Table of Contents

Developing for Drupal

The Drupal engine is open source. It is possible for each and every user to become a contributor. The fact remains that most Drupal users, even those skilled in programming arts, have never contributed to Drupal even though most of us had days where we thought to ourselves: "I wish Drupal could do this or that ...". Through this section, we hope to make Drupal more accessible to them.

The guide pages found here are collaborative, but not linked to particular Drupal versions. Because of this, documentation can become out of date. To combat this, we are moving most developer documentation into the Doxygen documentation that is versioned by CVS and generated from the source code. Look there for up-to-date and version-specific information.

- CVS log messages
- Browse CVS repository

Contributing to Drupal

Drupal is a collaborative, community-driven project. This means that the software and its supporting features (documentation, the drupal.org website) are collaboratively produced by users and developers all over the world.

There are several ways to contribute to Drupal:

- Improve or enhance the software
- Provide support and documentation for other users (e.g., by posting additions or updates to the Drupal Handbook or answering requests on user forums or issues).
- Provide financial support to Drupal development.

This section focuses on the first of these three.

Types of Contributions

There are two basic types of contributions you can make to Drupal's code base: (a) "contributed" modules or themes and (b) contributions to the drupal "core".

- *"Contributions"* are the community-produced modules and themes available on the Drupal site. To make a contribution, you need to apply for contributor privileges, produce your contribution, and then notify the contributions manager to request a review of your work before posting. As long as contributions meet some minimal criteria - they do what they claim to and have some demonstrable benefit without unduly replicating already-available functionality - they are approved.

  If you have major enhancements you wish to contribute, doing so via a contributed module is in many ways the easiest way to begin. Contributed code has a relatively low set of requirements to meet.

- In contrast, *changes to the Drupal core* are made through a thorough consultative process to ensure the overall integrity of the software.

  Changes to the Drupal core are generally of three types:

  - *Bug fixes.* These changes respond to identified problems in the existing code.
  - *New features.* These changes are enhancements on what is already available.
  - *Code maintenance.* These changes are to improve the quality of the code or bring it up to date with changes elsewhere in Drupal. This can include bringing code in line with coding standards, improving efficiency (e.g., eliminating unneeded database queries), introducing or improving in-line comments, and doing upgrades for compliance with a new release version.

While you can create your own issues, you can also begin by simply taking on existing tasks on the task list.

Task list

The Drupal bug database contains many issues classified as "bite-sized" tasks -- tasks that are well-defined and self-contained, and thus suitable for a volunteer looking to get involved with the project. You don't need broad or detailed knowledge of Drupal's design to take on one of these, just a pretty good idea of how things generally work, and familiarity with the coding guidelines. Each task is something a volunteer could pick off in a spare evening or two.

If you start one of these, please notify the other developers by mailing drupal-devel@drupal.org (of course, you should be subscribed to that list). If you have questions as you go, ask the dev list or update the task (updates are sent to the list automatically). Send the patch to the list when ready.

Bug reports

If you found a bug, send us the bug report and we will fix it provided you include enough diagnostic information for us to go on. Your bug reports play an essential role in making Drupal reliable.

Bug reports can be posted in connection with any project hosted on drupal.org. You can submit a new bug via the submit issue form. Provide a sensible title for the bug, and choose the project you think you have found the bug in. After previewing the submission, you will need to choose a related component and you will be able to provide more details about the bug, including the description of the problem itself. Please include any error messages you received and a detailed description of what you were doing at the time.

Note that you don't have to be logged in nor a member of drupal.org to submit bugs.

The first thing we will do when you report a bug is tell you to upgrade to the newest version of Drupal, and then see if the problem reproduces. So you'll probably save us both time if you upgrade and test with the latest version before sending in a bug report.

How to report bugs effectively

Summary

- The first aim of a bug report is to let the programmer see the failure with their own eyes. If you can't be with them to make it fail in front of them, give them detailed instructions so that they can make it fail for themselves.
- In case the first aim doesn't succeed, and the programmer can't see it failing themselves, the second aim of a bug report is to describe what went wrong. Describe everything in detail. State what you saw, and also state what you expected to see. Write down the error messages, especially if they have numbers in.

- When your computer does something unexpected, freeze. Do nothing until you're calm, and don't do anything that you think might be dangerous.
- By all means try to diagnose the fault yourself if you think you can, but if you do, you should still report the symptoms as well.
- Be ready to provide extra information if the programmer needs it. If they didn't need it, they wouldn't be asking for it. They aren't being deliberately awkward. Have version numbers at your fingertips, because they will probably be needed.
- Write clearly. Say what you mean, and make sure it can't be misinterpreted.
- Above all, be precise. Programmers like precision.

Introduction
Anybody who has written software for public use will probably have received at least one bad bug report. Reports that say nothing ("It doesn't work!"); reports that make no sense; reports that don't give enough information; reports that give wrong information. Reports of problems that turn out to be user error; reports of problems that turn out to be the fault of somebody else's program; reports of problems that turn out to be network failures.

There's a reason why technical support is seen as a horrible job to be in, and that reason is bad bug reports. However, not all bug reports are unpleasant: I maintain free software, when I'm not earning my living, and sometimes I receive wonderfully clear, helpful, informative bug reports.

In this essay I'll try to state clearly what makes a good bug report. Ideally I would like everybody in the world to read this essay before reporting any bugs to anybody. Certainly I would like everybody who reports bugs to me to have read it.

In a nutshell, the aim of a bug report is to enable the programmer to see the program failing in front of them. You can either show them in person, or give them careful and detailed instructions on how to make it fail. If they can make it fail, they will try to gather extra information until they know the cause. If they can't make it fail, they will have to ask you to gather that information for them.

In bug reports, try to make very clear what are actual facts ("I was at the computer and this happened") and what are speculations ("I think the problem might be this"). Leave out speculations if you want to, but don't leave out facts.

When you report a bug, you are doing so because you want the bug fixed. There is no point in swearing at the programmer or being deliberately unhelpful: it may be their fault and your problem, and you might be right to be angry with them, but the bug will get fixed faster if you help them by supplying all the information they need. Remember also that if the program is free, then the author is providing it out of kindness, so if too many people are rude to them then they may stop feeling kind.

"It doesn't work."
Give the programmer some credit for basic intelligence: if the program really didn't work at all, they would probably have noticed. Since they haven't noticed, it must be working for them. Therefore, either you are doing something differently from them, or your environment is different from theirs. They need information; providing this information is the purpose of a bug

report. More information is almost always better than less.

Many programs, particularly free ones, publish their list of known bugs. If you can find a list of known bugs, it's worth reading it to see if the bug you've just found is already known or not. If it's already known, it probably isn't worth reporting again, but if you think you have more information than the report in the bug list, you might want to contact the programmer anyway. They might be able to fix the bug more easily if you can give them information they didn't already have.

This essay is full of guidelines. None of them is an absolute rule. Particular programmers have particular ways they like bugs to be reported. If the program comes with its own set of bug-reporting guidelines, read them. If the guidelines that come with the program contradict the guidelines in this essay, follow the ones that come with the program!

If you are not reporting a bug but just asking for help using the program, you should state where you have already looked for the answer to your question. ("I looked in chapter 4 and section 5.2 but couldn't find anything that told me if this is possible.") This will let the programmer know where people will expect to find the answer, so they can make the documentation easier to use.

"Show me"
One of the very best ways you can report a bug is by showing it to the programmer. Stand them in front of your computer, fire up their software, and demonstrate the thing that goes wrong. Let them watch you start the machine, watch you run the software, watch how you interact with the software, and watch what the software does in response to your inputs.

They know that software like the back of their hand. They know which parts they trust, and they know which parts are likely to have faults. They know intuitively what to watch for. By the time the software does something obviously wrong, they may well have already noticed something subtly wrong earlier which might give them a clue. They can observe everything the computer does during the test run, and they can pick out the important bits for themselves.

This may not be enough. They may decide they need more information, and ask you to show them the same thing again. They may ask you to talk them through the procedure, so that they can reproduce the bug for themselves as many times as they want. They might try varying the procedure a few times, to see whether the problem occurs in only one case or in a family of related cases. If you're unlucky, they may need to sit down for a couple of hours with a set of development tools and really start investigating. But the most important thing is to have the programmer looking at the computer when it goes wrong. Once they can see the problem happening, they can usually take it from there and start trying to fix it.

"Show me how to show myself"
This is the era of the Internet. This is the era of worldwide communication. This is the era in which I can send my software to somebody in Russia at the touch of a button, and he can send me comments about it just as easily. But if he has a problem with my program, he can't have me standing in front of it while it fails. "Show me" is good when you can, but often you can't.

If you have to report a bug to a programmer who can't be present in person, the aim of the exercise is to enable them to reproduce the problem. You want the programmer to run their own copy of the program, do the same things to it, and make it fail in the same way. When they can see the problem happening in front of their eyes, then they can deal with it.

So tell them exactly what you did. If it's a graphical program, tell them which buttons you pressed and what order you pressed them in. If it's a program you run by typing a command, show them precisely what command you typed. Wherever possible, you should provide a verbatim transcript of the session, showing what commands you typed and what the computer output in response.

Give the programmer all the input you can think of. If the program reads from a file, you will probably need to send a copy of the file. If the program talks to another computer over a network, you probably can't send a copy of that computer, but you can at least say what kind of computer it is, and (if you can) what software is running on it.

"Works for me, so what goes wrong?"
If you give the programmer a long list of inputs and actions, and they fire up their own copy of the program and nothing goes wrong, then you haven't given them enough information. Possibly the fault doesn't show up on every computer; your system and theirs may differ in some way. Possibly you have misunderstood what the program is supposed to do, and you are both looking at exactly the same display but you think it's wrong and they know it's right.

So also describe what happened. Tell them exactly what you saw. Tell them why you think what you saw is wrong; better still, tell them exactly what you expected to see. If you say "and then it went wrong", you have left out some very important information.

If you saw error messages then tell the programmer, carefully and precisely, what they were. They are important! At this stage, the programmer is not trying to fix the problem: they're just trying to find it. They need to know what has gone wrong, and those error messages are the computer's best effort to tell you that. Write the errors down if you have no other easy way to remember them, but it's not worth reporting that the program generated an error unless you can also report what the error message was.

In particular, if the error message has numbers in it, do let the programmer have those numbers. Just because you can't see any meaning in them doesn't mean there isn't any. Numbers contain all kinds of information that can be read by programmers, and they are likely to contain vital clues. Numbers in error messages are there because the computer is too confused to report the error in words, but is doing the best it can to get the important information to you somehow.

At this stage, the programmer is effectively doing detective work. They don't know what's happened, and they can't get close enough to watch it happening for themselves, so they are searching for clues that might give it away. Error messages, incomprehensible strings of numbers, and even unexplained delays are all just as important as fingerprints at the scene of a crime. Keep them!

If you are using Unix, the program may have produced a core dump. Core dumps are a particularly good source of clues, so don't throw them away. On the other hand, most programmers don't like to receive huge core files by e-mail without warning, so ask before mailing one to anybody. Also, be aware that the core file contains a record of the complete state of the program: any "secrets" involved (maybe the program was handling a personal message, or dealing with confidential data) may be contained in the core file.

"So then, I tried..."
There are a lot of things you might do when an error or bug comes up. Many of them make the problem worse. A friend of mine at school deleted all her Word documents by mistake, and before calling in any expert help, she tried reinstalling Word, and then she tried running Defrag. Neither of these helped recover her files, and between them they scrambled her disk to the extent that no Undelete program in the world would have been able to recover anything. If she'd only left it alone, she might have had a chance.

Users like this are like a mongoose backed into a corner: with its back to the wall and seeing certain death staring it in the face, it attacks frantically, because doing something has to be better than doing nothing. This is not well adapted to the type of problems computers produce.

Instead of being a mongoose, be an antelope. When an antelope is confronted with something unexpected or frightening, it freezes. It stays absolutely still and tries not to attract any attention, while it stops and thinks and works out the best thing to do. (If antelopes had a technical support line, it would be telephoning it at this point.) Then, once it has decided what the safest thing to do is, it does it.

When something goes wrong, immediately stop doing anything. Don't touch any buttons at all. Look at the screen and notice everything out of the ordinary, and remember it or write it down. Then perhaps start cautiously pressing "OK" or "Cancel", whichever seems safest. Try to develop a reflex reaction - if a computer does anything unexpected, freeze.

If you manage to get out of the problem, whether by closing down the affected program or by rebooting the computer, a good thing to do is to try to make it happen again. Programmers like problems that they can reproduce more than once. Happy programmers fix bugs faster and more efficiently.

"I think the tachyon modulation must be wrongly polarised."
It isn't only non-programmers who produce bad bug reports. Some of the worst bug reports I've ever seen come from programmers, and even from good programmers.

I worked with another programmer once, who kept finding bugs in his own code and trying to fix them. Every so often he'd hit a bug he couldn't solve, and he'd call me over to help. "What's gone wrong?" I'd ask. He would reply by telling me his current opinion of what needed to be fixed.

This worked fine when his current opinion was right. It meant he'd already done half the work and we were able to finish the job together. It was efficient and useful.

But quite often he was wrong. We would work for some time trying to figure out why some particular part of the program was producing incorrect data, and eventually we would discover that it wasn't, that we'd been investigating a perfectly good piece of code for half an hour, and that the actual problem was somewhere else.

