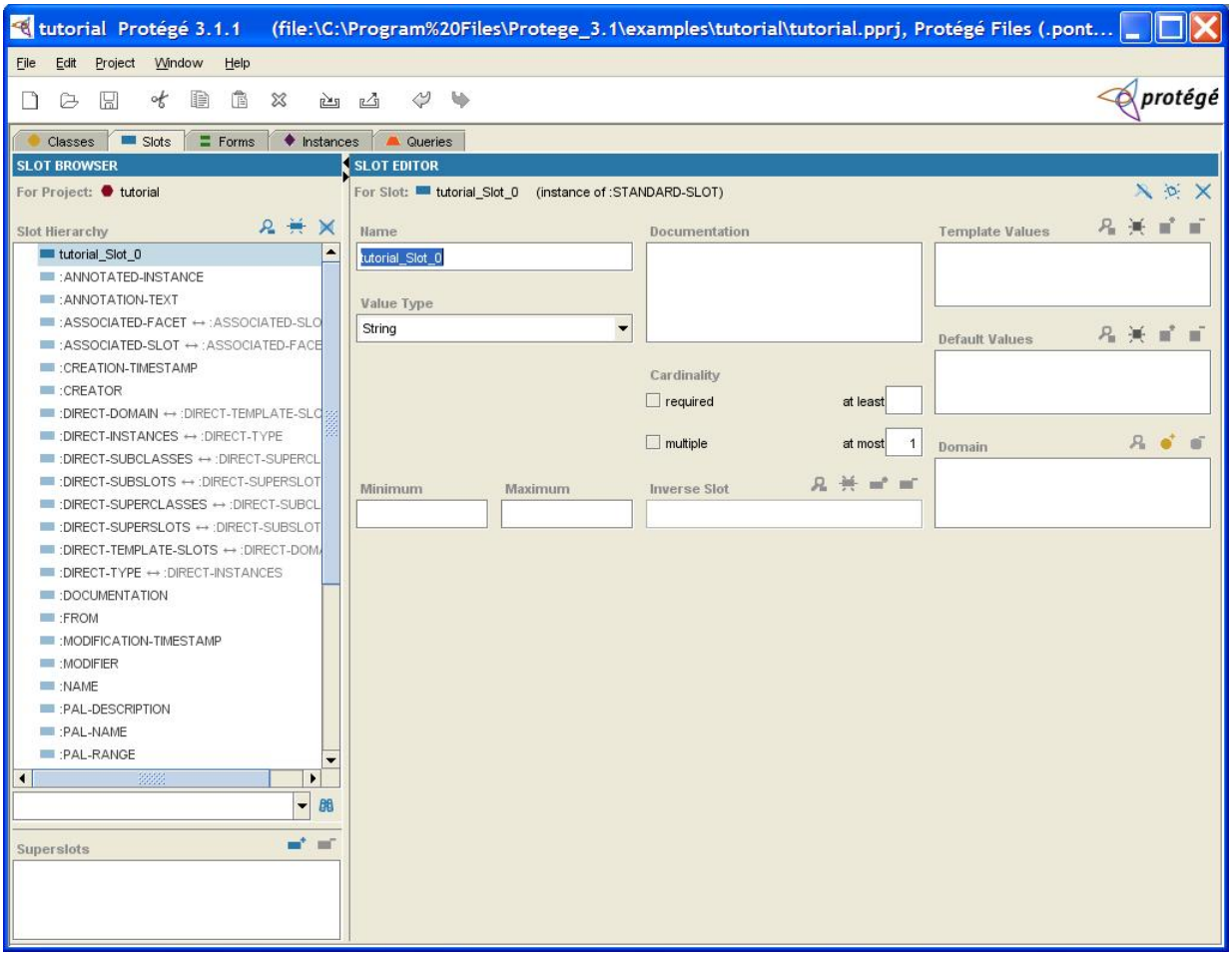
Creating a slot at the Slots Tab

There are several ways to create a slot. One way is to create a slot using the Slots Tab and then assign it to one or more classes. To create a slot called **name** using the Slots Tab:

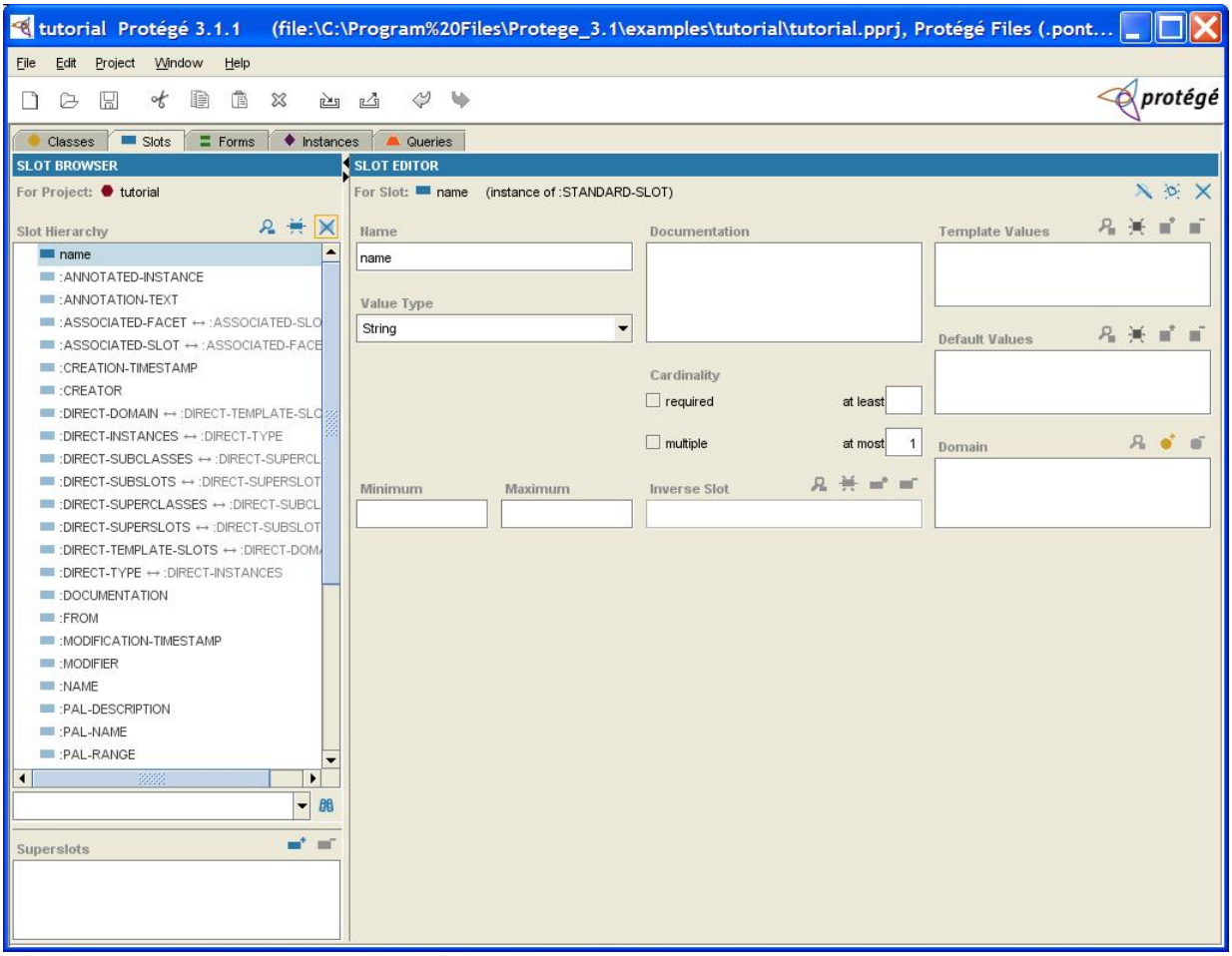
1. Click on the Slots Tab. Notice that the Slots Tab has a layout similar to the Classes Tab, with the slots listed on the left, and a Slot Editor, showing slot properties, on the right.



2. Click the Create Slot button at the top right of the Slot Hierarchy pane. A new slot is created. Just as when you create a class, the slot is given a generic name, such as "tutorial_Slot_0". This name is automatically highlighted in the Slot Editor when the slot is first created.



3. To name the slot, make sure the generic name is highlighted in the Name field of the Slot Editor. Type the name for the slot, **name**. A recommended convention is to make slot names lowercase, with words separated by an underscore. Making slot names lowercase, while making classe names start with an uppercase letter, helps you distinguish them in your ontology.



4. Notice that this slot has the default *Value Type* String. The value type places restrictions on the type of values a slot can have. A String slot is a slot that takes values that are any string of alphanumeric characters, possibly including spaces.

For this simple slot, we will not change any of the other facets in the Slot Editor.

Assigning a slot to a class

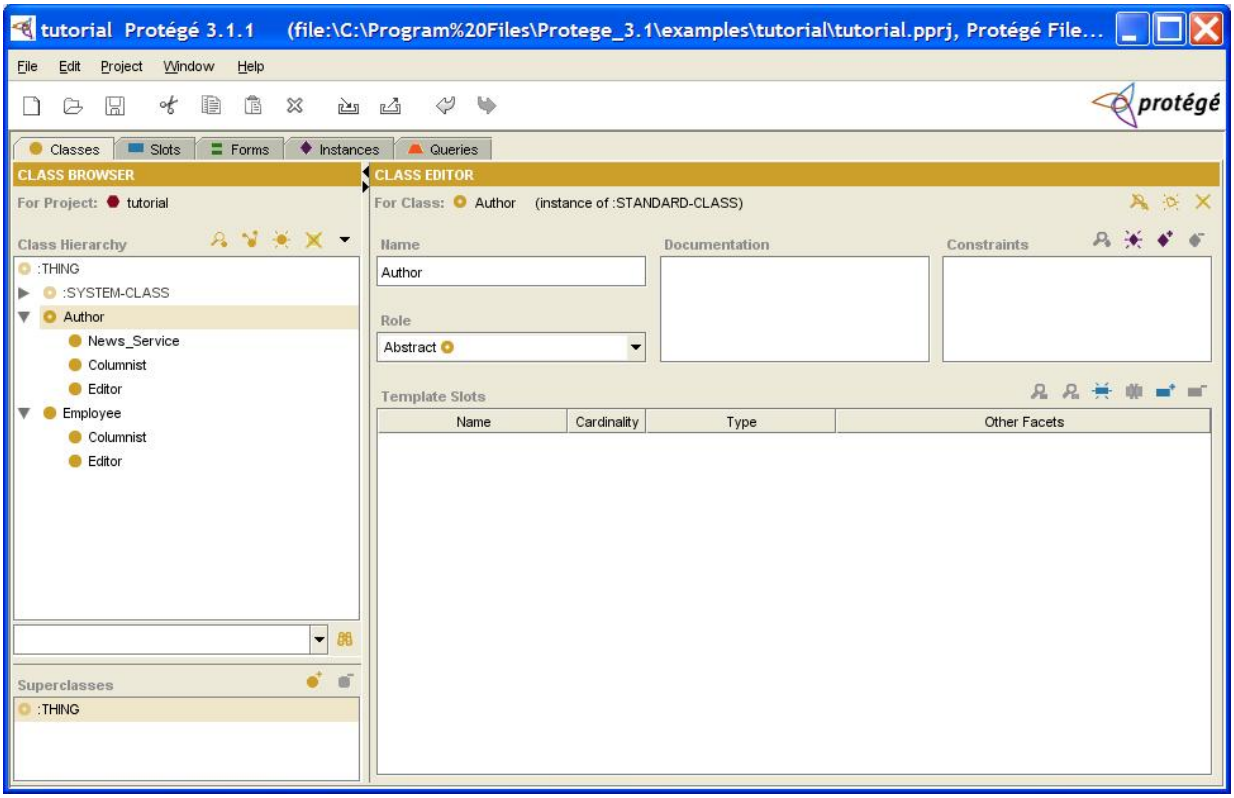
All we have done so far is create a slot which defines the general attribute **name**. To truly incorporate it in our ontology, we need to assign it to a class, e.g., we want every instance of the **Author** class to have a name.

To do this, we will return to the Classes Tab and edit the **Author** class. Any attributes you create or assign to a class are displayed in the Class Editor, to the right of the Class Browser. We have already used the Class Editor to change the name of the classes we have created, and also to change the role of the **Author** class to Abstract. Now we will use the Class Editor to view and name slots.

To assign the slot **name** to a class:

1. Click on the Classes Tab.

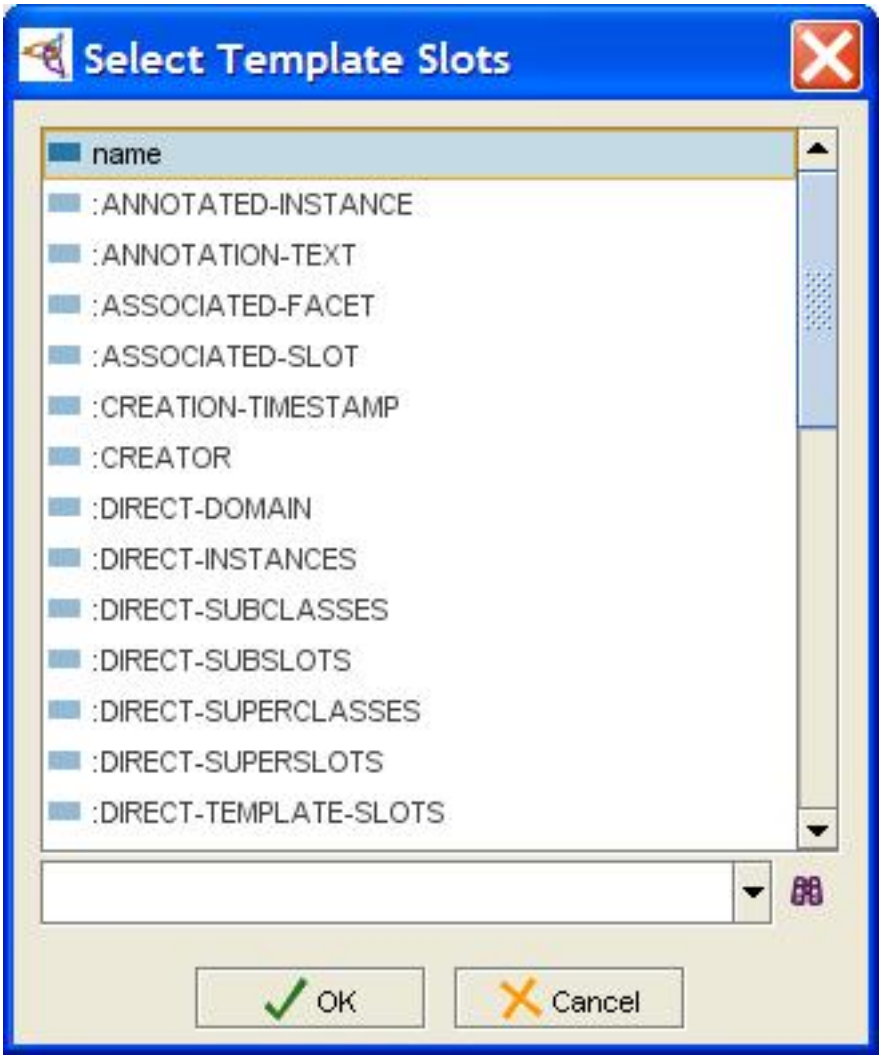
2. Highlight the class **Author** in the Class Hierarcy pane.



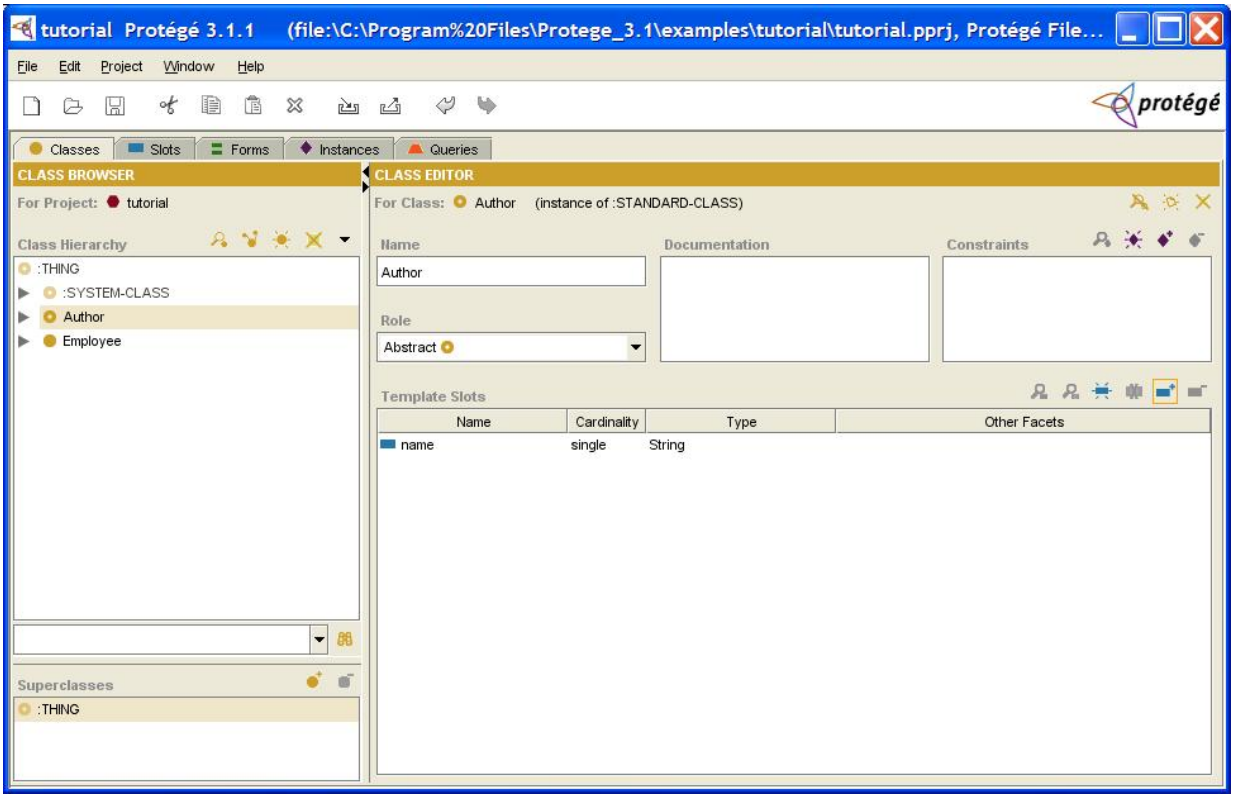
Take a moment to look at the Class Editor to the right. This area has a Name (which we have already changed to Author), the role (Abstract), as well as documentation and constraints. Below these fields is the Template Slots pane, which takes up the entire lower half of the Class Editor pane. This is the area that lists all the slots assigned to a class. Currently it is empty.

3. To add the slot to the class, click the Add Slot button. The slot buttons appear at the far right of the Protege-Frames window, just above the Template Slots pane.

4. When you click the Add Slot button, the Select Template Slots dialog box displays all the slots in your project in alphabetical order (with the exception of the Protege system classes, which are listed at the bottom of the slot hierarchy).



