



Developing Achievo Extensions

Part I – Basics

Ivo Jansch <ivo@achievo.org>
© 2002

Last modified: August 15, 2002

Contents

1 Pizza	4
1.1 Data model.....	4
2 Achievo modules	6
2.1 Creating the module	6
2.2 Installing the module in Achievo	7
3 The Pizza node	8
3.1 Creating the node class	8
3.2 Adding the new feature to the menu.....	10
3.3 Testing the node	11
3.4 Finishing the Pizza node.....	12
4 Security	14
5 Improving user-friendliness	15
5.1 Navigation improvements	15
5.2 Adding search functionality	15
6 Conclusion	17

Introduction

If you've been using Achievo for a while, and you want to develop your own extensions to the application, this is the guide for you!

After reading this guide (and working through the examples), you will have an understanding of how Achievo works and how you can extend it to meet the needs of your organisation. You will find how incredibly easy it is to develop extra features for Achievo.

From the start of the project, Achievo has been developed with extension in mind. A project management suite is typically an application that never completely fits to an organisation. Each organisation has its own way of doing things. Each organisation has its own workflow. With this in mind, we created a powerful backend that allows you to completely adapt Achievo to your needs.

In this guide, we will teach you the basics of Achievo's architecture. Later, in Part II we will dive deeper into the internals of Achievo, and teach you how you can make your extensions even more powerful and feature-rich. Finally in Part III, you will learn how to change existing Achievo functionality, without hacking the original Achievo code, and the most powerful features of the backend.

If you encounter any problems with the examples in this guide, or if you have any questions or comments, just drop me a mail (ivo@achievo.org). This is the first version of the guide, so some things may not be clear enough yet. I welcome any suggestions for improvements.

Prerequisites

In order to develop extensions for Achievo, you should have some knowledge about programming in PHP, since that is the language Achievo was written in. You don't need to be a guru, but knowledge of the basic concepts will help you understand this guide. If you have never used PHP before, find a tutorial on the web. The official PHP site, www.php.net, is a good starting point if you want to learn more about PHP.

Experience with databases is recommended. It's certainly not necessary to be an expert in SQL. (You will find out how little SQL you will actually have to write when developing Achievo extensions!) You should have some basic knowledge about databases however. We need to create some tables, so you need to know at least a bit of SQL. In this guide we concentrate on MySQL as a database server, but the instructions are quite similar for other databases.

We assume you have a working Achievo installation that you are able to develop on (Achievo 0.8 or newer). Don't use your company's production installation, or production database! Your fellow Achievo users will be very unhappy if you do that. Just create a copy of the database and a copy of the Achievo installation. (Don't forget to update Achievo's `config.inc.php` file so Achievo will use the database copy instead of the original, and don't forget to grant correct privileges to the databaseuser.) If you installed a new database, make sure you have some initial data in it: at least one security profile and a user.

Finally, all you need is a text editor. I can recommend SciTE (www.scintilla.org/SciTE.html), which is a free source code editor which features syntax highlighting. But of course you can use the text editor you are most comfortable with.

So. If any of the above did not scare you away, we are now ready to start.

1 Pizza

Pizza? Yes. Pizza.

The example we are going to use in this guide is a Pizza Database. This example is chosen for a few reasons. First, it's going to show you that you can add virtually anything to Achievo. This illustrates that Achievo is not branch-specific, and extensions for all kinds of businesses can be added. Second, it is a feature that is unlikely to ever exist in Achievo for real, so it can be used perfectly as an example application. And finally, I was hungry when I wrote this guide. If you like the guide, buy me a pizza sometime. :-)

Let me explain to you first what we're going to develop. Suppose we own a Pizza Restaurant, and for some obscure reason, we use Achievo to track time we spent on baking pizzas.

An ideal Achievo extension for our little restaurant would be to have a database where we can store what Pizzas we serve (to be used later to generate menu cards, keep track of orders etc.)

So what do we need? A feature in Achievo where we can manage Pizzas. Management of Pizzas includes saving new pizzas, editing existing ones and deleting pizzas we no longer serve.

1.1 Data model

Once you know what you want to develop, you will need to design a data model. This means, we're going to think about the database tables that we will need for our feature.