I'm sure he wouldn't do that to a doctor. "Doctor, I need a prescription for Hydroyoyodyne." People know not to say that to a doctor: you describe the symptoms, the actual discomforts and aches and pains and rashes and fevers, and you let the doctor do the diagnosis of what the problem is and what to do about it. Otherwise the doctor dismisses you as a hypochondriac or crackpot, and quite rightly so.

It's the same with programmers. Providing your own diagnosis might be helpful sometimes, but always state the symptoms. The diagnosis is an optional extra, and not an alternative to giving the symptoms. Equally, sending a modification to the code to fix the problem is a useful addition to a bug report but not an adequate substitute for one.

If a programmer asks you for extra information, don't make it up! Somebody reported a bug to me once, and I asked him to try a command that I knew wouldn't work. The reason I asked him to try it was that I wanted to know which of two different error messages it would give. Knowing which error message came back would give a vital clue. But he didn't actually try it - he just mailed me back and said "No, that won't work". It took me some time to persuade him to try it for real.

Using your intelligence to help the programmer is fine. Even if your deductions are wrong, the programmer should be grateful that you at least tried to make their life easier. But report the symptoms as well, or you may well make their life much more difficult instead.

"That's funny, it did it a moment ago."
Say "intermittent fault" to any programmer and watch their face fall. The easy problems are the ones where performing a simple sequence of actions will cause the failure to occur. The programmer can then repeat those actions under closely observed test conditions and watch what happens in great detail. Too many problems simply don't work that way: there will be programs that fail once a week, or fail once in a blue moon, or never fail when you try them in front of the programmer but always fail when you have a deadline coming up.

Most intermittent faults are not truly intermittent. Most of them have some logic somewhere. Some might occur when the machine is running out of memory, some might occur when another program tries to modify a critical file at the wrong moment, and some might occur only in the first half of every hour! (I've actually seen one of these.)

Also, if you can reproduce the bug but the programmer can't, it could very well be that their computer and your computer are different in some way and this difference is causing the problem. I had a program once whose window curled up into a little ball in the top left corner of the screen, and sat there and sulked. But it only did it on 800x600 screens; it was fine on my 1024x768 monitor.

The programmer will want to know anything you can find out about the problem. Try it on another machine, perhaps. Try it twice or three times and see how often it fails. If it goes wrong when you're doing serious work but not when you're trying to demonstrate it, it might be long running times or large files that make it fall over. Try to remember as much detail as you can about what you were doing to it when it did fall over, and if you see any patterns, mention them. Anything you can provide has to be some help. Even if it's only probabilistic (such as "it tends to crash more often when Emacs is running"), it might not provide direct clues to the cause of the problem, but it might help the programmer reproduce it.

Most importantly, the programmer will want to be sure of whether they're dealing with a true intermittent fault or a machine-specific fault. They will want to know lots of details about your computer, so they can work out how it differs from theirs. A lot of these details will depend on the particular program, but one thing you should definitely be ready to provide is version numbers. The version number of the program itself, and the version number of the operating system, and probably the version numbers of any other programs that are involved in the problem.

"So I loaded the disk on to my Windows . . ."
Writing clearly is essential in a bug report. If the programmer can't tell what you meant, you might as well not have said anything.

I get bug reports from all around the world. Many of them are from non-native English speakers, and a lot of those apologise for their poor English. In general, the bug reports with apologies for their poor English are actually very clear and useful. All the most unclear reports come from native English speakers who assume that I will understand them even if they don't make any effort to be clear or precise.

- Be specific. If you can do the same thing two different ways, state which one you used. "I selected Load" might mean "I clicked on Load" or "I pressed Alt-L". Say which you did. Sometimes it matters.
- Be verbose. Give more information rather than less. If you say too much, the programmer can ignore some of it. If you say too little, they have to come back and ask more questions. One bug report I received was a single sentence; every time I asked for more information, the reporter would reply with another single sentence. It took me several weeks to get a useful amount of information, because it turned up one short sentence at a time.
- Be careful of pronouns. Don't use words like "it", or references like "the window", when it's unclear what they mean. Consider this: "I started FooApp. It put up a warning window. I tried to close it and it crashed." It isn't clear what the user tried to close. Did they try to close the warning window, or the whole of FooApp? It makes a difference. Instead, you could say "I started FooApp, which put up a warning window. I tried to close the warning window, and FooApp crashed." This is longer and more repetitive, but also clearer and less easy to misunderstand.
- Read what you wrote. Read the report back to yourself, and see if you think it's clear. If you have listed a sequence of actions which should produce the failure, try following them yourself, to see if you missed a step.

Feature suggestions

How many times you have dreamed "Gee...I wish Drupal could do that" or "I like the xxx feature, but it should work better". If you want to improve Drupal, send us you wishes as a feature suggestions. Your suggestions play an essential role in making Drupal more usable and feature-rich.

The core features provided by Drupal are listed on the features page. You can submit a feature request by creating a new issue connected to the component the feature is related to. Please note that there is a Drupal contributed module named 'Features' which is used on the feature page mentioned above. Every module has a feature request subcategory, and thus **the 'Feature' module is not the appropriate place** to submit feature requests. To properly file a feature request, first choose the project it is related to and then after hitting preview set the other related options. You will be able to categorize the issue as a feature request with the Issue Information / Category dropdown.

Note that you don't have to be logged in nor to be a member of drupal.org to suggest features.

Patches

Patches are a way to distribute relatively small changes to code. They are the preferred way to contribute bug fixes and other proposed changes to Drupal's codebase.

Diff and patch

Diff and patch are two complementary tools for recording and applying changes between two sets of files.

We use them for content control even though we distribute our code via CVS. Why? Because diff and patch provide an immense amount of control. Patches can be submitted via e-mail and in plain text; maintainers can read and judge the patch before it ever gets near a tree. It allows maintainers to look at changes easily without blindly integrating them.

Diff is the first command in the set. It has the simple purpose to create a file called a *patch* or a *diff* which contains the differences between two text files or two groups of text files. Diff can write into different formats, although the unified difference format is preferred. The patches this command generates are much easier to distribute and allow maintainers to see quickly and easily what changed and to make a judgement.

Patch is diff's complement and takes a patch file generated by diff and applies it against a file or a group of files.

The actual usage of diff and patch is not complicated.

At its simplest, a diff command for comparing two files would be:

```
diff old.txt new.txt  > oldnew.patch
```

For drupal, we prefer patches in unified format, so we add -u to the command line:

```
diff –u old.txt new.txt > oldnew.patch
```

It is helpful to keep a reference in the patch file to which function was patched, so the following form of the command is often used. For example, if you have made a change in foo.module, to create a patch against the CVS tree:

```
cvs diff –u –F ^function foo.module > foo.patch
```

Or if you had downloaded Drupal instead of checking it out from CVS and were creating a patch against a local copy of foo.module:

```
diff –u –F ^function foo.module newfoo.module > foo.patch
```

Generally, however, a comparison of two source trees is often desired. A possible command to do so is:

```
diff –ruN old new > tree.diff
```

Once a patch is generated, the process of patching the file is even simpler. Based on our examples above, we could do:

```
patch < oldnew.patch
```

Or if you want to patch an entire directory, you should use:

```
patch –p0 –u < tree.diff
```

To unapply the patch, use:

```
patch –p0 –R < tree.diff
```

Diff on Windows

(against a cvs source with the cvs.exe built-in diff. do diff local files, you need a windows diff program, command line or visual)

Generic:

- find the cvs.exe of your cvs package (WinCVS, TortoiseCVS, cygwin, ...) and make sure it is in your PATH
- cd to your drupal root dir
- `cvs diff –u [[–r rev1|–D date1] [–r rev2|–D date2]] [file_to_diff] [> file_to_diff.patch]`
  - `–u`: unified format
  - `–r`: revision(s) to diff
    - no -r: compare the working file with the revision it was based on

        - one -r: compare that revision with your current working file
        - two -r: compare those two revisions
    - `-D`: use a date_spec to specify revisions. examples: "1972-09-24 20:05", "24 Sep 1972 20:05".
    - `file_to_diff`: path to the file or directory you want to diff. if you specify a directory, the output will include the diff of all differing files in this directory and all subdirectories.
    - `> file_to_diff.patch`: creates a patch - saves the diff in `file_to_diff.patch` instead of outputting it on stdout. if you send a patch, make sure it has the proper line endings
    - see the CVS manual for a complete list of and additional options

via WinCVS GUI

Just select the file you edited and right-mouse-click > "diff selection" (or press the "diff selected"-icon on the toolbar, or do Menubar > "Query" > "diff selection"). This brings up a "Diff settings" dialog box that offers some limited options as "revisions to diff" and "ignore whitespace/case" [*update 2003-Feb-07: starting with WinCvs 1.3b11, "Full diff options [are] available from the diff dialog"*]. The resulting diff is output to the WinCVS-Console and can be copied and pasted.

via WinCVS/TortoiseCVS external diff

- WinCVS: Menubar > "Admin" > "Preferences" > "WinCVS" > "External diff program ". This program will be invoked by the "Diff selection" when "Use the external diff" is checked.
- TortoiseCVS: CVS > "Preferences" > "External diff application". This program will be invoked by "CVS Diff ..."

Some external visual diff programs for Windows:

- Araxis Merge (commercial)
- ExamDiff
- CSDiff
- for those who can live w/ java: Guiffy (commercial)
- WinMerge
- you may find more here

Notes:

- While these programs do a nice job in showing file differences visually, side by side, *non of them* (as i can tell) allows to actually *save* the difference in unified format (most allow to save a *standard* diff, though) - **update**: TortoiseCVS lets you save patches. It does unified format by default. See its Make Patch option. Note that this 'Make Patch' option can make recursive patches when applied to directories.
- You *cannot* specify the "-u" in the External diff preferences (eg "diff -u") as this will result in "Unable to open 'diff -u' (The system cannot find the file specified.)". A workaround for this is to, in the preferences, specify a batch-file that calls the external diff with the -u option. Another workaround is meta-diff, which allows for launching of special diff programs for certain file types.)

line endings: an issue with using diff on windows is that generated patches have windows line endings, which makes them impossible to apply on unix boxes [1][2]. unfortunately, there seems to be no way to convince "cvs diff" to output unix line endings*. so the only way for making a proper patch on windows that i see is to convert / filter the output from "cvs diff" to unix line endings:

- filter: pipe "cvs diff"s output through some dos2unix tool (like the one from Robert B. Clark, or like cygwins's dos2unix / d2u):

  ```
  cvs diff [options] file_to_diff  | unix2dos -u > file_to_diff.patch
  ```

- convert: save "cvs diff"s output to a file:

  ```
  cvs diff [options] file_to_diff > file_to_diff.patch
  ```

  and manually convert `file_to_diff.patch` to unix line endings. every developers editor should be capable of this; besides, there are many dos2unix versions that operate on files.

Patch on Windows

I haven't found any Windows-GUI for patch, so the only choice is a Windows port of the Unix command-line tool. If you know of a Windows GUI for patch, please let me know

- Cygwin - a UNIX environment for Windows, including many standard UNIX-tools (including diff and patch)
- pre-compiled binaries, available from various places. Some I've found:
  - http://www.squirrel.nl/people/jvromans/tpj0403-0016b.html
  - http://www.gnu.org/software/emacs/windows/faq11.html#patch

  Note:

  I found many of the precompiled binaries have problems with pathnames etc. and do not work properly. So I would recommend installing cygwin - it takes a while, but after that, you have a nice Unix environment that works.

* and i tried a lot: checking out all files with unix line endings, various -kb options, external diffs, patched cvs versions ... nothing. for a discussion of this, check CVS and binary files

You can try also integrated diff/patch packages like GNU diffutils for Windows or get the GNU utilities for Win 32.

Creating and submitting patches

The process of submitting a patch can seem daunting at first. This text is a collection of suggestions which can greatly increase the chances of your change being accepted.

The easiest way to get set up for making and sending patches is to get CVS working. Then you can just type: `cvs diff -u -F^f [file to patch]` to generate a patch. To output it to a file, go: `cvs diff -u -F^f [file to patch] > [outfile]`

Coding style:
If your code deviates too much from the Code Conventions, it is more likely to be rejected without further review and without comment.

diff -u:
Use `diff -u` or `diff -urN` to create patches: when creating your patch, make sure to create it in "unified diff" format, as supplied by the `-u` argument to `diff`. Patches should be based in the root source directory, not in any lower subdirectory. Make sure to create patches against a "vanilla", or unmodified source tree.

diff -F^f:
Use the additional `-F^f` argument to `diff` to create patches that are easier to read. `-F^f` tells `diff` to include the last matching line in the header of the created patch. This will be the last function definition if the files adhere to the Drupal Code Conventions.

Describe your changes:
Describe the technical detail of the change(s) your patch includes and try to be as specific as possible. Note that we prefer technical reasoning above marketing: give us clear reasons why "this way" is good. Justify your changes and try to carry enough weight. It is important to note the version to which this patch applies.

Separate your changes:
Separate each logical change into its own patch. For example, if your changes include both bug fixes and performance enhancements, separate those changes into two or more patches. If your changes include an API update, and a new module which uses that new API, separate those into two patches.