5. Select **name** and click OK.



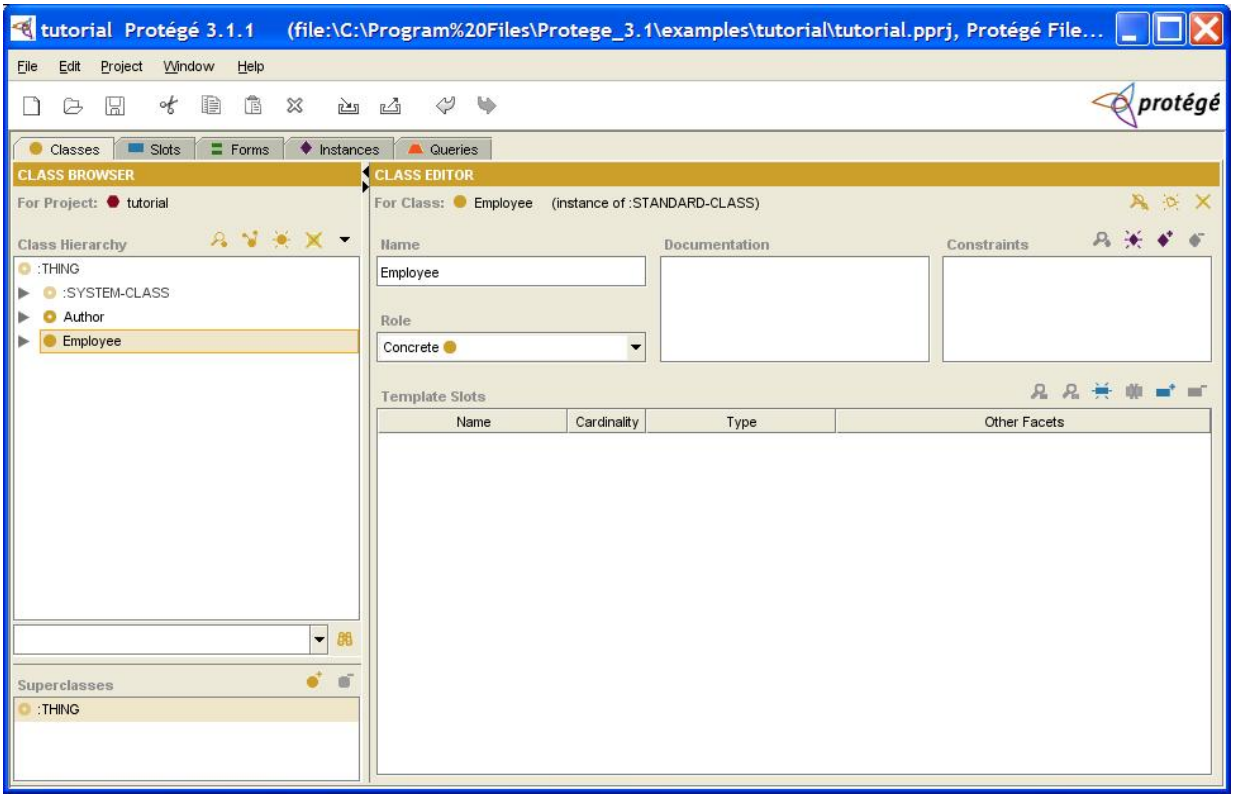
If you look at the Template Slots for **Author**, you see that **name** has been added to the list of Template Slots, along with its properties, in this case, its cardinality (an author can have only one name) and the slot type, String.

Creating a slot in the Classes Tab

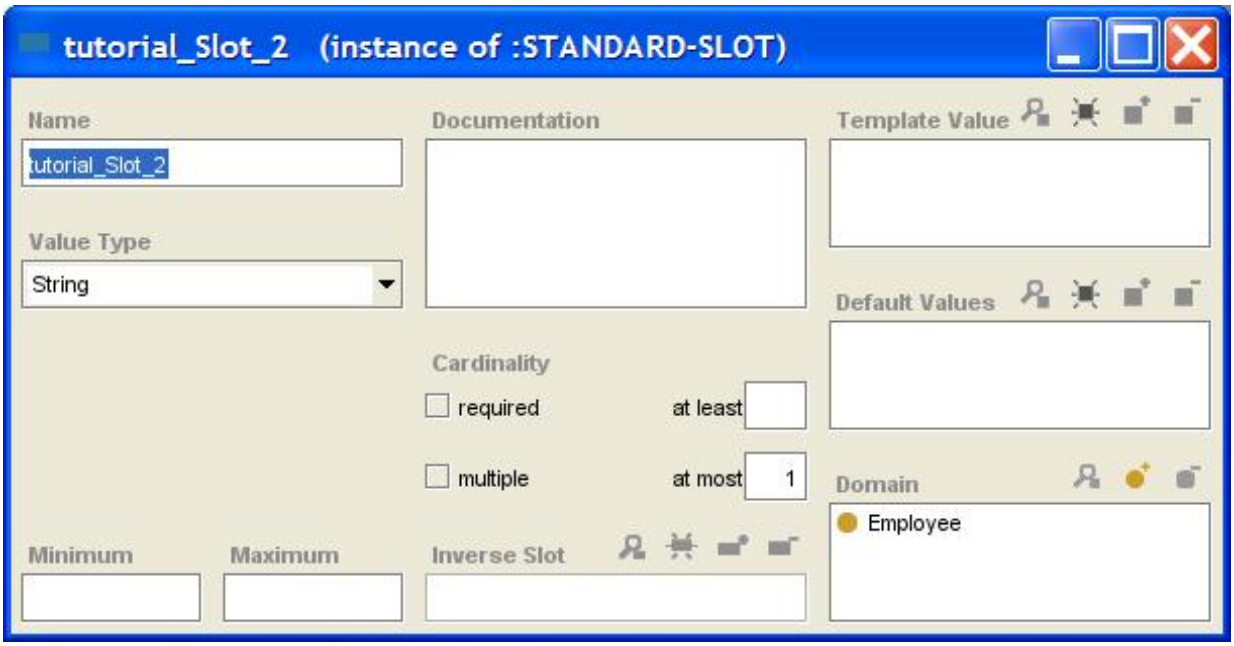
Switching between the Classes Tab and the Slots Tab can be tedious. In addition, since slots represent properties of classes, it would be simpler if we could create the slot directly from the Classes Tab, which is easy to do in Protege-Frames.

To create a slot for the class **Employee**:

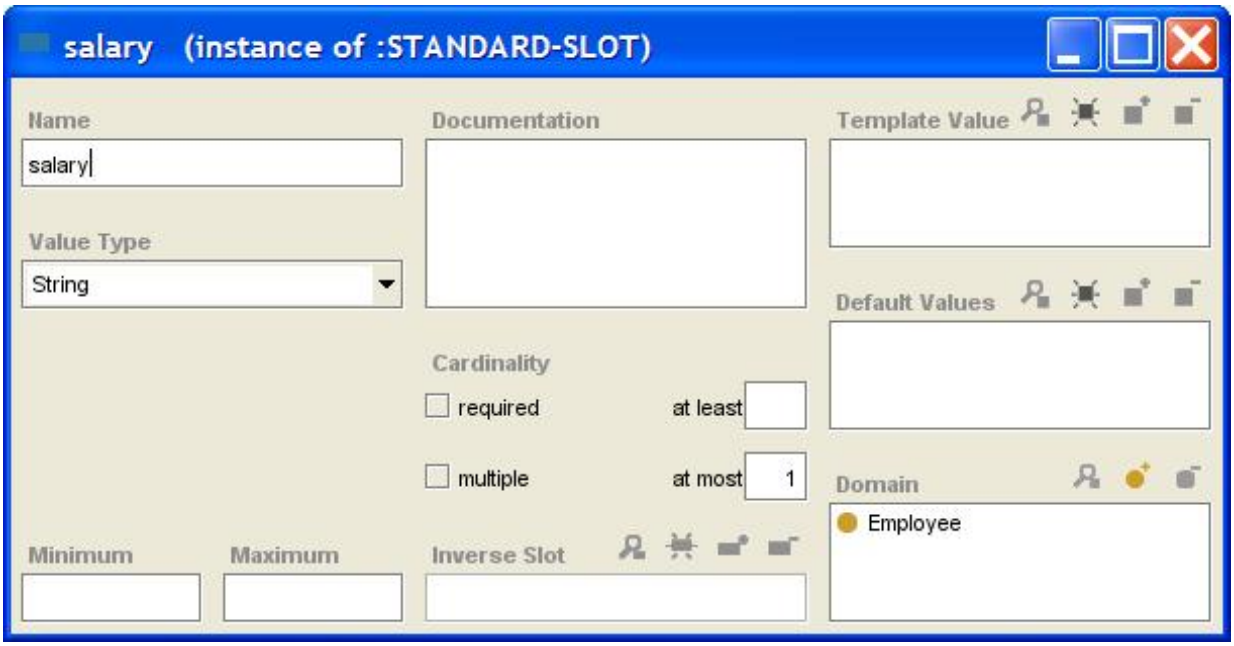
1. Select **Employee** in the Class Hierarchy pane.



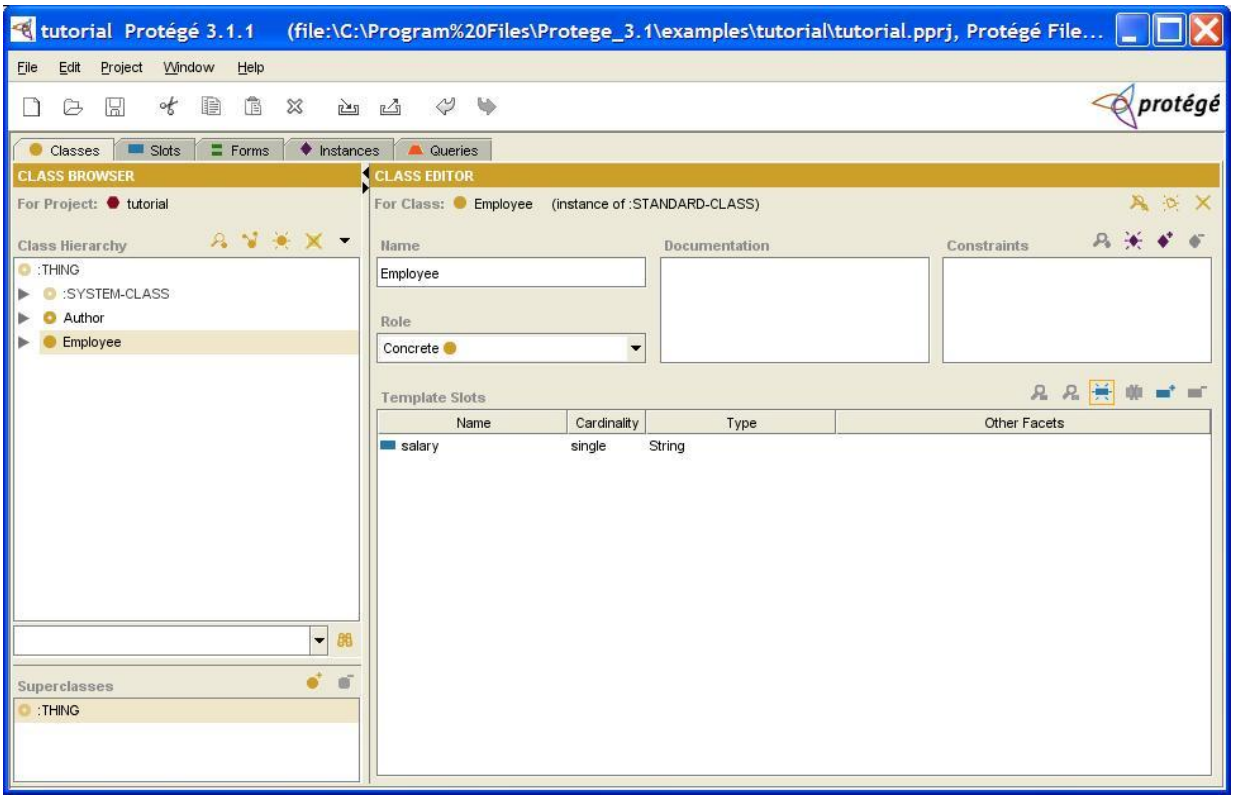
2. Click the Create Slot button at the far right of the Template Slots pane, which brings up the modeless slot form:



3. Type **salary** in the name field and hit **Enter**.



4. Return to the main window. (You can leave the slot editor dialog for **salary** open if you wish. You will return to it later to edit more properties of the **salary** slot). Notice that your slot now appears in the Template Slots pane when **Employee** is highlighted.

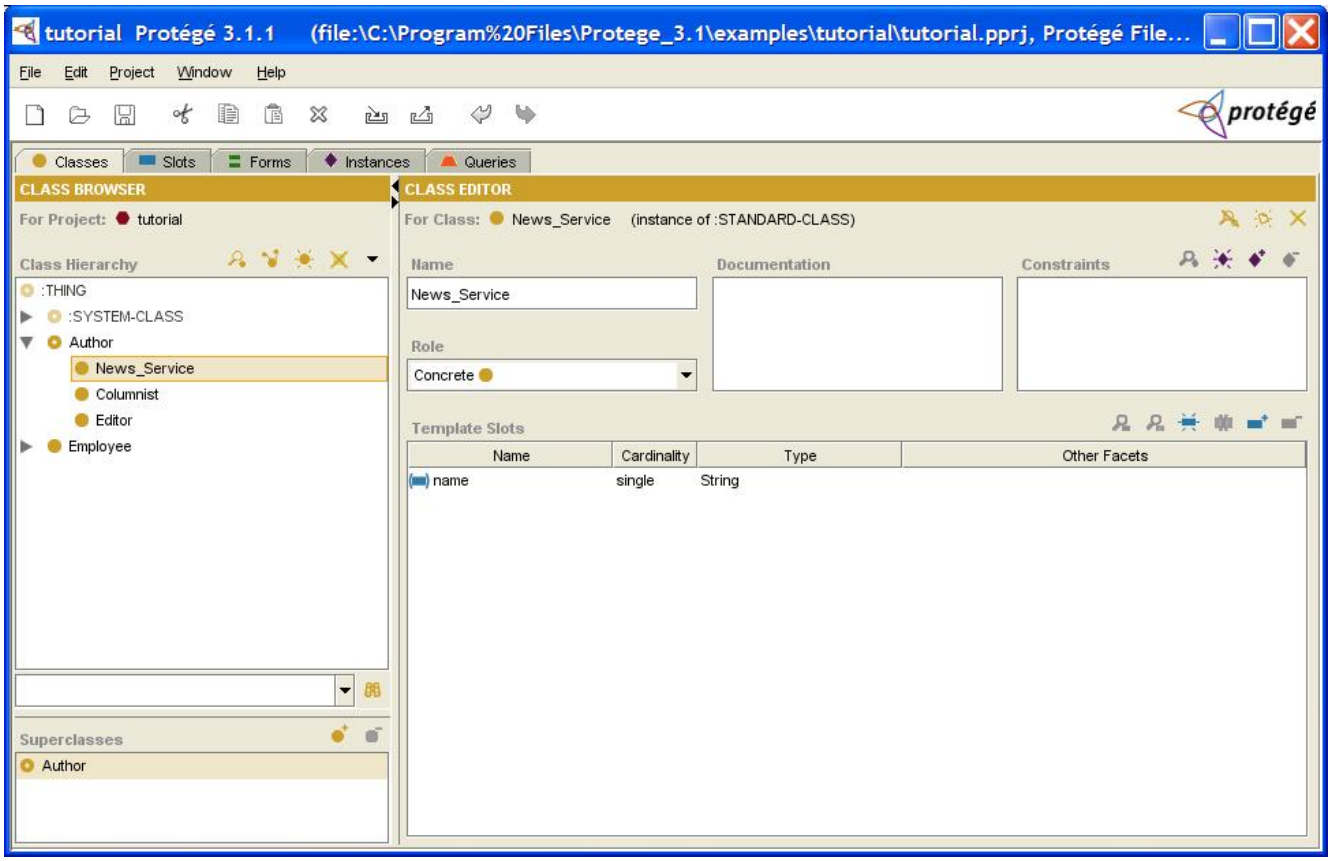


Slots and inheritance

We do *not* have to assign the **name** slot to every class where we want it to appear. Any subclass of a class automatically inherits all the slots of the superclass. For example, if you click on the **News_Service** class, you can see that:

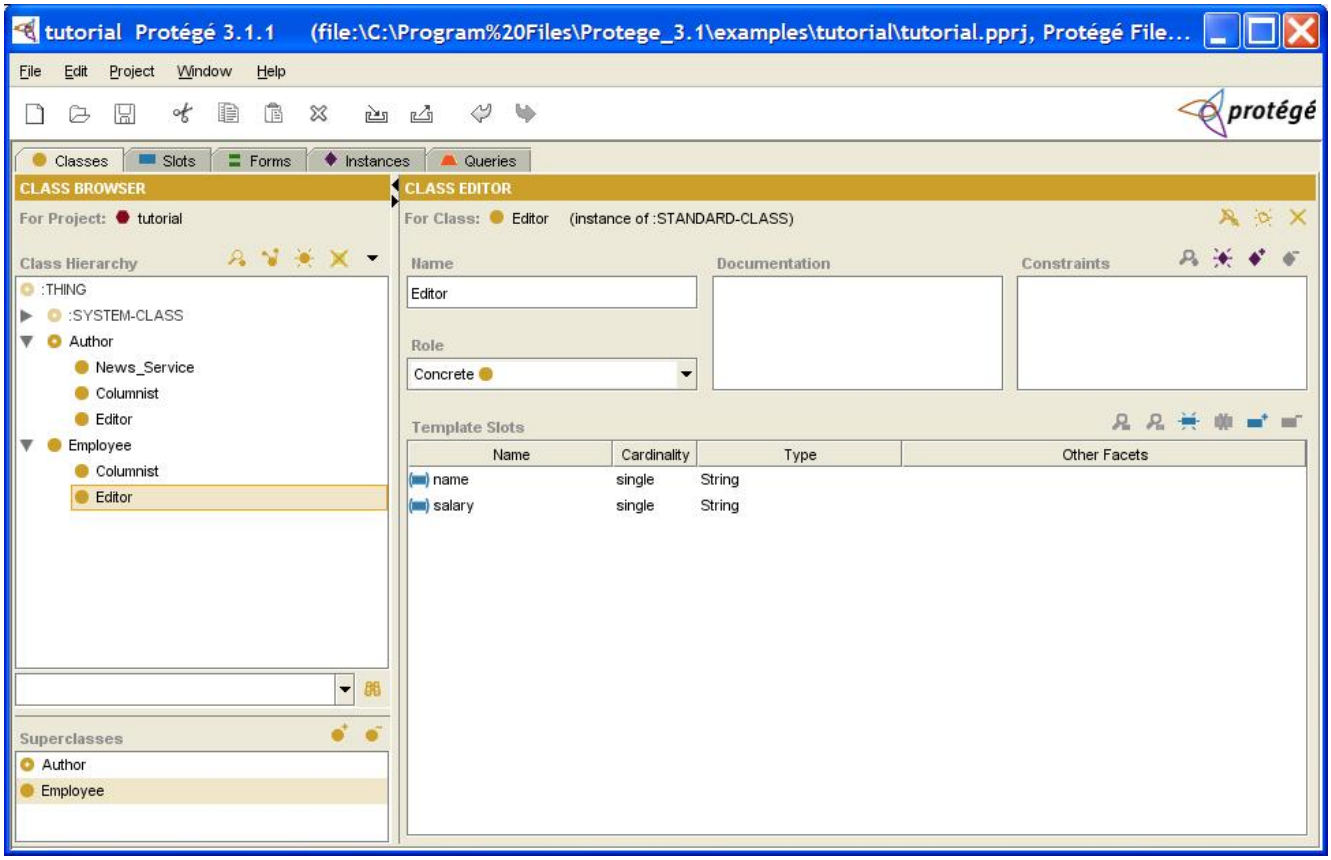
a) the **name** slot has already been assigned to this class through inheritance

b) the slot icon is different than it was for **Author**, i.e., the " inherited slot" icon shows that the slot has been inherited



Note: In the Protege-Frames menu bar, click Help | Icons to see a list of all the icons in the Protege User Interface, along with textual descriptions of what the icons mean.