Let's think of what we want to store about a pizza. A pizza of course has a name. It also should have a description, and a price. Maybe we also want to know the date that we entered the Pizza in the database. As you probably already know: in a database, a record should also have a unique id. While the name of a pizza might be unique, it's a good idea to use a separate numeric id as primary key. (Read some database tutorials if you want to know why.)

Let's leave it at that for now, to keep the example simple. (In Part 2, I will teach you how you can keep track of orders, how you can manage ingredients and some more advanced features, but to understand the basics, the fields I just mentioned are sufficient.)

So we should create a table in the Achievo database called 'pizza' with the fields we just described. This is what the table should look like:

Pizza	
Fieldname	Type
id	integer
name	string
description	text
price	float
entrydate	date

When we create the table in the database, using reasonable values for field lengths etc., the SQL script might look like:

```
CREATE TABLE pizza(  
  id int(10) NOT NULL,  
  name varchar(50),  
  description text,  
  price decimal(5,2),  
  entrydate date,  
  PRIMARY KEY (id)  
);
```

This script is for MySQL databases. If you use a different database, you might need to modify the script a bit to accommodate for some database dependent SQL statements.

After we run this script on the database, we are ready to start development.

2 Achievo modules

Everything we add to Achievo will be located in a so-called module. A module is a set of source files in a directory that belong to each other, and that together form the implementation of a feature.

We will not modify the original source code of Achievo, nor will we put our code between the existing Achievo code.

This happens for a good reason. When new Achievo versions are released, we want to be able to use our module with this new version, without having to write our code all over again. This also keeps our code nicely together. And finally, it makes it easier to distribute features you created. (We will create a module repository on www.achievo.org where you can upload your module, so other users can benefit from it.)

2.1 Creating the module

A module is stored in a directory. You could use the `modules/` directory in Achievo to store your module, but I prefer to reserve that directory for standard Achievo modules, and put my own modules in a directory outside of Achievo's directory structure.

Suppose you have Achievo installed in `/var/www/html/achievo-0.8/`, we could create the directory `/var/www/html/achievo_modules/` to store our own modules. It doesn't really matter how you name the directory and where you put it, as long as the directory is accessible by your webserver. So make sure you set the correct permissions on the directory once you created it. The directory has to be readable by the user that runs your webserver (usually something like 'nobody', or 'www').

We will call our module 'pizzaman' (short for 'pizza manager'). The first thing we do when creating the module is create the directory. It is best to name the directory after the module, so we create the directory `/var/www/html/achievo_modules/pizzaman/`.

An important rule when developing an Achievo module is that every module should have a file named `module.inc`, which contains a definition of the module. This definition specifies, among other things, what menu items should appear in Achievo when you activate the module.

For now, we will create a very basic implementation of a module, which we will complete later on.

Create a file called `module.inc` in the `achievo_modules/pizzaman` directory and put the following code in the file (don't worry if you don't understand it, I will explain it after you created the file):

```
<?php
    class mod_pizzaman extends atkModule
    {
    }
?>
```

You should already know that the `<?php` and `?>` markers indicate that we are writing a PHP source file. What we did here is create a class (yes, Achievo is ‘object-oriented’!) with the name `mod_pizzaman`. The name of this class is always ‘*mod_*’ plus the name of the module.

What we also specified is that this class extends `atkModule`. The `atkModule` class is what you might call ‘the mother of all modules’. If you derive your module class from `atkModule`, you actually prepare the module for use in Achievo. Achievo can now interface with your module. Later on, we will add some necessary code to the module, to indicate to Achievo what menu-items to show etc. For now, this bit of code suffices. Next, we will activate the module in Achievo.

2.2 Installing the module in Achievo

Open Achievo’s configuration file (`config.inc.php`), and look for a section named ‘external modules’ (near the end of the file). Add the following line to activate your module:

```
module("pizzaman", "../achievo_modules/pizzaman/");
```

This will tell Achievo to load the module ‘pizzaman’ and load it from the specified directory. The directory is relative to Achievo’s own directory, but you may also specify an absolute path here. Once the file is saved, we have activated the module. (Don’t forget the trailing slash at the end of the directory name. The example in `config.inc.php` doesn’t have this, but that’s an error.)