Verifying your patch
The CVS review team is overloaded reviewing patch submissions. Please make their lives easier by assuring the following:

- Test your code!
- Make sure your code is clean and secure. If your patch is just a quick hack, then don't set your issue to **Patch** status.
- Patch against HEAD. If you only have a patch against a prior revisision, then don't assign your issue to **Patch** status

Submitting your patch:
Patches should be submitted via the issue tracker. Create a bug report or feature request, attach your patch using the file upload form and set the issue's status to *patch*. Setting the status to patch is important as it adds the patch to the patch queue.

Rules of reviewing patches

1. Do review the code not the person.
2. Do not take a review personally. If you get a bad review deal with it and make your patch better. It is a learning experience.
3. Do help to review other peoples code as it will make your own code better. It will make your more critical and likely to spot your own mistakes. Might also teach you a trick or two

you didn't know about.

4. Do not feel obligated to review others code even if people review your code. (It comes highly recommended.)
5. Do give friendly suggestions on how a person can improve their code.
6. Do not demand that your code gets reviewed. Your time will come.
7. Do remind people nicely that it would be nice if someone reviewed your code, but only once a week.
8. Do this to get a good review:
9.
    - Do make sure the code actually works. Working code is a big plus.
    - Do make sure the patch is current with Drupal CVS. Feel free to refuse to review patches that don't apply nicely to Drupal CVS.
    - Do look at the code and make sure it follows the Drupal coding standards.
    - Do make sure the code uses available support functions and doesn't re-invent the wheel.
    - Do write documentation both in the code and for the users.
10. Do this when reviewing:
11.
    - Do make sure the patch does everything in item 8.
    - Do comment on the general coding style.
    - Do comment on the user interface.
    - Do make suggestions on how to improve the patch.
    - Do give your vote (+1/-1) as to whether this should be included in Drupal.
12. Just do it.

The revision process

Changes to the Drupal core are usually made after consideration, planning, and consultation. They are also made on a priority basis--fixes come before additions, and changes for which there is a high demand come before proposals that have gone relatively unnoticed. Any potential change has to be considered not only on its own merits but in relation to the aims and principles of the project as a whole.

The particular stages that a new feature goes through vary, but a typical cycle for a significant change might include:

- General discussion of the idea, for example through a posting in a drupal.org forum. This can be a chance to gauge support and interest, scope the issue, and get some direction and suggestions on approaches to take. If you're considering substantive changes, starting out at the discussion level - rather than jumping straight into code changes - can save you a lot of time.
- Posting an issue through the drupal.org project system.
- Discussion raising issues on the proposed direction or solution, which may include a real-time meeting through IRC.
  Individual Drupal community members may vote for (+1) or against (-1) the change. While

informal, this voting system can help quantify support.
- Producing a patch with specific proposed code changes.
- Review of the changes and further discussion.
- Revisions to address issues.
- Possible application of the patch.

The process of discussion and revision might be repeated several times to encompass diverse input. At any point in the process, the proposal might be:

- Shelved as impractical or inappropriate.
- Put off until other logically prior decisions are made.
- Rolled into another related initiative.
- Superceded by another change.

If you submit suggestions that don't end up being adopted, please don't be discouraged! It doesn't mean that your ideas weren't good--just that they didn't end up finding a place. The discussion itself may have beneficial outcomes. It's all part of collaboratively building a quality open source project.

Criteria for evaluating proposed changes

The following criteria are used by core developers in reviewing and approving proposed changes:

- *The changes support and enhance Drupal project aims*.
- *The proposed changes are current.* Especially for new features, priority is usually given to development for the "HEAD" (the most recent development version of the code, also referred to as the CVS version) as opposed to released versions. There may have been significant changes since the last release, so developing for the CVS version means that
- *The proposed change doesn't raise any significant issues or risks.* Specifically, issues that have been raised in the review process have been satisfactorily addressed.
- *The changes are well coded.* At a minimum, this means coding in accordance with the Drupal coding standards. But it also means that the coding is intelligent and compact. Elegant solutions will have greater support than cumbersome ones that accomplish the same result.
- *There is demonstrated demand and support for the change.* Demand is indicated by, e.g., comments on the drupal.org issues system or comments in forums or the drupal-dev email list.
- *The change will be used by a significant portion of the installed Drupal base* as opposed being relevant only to a small subset of Drupal users.
- *The benefits of the change justifies additional code and resource demands.* Every addition to the code base increases the quantity of

code that must be actively maintained (e.g., updated to reflect new design changes or documentation approaches). Also, added code increases the overall Drupal footprint through, e.g., added procedure calls or database queries. Benefits of a change must outweigh these costs.

Maintaining a project on drupal.org

Each *drupal.org project* (a contributed theme, module or translation) needs to be maintained in the contributions repository. Before creating a project page on drupal.org, apply for a CVS account and commit your project to the repository. If you are not using the drupal.org infrastructure, you can't setup a project page on drupal.org nor can you offer your module for download at drupal.org.

To get your project listed on drupal.org after it has been committed to CVS, fill in the form at http://drupal.org/node/add/project_project/. Make sure that the 'Short project name' matches the directory name in the CVS repository. For example, the *contributions/modules/my_module* module has the short name *my_module*.

Note that the newly created project will not be instantly available as it will need to be approved by one of the administrators. After that, it will appear soon after you committed some code/updates to the contributions repository. Once the project page became available, people will be able to file bugs against your project, add tasks or request new features. Your project will also become available for download.

Downloads and packaging

As soon your project page has been activated and assuming it is properly configured, drupal.org will automatically package your project and make it available for download. Projects are packaged once or twice a day so your project will not be available instantly.

Managing releases

Releases are handled using CVS branches. By default, only the *CVS HEAD version* (development version) of your project is packaged and offered for download.

However, if you branch your project using the *DRUPAL-4-5* branch name, drupal.org will package the *Drupal 4.5 compatible release* of your project. For this to work, you must use the correct branch names. A list of valid branch names can be found in the contributions repository's FAQ.txt.

As projects are only packaged once or twice a day, it might take up to 24 hours for new releases to become available on the website or for updates to propagate to the downloads.

If you found a bug that needs to be fixed in several releases of your project, make sure to commit the fix to the different branches unless you are no longer maintaining certain releases of your project.

Branching and releases are restricted to the `modules`, `themes`, `theme-engines` and `translations` directories in the contributions repository. Personal sandboxes in the `sandbox` directory can't be branched, won't be packaged and can't get a project page on drupal.org.

Orphaned projects

If you are no longer capable of maintaining your project, please add a note to your project page and ask in the forums whether someone is willing to take over maintenance. Proper communication is key so make sure to mark your project as orphaned. If you found a new maintainer or if you are willing to maintain an orphaned project, get in touch with a site maintainer so we can transfer maintainership.

Tips for contributing to the core

The following tips might improve the chances of your contributions being accepted:

- Take a step back and objectively evaluate whether the changes are appropriate for the Drupal core. Ask yourself:
  - Is the feature already implemented? Search the forums and issue tracker.
  - Could the feature be implemented as a contributed module rather than a patch to the core?
  - Will the change benefit a substantial portion of the Drupal install base?
  - Is the change sufficiently general for others to build upon cleanly?
- Be explanatory, provide descriptions and illustrations, make a good case. Don't count on others downloading, installing, and testing your changes. Rather, show them in a nutshell what your changes would mean. Anticipate and address questions or concerns. If appropriate, provide screenshots.
- Be friendly and respectful. Acknowledge the effort others put in.
- Be open to suggestions and to other ways of accomplishing what you're aiming for.
- Be persistent. If you don't get any response right away, don't necessarily give up. If you're still convinced your idea has merit, find another way to present it.
- Respond, in a timely way, to suggestions, requests, or issues raised. Revise your work accordingly.
- If some time has gone by, update your changes to work with the current CVS version.

Mailing lists

Newsletter

A read-only mailing list used for sending out the Drupal newsletter.

view archive Â· search archive Â· mailman page

Drupal-support

If you need help with installing, running or anything Drupal related this is the list to post your questions.

view archive Â· search archive Â· mailman page

Drupal-devel

This list is for those who want to either take part or just observe Drupal development.

view archive Â· search archive Â· mailman page

Drupal-docs

The place for non-programmers that want to contribute and work on documentation.

view archive Â· search archive Â· mailman page

Drupal-cvs

All CVS commits are posted to this list, a daily digest is also posted to drupal-devel though.

view archive Â· search archive Â· mailman page

Infrastructure

A mailing list for those maintaining the Drupal infrastructure, most notably the drupal.org website.

view archive Â· search archive Â· mailman page

Subscribe

Mail address:

Lists:
☐ newsletter
☐ drupal-support
☐ drupal-devel
☐ drupal-docs
☐ drupal-cvs
☐ infrastructure

Mailing of project issues

Every project issue for Drupal *with patch status* is emailed to the drupal-devel mailing list when updated. These are mailed to promote peer review of code potentially going into Drupal. Other issues are not emailed because it would make the mailing list less useful as email volume increases. You can subscribe to project issue updates for any contributed module, theme, or translation.

Coding standards

Drupal Coding Standards

Note: The Drupal Coding Standards applies to code that is to become a part of Drupal. This document is based on the PEAR Coding standards.

Indenting

Use an indent of 2 spaces, with no tabs.

Control Structures

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated of them:

```
if (condition1 || condition2) {
  action1;
}
elseif (condition3 && condition4) {
  action2;
}
else {
  defaultaction;
}
```

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

For switch statements:

```
switch (condition) {
  case 1:
    action1;
    break;

  case 2:
    action2;
    break;

  default:
    defaultaction;
    break;
}
```

Function Calls

Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
$var = foo($bar, $baz, $quux);
```

As displayed above, there should be one space on either side of an equals sign used to assign the return value of a function to a variable. In the case of a block of related assignments, more space may be inserted to promote readability:

```
$short         = foo($bar);
$long_variable = foo($baz);
```

Function Declarations

```
function funstuff_system($field) {
  $system["description"] = t("This module insert funny text into posts randomly.");
  return $system[$field];
}
```

Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate.

Comments

Inline documentation for classes should follow the Doxygen convention. More information about Doxygen can be found here:

- Document block syntax
- Comment commands

Note that Drupal uses the following docblock syntax:

```
/**
 * Comments.
 */
```

And all Doxygen commands should be prefixed with a @ instead of a /.

Non-documentation comments are strongly encouraged. A general rule of thumb is that if you look at a section of code and think "Wow, I don't want to try and describe that", you need to comment it before you forget how it works.

C style comments (/* */) and standard C++ comments (//) are both fine. Use of Perl/shell style comments (#) is discouraged.

Including Code

Anywhere you are unconditionally including a class file, use require_once(). Anywhere you are conditionally including a class file (for example, factory methods), use include_once(). Either of these will ensure that class files are included only once. They share the same file list, so you don't need to worry about mixing them - a file included with require_once() will not be included again by include_once().

Note: include_once() and require_once() are statements, not functions. You don't need parentheses around the filename to be included.

PHP Code Tags

Always use <?php ?> to delimit PHP code, not the <? ?> shorthand. This is required for Drupal compliance and is also the most portable way to include PHP code on differing operating systems and setups.

Header Comment Blocks

All source code files in the core Drupal distribution should contain the following comment block as the header:

```
<?php
/* $Id$ */
```

This tag will be expanded by the CVS to contain useful information

```
<?php
/* $Id: CODING_STANDARDS.html,v 1.4 2004/10/27 11:55:32 uwe Exp $ */
```

Using CVS

Include the Id CVS keyword in each file. As each file is edited, add this tag if it's not yet present (or replace existing forms such as "Last Modified:", etc.).

The rest of this section assumes that you have basic knowledge about CVS tags and branches.

CVS tags are used to label which revisions of the files in your package belong to a given release. Below is a list of the required CVS tags:

DRUPAL-X-Y
: (required) Used for tagging a release. If you don't use it, there's no way to go back and retrieve your package from the CVS server in the state it was in at the time of the release.

Example URLs

Use "example.com" for all example URLs, per RFC 2606.

Naming Conventions

Functions and Methods

Functions and methods should be named using lower caps and words should be separated with an underscore. Functions should in addition have the grouping/module name as a prefix, to avoid name collisions between modules.

Private class members (meaning class members that are intended to be used only from within the same class in which they are declared; PHP 4 does not support truly-enforceable private namespaces) are preceded by a single underscore. For example:

```
_node_get()
```

```
$this->_status
```

Constants

Constants should always be all-uppercase, with underscores to separate words. Prefix constant names with the uppercased name of the module they are a part of.

Global Variables

If you need to define global variables, their name should start with a single underscore followed by the module/theme name and another underscore.

Filenames

All documentation files should have the filename extension ".txt" to make viewing them on Windows systems easier. Also, the filenames for such files should be all-caps (e.g. README.txt instead of readme.txt) while the extension itself is all-lowercase (i.e. txt instead of TXT).

Examples: README.txt, INSTALL.txt, TODO.txt, CHANGELOG.txt etc.

Doxygen formatting conventions

Doxygen is a documentation generation system. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.

There is an excellent manual at the Doxygen site. The following notes pertain to the Drupal implementation of Doxygen.

General documentation syntax

To document a block of code, the syntax we use is:

```
/**
 * Documentation here
 */
```

Doxygen will parse any comments located in such a block. Our style is to use as few Doxygen-specific commands as possible, so as to keep the source legible. Any mentions of functions or file names within the documentation will automatically link to the referenced code, so typically no markup need be introduced to produce links.

Documenting files

It is good practice to provide a comment describing what a file does at the start of it. For example:

```
<?php
/* $Id: theme.inc,v 1.202 2004/07/08 16:08:21 dries Exp $ */

/**
 * @file
 * The theme system, which controls the output of Drupal.
 *
 * The theme system allows for nearly all output of the Drupal system to be
 * customized by user themes.
 */
```

The line immediately following the @file directive is a short description that will be shown in the list of all files in the generated documentation. Further description may follow after a blank line.

Documenting functions

All functions that may be called by other files should be documented; private functions optionally may be documented as well. A function documentation block should immediately precede the declaration of the function itself, like so:

```
/**
 * Verify the syntax of the given e-mail address.
 *
 * Empty e-mail addresses are allowed. See RFC 2822 for details.
 *
 * @param $mail
 *   A string containing an email address.
 * @return
 *   TRUE if the address is in a valid format.
 */
function valid_email_address($mail) {
```

The first line of the block should contain a brief description of what the function does. A longer description with usage notes may follow after a blank line. Each parameter should be listed with a @param directive, with a description indented on the following line. After all the parameters, a @return directive should be used to document the return value if there is one. Functions that are easily described in one line may omit these directives, as follows:

```
/**
 * Convert an associative array to an anonymous object.
 */
function array2object($array) {
```

The parameters and return value must be described within this one-line description in this case.

Documenting hook implementations

Many modules consist largely of hook implementations. If the implementation is rather standard and does not require more explanation than the hook reference provides, a shorthand documentation form may be used:

```
/**
 * Implementation of hook_help().
 */
function blog_help($section) {
```

This generates a link to the hook reference, reminds the developer that this is a hook implementation, and avoids having to document parameters and return values that are the same for every implementation of the hook.

Documenting themeable functions

In order to provide a quick reference for theme developers, we tag all themeable functions so that Doxygen can group them on one page. To do this, add a grouping instruction to the documentation of all such functions:

```
/**
 * Format a query pager.
 *
 * ...
 * @ingroup themeable
 */
function theme_pager($tags = array(), $limit = 10, $element = 0, $attributes = array()) {
  ...
}
```

The same pattern can be used for other functions scattered across multiple files that need to be grouped on a single page.

Comments

Inline documentation for classes should follow the PHPDoc convention, similar to Javadoc. More information about PHPDoc can be found here:
http://phpdocu.sourceforge.net/spec/howto.html

Non-documentation comments are strongly encouraged. A general rule of thumb is that if you look at a section of code and think "Wow, I don't want to try and describe that", you need to comment it before you forget how it works.

C style comments (`/* */`) and standard C++ comments (`//`) are both fine. Use of Perl/shell style comments (`#`) is discouraged.

Indenting

Use an indent of 2 spaces, with no tabs. No trailing spaces.

PHP Code tags

Always use `<?php ?>` to delimit PHP code, not the `<? ?>` shorthand. This is required for Drupal compliance and is also the most portable way to include PHP code on differing operating systems and setups.

SQL naming conventions

- Don't use (ANSI) SQL / MySQL / PostgreSQL / MS SQL Server / ... Reserved Words for column and/or table names. Even if this may work with your (MySQL) installation, it may not with others or with other databases. Some references:
  - (ANSI) SQL Reserved Words
  - MySQL Reserved Words: 4.x, 3.23.x, 3.21.x
  - PostgreSQL Reserved Words
  - MS SQL Server Reserved Words

  Some commonly misused keywords: `TIMESTAMP, TYPE, TYPES, MODULE, DATA, DATE, TIME, ...`
  See also [bug] SQL Reserved Words.

- Capitalization, Indentation
  - UPPERCASE reserved words
  - lowercase (or Capitalize) table names
  - lowercase column names

Example:

```
SELECT r.rid, p.perm
FROM {role} r
  LEFT JOIN {permission} p ON r.rid = p.rid   -- may be on one line with prev.
ORDER BY name
```

- Naming
  - Use plural or collective nouns for table names since they are sets and not scalar values.
  - Name every constraint (primary, foreign, unique keys) yourself. Otherwise you'll see funny-looking system-generated names in error messages. This happened with the `moderation_roles` table which initially defined a key without explicite name as `KEY (mid)`. This got mysqldump'ed as `KEY mid (mid)` which resulted in a syntax error as `mid()` is a mysql function (see [bug] mysql --ansi cannot import install database).
  - Index names should begin with the name of the table they depend on, eg. `INDEX users_sid_idx`.

References:

- Joe Celko - Ten Things I Hate About You
- Joe Celko - SQL for Smarties: Advanced SQL Programming
- RDBMS Naming conventions
- SQL Naming Conventions

Functions

Functions should be named using lower caps and words should be separated with an underscore. Functions should have the grouping/module name as a prefix, to avoid name collisions between modules.

Constants

Constants should always be all-uppercase, with underscores to separate words. Prefix constant names with the uppercased name of the module they are a part of.

Control structures

These include `if`, `for`, `while`, `switch`, etc. Here is an example `if` statement, since it is the most complicated of them:

```
if ((condition1) || (condition2)) {
  action1;
}
elseif ((condition3) && (condition4)) {
  action2;
}
else {
  defaultaction;
}
```

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

For switch statements:

```
switch (condition) {
  case 1:
    action1;
    break;

  case 2:
    action2;
    break;

  default:
    defaultaction;
    break;
}
```

Header comment blocks

All Drupal source code files should start with a header containing the RCS Id keyword:

```
<?php
// $Id: CODING_STANDARDS,v 1.1 2001/11/05 07:32:17 natrak Exp $
```

Note that everything after the starting $Id and before the closing $ is automatically generated by CVS - *you shouldn't edit this manually*. If you add a new file to CVS, just write `// $Id$`.

CVS

CVS (the *concurrent version system*) is a tool to manage software revisions and release control in a multi-developer, multi-directory, multi-group environment. It comes in very handy to maintain local modifications.

Thus, CVS helps you if you are part of a group of people working on the same project. In large software development projects, it's usually necessary for more than one software developer to be modifying modules of the code at the same time. Without CVS, it is all too easy to overwrite each others' changes unless you are extremely careful.

In addition, CVS helps to keep track of all changes. Therefore, the CVS server has been setup to mail all CVS commits to all maintainers. Thus, it does not require any effort to inform the other people about the work you have done, and by reading the mails everyone is kept up to date.

CVS concepts

Drupal uses Concurrent Versions System (CVS) to co-ordinate development, which can be thought of as a system for controlling the contents of a library. Describing CVS in terms of a library, books, and editions is only a metaphor - unlike a real, phisical library, no matter how many people check out a book, it will always be available to the next person.

As with any library though, there are rules of behaviour towards other users and how you treat the library materials, no need to worry about overdue library tickets or leaving coffee stains on pages, but please be sure to act responsibly when contributing content to the library.

Repository

A repository can be thought of as a book, Drupal has two repositories which can be checked out (downloaded to your local computer):

- **Drupal**
  the core Drupal code, i.e. what is downloaded as 'Drupal'.
- **Contributions**
  modules, themes, translations, etc. supplied by contributors, i.e. all Drupal material that is not in the core.

Branch

CVS tracks different versions of content. Imagine different editions of a text book, each edition including amendments and additions - each edition is known as a 'branch', and has a branch tag to identify it, e.g. *Drupal 4.5*. Every branch corresponds to a particular version of Drupal that gets released.

Head

This is a special edition of the book (repository) which has not been published yet, or in CVS speak has not been "branched" yet. Think of "head" as a manuscript of the next edition of the book. Amendments and additions are added to the manuscript, it's proof-read and tested, and once it's ready it gets published (branched) as a new edition.

Working copy/Work area

Users can check out a copy of a book to work on locally, the original remains in the library and can be checked out by other users. When checking out a book remember that it is an edition (branch or head) of the book (repository). The copy of the book which is on the user's local computer is known as the "Working copy", "Work area" or "Work directory".

Changes the user makes to the local copy can be sent back to the library for inclusion in the repository.

Project

Modules, themes or translations can be added (committed) by users to the Contributions book (repository) at any time. It's possible to add to the manuscript (head) of Contributions, or any of the previous editions (branches). The website drupal.org tracks user additions to the Contributions book (repository) as "Projects". Each project has a description page from which it can be downloaded, and where issues and feature requests can be added by users.

Patch

Only a few Drupal developers are able to make changes directly to the Drupal book (repository). All other users who want to include changes they've made while working on their local copy of the Drupal book (repository), must create a file showing the differences between the local version and the version at drupal.org. This differences file is known as a "patch", and is sent to the rest of the development community by creating an "issue" at drupal.org and attaching the file.

Using CVS with branches and tags

To manage the different Drupal versions, we use tags and branches. A branch specifies a major Drupal version. For example, all 4.4.x versions belong in the DRUPAL-4-4 branch. Whenever we release a specific version, we create a tag. A tag is a marker which defines a snapshot of all the files in the CVS at a certain moment. For example, the tag DRUPAL-4-4-0 specifies all files at the time of the 4.4.0 release. The HEAD branch is special and is used to refer to the latest development version.

For an up-to-date complete list of branches and tags, see "Show files using tag:" at ViewCVS (at the bottom).

Here's a quick guide on using tags and branches. This assumes you have successfully checked out the 'main' and 'contributions' repositories. In my case, I usually make a folder in my home directory for each cvs server. In Drupal's case, `cvs.drupal.org`. For instance, you've made the CVS folder and here you check out your copy of the CVS version of Drupal.

```
~cvs.drupal.org$ cvs -d :pserver:anonymous@cvs.drupal.org:/cvs/drupal login
~cvs.drupal.org$ cvs co drupal
```

This should leave you with a folder `cvs.drupal.org/drupal` which contains the current CVS code. You can keep this up-to-date by going into the `cvs.drupal.org/drupal` directory and using the command

```
~/cvs.drupal.org/drupal$ cvs update -dP
```

which will give you the latest copy, create any new directories that exist in the repository (-d), and trim unused directories (-P). Note that you don't need to specfiy the server at this point since the `drupal` directory contains a `CVS` folder that contains the repository and root information.

You've also done this for the contributions,

```
~cvs.drupal.org$ cvs -d :pserver:anonymous@cvs.drupal.org:/cvs/drupal-contrib login
~cvs.drupal.org$ cvs co contributions
```

Which leaves you with the latest contributions in the `cvs.drupal.org/contributions` directory.

But now you want to have a nice copy of the 4.3.0 version, and you don't want to have to download the tgz file all the time. The CVS maintainer has branched the drupal repository and tagged it to keep track of this release. If you download it directly with the release tag it's going to overwrite your drupal folder. You probably want to keep it simple and use this command:

```
~cvs.drupal.org$ cvs -d :pserver:anonymous@cvs.drupal.org:/cvs/drupal -q checkout -d drupal-4.3 -r DRUPAL-4-3 drupal
```

That's going to create a new directory `cvs.drupal.org/drupal-4.3.0` that contains the 4.3.0 version. Once it's been checked out, you don't need to worry about specifying it again. The `CVS` directory in the `drupal-4.3.0` directory has the tag information along with repository and root information like we saw before. Just go into the `drupal-4.3` folder and execute