Classes with more than one superclass inherit the slots from both classes. For example, if you click on the **Editor** class, you can see that it has inherited the **name** slot from **Author**, and the **salary** slot from **Employee**. This multiple inheritance is an essential part of Protege-Frames.


tutorial Protégé 3.1.1 (file:\C:\Program%20Files\Protege_3.1\examples\tutorial\tutorial.pprj, Protégé File...
File Edit Project Window Help
protégé
Classes Slots Forms Instances Queries
CLASS BROWSER
For Project: tutorial
Class Hierarchy
:THING
:SYSTEM-CLASS
Author
News_Service
Columnist
Editor
Employee
Columnist
Editor
Superclasses
Author
Employee
CLASS EDITOR
For Class: Editor (instance of :STANDARD-CLASS)
Name
Editor
Documentation
Constraints
Role
Concrete
Template Slots
Name
Cardinality
Type
Other Facets
name
single
String
salary
single
String

Creating slot facets

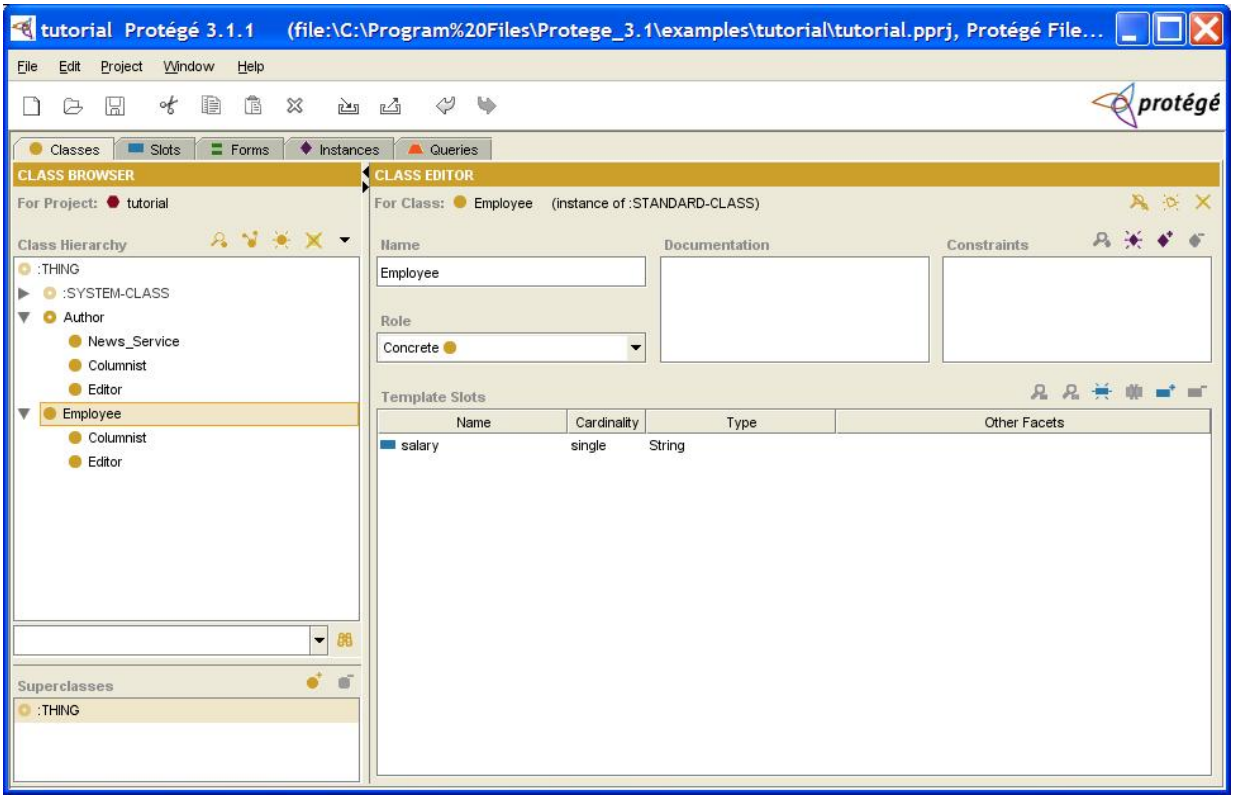
The slots we have created so far are very simple. However, slots themselves can have properties. For instance, a salary is always a number. You can also use slots to create relationships between classes. The properties of a slot, called facets, can be created and edited either from the Classes Tab using the slot specification dialog, or from the Slots Tab in the Slot Editor pane.

- Defining slot facets for salary
- Creating a relationship using slots

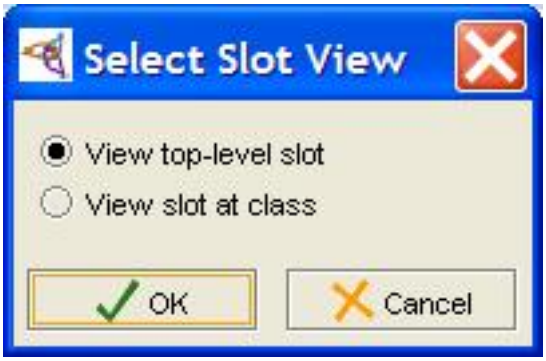
Defining slot facets for salary

We can define some facets for the **salary** slot we created earlier.

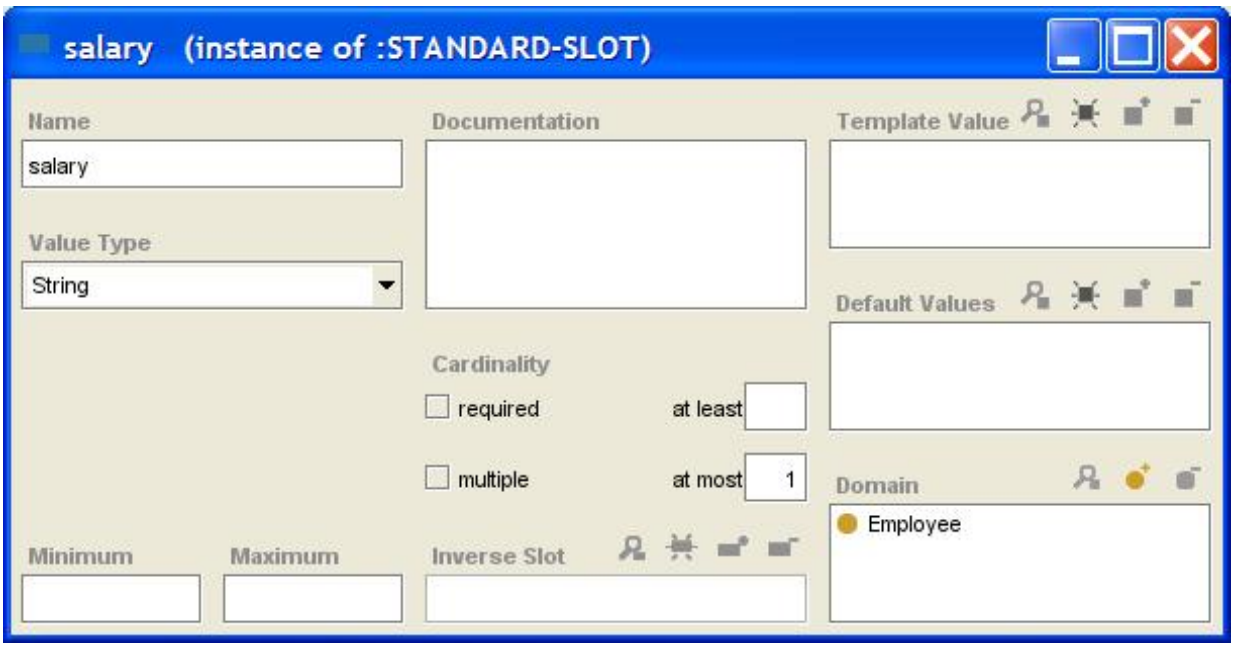
1. Select the **Employee** class in the Class Hierarchy pane.



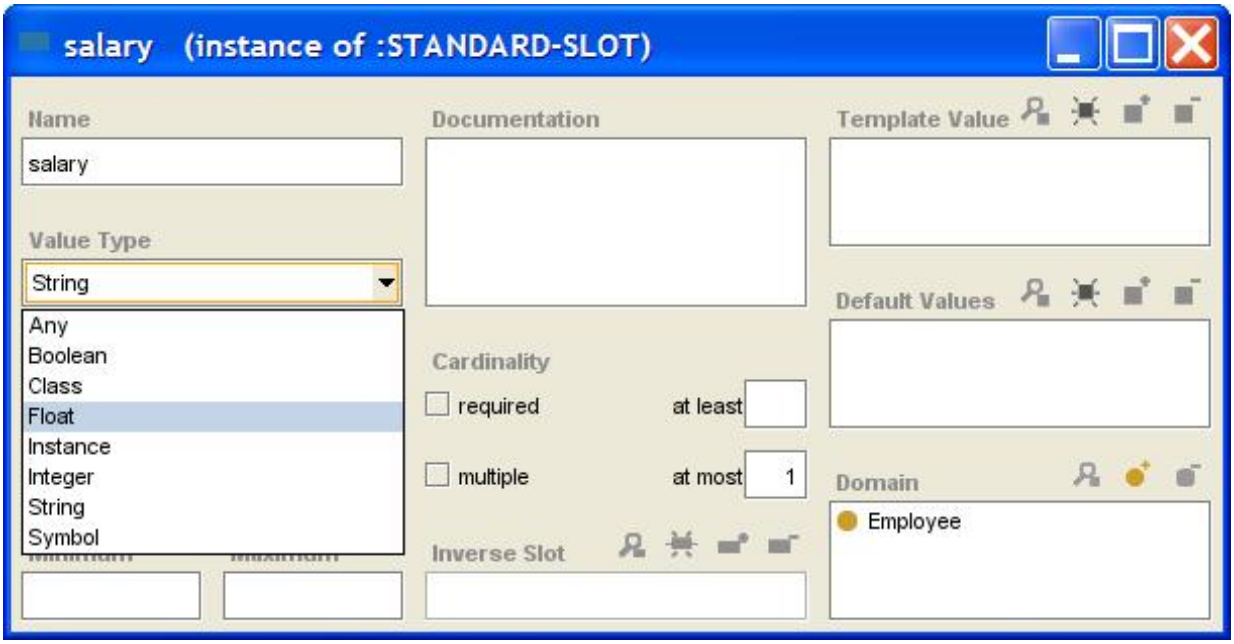
2. Double-click on the **salary** slot in the Template Slots pane to open the slot form. A dialog box asks you to select the Slot View. When you edit a slot, you can choose whether your edits will apply to the slot and all classes with that slot -- to the "top-level" slot -- or whether you just want them to apply to this class and all of its children.



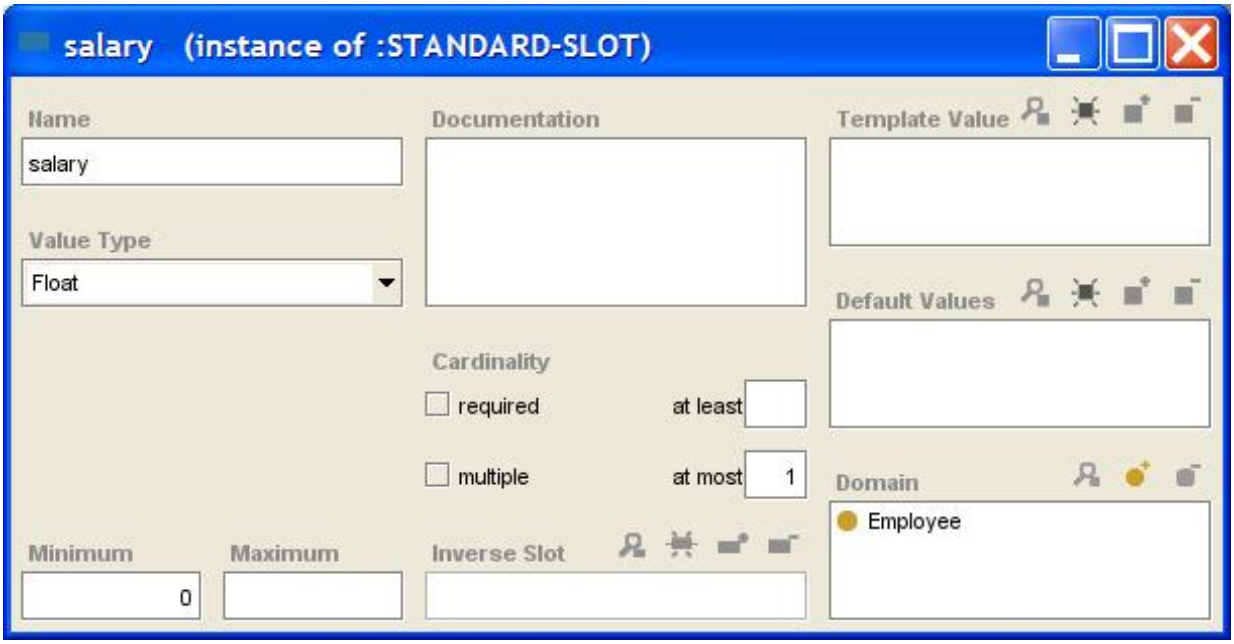
3. In this case, we want to view and edit the top-level slot. Make sure that the **View top-level slot** option is selected and click **OK**. This will change the definition of the slot everywhere it appears in the ontology.



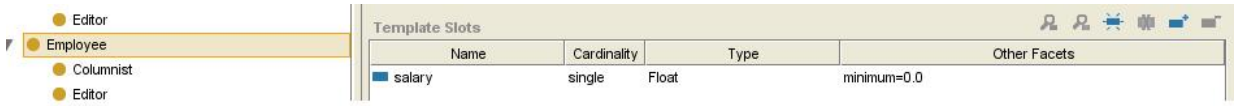
4. In the resulting slot form, select **Float** from the **Value Type** menu at the left. When you create instances, you will only be able to enter a valid floating point number for this slot.



5. Enter 0 (zero) in the Minimum field at the lower left. This specifies that for each instance of **Employee**, the value that is entered for **salary** must be non-negative.



6. Close the Edit Slot dialog box. You can see that the description of the slot has changed in the Template Slots pane. The type is now set to Float, and the minimum value of 0.0 appears in the Other Facets column.

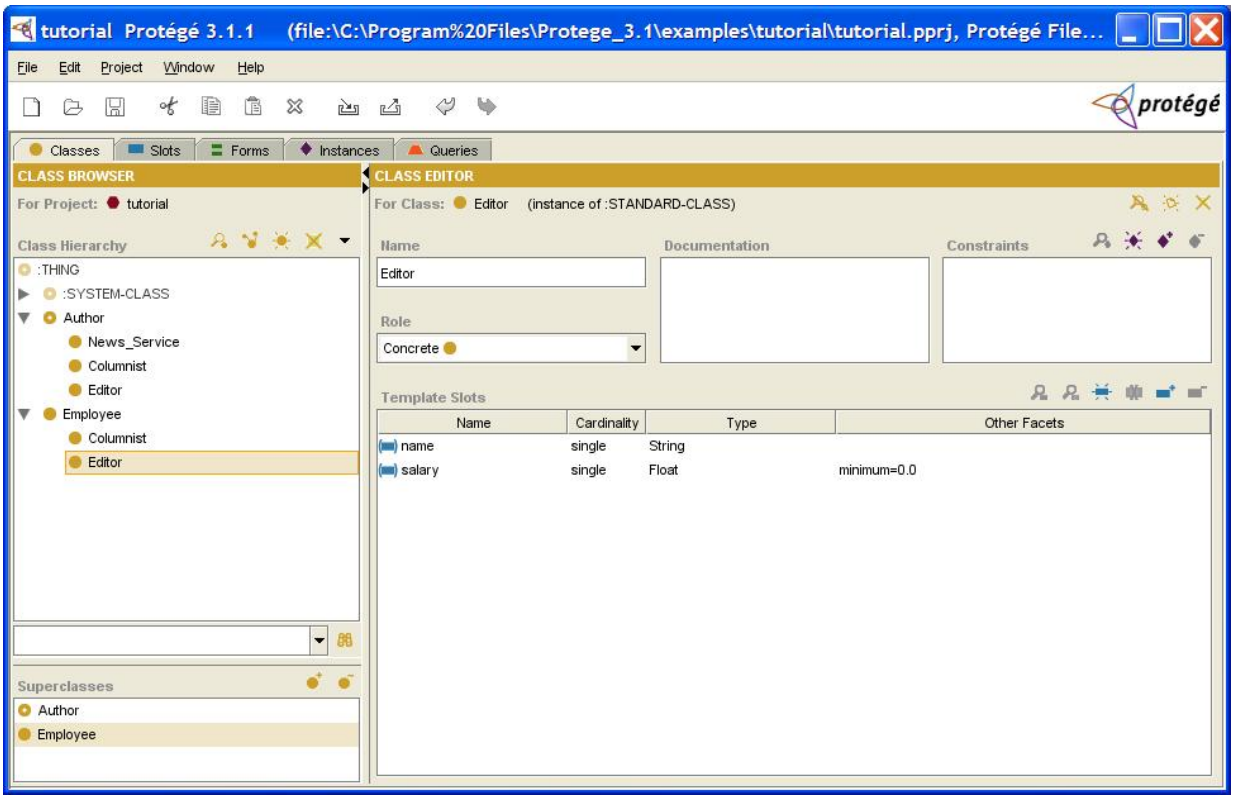


Note: Making edits at the top-level slot means that you are modifying the fundamental definition of the slot. It is the same as making changes at the Slots Tab. If you select the **View slot at class** option instead, any changes you make will only apply to this class and its children. For example, if you started a program that allowed people to pay to intern at the newspaper, you could create a new subclass of **Employee**, **Paying_intern**, open the **salary** slot, select **View slot at class**, and delete the minimum value. This would create a "slot override", which would only affect **Paying_intern**, while other employees would still be required to make a non-negative salary.

Creating a relationship using slots

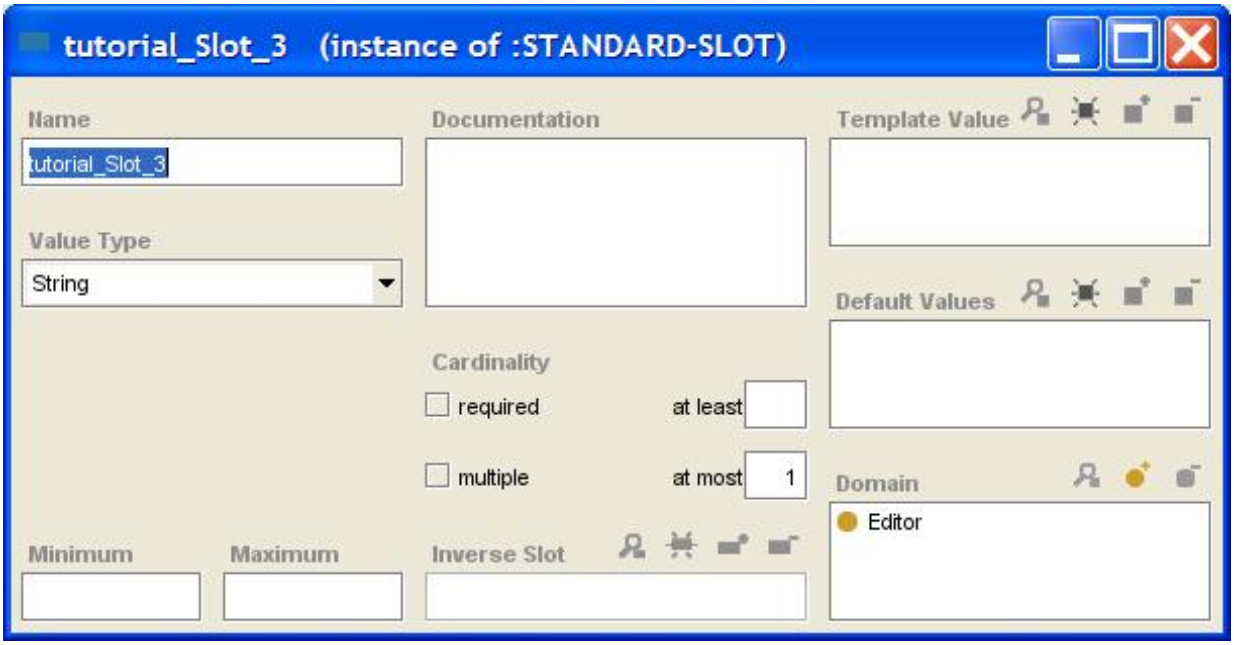
Protege-Frames also allows you to create slots that can be used to describe relationships between classes that are not defined by the class hierarchy. This can be done using slots of type Instance or of type Class. For example, an editor may be *responsible for* one or more employees. We can create a slot that represents this relationship between the **Editor** and **Employee** classes as follows:

1. Select **Editor** in the Class Hierarchy pane.



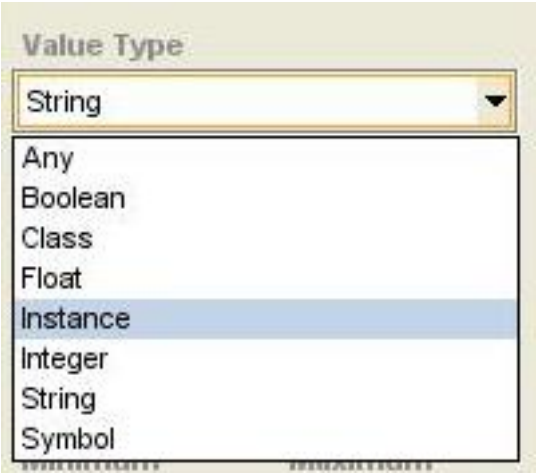
2.

button to create and attach a new slot to **Editor**.



3. In the resulting slot form, type **responsible_for** in the Name field.

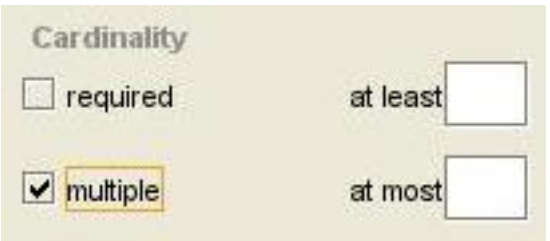
4. Select Instance from the Value Type menu.