You may instead put this line in the file `modules/config.modules.inc`, but again, I prefer to keep this directory for ‘official’ Achievo modules. This will also help you when upgrading, since you usually keep your configuration file when you switch to a new version of Achievo.

Before we continue, I first want you to change the file `atkconf.inc` in the Achievo directory. Find the entry named `$config_debug` and change its value to 1. This will turn on the debugger, which is helpful when developing extensions. With the debugger turned on, Achievo will show you a lot of information, including error messages and warnings if there are any.

You may now browse to your Achievo test installation with your favorite browser. Nothing happens yet, since we still haven’t implemented any functionality. But at least you should not get any error messages. If you see errors or warnings, revise the steps taken so far. Notice the lines that appear near the bottom of the screen: they contain the debug information. It’s a time-stamped log of what happens when a screen is rendered.

If everything went smooth, we now get to the point of this guide: implementing the user-interface for our pizza database!

3 The Pizza node

The what?

Before you can understand the title of this chapter, I should tell you what a node is. A node is Achievo's term for a class that implements an 'informational unit'. For example, projects, customers and activities are pieces of information in Achievo. Every type of information is represented by a node class. There is a class for managing projects, for managing customers and so on.

A node defines how information is structured. It tells the system how to handle records, and how to create a user-interface for managing the information. For example, the projects node tells Achievo that a project is made up of an id, a name, description etc. In other words, the node class is the link between the database table and the user-interface. You will see later on that we don't even need to implement a user-interface. If the nodes are defined correctly, the user-interface is automatically generated for us by the application.

People familiar with Java might compare the node concept with Java-Beans. Although a node is completely different from a bean, the concepts are similar.

Every type of information is represented by a node class, so what we have to do is create a node class for our Pizza table.

3.1 Creating the node class

We first have to determine a name for the node. It is best to use a name that describes the information that the node represents. The node that represents project management is called 'project'; the node that represents customers is called 'customer' and so on. Since our node will handle management of pizzas, we will call it 'pizza'.

Every node is located in its own file. The name of the file is important. It should always be '*class.<the name of your node>.inc*', in our case '*class.pizza.inc*'. If you name it differently, Achievo won't be able to find it.

Let's create the file, and put the first lines of code in it (again, I will first show you the code and afterwards explain it):

```
<?php

    class pizza extends atkNode
    {
        function pizza()
        {
            $this->atkNode("pizza");
        }
    }

?>
```

Like with the module, we start with defining the class. Every node extends the atkNode class, the mother of all nodes. Doing this makes it possible for Achievo to interface with the class.

We also add one function to the class, with the same name as the class itself. This is called a constructor function. The function with the same name as the class is always called when a

node is created. In this function, we initialize the base-class, telling it the name of the node, with this line:

```
$this->atkNode("pizza");
```

If all this talk about constructors and base classes is abracadabra to you, don't worry. It's not essential that you understand this. These are just lines that always need to be there. Do the same in your own nodes and it will work. Trust me.

Now it's time to implement some functionality in the pizza node. We start by telling the node what database fields the pizza table has. Let's start with the id and name fields. We do this by adding a few lines to the constructor:

```
function pizza()
{
    $this->atkNode("pizza");

    $this->add(new atkAttribute("id", AF_PRIMARY |
                                AF_HIDE |
                                AF_AUTO_INCREMENT));
    $this->add(new atkAttribute("name", AF_UNIQUE | AF_OBLIGATORY));
}
```

The two lines we just added add so-called 'attributes' to the node. Each attribute represents a field in the database. Let's take a closer look at the first line we added, so I can explain to you exactly what it does:

```
$this->add
```

This is the function-call to add attributes to our node. `$this` is the node itself.

```
new atkAttribute
```

Here, we create an `atkAttribute`, a database field representation. There are a lot of different types of attributes. Each type of field may have its own kind of attribute. For example, there is an `atkDateAttribute` for manipulating dates. The `atkAttribute` is the default, most common attribute. We will find out more about other attributes later on.

```
"id"
```

This is the name of the field in the database. The name we specify here has to be exactly the name of the field in the database.

```
AF_PRIMARY | AF_HIDE | AF_AUTO_INCREMENT
```

The second parameter to the `atkAttribute` function-call represents the so-called 'flags'. Flags influence the behavior of the attribute. Each flag starts with 'AF_'. You can specify more than one flags by separating them with a '|'. There are over 30 different flags. We won't get to see all of them in this guide. For now, I'll explain the flags as we encounter them.