```
~/cvs.drupal.org/drupal-4.3$ cvs update -dP
```

to keep your copy of the 4.3.0 branch up to date.

Windows

You may have noticed this was geared towards the linux user. Sorry, I haven't used the windows clients for CVS. I'm sure they would work, I just haven't tried them (for very long). You could probably do this same thing on your windows box by installing one of the windows GUIs or using Cygwin, a GNU/UNIX client for windows. Probably the easiest way it to use TortoiseCVS.

Available Branches

The available branches currently are:

- HEAD
- DRUPAL-4-6
- DRUPAL-4-5
- DRUPAL-4-4
- DRUPAL-4-3
- DRUPAL-4-2
- DRUPAL-4-1
- DRUPAL-4-0
- DRUPAL-3-0

The DRUPAL-4-3-0 tag we used above is a marker in the DRUPAL-4-3 branch. Other tags for DRUPAL-4-3 are DRUPAL-4-3-1 and DRUPAL-4-3-2.

Apply for contributions CVS access

CVS GUIs and clients

There are a number of CVS 'front-ends' or GUIs which aim to improve on the command-line tools of CVS. These tools are grouped here by operating system/platform.

Cross-platform CVS clients

There are a number of GUI clients which run on multiple operating systems and platforms

Eclipse CVS plug-in

Perhaps the best CVS front-end I've used is the cross-platform tool Eclipse (www.eclipse.org). It has the functionality to do almost anything you can do with the command line, but you can use the console if you need to. As well, you can use it to edit the PHP code with another plugin.

CVS front ends for Windows

There are many CVS GUI front ends for Windows. Please add a new section if you know of additional ones

TortoiseCVS

TortoiseCVS lets you work with files under CVS version control directly from Windows Explorer. It's freely available under the GPL. The following tutorial teaches how to use TortoiseCVS with Drupal.

- Download TortoiseCVS from http://www.tortoisecvs.org/download.shtml and install it.
- In Windows Explorer, select the folder under which you want the Drupal source directory to live. Right-click on it. There are two new sections in the context menu - **CVS Checkout** and **CVS >**. Select **CVS Checkout**.
- Fill in the following fields:

  Protocol: `Password server (:pserver:)`
  Server: `[cvs.]drupal.org`
  Repository folder: `/cvs/drupal` (main distro) or `/cvs/drupal-contrib` (contributions)
  User name: `anonymous`
  Module: `drupal` (main distro) or

  `contributions`
  (contributions)

  and press "OK".

- You will be asked for password. Enter anonymous and press "OK".
- A log window which monitors the checkout process will appear. Checking out the whole CVS repository will take a while.
- If everything works, you will see the message "Success, CVS operation completed" at the end of the log. A new directory (named like the module selected before) with the sources will be created.
- To bring your Drupal source tree up-to-date, select it's root folder ("drupal" / "contributions"), right-click it and do a "CVS Update".

The process above retrieves the freshest files from the repository (the so-called HEAD branch). These are sometimes unstable. To get Drupal modules and themes that are stable and ready for production (which you can also download from the Drupal downloads page), follow the process described above, but before hitting "OK" you need to:

- Click on the "Revision" tab on the CVS checkout dialog.
- Enable "Get tag/branch".
- Enter `DRUPAL-4-1-0` or `DRUPAL-4-00` depending on the version you are using in the tag/branch field.
- Hit OK.

You can also generate patch files with TortoiseCVS. Just select the files which you have patched in Windows Explorer. Then right click into the `CVS => Make Patch` menu item. Then you may wish to read Creating and sending your patches

WinCVS

WinCVS is another graphical CVS client available for MS Windows and for Macs. You can download the latest version from http://www.wincvs.org/. The checkout / update process is similar to the one described above.

CVS on Mac OS X

There are a number of good CVS clients for Mac OS X.

CVL: point and click CVS

For Mac users who are unused to the command line, CVS can at first look a bit daunting. Fortunately there is an application called CVL that provides control of CVS through a point and click interface. Most of the instructions for using CVL will also apply when using other applications to control CVS.

Setting up/step by step CVS

Here is a step-by-step guide to installing and setting up CVL.

1. **Install CVS.** The first step of using any application is of course to install it - CVS is installed by default with the Apple Developer tools, so if you haven't installed these yet, download the latest version and install them. They also include a lot of other useful stuff like Project Builder and File Merge (Apple Developer Membership is required, but free).

2. **Install CVL**

   The CVL package and complete installation instructions are available at http://www.sente.ch/software/cvl/

3. **Setup CVL to work with CVS.**

   Set CVS to ignore the 'hidden' .DS_Store files which OS X creates in each folder. To do this you need to open the Terminal (Application->Utilities->Terminal), and type the following:

   ```
   cd
   ```

   takes you to root of your account

   ```
   pico
   ```

   opens the Pico text application

   ```
   .DS_Store
   ```

   specifies which file types you want CVS to ignore

   Now press the keys: **Control** and **x** at the same time

   This closes the document you've just written, it will ask you if you want to save it - press **y** for yes, then type in the name of the file **.cvsignore** (note the '**.**') and press return. You've finished with the Terminal, so you can quit it.

4. Create a folder to put the CVS files into. The best place to do this is in the 'Sites' folder, to make it easy to use them through the Apache server built into your system. You can name the folder anything you want, my one is called 'drupal_cvs'.

5. **Step five** open the CVL application, you now need to Checkout (download) the latest version of the Drupal CVS like this:

   Tools->Repositories->Show Repositories

   Click Add

   Choose a repository dialog box will appear.

   In **Repository type** choose **pserver**.

   CVS User: your **Username** (that you applied for CVS with)

   Host: cvs.drupal.org

   Path: /cvs/drupal (main distro) or **/cvs/drupal-contrib** (contributions)

   Password: your **password** (that you applied for CVS with)

   Click **Add**.

   Next go to Tools->Repositories->Show Repositories

   The Drupal repository is now listed in the **Repositories** window. Select it and press **Checkout...**

   Checkout Module dialog box appears.

   Choose Module: drupal (main distro) or **contributions** (contributions)

   New work area location: Choose... select the folder you created in **step four**.

   Press **Checkout**.

   Wait patiently, this may take some time, as the whole of the Repository needs to be downloaded - you can see this happening if you open the console window (Tools->Console->Show Console), don't worry if you don't see anything at first, CVL usually thinks about what it's doing for a minute or two before taking action.

   When this is finished you will have a copy of the Drupal repository files in the folder you created on your hard drive, this is your **Work Area**, where you work on projects before uploading them to the repository for others to use.

Basic CVS with CVL

You now have a **Work Area** on your hard drive which is a mirror of the Repository on the Drupal server. You can see this by using the CVL menu Work Area->Open Recent and selecting the repository you just downloaded (drupal or contributions).

You can use this work area in the same way you would any other folder on your hard drive - create new files with BBEdit (or whatever you use), drag files to the trash, add new folders, delete folders - it's just a regular folder.

Once you've done some work you want to upload back to the Drupal server here's what you do:

Update the CVS by selecting the folder the new work is in, then **Control+Click** on the folder and choose **Update** from the contextual menu that pops up (or through the menu File->Update). CVS now shows any new files or folders that you have added (with a blue * in front).

Next you need to tell CVS to mark the files and folders for upload next time you send your changes to the Drupal repository. To do this select the files and folders and **Control+Click**, choose **Add To Work Area** (or through the menu File->Add To Work Area).

To upload your work to the Drupal repository, select your files and folders and **Control+Click**, choose **Commit...** (or through the menu File->Commit...). CVS will now add your work to the Drupal repository.

Preparing a project

New module, theme or translation projects should be started in the Drupal CVS contributions repository.

In the Finder, go to the folder where you saved the CVS contributions working copy, and create a new folder in the appropriate subfolder. For example, if you are working on a new module, create new folder in the module folder. Name the new folder to whatever you want to call the project. Try to make the name short and descriptive. Avoid spaces, use "_" to separate words, but read the Developer guidelines to understand how underscores in module names may interact with code behaviour.

In CVL, open the CVS contributions work area, navigate to the folder containing the new folder you just created, control-click on it and select "**Refresh**" from the menu that pops up.

The new folder, and any files you put in it, should now show up in CVL with a blue * next to it.

The blue * signifies that the files have not been added to the work area yet.

In CVL select the new folder, control-click on it and select "**Mark File(s) for Addition**" from the menu that pops up.

The blue * will now change to a green **+** next to each of the new files and folders, signifying that the files are part of the working copy and can be added to the repository at drupal.org once you want to commit them.

Committing a project

Once your new module, theme or translation is complete you may want to add it to Drupal's contributed repository, and create a project for it at www.drupal.org.

1. **Add your project to CVS**

   Your new project should first be added to the trunk of the contrib repository, which is know as the 'cvs' or 'HEAD' branch. (see Setting up)

   Add the files to the 'cvs' version of the contrib repository (see Preparing a project). Once added to CVS, the project folder and each of it's files will have a green '+' next to it, this means they are ready to be commited.

   Select the project's folder, for example:

   - **banana.module**
     contrib/modules/banana/banana.module
     select folder '*banana*'

   With the project folder selected, control-click, select 'Commit...'

   A dialog box will appear into which you can type a log message. The log message should briefly explain what new features have been added to this version of the files or what bugs have been fixed.

2. **Add your project to the Drupal Project tracker**

   Your files are now in the contrib repository, now you need to make drupal.org aware of your new project.

   By creating a 'project' at www.drupal.org the files in CVS become available for download on the 'Downloads' page, it also allows users to submit feature requests and bug reports for the project.

   Log in to www.drupal.org, in the side account block click on "create content", then click "project".

   Fill in the project form page to create the new project. The project will apear on drupal.org in a day or two.

Drupal CVS repositories

Main repository

There are two ways to access the latest Drupal sources in the main CVS repository. If you just want to have a quick look at some files, use the ViewCVS web interface. If you need the complete source tree to study and work with the code, follow these steps:

- If you don't have it yet, install a recent copy of CVS. If you are on Windows, you may check CVS front ends for Windows). Mac OS X users may find the tutorial on the CVL front end for CVS in section CVL: point and click CVS helpful.
- Login by running the command:

  ```
  $ cvs -d:pserver:anonymous@cvs.drupal.org:/cvs/drupal login
  ```

  The required password is 'anonymous' (without the quotes).

- To check out the latest drupal sources, run the command:

  ```
  $ cvs -d:pserver:anonymous@cvs.drupal.org:/cvs/drupal checkout drupal
  ```

  This will create a directory called `drupal` containing the latest drupal source tree.

- Once you have a copy of the Drupal source tree, use

  ```
  $ cvs update -dP
  ```

  in the source root dir to update all files to it's latest versions (`-d`: Create any (new) directories that exist in the repository if they're missing from the working directory. `-P`: Prune empty directories - directories that got removed in the repository will be removed in your working copy, too).

If you can't or don't want to use CVS, you can download nightly CVS snapshots from http://drupal.org/files/projects/drupal-cvs.tar.gz.

Contributions repository

The Contributions repository is a seperate CVS repository where people can submit their modules, themes, translations, etc. See the contributions FAQ.txt and README.txt for more information.

As the Main repository, you can browse it via the web interface. For anonymous (read-only) access, do the following:

- Login by running the command

```
$ cvs -d:pserver:anonymous@cvs.drupal.org:/cvs/drupal-contrib login
```

The required password is 'anonymous' (without the quotes).

- To check out the latest drupal contributions, run the command:

```
$ cvs -d:pserver:anonymous@cvs.drupal.org:/cvs/drupal-contrib checkout contributions
```

- To check out contributions for a certain Drupal version, do

```
$ cvs -d:pserver:anonymous@cvs.drupal.org:/cvs/drupal-contrib checkout
      -r <version tag> contributions
```

where <*version tag*> is one of the tags listed under "Q: How do I control the releases of my module/theme?" here.

- To update your tree to the latest version, do

```
$ cvs update -dP
```

in the source root dir.

If you want to add your own modules, themes, translations, etc., you need CVS write access:

Promoting a project to be an official release

To promote a project from the HEAD of the CVS tree to an official release state, the author needs to move the CVS tag.

For instance, to promote the CVS HEAD of the French translation to the official 4.5 release, using the command line from within inside the contributions/translation/fr, just do:

```
$ cvs up -dA
$ cvs tag -F DRUPAL-4-5
```

Adding a file to the CVS repository

If you would like to add a file or a directory, first you need to download the parent directory. Once again: the *parent* directory, not the directory you want to add something to, but the parent of it. I can not find any logic in this, but this is so.

First, issue the following command:

```
export
CVSROOT=:pserver:icvslogin:cvspasswd@cvs.drupal.org:/cvs/drupal-contrib
```

To add a new file to a module:

```
cvs co contributions/modules/modulename
cd contributions/modules
```

```
cp sourcefile modulename
cvs add modules/sourcefilename
cvs commit
```

Say you want to create a sandbox named `mysandbox`. Then do the following:

```
cvs co contributions/sandbox/weblinks [1]
cd contributions
mkdir sandbox/mysandbox
cvs add sandbox/mysandbox
cvs commit
```

[1] does not really matter which directory, just sg. from sandbox. I like this one, because it is small.

Of course, you may do several things in one commit, adding files, removing files, updating files. Neither remove (cvs remove) nor update (cvs update) is such a tedious process. Warning: you need to check out with your CVS account, because CVS ignores CVSROOT for existing checkouts.

Tracking Drupal source with CVS

Note: The following assumes you have both basic knowledge of CVS and your own local repository set up and working.

If you've been modifying the Drupal source code for your own purposes (or developing a module or theme) and manually applying your changes to the Drupal source every time it updates, you may be glad to learn that CVS can help make this easier.

This is usually referred to as 'tracking third-party sources' and requires knowledge of the CVS concepts branching, release tags, and the vendor tag. We'll work through an example here and explain these concepts as we go.

Example

Lets assume we'd like to track current Drupal CVS HEAD, and start by downloading the source. In this case we'll export using anonymous CVS (we could also just download a tarball).

Begin by logging in to the anonymous CVS server, the required password is 'anonymous':

```
cvs -d:pserver:anonymous@cvs.drupal.org:/cvs/drupal login
```

Then export the newest development version of drupal using the HEAD release tag:

```
cvs -d:pserver:anonymous@cvs.drupal.org:/cvs/drupal export -r HEAD drupal
```

Now that we have a local copy of the drupal source we can import it into our own CVS repository. In this example we import with a log message including the date '`-m "message text"`', a module location/name of '`sites/drupal`' (customize that to suit your own CVS repository), a vendor tag of '`drupal`' and a release tag of '`HEAD20040110`'. We also use the -ko

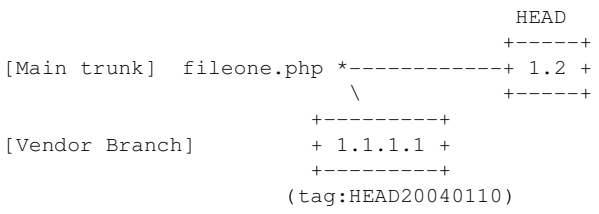
option to prevent keyword expansion (this preserves the CVS $ Id $ tags used on drupal.org):

```
cd drupalcvs import -ko -m "Import CVS HEAD on Jan 10th 2004" sites/drupal drupal HEAD20040110
```

Before we can customize we need to checkout into a working directory. Then we can modify a file or files and commit:

```
cvs checkout drupalcd drupal...modify a file or files...cvs commit
```

We now have a drupal module with a special 'vendor branch' (identified by the vendor tag), which contains the drupal source files we imported, and a main trunk with our modified files. Any files modified at this point are now HEAD on the main trunk of the module, whilst the unmodified files remain HEAD on the vendor branch (HEAD being what is produced by `cvs update`). For an individual file (fileone.php) the version history now looks like something like this:



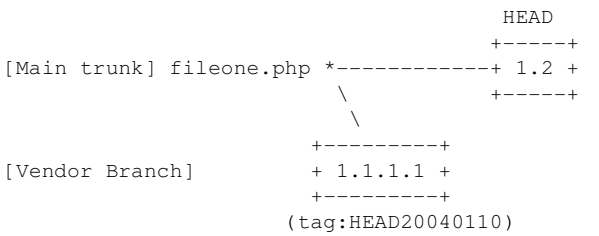
Updating the vendor branch

At some later point the drupal source code will have been updated and we'll want to add the updated version to our repository. We do this by repeating the process described above, we get a fresh copy of the source from drupal.org, and import using the same vendor tag but change the release tag from '`HEAD20040110`' to reflect the newer version:

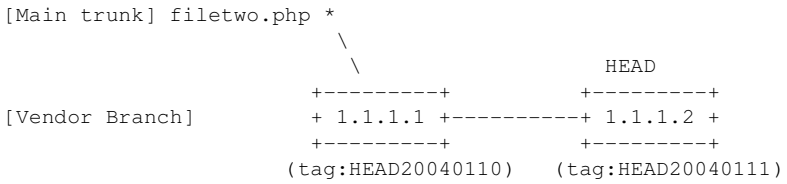
```
cvs import -ko -m "Import CVS HEAD on Jan 11th 2004" sites/drupal drupal HEAD20040111
```

This updates the vendor branch, a single files revision history can now appear four different ways, depending on whether it has been modified by us, by the vendor (drupal.org), by both, or not at all.

If the file was modified only by us, our modified version remains the head revision:



If the file was modified only by the vendor, the new version becomes the HEAD revision:



And if the file was modified by both us and the vendor:

Our version of filethree.php remains the HEAD revision, but this is clearly not desirable since it doesn't carry the latest changes. In fact, during our import of the latest source CVS would have warned us of conflicts between the two versions of filethree.php, we need to merge the changes to remove this conflict:

```
cvs checkout -jHEAD20040110 -jHEAD20040111 drupal
```

Examine the merged file to ensure the changes CVS made were sane and then '`cvs commit`' the changes back to the main trunk. Leaving us with a new revision which becomes HEAD:

Summary

It should now be clear that using the CVS vendor tag to create a vendor branch in your own drupal module you can track changes to the drupal source code whilst also maintaining and developing your own customizations and new features for drupal. This example has been kept very simple for the purposes of explanation, but the basic process can be used to achieve many different things, some examples:

- Track a specific release of Drupal (e.g. 4.3, or 4.2), instead of the development (CVS HEAD) version.
- Maintain your customized sites with modules, themes, static pages, images etc all added to your CVS repository, whilst still tracking and importing updates to the drupal core.
- Branch your module to maintain several customized web sites off a single tracked branch of the drupal core.

Reading the following additional resources is highly recommended.

Additional resources

- Article by Nick Patavalis: The mechanics and a methodology for tracking 3rd party sources with CVS
- The section ''Tracking third-party sources'' in the CVS manual
- The section ''Tracking third-party sources'' in a book on CVS

Sandbox maintenance rules

1. Always document your changes.
2. Split different set of patches into different directories. It takes longer to find the set of files relating to one change if it is mixed in with 2 other patches.
3. Keep the documentation current. Try to keep some track of your reasoning too. If I read in a README that change X wasn't a good idea after all it makes the reviewer wonder why.
4. Document the status of your patch. It is important to know if this is an early test, or considered stable and workable but the author of the patch.
5. All patches should be against the latest CVS version of Drupal, and include in the README when it was last synced.
6. Don't use a sandbox for developing modules. There is a different directory structure for that.
7. If your patch is 4 lines long don't bother to put it in a sandbox. Just mail it to the devel list and find out quicker if people like it or not. Small patches are quick to check and find out if work. Sandboxes should be for more extensive changes.
8. Try to maintain patches in the sandbox. They are so much easier to check than compete files. If you are using CVS then you can use diff (cvs -H diff)
9. Please make sure your script passes the code-style.pl script. It isn't perfect, and sometimes a bit too strict, but it will ensure some level of compliance with the coding standards.

Additional references

- CVS book
- CVS docs
- CVS FAQ
- CVS guide from TLDP

Drupal's APIs

If you are interested in developing Drupal modules or hacking away at the Drupal core then this is the place to find details about all the functions and classes defined in Drupal.

We now use Doxygen to automatically generate documentation from the latest drupal sources. This allows us to ensure that documentation is up-to-date, and to simultaneously track multiple versions of the documentation.

API Documentation is available from drupaldocs.org for:

- Drupal CVS HEAD
- Drupal 4.5.x
- Drupal 4.4.x

Please also read the Drupal Coding Standards page, which contains some guidelines for writing Doxygen comments.

Module developer's guide

Developer documentation can be found at http://drupaldocs.org/ and in the remainder of the Drupal developer's guide below.

- drupaldocs.org documents the Drupal APIs and presents an overview of Drupal's building blocks along with handy examples.
- The Drupal developer guide provides guidlines as how to upgrade your modules (API changes) along with development tips/tutorials.

Introduction to Drupal modules

When developing Drupal it became clear that we wanted to have a system which is as modular as possible. A modular design will provide flexibility, adaptability, and continuity which in turn allows people to customize the site to their needs and likings.

A Drupal module is simply a file containing a set of routines written in PHP. When used, the module code executes entirely within the context of the site. Hence it can use all the functions and access all variables and structures of the main engine. In fact, a module is not any different from a regular PHP file: it is more of a notion that automatically leads to good design principles and a good development model. Modularity better suits the open-source development model, because otherwise you can't easily have people working in parallel without risk of interference.

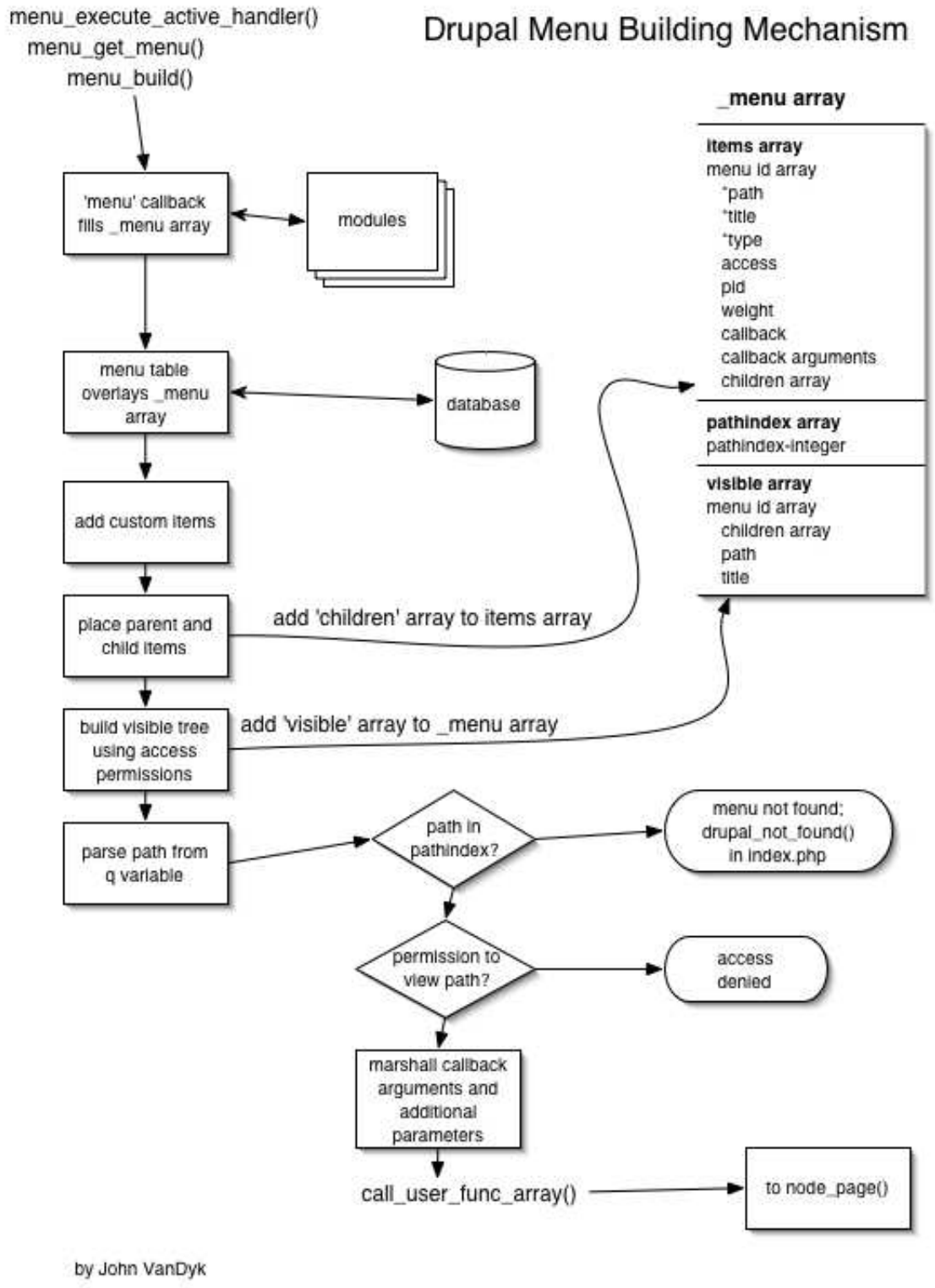
The idea is to be able to run random code at given places in the engine. This random code should then be able to do whatever needed to enhance the functionality. The places where code can be executed are called "hooks" and are defined by a fixed interface.

In places where hooks are made available, the engine calls each module's exported functions. This is done by iterating through the modules directory where all modules must reside. Say your module is named foo (i.e. modules/foo.module) and if there was a hook called bar, the engine will call foo_bar() if this was exported by your module.

See also the overview of module hooks, which is generated from the Drupal source code.

Drupal's menu building mechanism

(Note: this is an analysis of the menu building mechanism in pre-4.5 CVS as of August 2004. It does not include menu caching.)


menu_execute_active_handler()
menu_get_menu()
menu_build()
Drupal Menu Building Mechanism
'menu' callback fills _menu array
modules
menu table overlays _menu array
database
add custom items
place parent and child items
add 'children' array to items array
build visible tree using access permissions
add 'visible' array to _menu array
parse path from q variable
path in pathindex?
menu not found; drupal_not_found() in index.php
permission to view path?
access denied
marshall callback arguments and additional parameters
call_user_func_array()
to node_page()
_menu array
items array
menu id array
*path
*title
*type
access
pid
weight
callback
callback arguments
children array
pathindex array
pathindex-integer
visible array
menu id array
children array
path
title
by John VanDyk

This continues our examination of how Drupal serves pages. We are looking specifically at how the menu system works and is built, from a technical perspective. See the excellent overview in the menu system documentation.

We begin in index.php, where menu_execute_active_handler() has been called. Diving in from menu_execute_active_handler(), we immediately set the $menu variable by calling menu_get_menu(). The latter function declares the global $_menu array (note the underline, it means a 'super global', which is a predefined array in PHP lore) and calls _menu_build() to fill the array, then returns $_menu. Although menu_get_menu() initializes the $_menu array, the _menu_build() function actually reinitializes the $_menu array. Then it sets up two main arrays within $_menu: the items array and the path index array.

The items array is an array keyed to integers. Each entry contains the following fields:

Required fields		