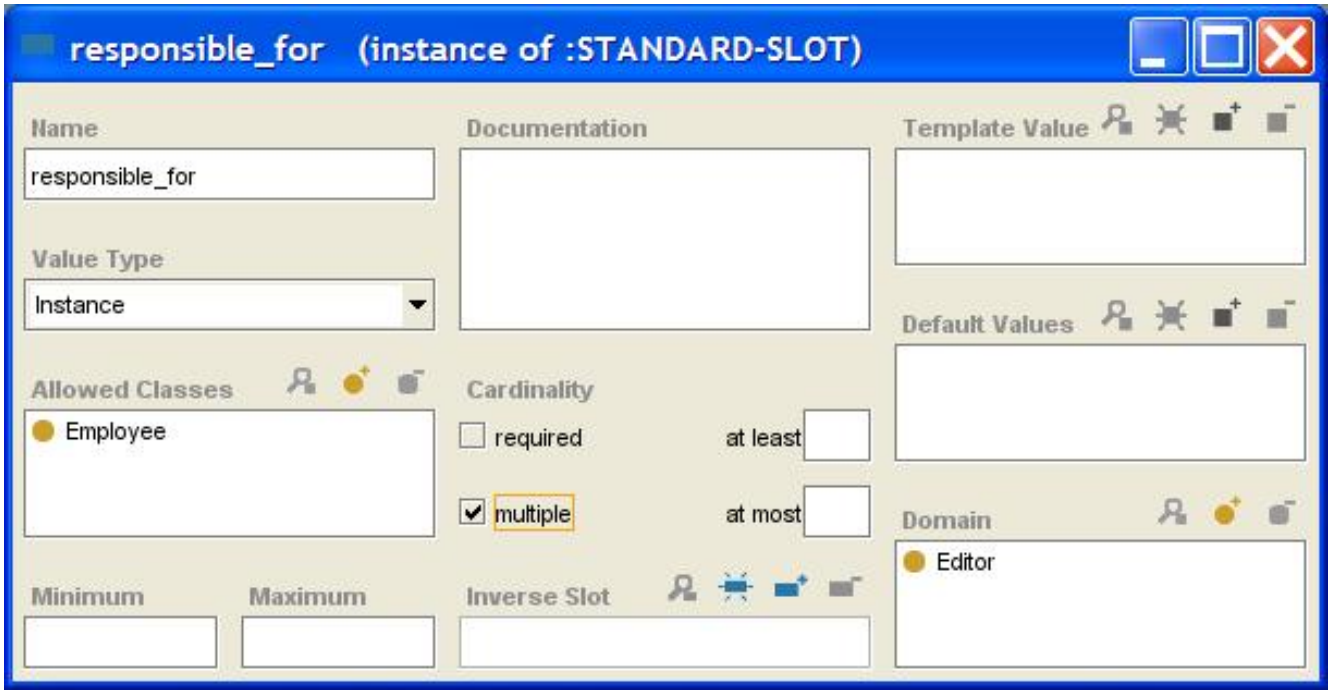
A new field, Allowed Classes, is displayed below the Value Type menu.

5. Click the Add Class button that appears at the top right of the Allowed Classes area. A Select Classes dialog box displays all the classes in the project. Select **Employee** and click OK.

6. To allow an editor to be responsible for more than one employee, click multiple in the Cardinality area to the right of the Value Type area.



After completing steps 1 through 6, the slot form for **responsible_for** looks as follows:



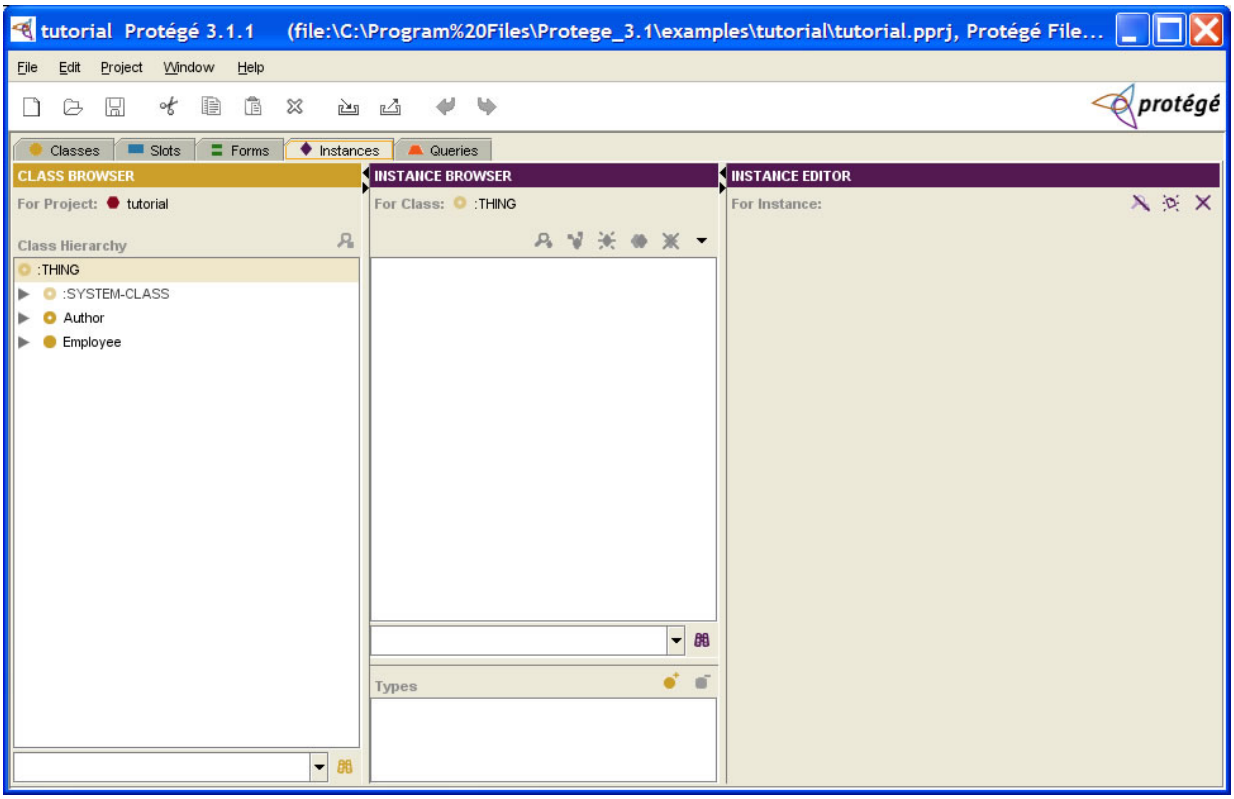
What have we done? We have created a slot that can hold one or more *instances* of the **Employee** class as its value. Later, when we create an instance of the **Editor** class, and we want to specify who that editor is responsible for, we can choose one or more instances of the **Employee** class to fill in the value of the **responsible_for** slot.

Entering instances

Instances are the actual data in your knowledge base. In general, it is a good idea to make sure you have structured your project as well as you can before entering extensive numbers of instances. If you have to make changes to your class or slot structure after instances have been entered, you may lose some information. In addition, if you add slots, you will have to go back and fill in the slot values for all instances that were created previously.

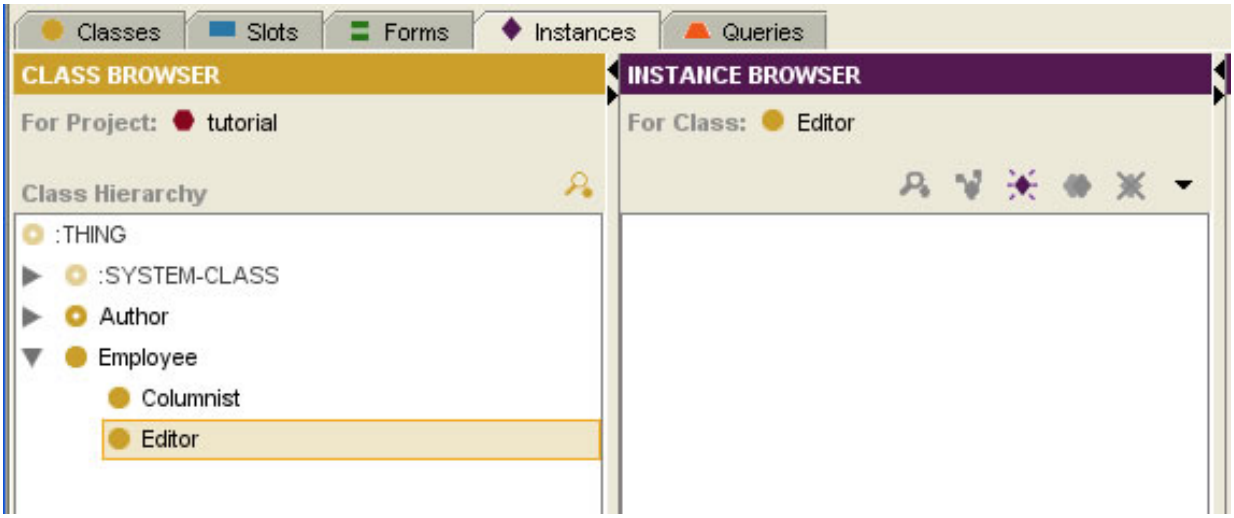
In this section, you will create two instances of the **Editor** class:

1. Click on the Instances Tab. This tab has three panes. The first, at the far left, displays the class hierarchy. The middle pane, which is currently blank, will show the list of instances you create for a particular class. The third pane shows the Instance Editor, which displays the form for the currently highlighted instance where you can enter slot values.

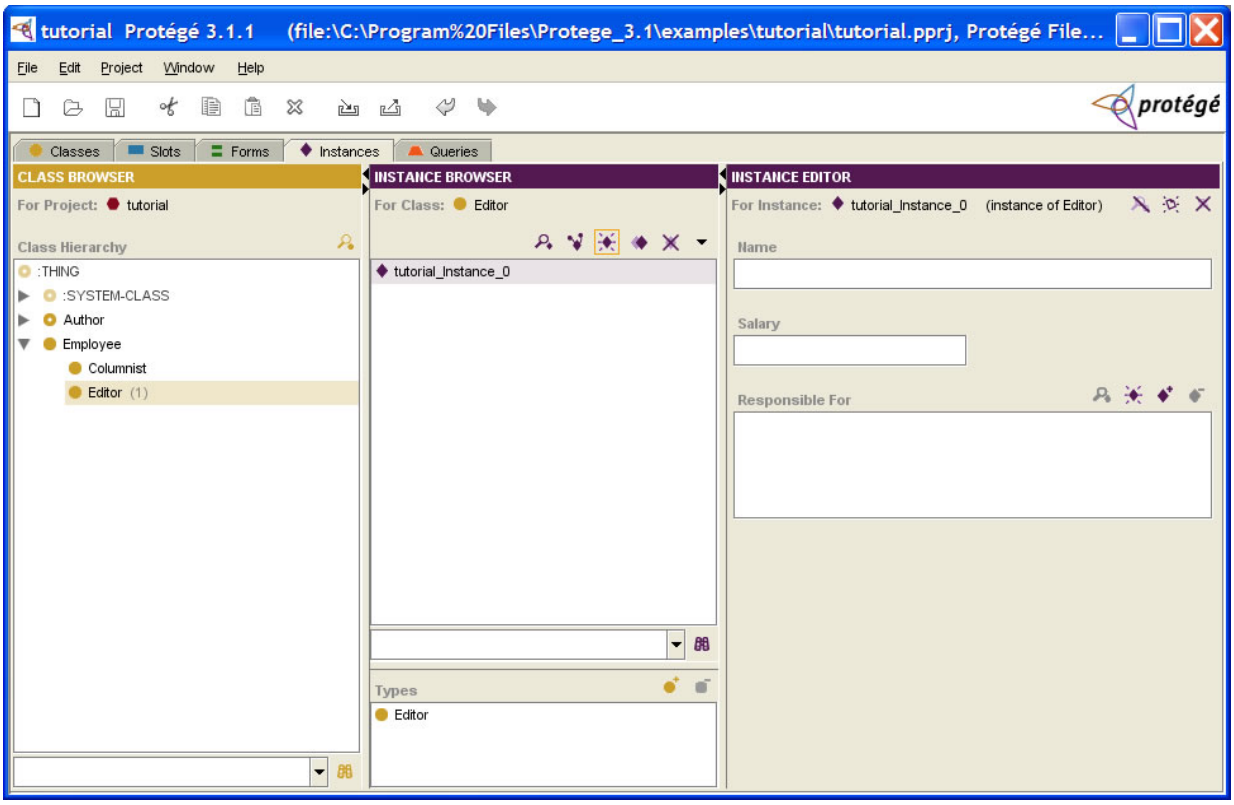


2. Expand the subclasses of **Employee**.

3. Click on **Editor**. The **Create Instance** button in the Instance Browser becomes active, indicating you can now create an instance.



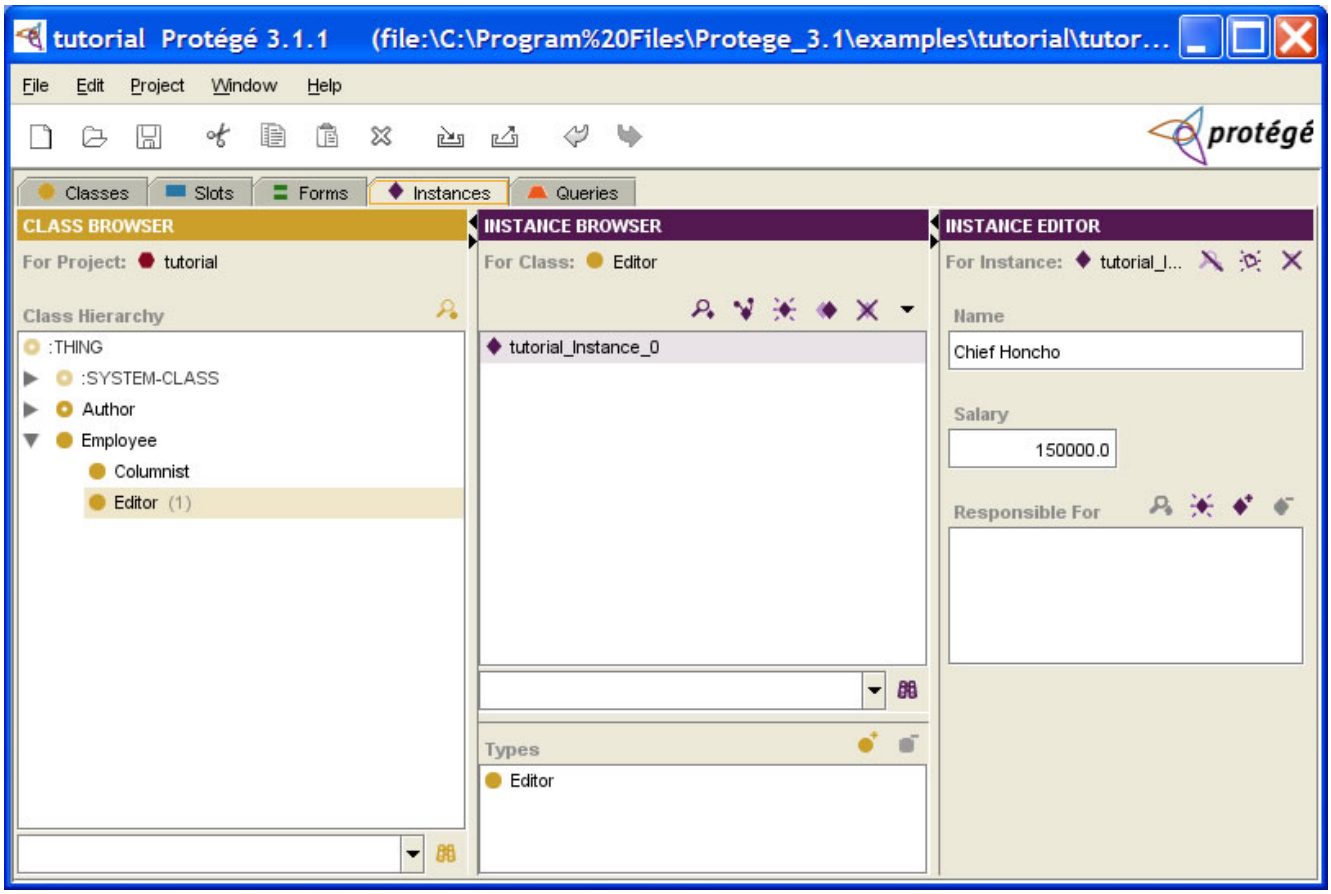
4. Click on the **Create Instance** button. An instance is created, and the Instance Editor form appears. You can see it has a number of fields, one for each slot you created. You use these fields to enter the values for the slots. Notice that the display for the **Editor** class in the Class Hierarchy pane changed when you created the instance. The (1) in parentheses after **Editor** indicates that this class now has one instance.



5. Enter *Chief Honcho* in the **Name** field.

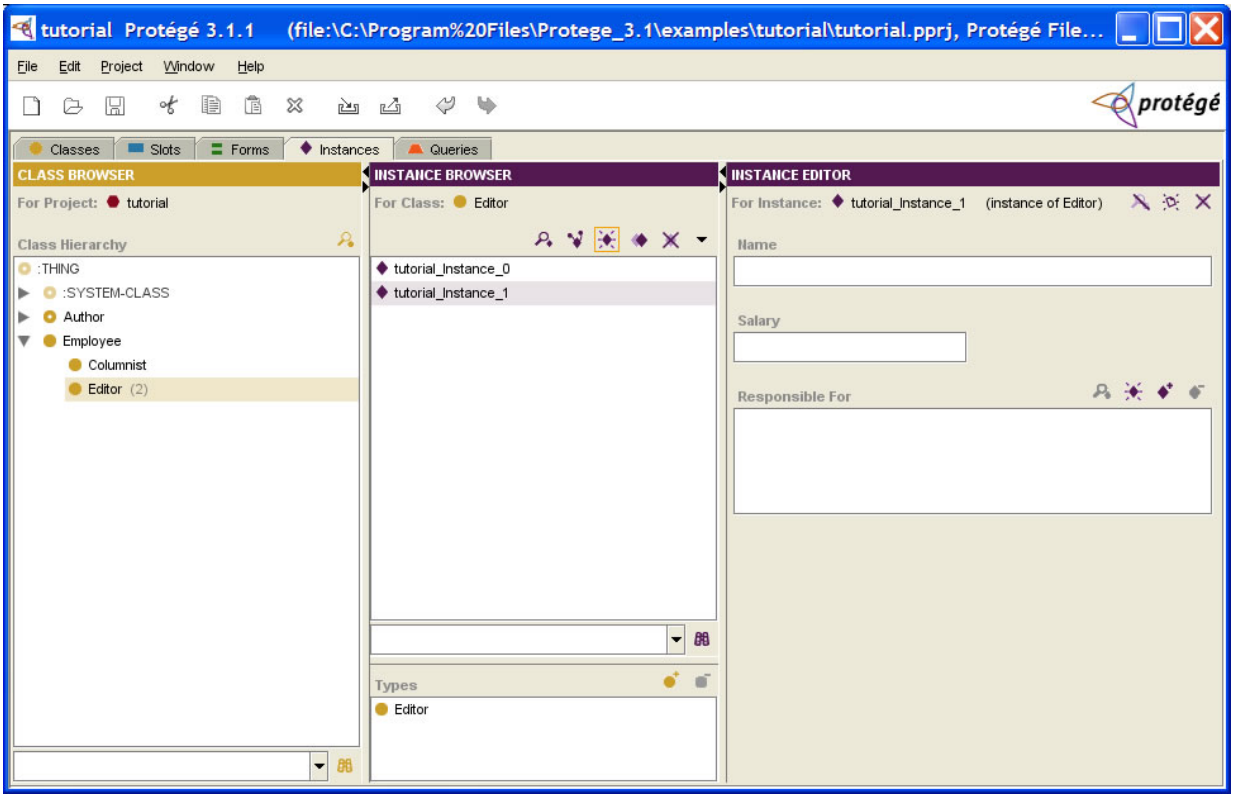
6. Enter *150000* in the **Salary** field. (Notice that the characters in this field will appear in red if anything other than a valid float value is entered. In Protege, if you attempt to enter slot values that do not satisfy the restrictions of the slot, the value will appear in red).