The `AF_PRIMARY` flag indicates to Achievo that this field is the primary key of the table. You should specify this flag for the same fields that you indicated as primary keys in the database. Achievo uses this information to determine which record the user is manipulating.

The `AF_HIDE` flag specifies that this attribute is hidden. The user of Achievo will never see this attribute. Primary keys are usually used internally, but of no use to the user. So most of the time, you will want to hide the primary key attribute.

The `AF_AUTO_INCREMENT` flag indicates that the id field is auto-incremented when a new Pizza is added to the database. Achievo takes care of creating the correct value for the id when this flag is set. (There are ways to solve this in the database, but Achievo uses a database independent solution.)

The next line adds the 'name' field to the node. It is similar to the previous line, but now we use the `AF_UNIQUE` and `AF_OBLIGATORY` flags as a parameter. These flags tell Achievo that the name of a pizza should be unique, and that 'name' is a required field. So there may not be a pizza without a name and no two pizzas can be called the same. Achievo takes care of checking these constraints.

Now there's only one line left to implement in this file before we can test out what we have created so far. Just below the last line where we added the name attribute, we add one line that tells Achievo which table in the database is used to store pizzas:

```
$this->add(new atkAttribute("name", AF_UNIQUE|AF_OBLIGATORY));

$this->setTable("pizza");
}
```

Now, our node class is ready for a first test. Save and close the file. We haven't added the fields for description, price and entrydate yet, but we will add these soon.

3.2 Adding the new feature to the menu

Before we can test our pizza management, we must add a menu-item to Achievo that gives access to our new feature. We do this in our *module.inc* file. We created this file before, but did not put any useful code in it. We are going to do that just now.

Add the following function to the module in the *module.inc* file:

```
class mod_pizza extends atkModule
{
    function getMenuItems()
    {
        menuitem("pizza", dispatch_url("pizzaman.pizza", "admin"));
    }
}
```

The function `getMenuItems()` is called by Achievo when it displays the menu, to determine the menu-items that your module wants to display. In this function, you use the `menuitem()` function to add an item to the menu.

This function takes two parameters. The name of the menuitem ('pizza' in this case) and the url that should be displayed when the user clicks the menu item. This can be any valid url. If for example we would have specified `menuitem("pizza", "http://www.pizza.com")`, the user would visit the pizza.com website when clicking on the link. But in our case, we use the `dispatch_url()` function to generate the correct url for us. The `dispatch_url` function takes two parameters. The first one is the name of the node that we want to use. We just created the pizza node, so we specify that here.

Note that we precede the node name with the module name and a dot. There may be more than one module using a node called "pizza". So we always have to specify the module name along with it. In our case the module is called "pizzaman" and the node is called "pizza". So the full parameter is "pizzaman.pizza".

The second parameter is the *'action'* we want to perform on the node. You will learn more about actions later on, but for now, it suffices to say that the *'admin'* action will open up the pizza administration screen. In other words, the place where we can add, edit and remove pizzas.

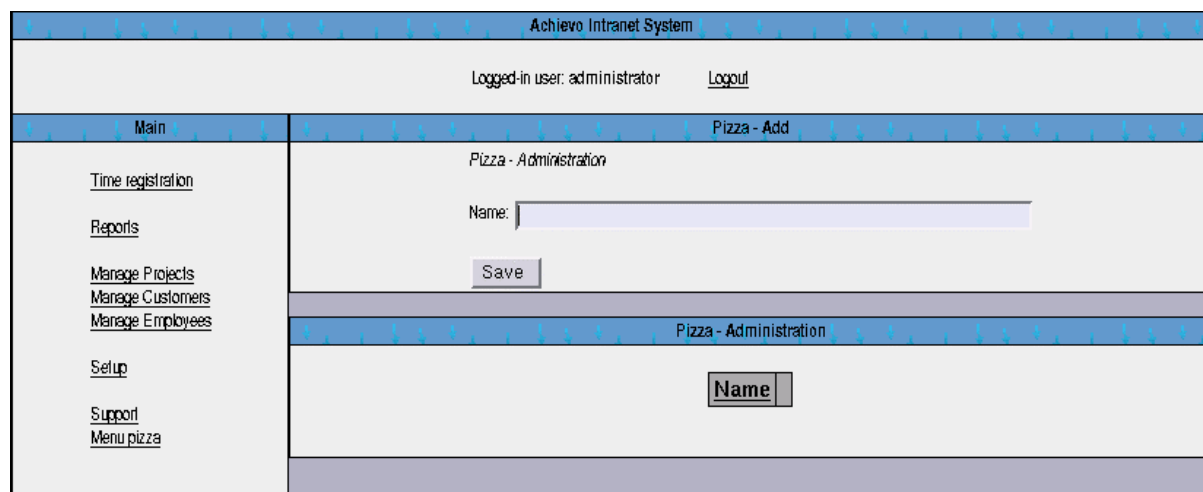
Save and close the file. We can now test our new feature!