path	string	the partial URL to the page for this menu item
title	string	the title that this menu item will have in the menu
type	integer	a constant denoting the menu item type (see comments in menu.inc)
Optional fields		
access	boolean	
pid	integer	
weight	integer	
callback	string	name of the function to be called if this menu item is selected
callback arguments	array	

An array called $menu_item_list is populated by sending a 'menu' callback to all modules with 'menu' hooks (that is, they have a function called foo_menu() where foo is the name of the module). So each module has a chance to register its own menu items. It is interesting that when the node module receives the menu callback through node_menu(), and the path is something like 'node/1' as it is in our present case, the complete node is actually loaded via the node_load() function so it can be examined for permissions. The $node variable into which it was loaded then goes out of scope, so the node is gone and needs to be rebuilt completely later on. This seems like a golden opportunity for the node module to cache the node.

The $menu_item_list array is normalized by making sure each array entry has a path, type and weight entry. As each entry is examined, the path index array of the $_menu array is checked to see if the path of this menu item exists. If an equivalent path is already there in the path index array, it is blasted away. The path index of this menu item is then added as a key with the value being the menu id. In the items array of the $_menu array, the menu id is used as the key and

the entire array entry is the value.

Note: the $temp_mid and $mid variables seem to do the same thing. Why, syntactically, cannot only one be used?

The path index array contained 76 items when serving out a simple node with only the default modules enabled.

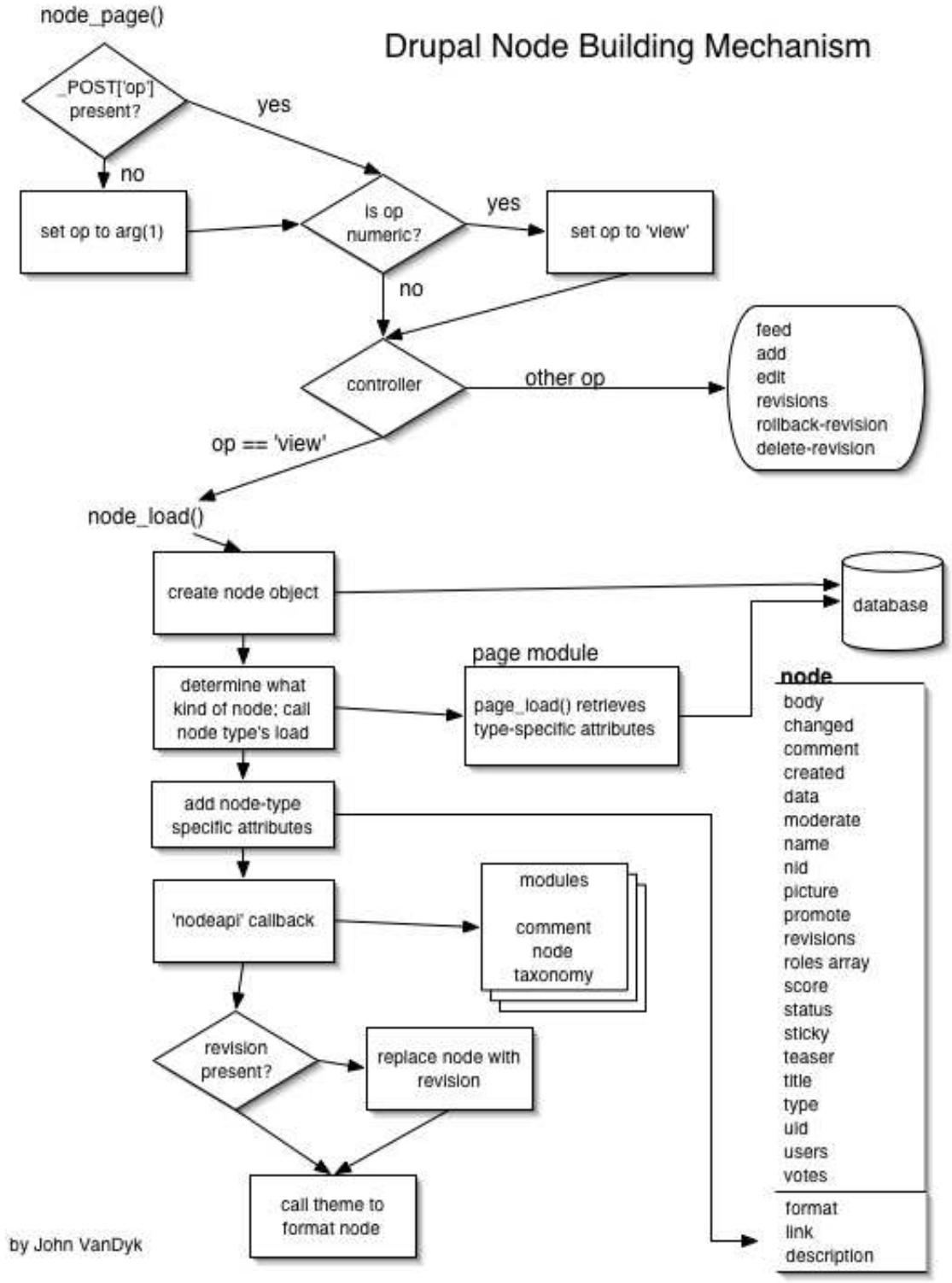
Next the menu table from the database is fetched and its contents are used to move the position of existing menu items from their current menu ids to the menu ids saved in the database. The comments says "reassigning menu IDs as needed." This is probably to detect if the user has customized the menu entries using the menu module. The path index array entries generated from the database can be recognized because their values are strings, whereas up til now the values in the path index array have been integers.

Now I get sort of lost. It looks like the code is looking at paths to determine which menu items are children of other menu items. Then _menu_build_visible_tree is a recursive function that builds a third subarray inside $_menu, to go along with items and path index. It is called visible and takes into account the access attribute and whether or not the item is hidden in order to filter the items array. As an anonymous user, all items but the Navigation menu item are filtered out. See also the comments in menu.inc for menu_get_menu(). In fact, read all the comments in menu.inc!

Now the path is parsed out from the q parameter of the URL. Since node/1 is present in the path index, we successfully found a menu item. It points to menu item -44 in our case, to be precise, but there must be a bug in the Zend IDE because it shows item -44 as null. Anyway, the menu item entry is checked for callback arguments (there are none) and for additional parameters (also none), and execution is passed off to node_page() through the call_user_func_array function.

Drupal's node building mechanism

(This walkthrough done on pre-4.5 CVS code in August 2004.)



The node_page controller checks for a $_POST['op'] entry and, failing that, sets $op to arg(1) which in this case is the '1' in node/1. A numeric $op is set to arg(2) if arg(2) exists, but in this

case it doesn't ('1' is the end of the URL, remember?) so the $op is hardcoded to 'view'. Thus, we succeed in the 'view' case of the switch statement, and are shunted over to node_load(). The function node_load() takes two arguments, $conditions (an array with nid set to desired node id -- other conditions can be defined to further restrict the upcoming database query) for which we use arg(1), and $revision, for which we use _GET['revision']. The 'revision' key of the _GET array is unset so we need to make brief stop at error_handler because of an undefined index error. That doesn't stop us, though, and we continue pell-mell into node_load using the default $revision of -1 (that is, the current revision). The actual query that ends up being run is

SELECT n.*, u.uid, u.name, u.picture, u.data FROM node n INNER JOIN users u on u.uid WHERE n = '1'

We get back a joined row from the database as an object. The data field from the users table is serialized, so it must be unserialized. This data field contains the user's roles. How does this relate to the user_roles table? Note that the comment "// Unserialize the revisions and user data fields" should be moved up before the call to drupal_unpack().

We now have a complete node that looks like the following:

Attribute	Value	
body	This is a test node body	
changed	1089859653	
comment	2	
created	1089857673	
data	a:1:{s:5... (serialized data)	
moderate	0	
name	admin	
nid	1	
picture	''	
promote	1	
revisions	''	
roles	array containing one key-value pair, 0 = '2'	
score	0	
status	1	
sticky	0	
teaser	This is a test node body	
title	Test	
type	page	
uid	1	
users	''	
votes	0	

All of the above are strings except the roles array.

So now we have a node loaded from the database. It's time to notify the appropriate module that this has happened. We do this via the node_invoke($node, 'load') call. The module called via this callback may return an array of key-value pairs, which will be added to the node above.

The node_invoke() function asks node_get_module_name() to determine the name of the module that corresponds with the node's type. In this case, the node type is a page, so the page.module is the one we'll call, and the specific name of the function we'll call is page_load(). If the name of the node type has a hyphen in it, the left part is used. E.g., if the node type is

page-foo, the page module is used.

The page_load() function turns out to be really simple. It just retrieves the format, link and description columns from the page table. The 'format' column specifies whether we're dealing with a HTML or PHP page. The 'link' and 'description' fields are used to generate a link to the newly created page, however, those will be deprecated with the improved menu system. To that extend, the core themes no longer use this information (unlike some older themes in the contributions repository). We return to node_load(), where the format, link and description key-value pairs are added to the node's definition.

Now it's time to call the node_invoke_nodeapi() function to allow other modules to do their thing. We check each module for a function that begins with the module's name and ends with _nodeapi(). We hit paydirt with the comment module, which has a function called comment_nodeapi(&$node, $op, arg = 0). Note that the node is passed in by reference so that any changes made by the module will be reflected in the actual node object we built. The $op argument is 'load', in this case. However, this doesn't match any of comment_nodeapi()'s symbols in its controller ('settings', 'fields', 'form admin', 'validate' and 'delete' match). So nothing happens.

Our second hit is node_nodeapi(&$node, $op, $arg = 0) in the node.module itself. Again, no symbols are matched in the controller so we just return.

We'll try again with taxonomy_nodeapi(&$node, $op, $arg = 0). Again, no symbols match; the taxonomy module is concerned only with inserts, updates and deletes, not loads.

Note that any of these modules could have done anything to the node if they had wished.

Next, the node is replaced with the appropriate revision of the node, if present as an attribute of $node. It is odd that this occurs here, as all the work that may have been done by modules is summarily blown away if a revision other than the default revision is found.

Finally, back in node_page(), we're ready to get down to business and actually produce some output. This is done with the statement

print theme('page', node_show($node, arg(3)), $node->title);

And what that statement calls is complex enough to again warrant another commentary. (Not yet done.)

How Drupal handles access

I believe this page should explain how user_access table works.
1.- Drupal checks if the user has access to that module, if he does ...
2.- The he checks the user _access page where gid is the role, view should be 1 and realm should be "all". If there is no access given in that table, he will not give the access to the user.

I believe there is not enough documentation on how to use node access, and hopefully this page will have more information as people contribute.

Drupal's page serving mechanism

This is a commentary on the process Drupal goes through when serving a page. For convenience, we will choose the following URL, which asks Drupal to display the first node for us. (A node is a thing, usually a web page.)