The Instances Tab now looks as follows. Note that the instance still has a generic name in the Instance Browser, such as "tutorial_instance_0". You will learn how to change this in the next section.



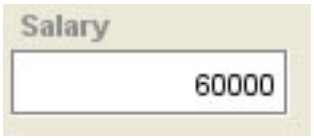
To create another instance:

1. Click the **Create Instance** button in the Instance Browser.



2. Type *Mr. Science* in the **Name** field.

3. Type *60000* in the **Salary** field.



Now that you have created more than one instance, you can define a relationship between them, e.g., you could say that Chief Honcho is "responsible for" Mr. Science. Before doing so, in order to make working with instances easier, you want to set a display slot for the **Editor** class. Protege will display the value of the display slot every time it displays instances of the class. Setting the display slot is covered in the next section.

Setting the display slot

For each class in your ontology, you can specify one of its slots to be a *display slot*. Protege will display the value of this slot any time it displays instances of the class. If you do not set a display slot, Protege will display the underlying system generated name for your instances such as "tutorial_Instance_0". It is usually very helpful to set a display slot for all classes that will have instances. In fact, you may choose to set the display slots for your classes before you even start creating instances.

To set the display slot for the **Editor** class:

1. Select the Instances Tab.
2. Select **Editor** in the Class Hierarchy pane.
3. Click the Instance Menu icon (arrowhead that points downward) at the top right of the Instance Browser.

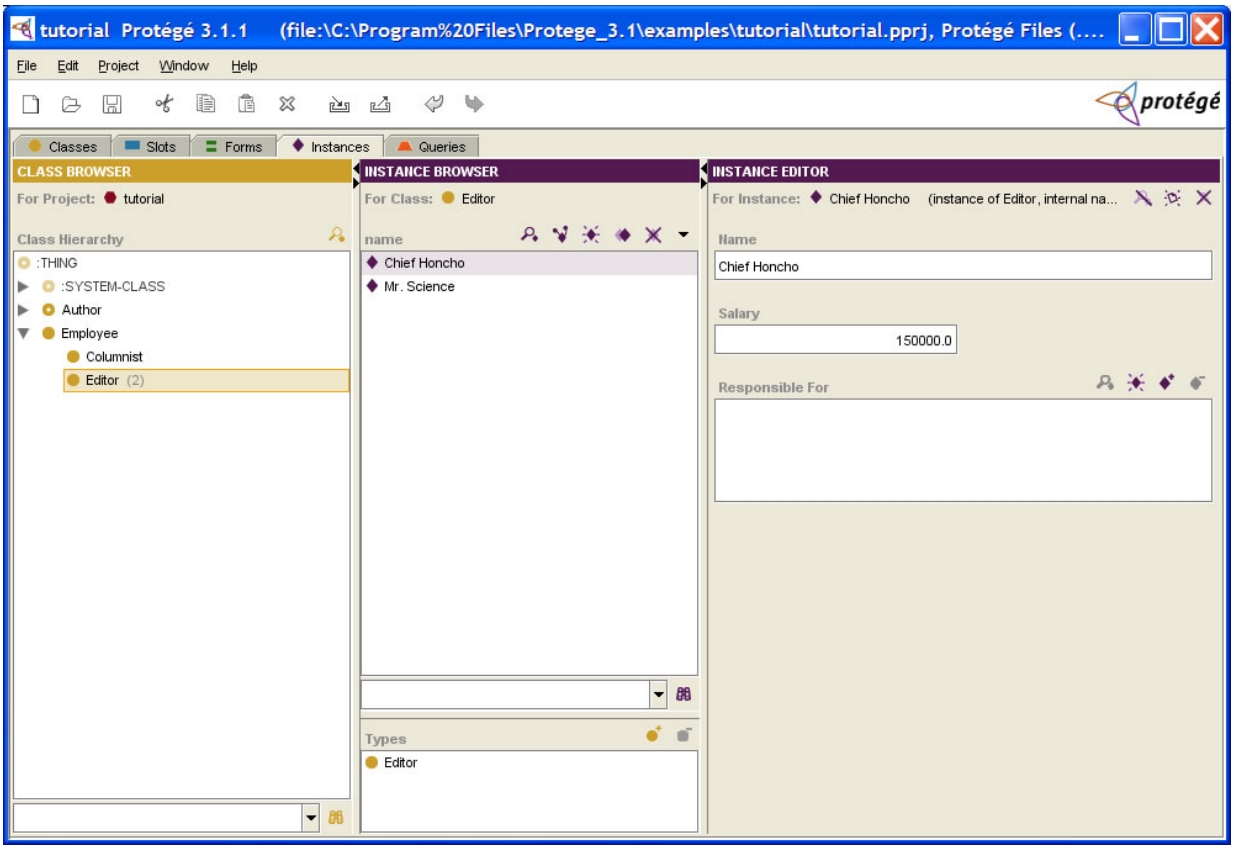
4. Select **Set Display Slot** from the menu.

5. Select **name** from the **Display Slot** menu.
6. The display of the list of instances in the Instance Browser changes to reflect the display slot you have chosen. Instances of the Editor class are now listed by the value of the name slot. (Instances are always displayed by alphabetical or numerical order). From now on, you will be able to browse instances of **Editor** by **name** in the Instances Tab, and everywhere else in the Protege User Interface where lists of instances are displayed.

Creating a relationship between instances

In this section, you will modify the *Chief Honcho* instance to make him responsible for *Mr. Science*:

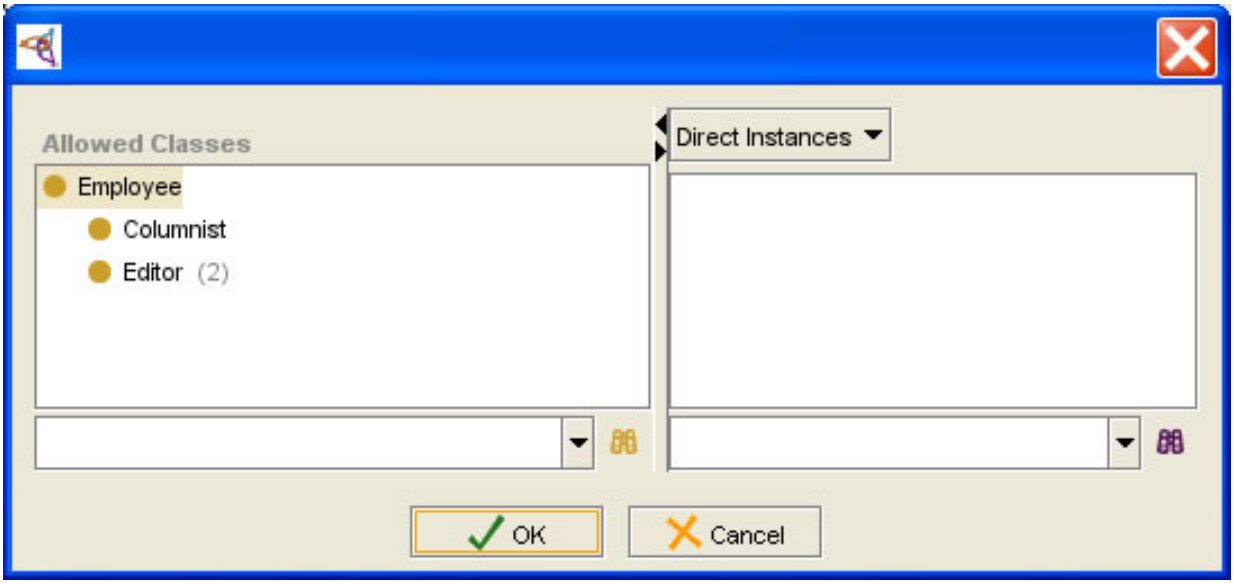
1. Click on the Instances Tab, expand the **Employee** class in the Class Hierarchy pane, and select the **Editor** class. Instances of **Editor** are now listed in the Instance Browser.



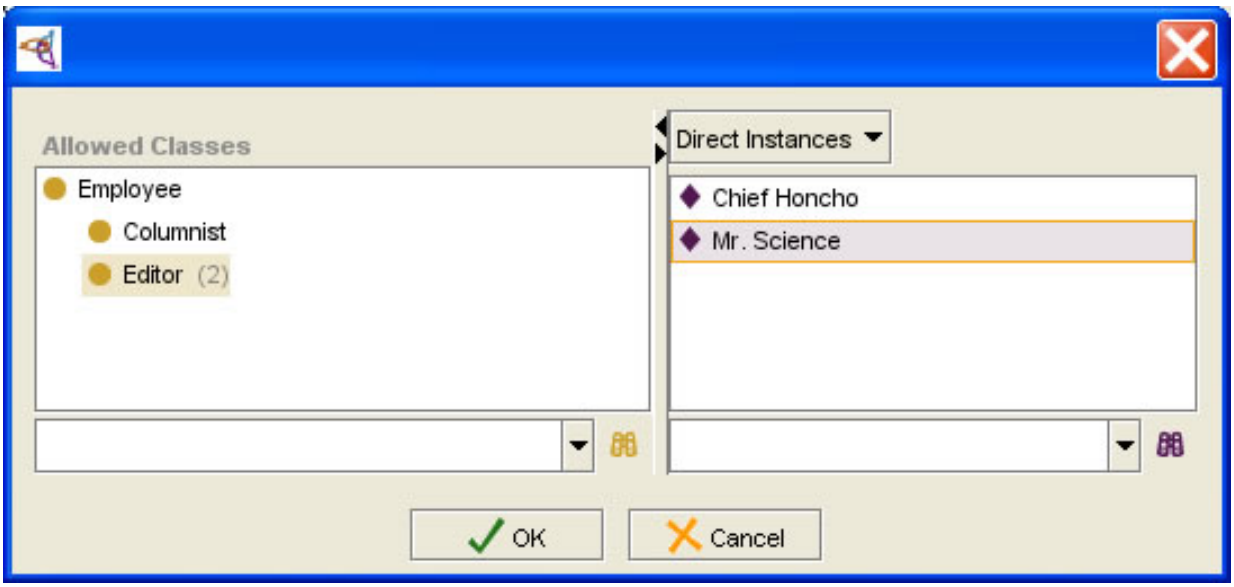
2. Select *Chief Honcho* in the Instance Browser. You will see the slots for *Chief Honcho* in the Instance Editor form, including the **Responsible For** slot. Notice that Protege-Frames uses the slot names in the instance form, but automatically converts underscores to spaces and capitalizes the first letter of each word.

3. Click on the **Add Instance** button at the top right of the **Responsible For** field.

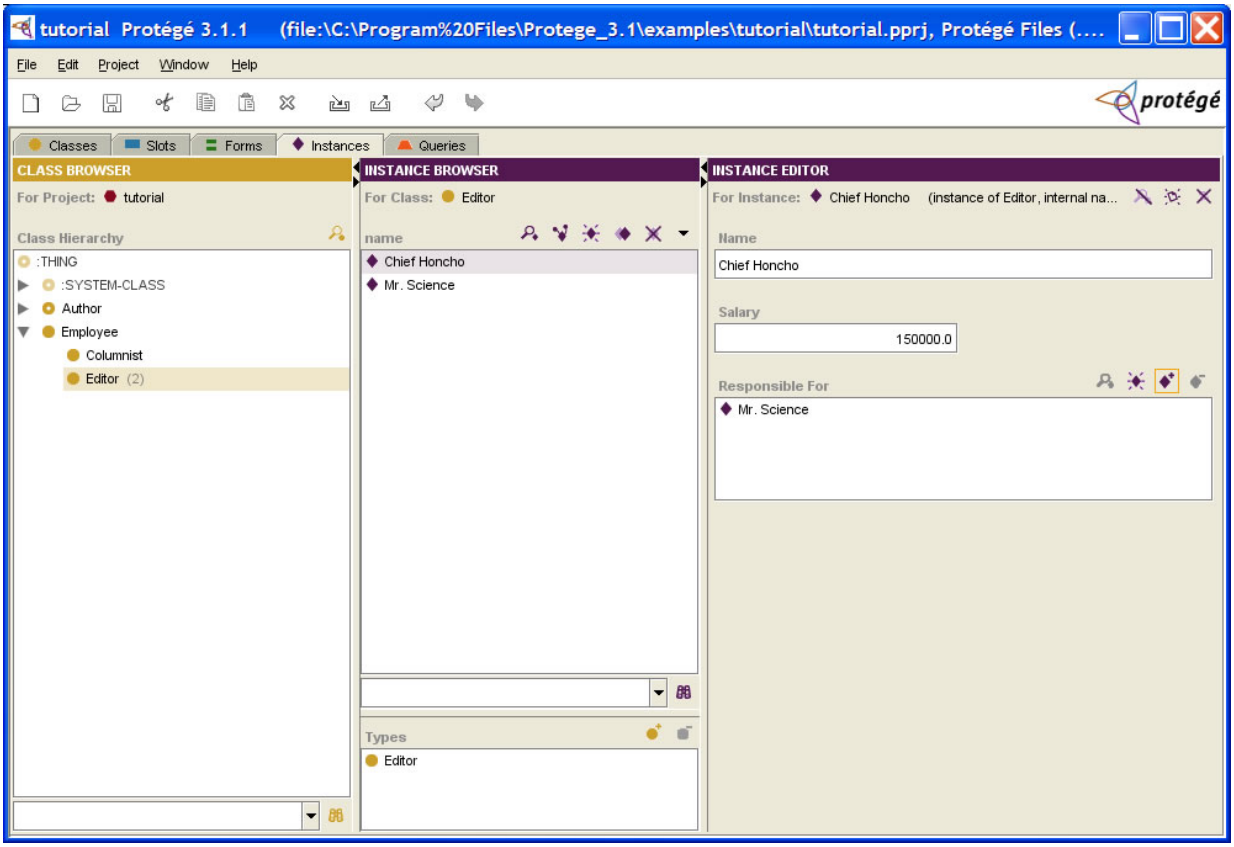
4. A dialog box opens to display two panes. On the left-hand side you see the hierarchy of allowed classes for the **responsible_for** slot.



5. Single-click on **Editor**. On the right-hand side of the dialog all of the instances of the **Editor** class are displayed. Select *Mr. Science* and click **OK**



6. You have now successfully created a relationship in your ontology specifying that Chief Honcho is "responsible for" Mr. Science.


tutorial Protégé 3.1.1 (file:\C:\Program%20Files\Protege_3.1\examples\tutorial\tutorial.pprj, Protégé Files (....
File Edit Project Window Help
protégé
Classes Slots Forms Instances Queries
CLASS BROWSER
For Project: tutorial
Class Hierarchy
:THING
:SYSTEM-CLASS
Author
Employee
Columnist
Editor (2)
INSTANCE BROWSER
For Class: Editor
name
Chief Honcho
Mr. Science
Types
Editor
INSTANCE EDITOR
For Instance: Chief Honcho (instance of Editor, internal na...
Name
Chief Honcho
Salary
150000.0
Responsible For
Mr. Science

Customizing a form

For each class in your ontology, Protege generates a default form that you can use to enter instance data. Forms contain a data entry field, or "widget" for each slot that is attached to a class. There are different types of widgets to handle the various slot value types, e.g. Protege assigns a "TextFieldWidget" to slots with String value types, an "IntegerFieldWidget" for slots with Integer value types, an "InstanceListWidget" for slots with Instance value types and cardinality multiple, etc.

If you do not like the default form that Protege generates for your class, you can customize it via the Forms Tab. Among other things, you can resize widgets, drag widgets to different locations on the form, hide widgets, and even select different widget types.

To dynamically see how the changes you make in the Forms Tab in the next sections are reflected in the Instance Editor, go to the Instances Tab, and double-click on *Chief Honcho* in the Instance Browser to display a separate, modeless Instance Editor window. Please note that if you created the slots for the **Editor** class in a different order than was outlined in this tutorial, your form may not look exactly like the screenshots in the following sections.

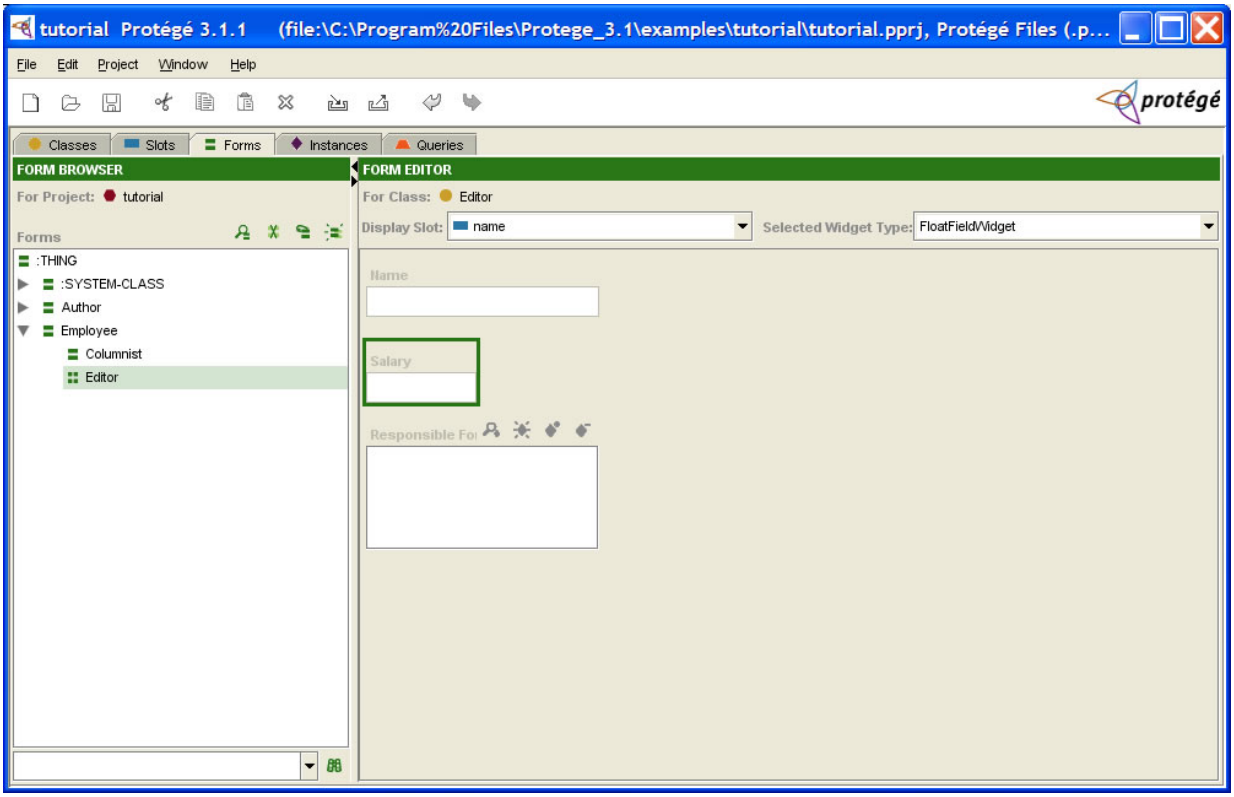
- Resizing a widget
- Moving a widget
- Customizing widget buttons
- Hiding a widget
- Displaying a hidden widget
- Using the default layout

Resizing a widget

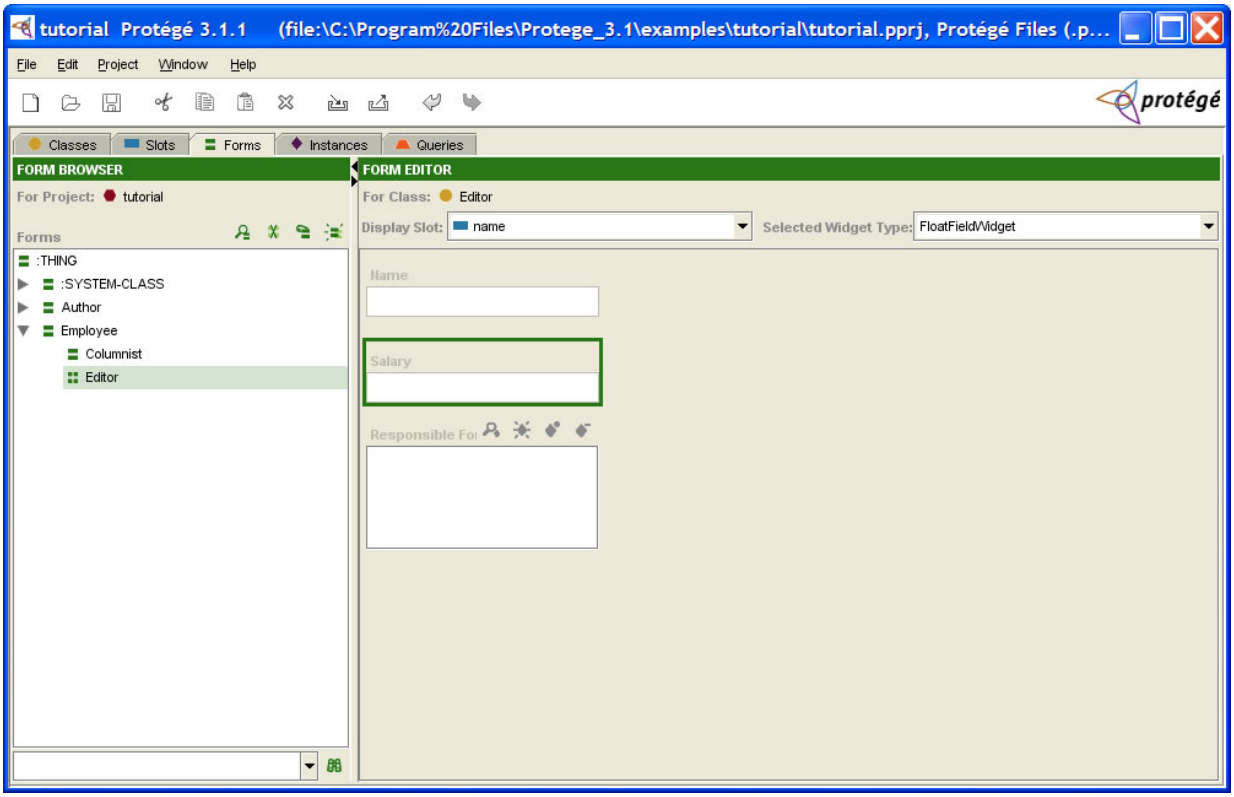
You can resize a selected widget by dragging any edge or corner. To resize a widget by dragging:

1. Click on the Forms Tab.

2. Make sure **Editor** is selected in the Form Browser on the left. Next, select the FloatFieldWidget for the **salary** slot by clicking on it in the Form Editor on the right-hand side. It is outlined in green to show it has been selected. Notice that the Selected Widget Type combo box at the top right tells you that this is a FloatFieldWidget, which is a widget used for entering floating point numbers.