3.3 Testing the node

Browse to the Achievo installation. Login in as administrator. This is necessary at this moment, since we haven't implemented any security yet for our pizza node. Achievo is quite strict with its security, so if we haven't specified anything, only the administrator is allowed to use the node.

Once you have logged in, you should immediately see a new entry in the menu, called *'menu_pizza'* (I will teach you later how to change the display texts).

When you click the menu-item, you should see the following screen:



The pizza screen is divided into two parts. The above part is for adding new pizzas. In the box below that, there's an area where existing pizzas are shown. Of course, the list is initially empty. But go ahead, enter a pizza name and press the *'save'* button.

See how we created a basic application to add, edit and delete pizzas in roughly ten lines of code? On the Achievo mailinglist, I used to tell people *'this or that can be done in 3 lines of code'* if they wanted to implement new features. Now you can see for yourself that it really takes a very small amount of programming to build something.

Another thing to notice: we haven't programmed any SQL queries! Achievo's backend, ATK, takes care of this.

Now that we have our basic class files in place, we can finish the application. After all, we defined that a Pizza had more than just a name.

3.4 Finishing the Pizza node

In paragraph 1.1, we decided that a Pizza should have an id, a name, a description, a price and an entrydate. We already have the id and name fields; we now implement the other fields.

A few paragraphs back, I introduced the 'atkAttribute' to you. Before adding the fields, we must determine which atkAttributes to use. Take a look at the atk/attributes directory in Achievo. There are a lot of attributes, each of which represents a different way of editing data.

For the description, we could use a regular atkAttribute, but then the user-interface would only provide us with a single line for editing the description, whereas we defined the description field in the database as 'text', so we can put multiple lines of data in it. The attribute we will use is the atkTextAttribute. This attribute represents a multi-line edit-box.

For the price, we will use the atkNumberAttribute. On screen, this attribute looks the same as the regular atkAttribute, but this attribute ensures that the user only enters valid numbers. Ofcourse, the database would not accept anything other than a valid number, but we don't want the user to see SQL errors. We want the application to nicely handle this, so we use the atkNumberAttribute.

The 'entrydate' field finally is represented by the atkDateAttribute. This attribute provides a nice way of entering valid dates.

So we add three lines of code to the pizza node class in class.pizza.inc:

```
$this->add(new atkAttribute("name", AF_UNIQUE|AF_OBLIGATORY));

$this->add(new atkTextAttribute("description"));
$this->add(new atkNumberAttribute("price"));
$this->add(new atkDateAttribute("entrydate"));

$this->setTable("pizza");
```

If you are running Achievo 0.8.x, you're done. If you're running Achievo 0.9 or higher, you need to add a few extra lines to the top of the class file:

```
<?php

useattrib("atktextattribute");
useattrib("atknumberattribute");
useattrib("atkdateattribute");

class pizza extends atkNode
```

Before Achievo 0.9, the entire codebase was always loaded, so all attributes could be used at all times. As the number of attributes grew, this became a performance problem. So in Achievo 0.9, you have to explicitly specify which attributes you use, so the backend can include the correct files. That's why since Achievo 0.9, you need these lines.

We have now implemented all fields. Our pizza database can be managed.