3. Click on the right side of the widget and drag to extend it. Try to align the right side of this widget with the right side of the widget for the **name** slot.

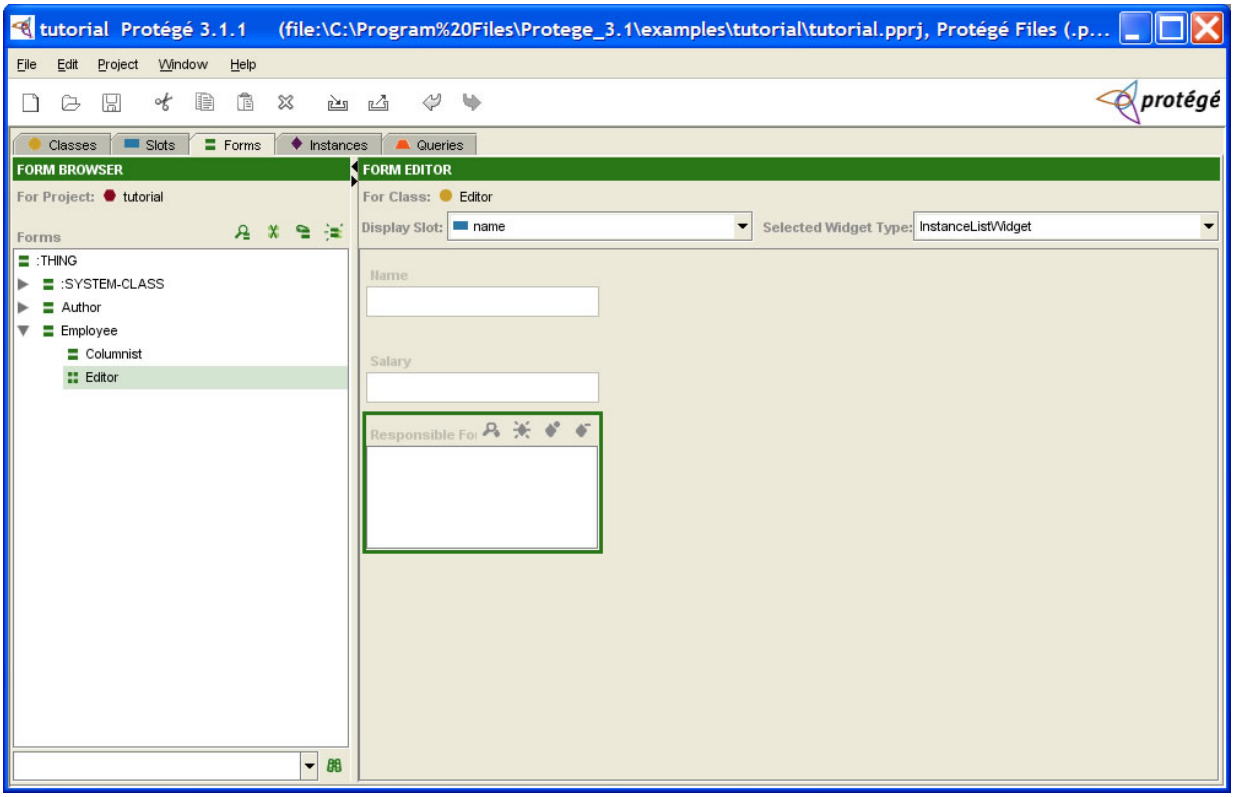


4. Notice that the icon in front of **Editor** in the Form Browser has changed. The new icon shows that the form for this class has been customized.

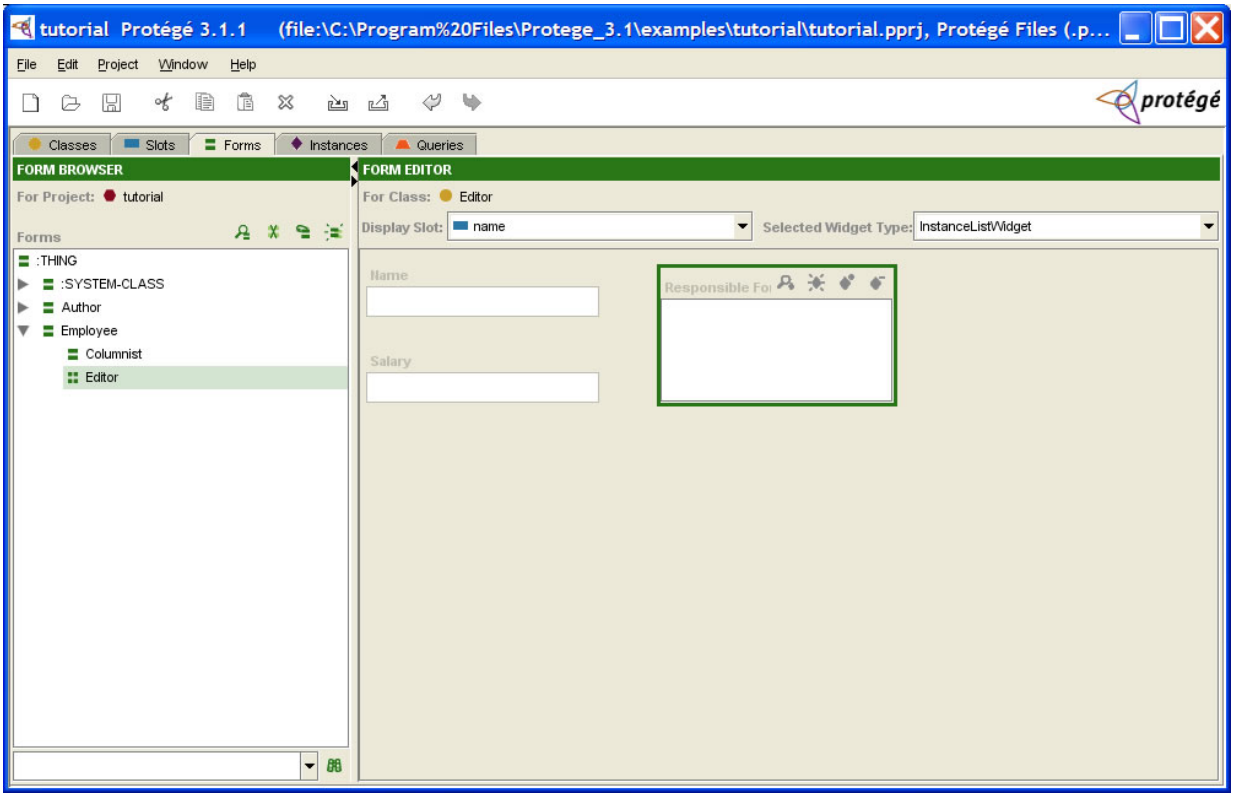
Moving a widget

You can also move any widget via dragging:

1. Select the InstanceListWidget for the **responsible_for** slot.



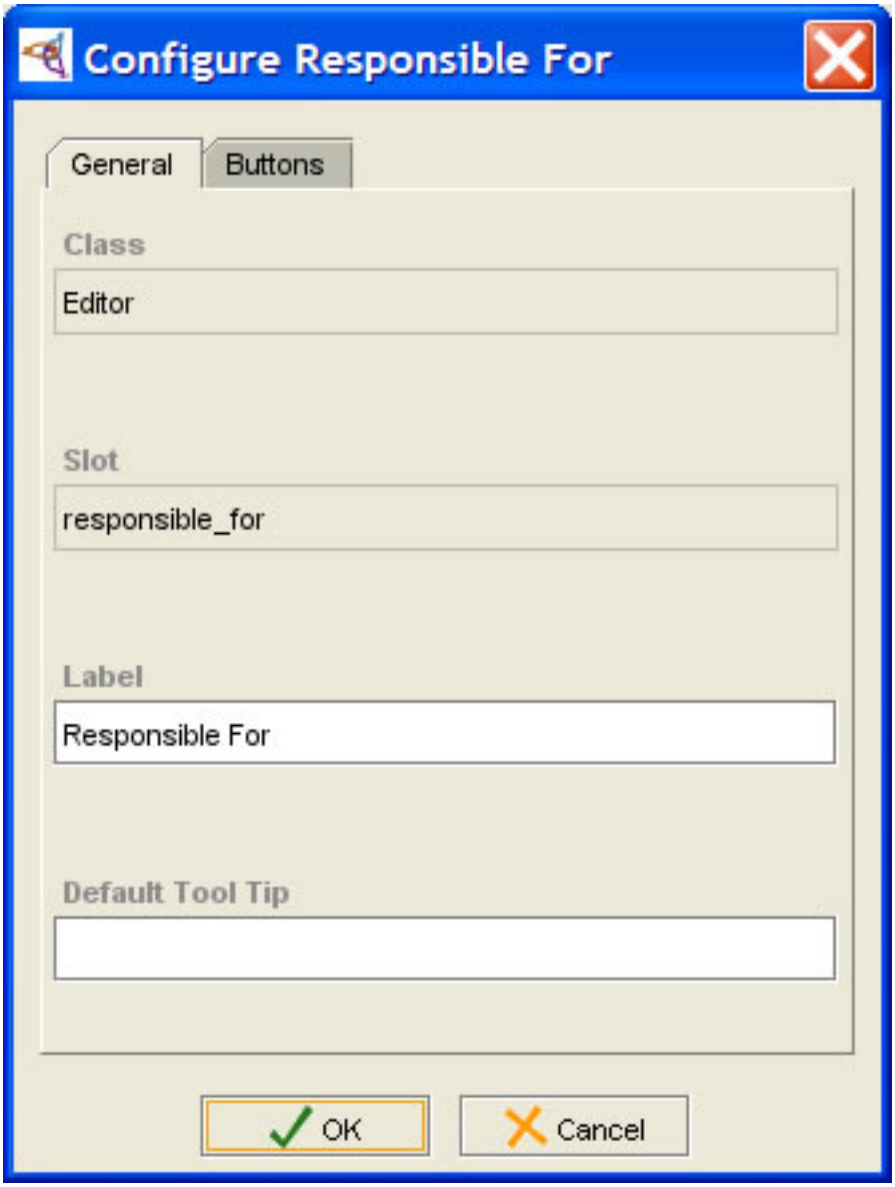
2. Drag it to the top right of the form, so that the top of the widget aligns with the top of the widget for the **name** slot.



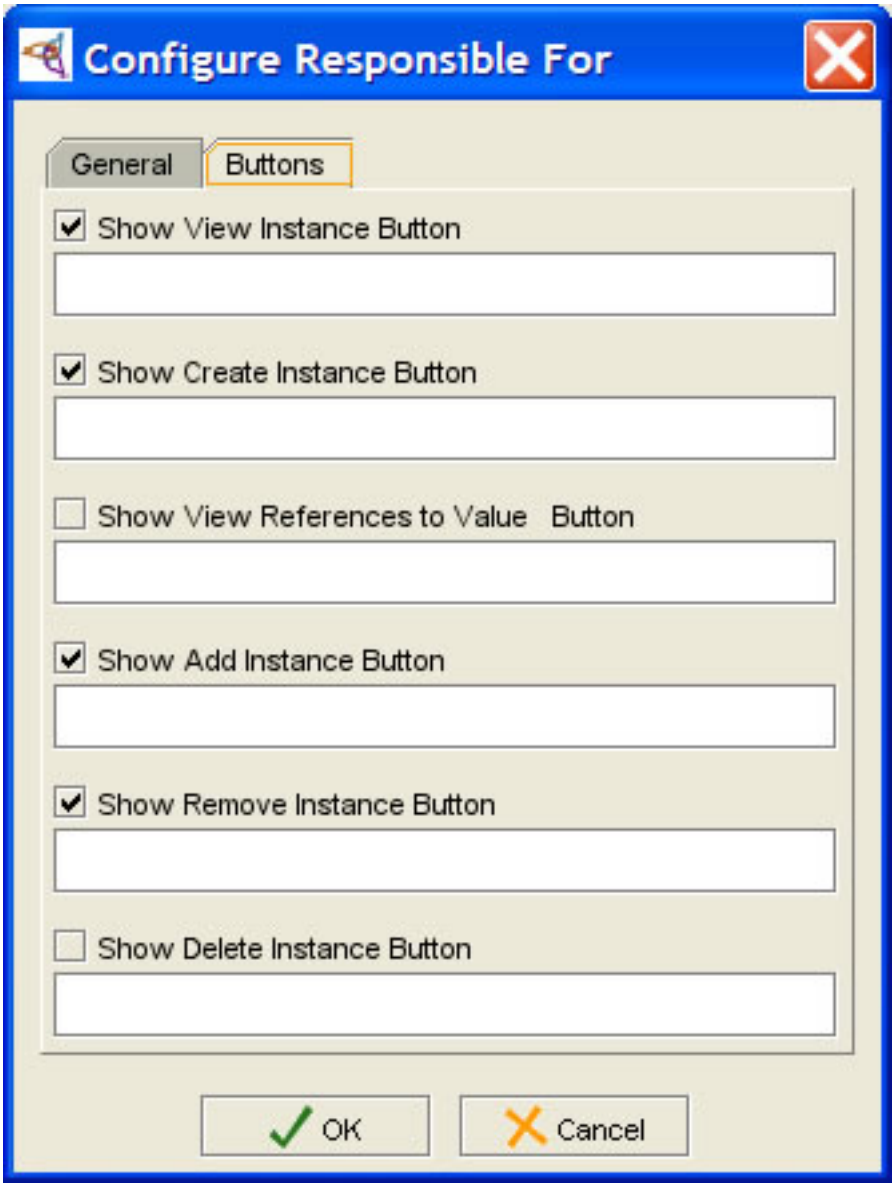
Customizing widget buttons

You can customize widgets to display a different label or a different set of buttons than the default. For example, you may want to be able to delete an instance from your project by clicking on a button in the InstanceListWidget that was assigned to the **responsible_for** slot. To display a delete button:

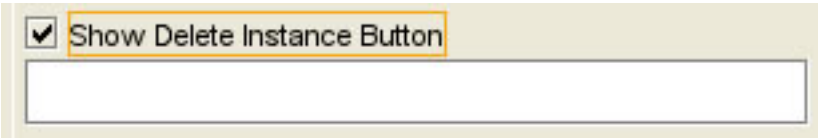
1. Double-click on the InstanceListWidget labeled **Responsible For** in the Form Editor.



2. Click the Buttons tab.



3. Click the check box in front of Show Delete Instance Button to select it.

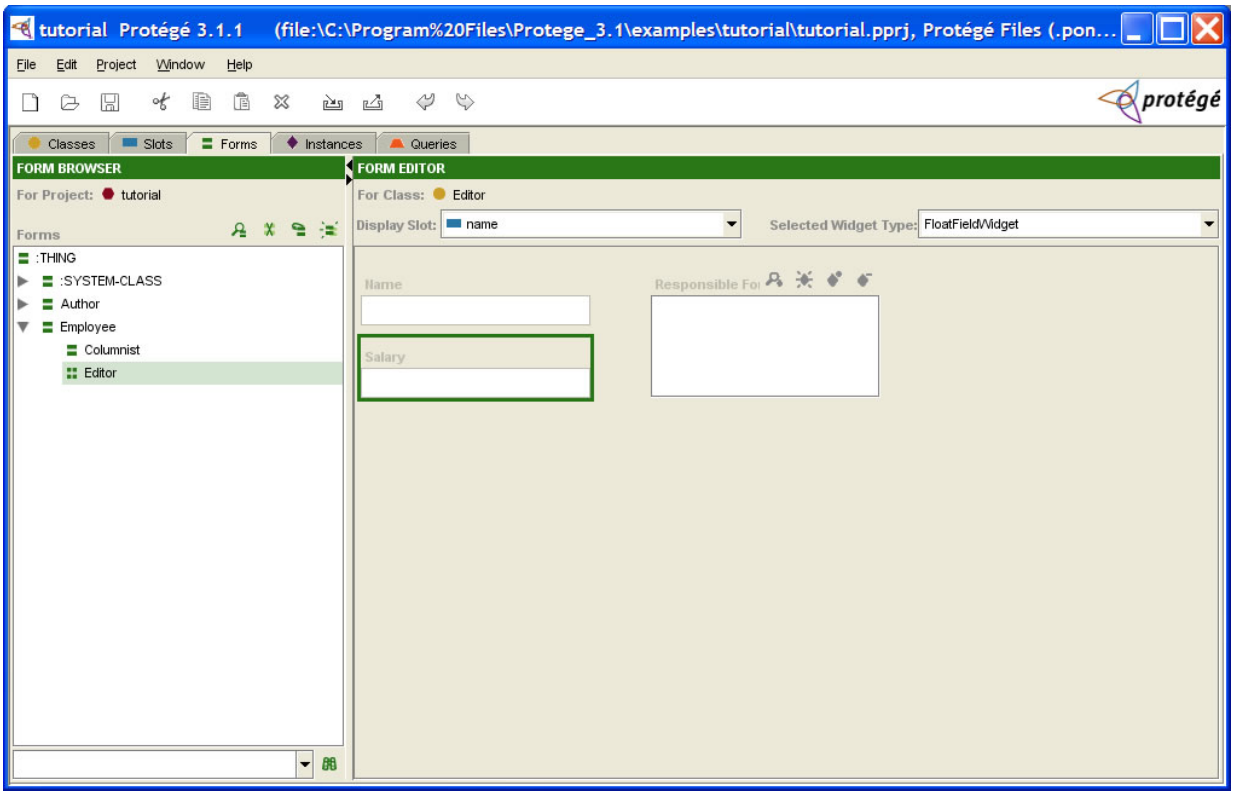


4. Click **OK**. The InstanceListWidget for the **responsible_for** slot now has five buttons.

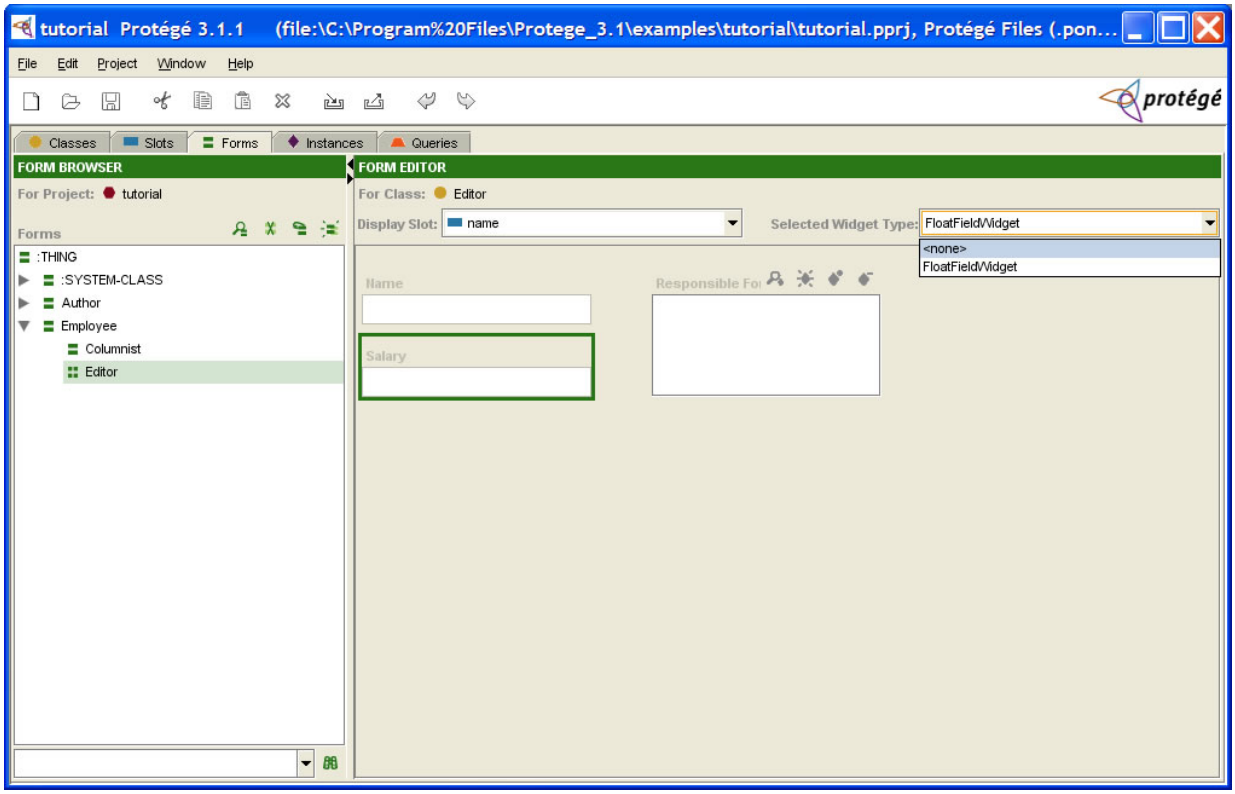
Hiding a widget

You can hide a widget so it cannot be seen in the Form and Instance Editors (this does *not* remove any information from your ontology). For example, you might want to hide the widget for the **salary** slot. To do this:

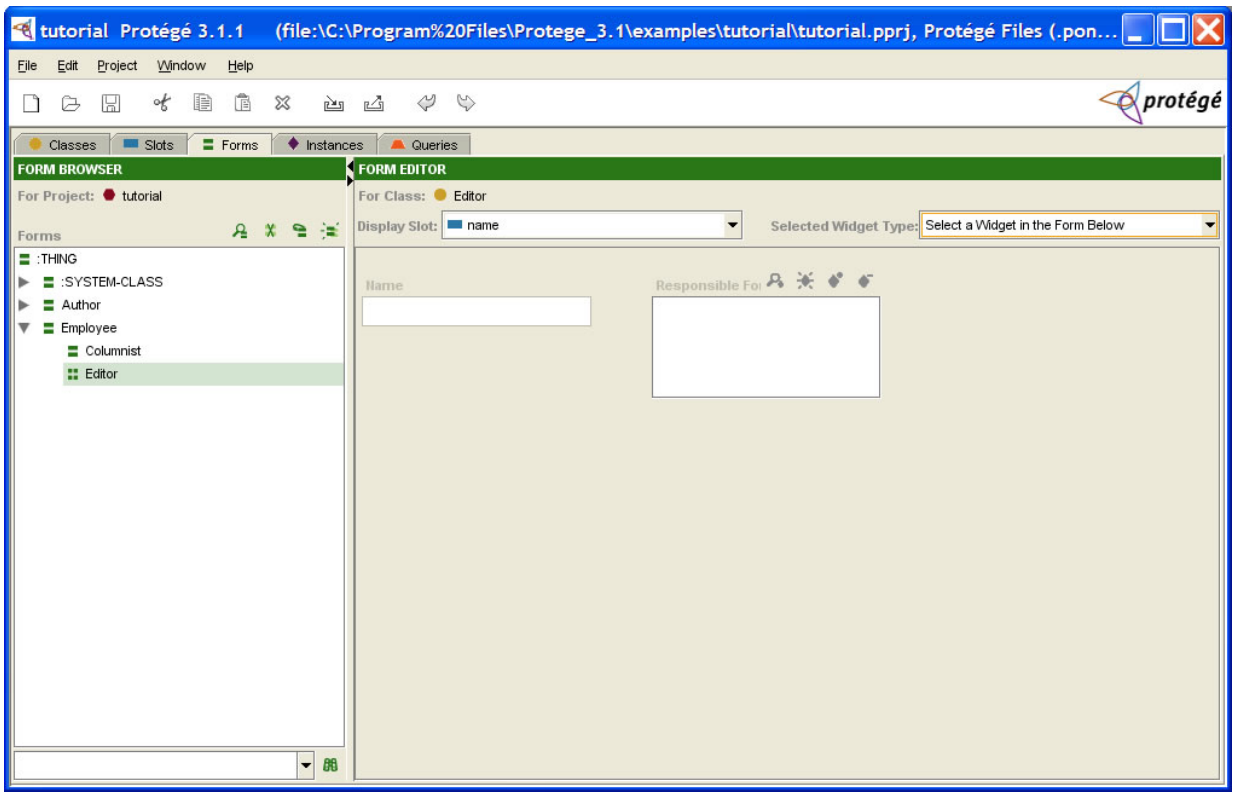
1. Select the FloatFieldWidget labeled **Salary** in the Form Editor.



2. Select "<none>" from the Selected Widget Type combo box.



3. The widget for the **salary** slot is no longer visible on the Form Editor.

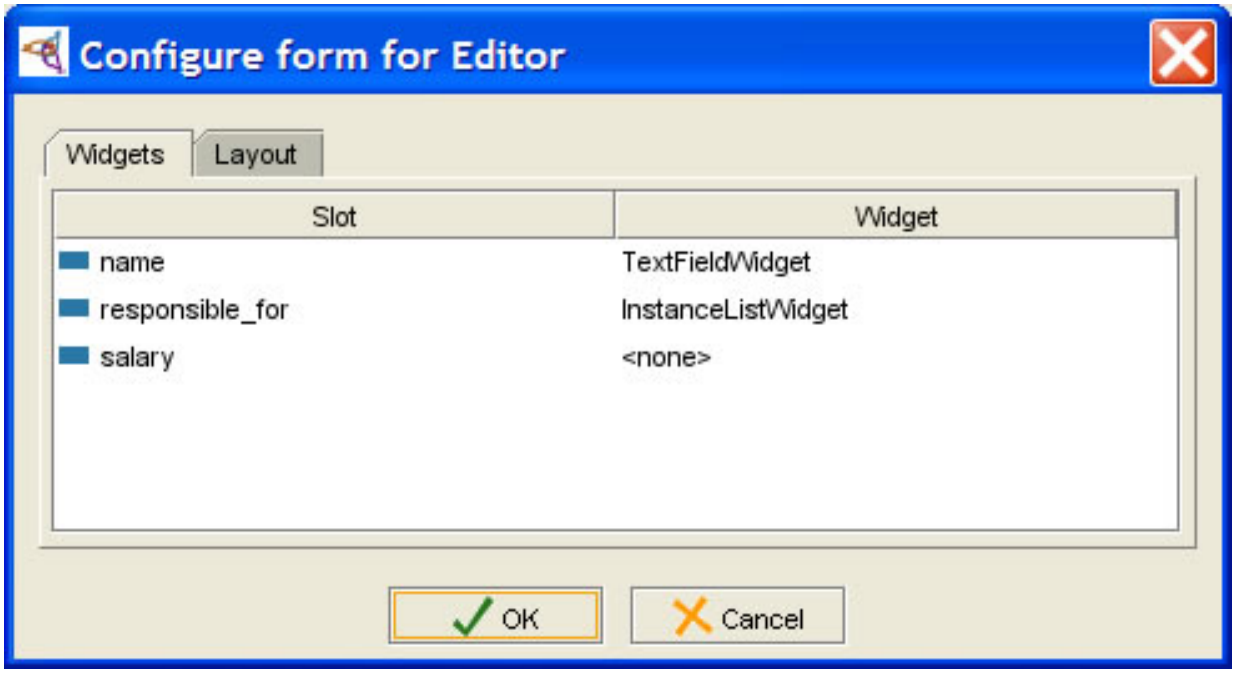


Displaying a hidden widget

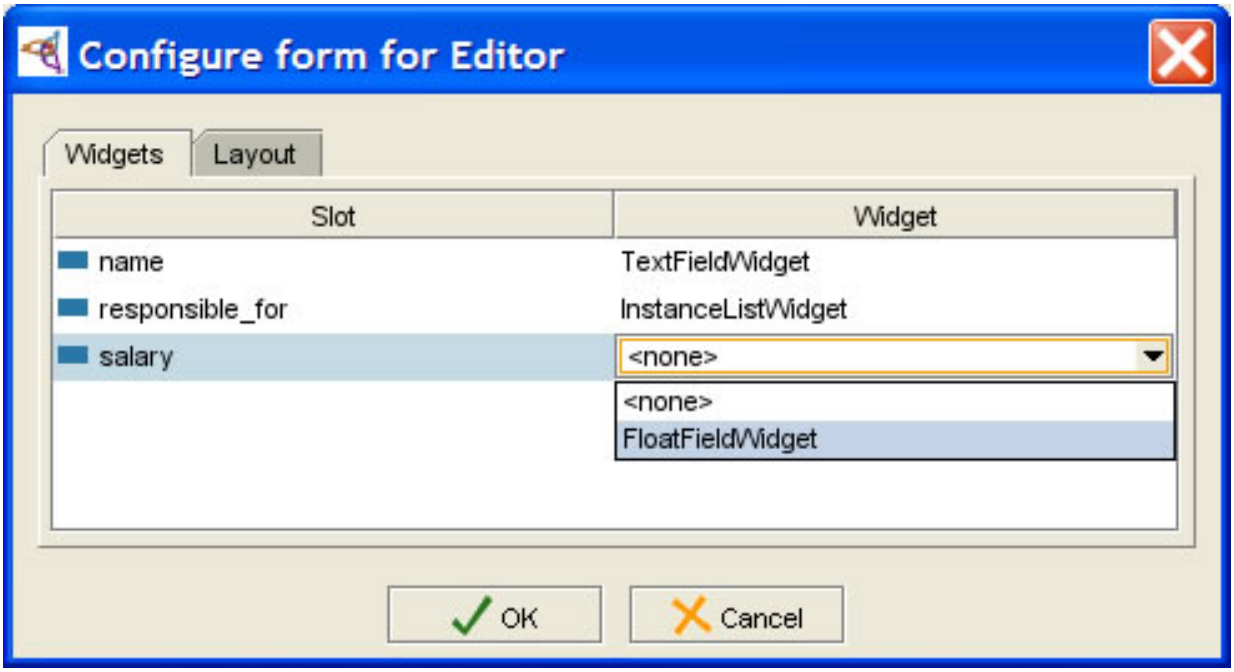
To restore a hidden widget to view:

1. In the Form Browser, click the **View Form Customizations** button at the top right of the Form Browser.

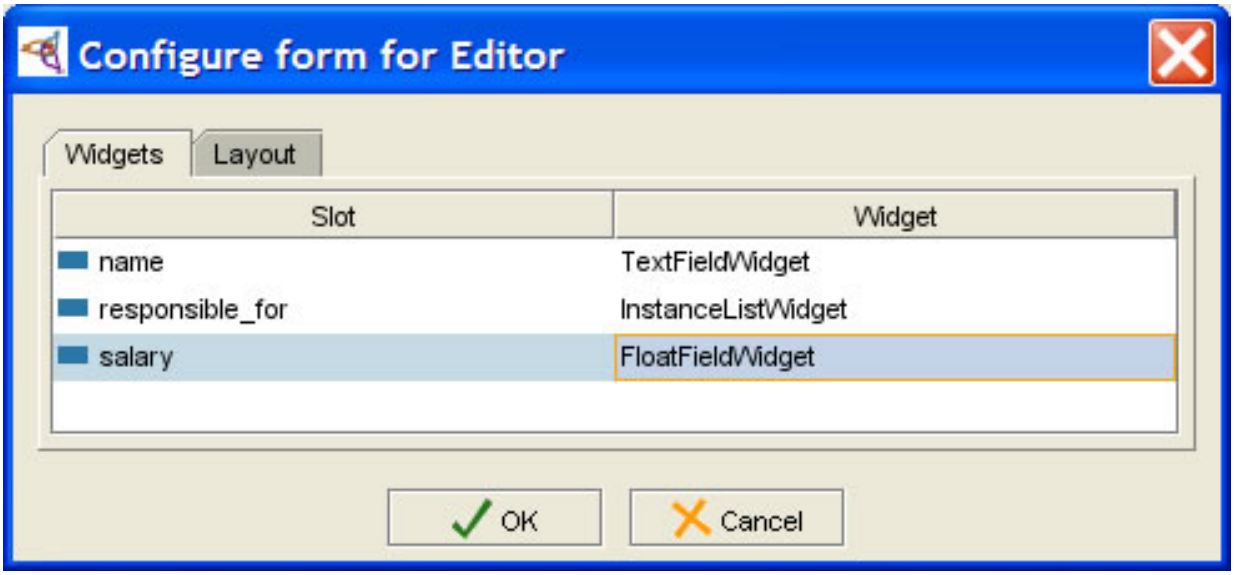
2. In the Configure form dialog box, you can see a list of all the slots and their corresponding widgets.



3. Click on "<none>" in the right hand column after **salary** to display different widget options for this slot, and select FloatFieldWidget.



4. Click **OK**.



5. The FloatFieldWidget for the **salary** slot is now visible again in the Form Editor.

Using the default layout

If you are unhappy with any form customizations that you have made, Protege-Frames provides a short-cut to rearrange all the visible widgets on a form into a standard order, accommodating their current size.

To have Protege-Frames auto-arrange all the widgets back to the default layout:

1. Make sure **Editor** is selected in the Form Browser.
2. Click the **Remove Form Customizations** button at the top right of the Form Browser.
3. The form adjusts to show the default layout. If you have hidden any widgets, they will be redisplayed. Notice also that the icon in front of Editor has changed back to the default form icon .

Creating and saving a query

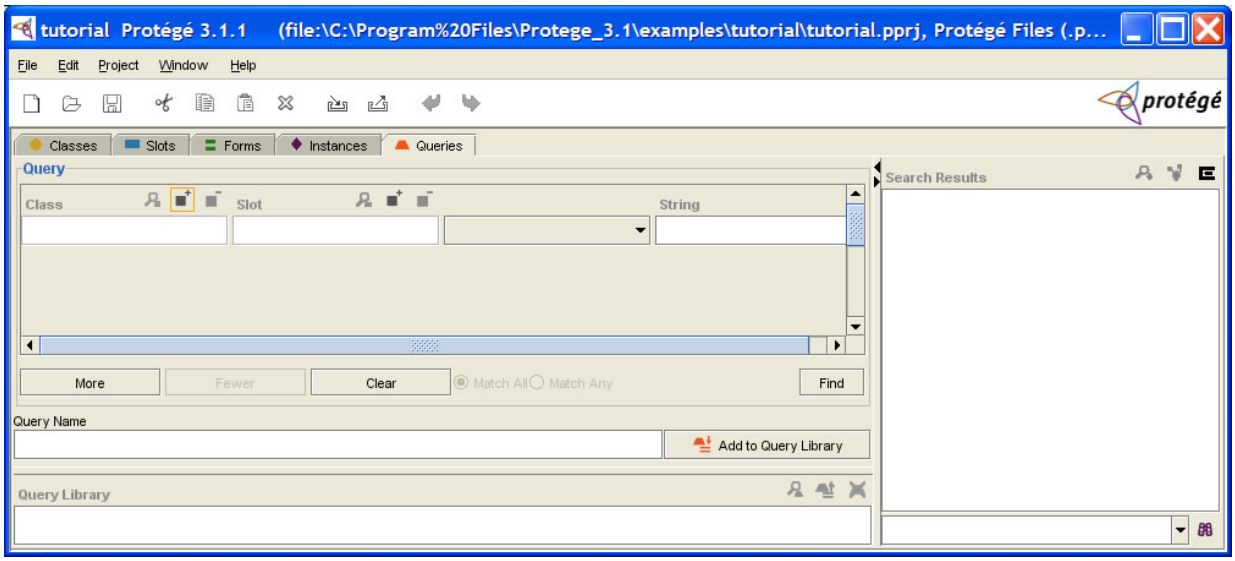
The Queries tab allows you to query your project and locate all instances that match criteria you specify. To create a query, you must select one or more classes, and one or more slots within that class. You can also save queries in the Query Library for future recall.

- Creating a query
- Running a query
- Saving a query
- Retrieving a query

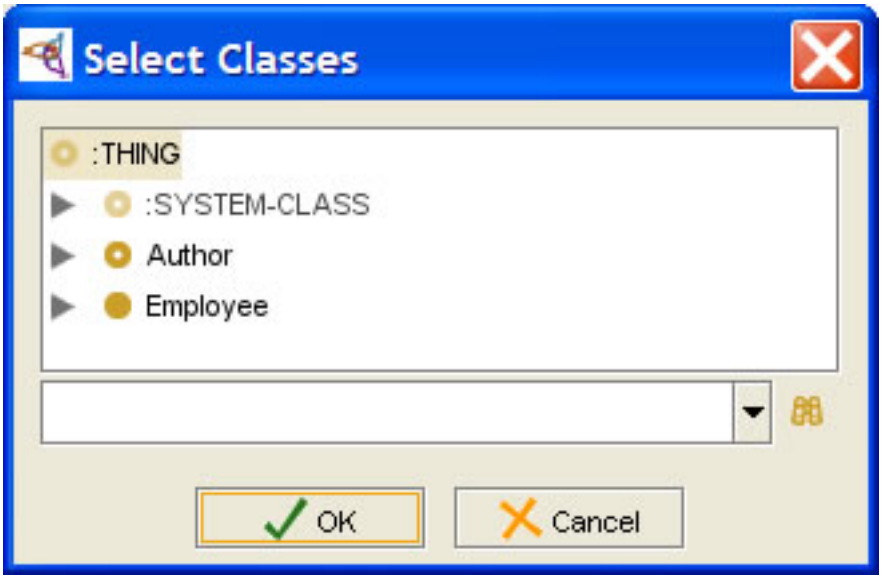
Creating a query

Suppose you are interested in locating all employees who have a salary greater than $75,000 dollars a year. To create the query:

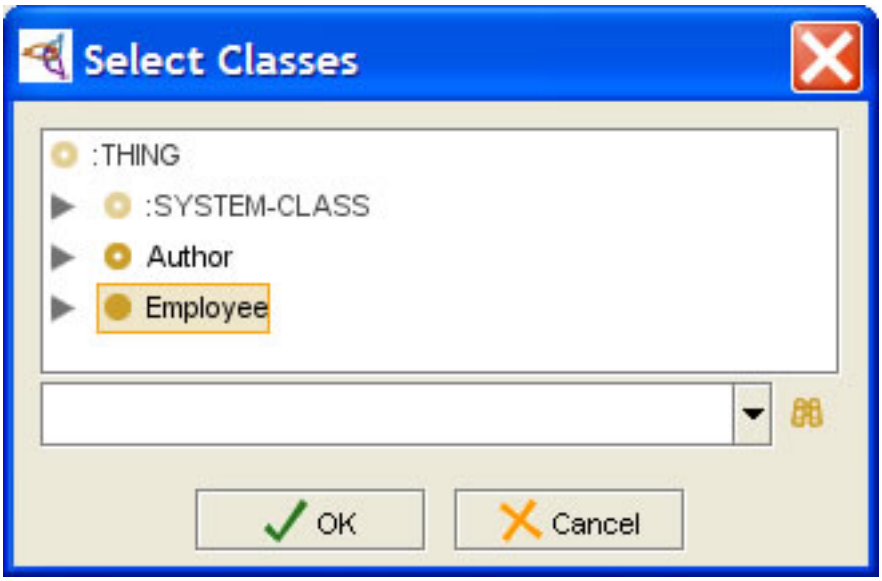
1. Click on the Queries tab.