When you edit a Pizza, you should now see a screen that looks like this:

The screenshot shows a web-based form titled "Pizza - Edit". The form is contained within a window with a blue header bar. The breadcrumb navigation shows "Pizza - Administration | Pizza - Edit". The form fields are as follows:

- Name:** A text input field containing "Athena".
- Description:** A large text area containing "Feta and other Greek ingredients."
- Price:** A text input field containing "6.50".
- Entry date:** A date selection field with three parts: a month dropdown menu showing "June", a day dropdown menu showing "Saturday 29", and a year input field showing "2002".

At the bottom of the form, there are three buttons: "Save and close", "Save", and "Cancel".

In chapter 5, we will greatly improve the user-friendliness of our pizza manager, by tweaking the code just a little. However, we first have to discuss a more important topic.

4 Security

Currently, only the ‘administrator’ user can manage pizzas. Of course this is not the way we want it. We want to be able to use the ‘security profiles’ feature of Achievo for granting users the right to manage pizzas.

As you will soon see, this only requires one line of code, but first I want to take the opportunity to explain Achievo’s security system a bit.

As you know by now, the functionality is implemented in nodes. Nodes are ‘action-driven’, which means that on each node, you can execute certain actions. Actions are for example ‘edit’, ‘delete’, ‘add’, which show the screens for editing, adding or deleting records. Then you have the action ‘admin’, which brings up the administration screen (where you view your pizzas).

It’s even possible to create new actions, as you will see in part 2 of this guide.

The security scheme used by Achievo is mapped to nodes and their actions. For any action on any node, you can grant someone the right to perform that action. This allows for fine-grained security settings. You can for example grant someone the right to edit pizzas, but deny the right to add new ones.

It’s also possible to group actions, if you want less fine-grained security. For example, you could argue that if one is allowed to execute the ‘admin’ action (the screen where you can see the pizzas and edit/delete/add them), one should automatically be allowed to execute edit, add and delete. We will handle these situations in part 2.

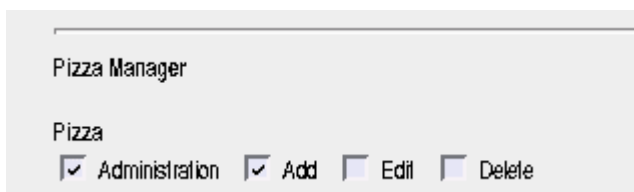
Since the list of possible actions on each node may vary, we have to tell the system which actions we want to add to the security profiles section.

We do this by adding one function to the *mod_pizza* class in our module.inc file:

```
function getNodes()  
{  
    registerNode("pizzaman.pizza", array("admin", "add", "edit", "delete"));  
}
```

In this function, we tell the system that we have a node called ‘pizza’, that belongs to the ‘pizzaman’ module, and that has 4 actions.

After adding this function, you can go to the ‘Security Profiles’ feature in Achievo, in the ‘Employee Administration’ menu, and now you can grant rights to manage pizzas to users, as can be seen in the following partial screenshot:



5 Improving user-friendliness

By now we have a very basic pizza manager, which allows us to add, edit and delete pizzas. By adding just a little bit of extra code, we can greatly improve the usability of our pizza manager.

5.1 Navigation improvements

As you have seen, the screen for managing the pizzas contains a simple list, that can be sorted by clicking on a column header, and navigation is done on a page by page basis: about 25 pizzas (depending on your Achievo configuration) are shown on one page.

We can improve usability by setting the default sort order to the pizza name, and by adding an alphabetical index. This index is useful when we have a lot of pizzas. By clicking on a letter of the alphabet, the screen shows only pizzas starting with that letter.

These two enhancements can be done in two small lines, which we add to the constructor of the pizza class, in the same spot where we told the system what database table it needed to use:

```
$this->add(new atkDateAttribute("entrydate"));

$this->setIndex("name");
$this->setOrder("name");
$this->setTable("pizza");
```

The first line we added specifies that the alphabetical index should be based on the pizza name.

The second line sets the default sort order to the name field as well. Note that after adding this line, we can still sort the list by any other column by just clicking the column header.

5.2 Adding search functionality

Sometimes, you're looking for a pizza, but you just can't find it in the list. We can add search functionality to the Pizza manager very easily, allowing you to search for any value in any of the fields of the table.

In the list of Pizzas, it would be nice to be able to search directly for a name, or a description. In a separate screen, we can then search for other fields, like prices.