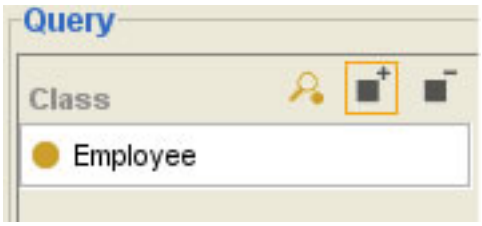
2. Click the **Select Cls** button above the Class text box in the Query pane.



3. Select **Employee** from the Select Classes dialog box, then click OK.

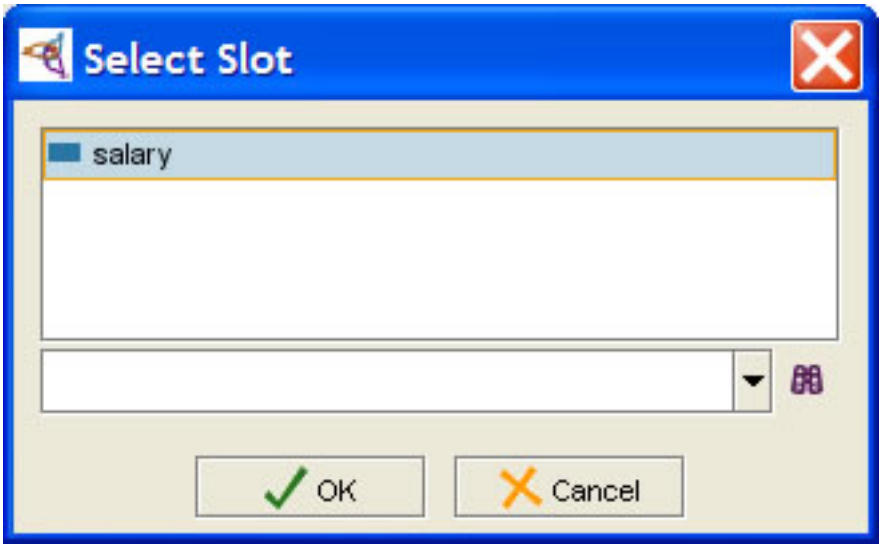


Employee is now displayed in the Class text box.

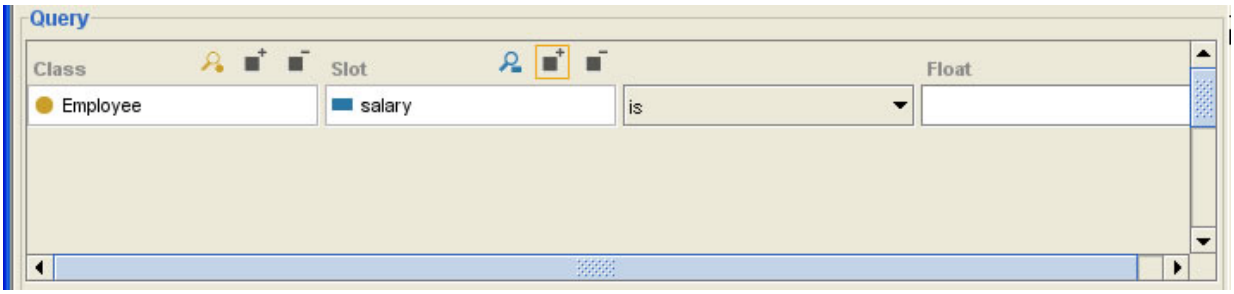


4. Click the Select Slot button above the Slot text box.

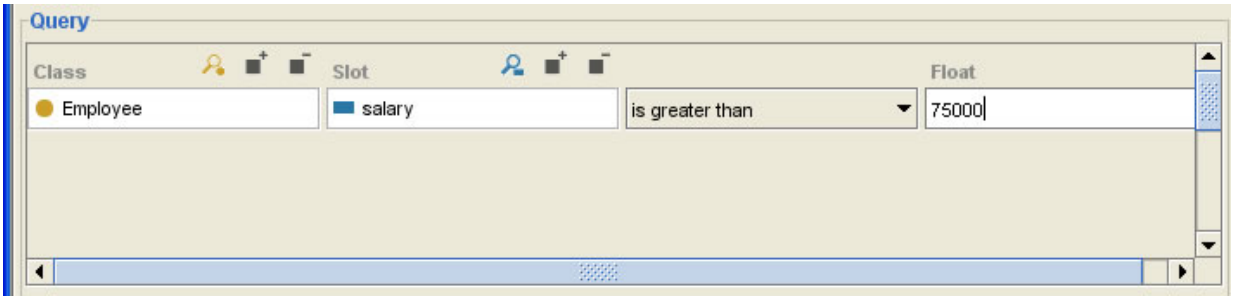
5. Select **salary** from the Select Slot dialog box and click OK.



The menu to the right of the Slot text box is now active, and the text box at the far right of the window reminds you that the value type of the slot you have selected is a float.



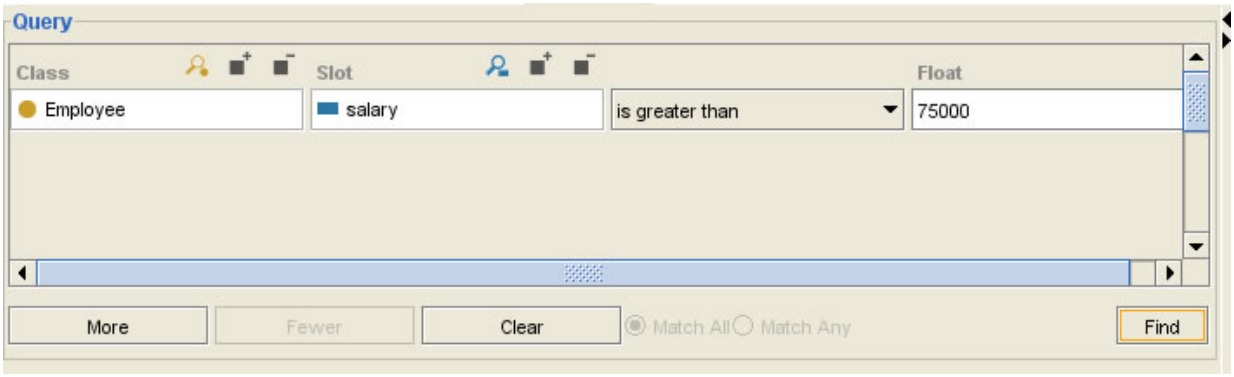
6. Select "is greater than" from the drop down menu. Next, enter "75000" in the Float text box.



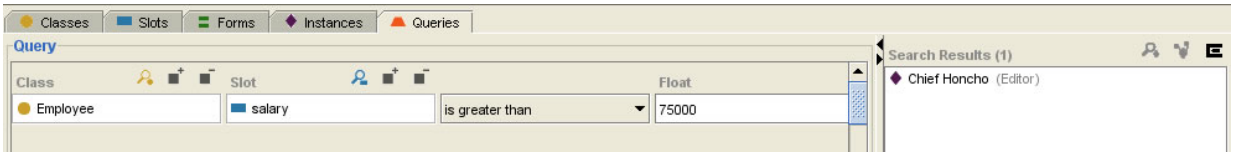
Running a query

Now that you have set up your query, you can run it and view the results.

1. To run the query, click the Find button near the bottom right of the Query pane.



2. The results are shown in the Search Results pane at the far right. If you cannot see the results, you may need to enlarge your window or move the slider bar.

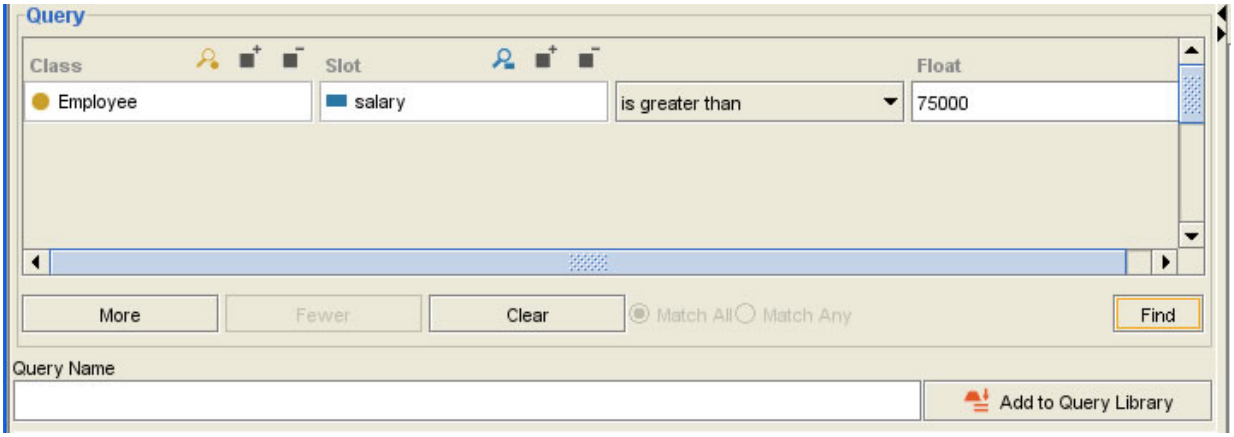


Note: You can double-click on any instance in the Search Results pane to view the Instance Editor.

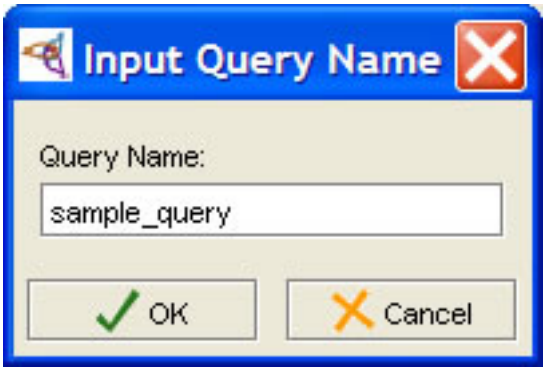
Saving a query

You can save any query before or after it is executed. To save a query in the Query Library:

1. Click the **Add to Query Library** button to the right of the Query Name text box.

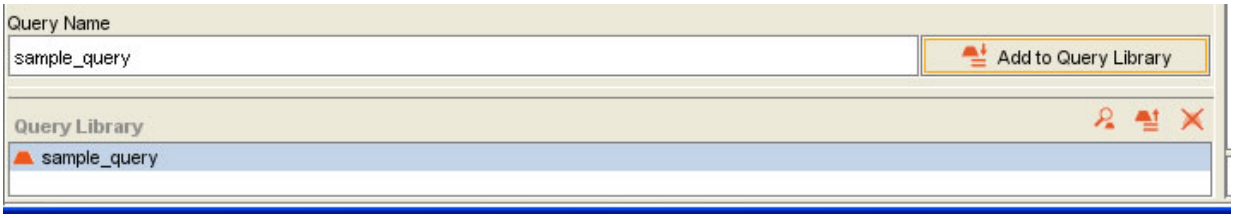


2. Type "sample_query" in the Input Query Name dialog box.



Note: You may also type a query name directly into the Query Name text box on the main window and click the **Add to Query Library** button.

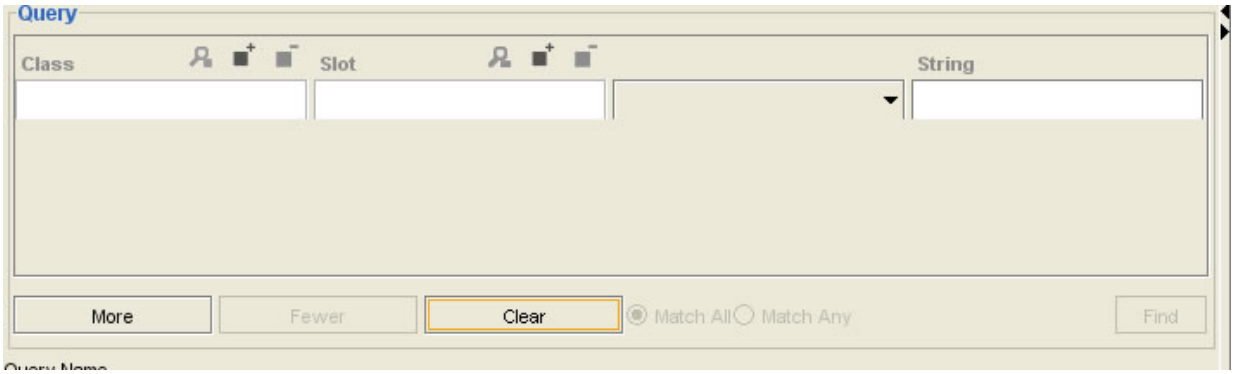
3. Click **OK**. The name is displayed in the Query Name text box and the query is listed in the Query Library pane below.



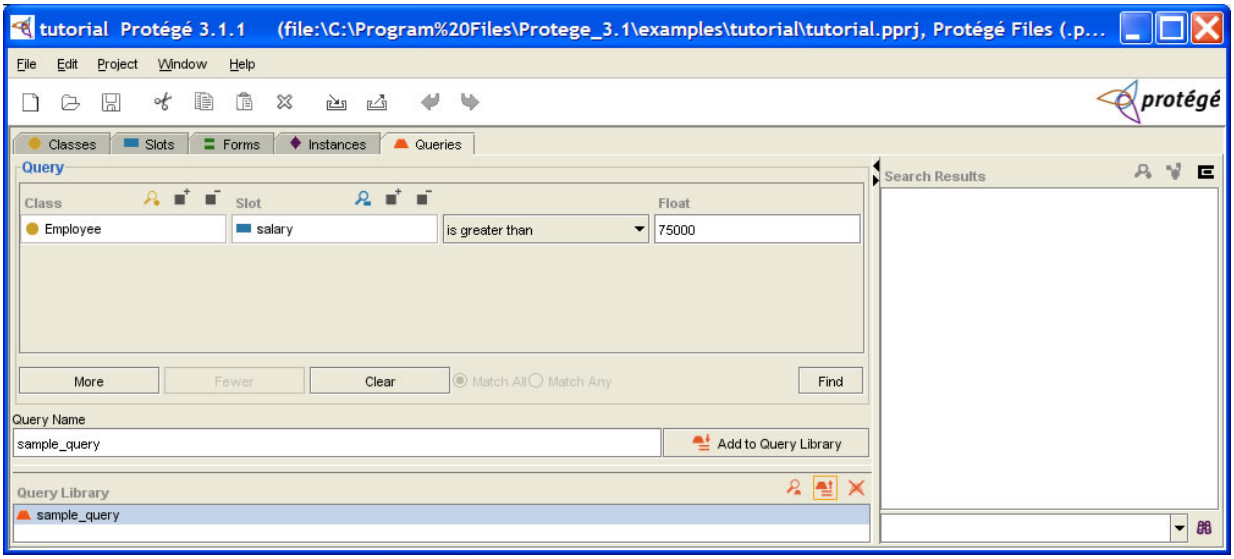
Retrieving a query

To retrieve a saved query, you can select it from the Query Library:

1. First, because we are running the same query again, click the Clear button to clear the query results. Otherwise, you will not see a change.

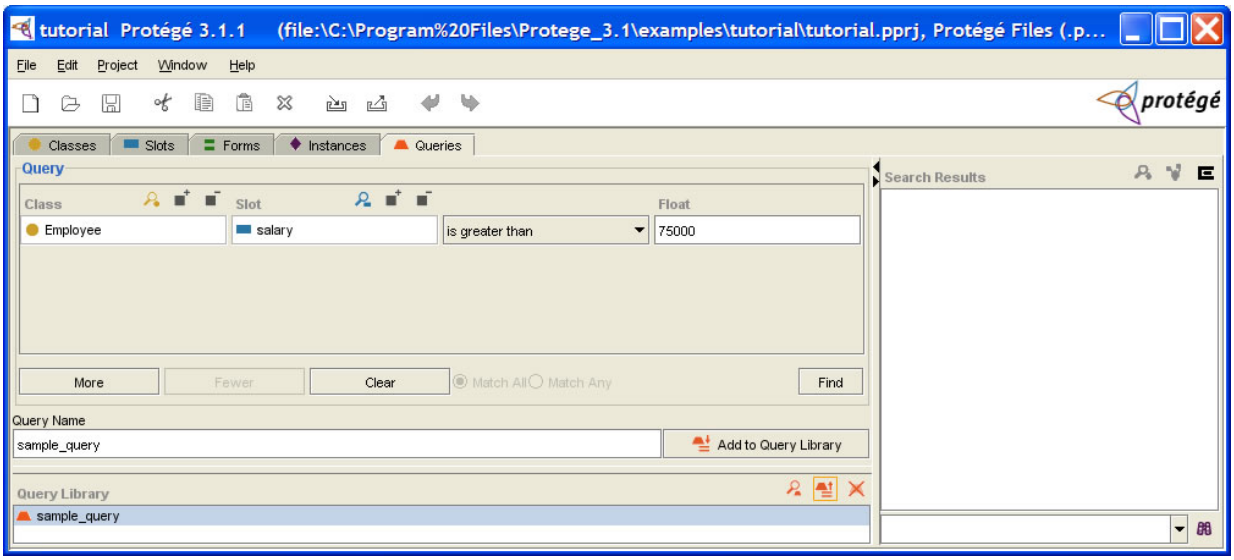


2. Select *sample_query* in the Query Library at the bottom of the screen. (If you cannot see the Query Library, you may need to enlarge your window or move the slider bar).

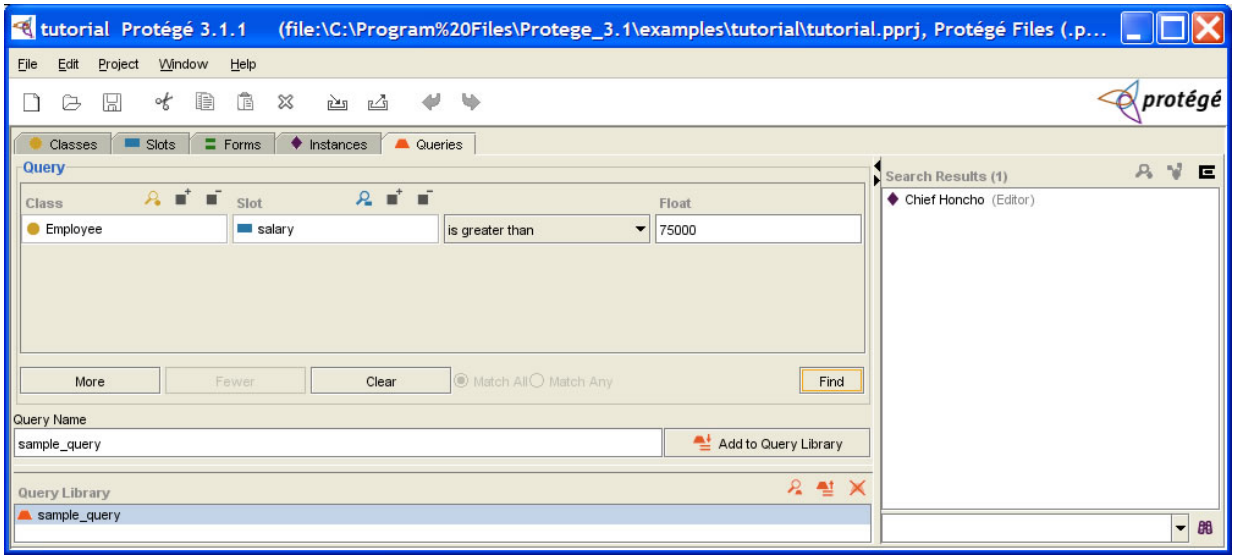


3. Click the **Retrieve Query** button to retrieve the query.

4. The saved query is now displayed at the top of the window. If you wish, you can now alter the query. You can also combine queries by clicking the **More** button.



5. To run the query, click the Find button.


tutorial Protégé 3.1.1 (file:\C:\Program%20Files\Protege_3.1\examples\tutorial\tutorial.pprj, Protégé Files (.p...
File Edit Project Window Help
protégé
Classes Slots Forms Instances Queries
Query
Class
Slot
Float
Employee
salary
is greater than
75000
More
Fewer
Clear
Match All
Match Any
Find
Query Name
sample_query
Add to Query Library
Query Library
sample_query
Search Results (1)
Chief Honcho (Editor)