As we've seen before, much of the behavior of the application can be influenced by *flags*. Enabling search functionality consists of just that: adding flags to the fields you want to be able to search on.

We go to the two lines from the class.pizza.inc file that were used to add the name and the description fields, and change them like this:

```
$this->add(new atkAttribute("name", AF_UNIQUE |
                             AF_OBLIGATORY |
                             AF_SEARCHABLE));

$this->add(new atkTextAttribute("description", 15,
                              AF_OBLIGATORY | AF_SEARCHABLE));
```

With the AF_SEARCHABLE flag, we indicate that these fields are directly searchable from within the list of pizzas.

Note that the atkTextAttribute has 3 parameters. To specify the flags, which are the third parameter, we also have to specify the second parameter, which indicates the number of lines that you can see on the screen when editing the description. When your descriptions are large, it makes sense to put this value to something like 15. If they're small, 3 might be enough. While we're editing the flags, we also add the AF_OBLIGATORY flag to the description field, which makes the description a required field.

After you added the two flags, browse the pizza manager. You will see a screen that looks like this:

The screenshot shows the 'Achievo Intranet System' interface. At the top, it says 'Logged-in user: administrator' and 'Logout'. The main content area is titled 'Pizza - Administration' and contains a search bar, an alphabetical index (A-Z), and a table of pizzas. The table has columns for Name, Description, Price, and Entrydate. Below the table, there is a 'Search (Extended)' button.

Name	Description	Price	Entrydate	
Athena	Feta and other Greek ingredients.	6.50	June 29 2002	Edit Delete
Funghi	Pizza with lots of mushrooms.	5.50	June 29 2002	Edit Delete
Quattro	Stagioni	6.50	June 29 2002	Edit Delete

Notice how the name and the description are now searchable. Also notice how we, automatically, also have an 'Extended' search option, allowing you to do advanced searches on all fields of the database.

We now have a very useable Pizza manager. On to the conclusion!

6 Conclusion

Ok, we're finally done!

Let's take a look at what we have done so far. Even though I have rambled on for 16 pages by now, look at how little code we actually had to write! It took only about 20 lines of code, to create a complete pizza manager application!

Yes. Developing Achievo extensions really takes that little amount of code.

And this is only the beginning. In part 2 and 3 of this guide, which I will start to write soon, I will show you a lot of powerful features, like creating relations between nodes or customizing standard features. But for now, just play around with the code you just wrote. Try to write your own little applications. Find out how easy you can add features to Achievo to make it fit your organization even more.

Should you encounter any problems during development, subscribe to our mailinglist by sending an empty message to achievo-subscribe@achievo.org, and we will help you out. Also, you might want to join channel #achievo, on KreyNet IRC Network (irc.krey.net), where development of Achievo can be discussed.

Good luck, and have fun!
Ivo Jansch <ivo@achievo.org>

Appendix

The complete code used in this guide can be found in this appendix:

module.inc

```
<?php

class mod_pizzaman extends atkModule
{
    function getMenuItems()
    {
        menuitem("pizza", dispatch_url("pizzaman.pizza", "admin"));
    }

    function getNodes()
    {
        registerNode("pizzaman.pizza", array("admin",
                                             "add",
                                             "edit",
                                             "delete"));
    }
}

?>
```

class.pizza.inc

```
<?php

// The next three lines should only be added when
// using Achievo 0.9 or higher.

useattrib("atktextattribute");
useattrib("atknumberattribute");
useattrib("atkdateattribute");

class pizza extends atkNode
{
function pizza()
{
$this->atkNode("pizza");

$this->add(new atkAttribute("id", AF_PRIMARY|
                           AF_HIDE|
                           AF_AUTO_INCREMENT));
$this->add(new atkAttribute("name", AF_UNIQUE|
                           AF_OBLIGATORY|
                           AF_SEARCHABLE));

$this->add(new atkTextAttribute("description", 15,
                              AF_OBLIGATORY|
                              AF_SEARCHABLE));

$this->add(new atkNumberAttribute("price"));
$this->add(new atkDateAttribute("entrydate"));

$this->setIndex("name");
$this->setOrder("name");
$this->setTable("pizza");

}
}
?>
```