```
http://127.0.0.1/~vandyk/drupal/?q=node/1
```

A visual companion to this narration can be found here; you may want to print it out and follow along. Before we start, let's dissect the URL. I'm running on an OS X machine, so the site I'm serving lives at /Users/vandyk/Sites/. The drupal directory contains a checkout of the latest Drupal CVS tree. It looks like this:

```
CHANGELOG.txt
cron.php
CVS/
database/
favicon.ico
includes/
index.php
INSTALL.txt
LICENSE.txt
MAINTAINERS.txt
misc/
modules/
phpinfo.php
scripts/
themes/
tiptoe.txt
update.php
xmlrpc.php
```

So the URL above will be be requesting the root directory / of the Drupal site. Apache translates that into `index.php`. One variable/value pair is passed along with the request: the variable 'q' is set to the value 'node/1'.

So, let's pick up the show with the execution of index.php, which looks very simple and is only a few lines long.

Let's take a broad look at what happens during the execution of `index.php`. First, the `includes/bootstrap.inc` file is included, bringing in all the functions that are necessary to get Drupal's machinery up and running. There's a call to `drupal_page_header()`, which starts a timer, sets up caching, and notifies interested modules that the request is beginning. Next, the `includes/common.inc` file is included, giving access to a wide variety of utility

functions such as path formatting functions, form generation and validation, etc. The call to `fix_gpc_magic()` is there to check on the status of PHP "magic quotes" and to ensure that all escaped quotes enter Drupal's database consistently. Drupal then builds its navigation menu and sets the variable `$status` to the result of that operation. In the switch statement, Drupal checks for cases in which a Not Found or Access Denied message needs to be generated, and finally a call to `drupal_page_footer()`, which notifies all interested modules that the request is ending. Drupal closes up shop and the page is served. Simple, eh?

Let's delve a little more deeply into the process outlined above.

The first line of `index.php` includes the `includes/bootstrap.inc` file, but it also executes code towards the end of `bootstrap.inc`. First, it destroys any previous variable named `$conf`. Next, it calls `conf_init()`. This function allows Drupal to use site-specific configuration files, if it finds them. The name of the site-specific configuration file is based on the hostname of the server, as reported by PHP. `conf_init` returns the name of the site-specific configuration file; if no site-specific configuration file is found, sets the variable `$config` equal to the string `$confdir/default`. Next, it includes the named configuration file. Thus, in the default case it will include `sites/default/settings.php`. The code in `conf_init()` would be easier to understand if the variable `$file` were instead called `$potential_filename`. Likewise `$conf_filename` would be a better choice than `$config`.

The selected configuration file (normally `/sites/default/settings.php`) is now parsed, setting the `$db_url variable`, the optional `$db_prefix variable`, the `$base_url` for the website, and the `$languages` array (default is `"en"=>"english"`).

The `database.inc` file is now parsed, with the primary goal of initializing a connection to the database. If MySQL is being used, the `database.mysql.inc` files is brought in. Although the global variables `$db_prefix`, `$db_type`, and `$db_url` are set, the most useful result of parsing `database.inc` is a global variable called `$active_db` which contains the database connection handle.

Now that the database connection is set up, it's time to start a session by including the `includes/session.inc` file. Oddly, in this include file the executable code is located at the top of the file instead of the bottom. What the code does is to tell PHP to use Drupal's own session storage functions (located in this file) instead of the default PHP session code. A call to PHP's `session_start()` function thus calls Drupal's `sess_open()` and `sess_read()` functions. The `sess_read()` function creates a global `$user object` and sets the `$user->roles` array appropriately. Since I am running as an anonymous user, the `$user->roles` array contains one entry, `1->"anonymous user"`.

We have a database connection, a session has been set up...now it's time to get things set up for modules. The `includes/module.inc` file is included but no actual code is executed.

The last thing `bootstrap.inc` does is to set up the global variable `$conf`, an array of configuration options. It does this by calling the `variable_init()` function. If a per-site configuration file exists and has already populated the `$conf` variable, this populated array is passed in to `variable_init()`. Otherwise, the `$conf` variable is null and an empty array is passed in. In both cases, a populated array of name-value pairs is returned and assigned to the

global $conf variable, where it will live for the duration of this request. It should be noted that name-value pairs in the per-site configuration file have precedence over name-value pairs retrieved from the "variable" table by variable_init().

We're done with bootstrap.inc! Now it's time to go back to index.php and call drupal_page_header(). This function has two responsibilities. First, it starts a timer if $conf['dev_timer'] is set; that is, if you are keeping track of page execution times. Second, if caching has been enabled it retrieves the cached page, calls module_invoke_all() for the 'init' and 'exit' hooks, and exits. If caching is not enabled or the page is not being served to an anonymous user (or several other special cases, like when feedback needs to be sent to a user), it simply exits and returns control to index.php.

Back at index.php, we find an include statement for common.inc. This file is chock-full of miscellaneous utility goodness, all kept in one file for performance reasons. But in addition to putting all these utility functions into our namespace, common.inc includes some files on its own. They include theme.inc, for theme support; pager.inc for paging through large datasets (it has nothing to do with calling your pager); and menu.inc. In menu.inc, many constants are defined that are used later by the menu system.

The next inclusion that common.inc makes is xmlrpc.inc, with all sorts of functions for dealing with XML-RPC calls. Although one would expect a quick check of whether or not this request is actually an XML-RPC call, no such check is done here. Instead, over 30 variable assignments are made, apparently so that if this request turns to actually be an XML-RPC call, they will be ready. An xmlrpc_init() function instead may help performance here?

A small tablesort.inc file is included as well, containing functions that help behind the scenes with sortable tables. Given the paucity of code here, a performance boost could be gained by moving these into common.inc itself.

The last include done by common.inc is file.inc, which contains common file handling functions. The constants FILE_DOWNLOADS_PUBLIC = 1 and FILE_DOWNLOADS_PRIVATE = 2 are set here, as well as the FILE_SEPARATOR, which is \\ for Windows machines and / for all others.

Finally, with includes finished, common.inc sets PHP's error handler to the error_handler() function in the common.inc file. This error handler creates a watchdog entry to record the error and, if any error reporting is enabled via the error_reporting directive in PHP's configuration file (php.ini<code>), it prints the error message to the screen. Drupal's <code>error_handler() does not use the last parameter $variables, which is an array that points to the active symbol table at the point the error occurred. The comment "// set error handler:" at the end of common.inc is redundant, as it is readily apparent what the function call to set_error_handler() does.

The Content-Type header is now sent to the browser as a hard coded string: "Content-Type: text/html; charset=utf-8".

If you remember that the URL we are serving ends with `/~vandyk/drupal/?q=node/1`, you'll note that the variable `q` has been set. Drupal now parses this out and checks for any path aliasing for the value of `q`. If the value of `q` is a path alias, Drupal replaces the value of `q` with the actual path that the value of `q` is aliased to. This sleight-of-hand happens before any modules see the value of `q`. Cool.

Module initialization now happens via the `module_init()<code> function. This function runs <code>require_once()<code> on the <code>admin, filter, system, user` and `watchdog` modules. The filter module defines `FILTER_HTML*` and `FILTER_STYLE*` constants while being included. Next, other modules are `include_once`'d via `module_list()`. In order to be loaded, a module must (1) be enabled (that is, the status column of the "system" database table must be set to 1), and (2) Drupal's throttle mechanism must determine whether or not the module is eligible for exclusion when load is high. First, it determines whether the module is eligible by looking at the throttle column of the "system" database table; then, if the module is eligible, it looks at `$conf["throttle_level"]` to see whether the load is high enough to exclude the module. Once all modules have been `include_once`'d and their names added to the `$list` local array, the array is sorted by module name and returned. The returned `$list` is discarded because the `module_list()` invocation is not part of an assignment (e.g., it is simply `module_list()` and not `$module_list = module_list()`). The strategy here is to keep the module list inside a static variable called `$list` inside the `module_list()` function. The next time `module_list()` is called, it will simply return its static variable `$list` rather than rebuilding the whole array. We see that as we follow the final objective of `module_init()`; that is, to send all modules the "init" callback.

To see how the callbacks work let's step through the init callback for the first module. First `module_invoke_all()` is called and passed the string enumerating which callback is to be called. This string could be anything; it is simply a symbol that call modules have agreed to abide by, by convention. In this case it is the string "init".

The `module_invoke_all()` function now steps through the list of modules it got from calling `module_list()`. The first one is `"admin"`, so it calls `module_invoke("admin","init")`. The `module_invoke()` function simply puts the two together to get the name of the function it will call. In this case the name of the function to call is `"admin_init()"`. If a function by this name exists, the function is called and the returned result, if any, ends up in an array called `$return` which is returned after all modules have been invoked. The lesson learned here is that if you are writing a module and intend to return a value from a callback, you must return it as an array. [Jonathan Chaffer: *Each "hook" (our word for what you call a callback) defines its own return type. See the full list of hooks available to module developers, with documentation about what they are expected to return.*]

Back to `common.inc`. There is a check for suspicious input data. To find out whether or not the user has permission to bypass this check, `user_access()` is called. This retrieves the user's permissions and stashes them in a static variable called `$perm`. Whether or not a user has permission for a given action is determined by a simple substring search for the name of the permission (e.g., "bypass input data check") within the `$perm` string. Our `$perm` string, as an anonymous user, is currently "0access content, ". Why the 0 at the beginning of the string?

Because $perm is initialized to 0 by user_access().

The actual check for suspicious input data is carried out by valid_input_data() which lives in common.inc. It simply goes through an array it's been handed (in this case the $_REQUEST array) and checks all keys and values for the following "evil" strings: javascript, expression, alert, dynsrc, datasrc, data, lowsrc, applet, script, object, style, embed, form, blink, meta, html, frame, iframe, layer, ilayer, head, frameset, xml. If any of these are matched watchdog records a warning and Drupal dies (in the PHP sense). I wondered why both the keys and values of the $_REQUEST array are examined. This seems very time-consuming. Also, would it die if my URL ended with "/?xml=true" or "/?format=xml"?

The next step in common.inc's executable code is a call to locale_init() to set up locale data. If the user is not an anonymous user and has a language preference set up, the two-character language key is returned; otherwise, the key of the single-entry global array $language is returned. In our case, that's "en".

The last gasp of common.inc is to call init_theme(). You'd think that for consistency this would be called theme_init() (of course, that would be a namespace clash with a callback of the same name). This finds out which themes are available, which the user has selected, and then include_once's the chosen theme. If the user's selected theme is not available, the value at $conf["theme_default"] is used. In our case, we are an anonymous user with no theme selected, so the default xtemplate theme is used. Thus, the file themes/xtemplate/xtemplate.theme is include_once'd. The inclusion of xtemplate.theme calls include_once("themes/xtemplate/xtemplate.inc"), and creates a new object called xtemplate as a global variable. Inside this object is an xtemplate object called "template" with lots of attributes. Then there is a nonfunctional line where SetNullBlock is called. A comment indicates that someone is aware that this doesn't work.

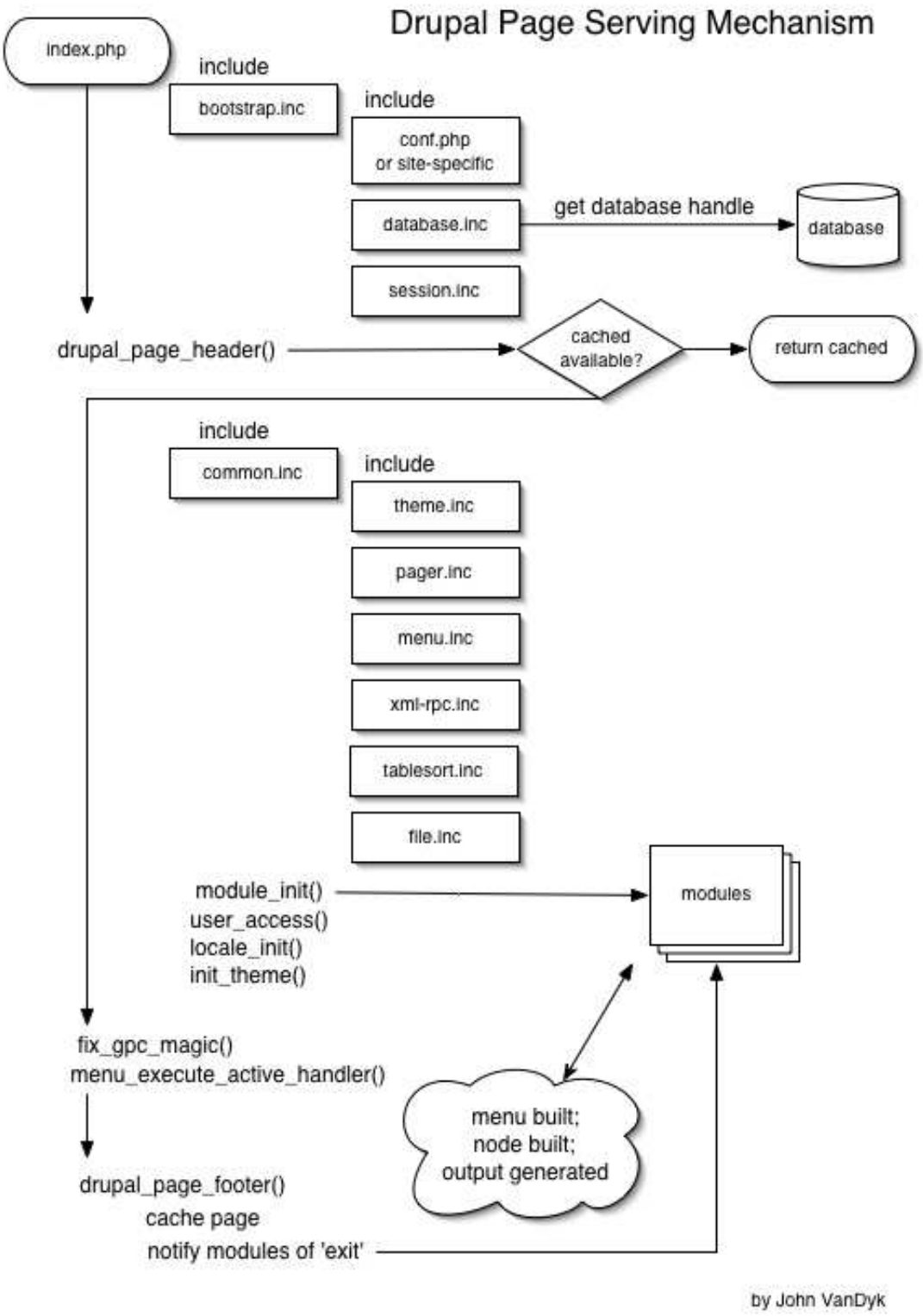
Now we're back to index.php! A call to fix_gpc_magic() is in order. The "gpc" stands for Get, Post, Cookie: the three places that unescaped quotes may be found. If deemed necessary by the status of the boolean magic_quotes_gpc directive in PHP's configuration file (php.ini), slashes will be stripped from $_GET, $_POST, $_COOKIE, and $_REQUEST arrays. It seems odd that the function is not called fix_gpc_magic_quotes, since it is the "magic quotes" that are being fixed, not the magic. In my distribution of PHP, the magic_quotes_gpc directive is set to "Off", so slashes do not need to be stripped.

The next step is to set up menus. This step is crucial. The menu system doesn't just handle displaying menus to the user, but also determines what function will be handed the responsibility of displaying the page. The "q" variable (we usually call the Drupal path) is matched against the available menu items to find the appropriate callback to use. Much more information on this topic is available in the menu system documentation for developers. We jump to menu_execute_active_handler() in menu.inc. This sets up a $_menu array consisting of items, local tasks, path index, and visible arrays. Then the system realizes that we're not going to be building any menus for an anonymous user and bows out. The real meat of the node creation and formatting happens here, but is complex enough for a separate commentary. Back in index.php, the switch statement doesn't match either case and we approach the last call in the file, to drupal_page_footer in common.inc. This takes care of

caching the page we've built if caching is enabled (it's not) and calls `module_invoke_all()` with the "exit" callback symbol.

Although you may think we're done, PHP's session handler still needs to tidy up. It calls `sess_write()` in `session.inc` to update the session database table, then `sess_close()` which simply returns 1.

We're done.


Drupal Page Serving Mechanism
index.php
include
bootstrap.inc
include
conf.php
or site-specific
database.inc
get database handle
database
session.inc
drupal_page_header()
cached
available?
return cached
include
common.inc
include
theme.inc
pager.inc
menu.inc
xml-rpc.inc
tablesort.inc
file.inc
module_init()
user_access()
locale_init()
init_theme()
modules
fix_gpc_magic()
menu_execute_active_handler()
menu built;
node built;
output generated
drupal_page_footer()
cache page
notify modules of 'exit'
by John VanDyk

Creating modules - a tutorial

This tutorial describes how to create a module for Drupal 4.5.*. It is an update to the tutorial for Drupal 4.3. Please see comments there, also.

A module is a collection of functions that link into Drupal, providing additional functionality to your Drupal installation. After reading this tutorial, you will be able to create a basic block module and use it as a template for more advanced modules and node modules.

This tutorial will not necessarily prepare you to write modules for release into the wild. It does not cover caching, nor does it elaborate on permissions or security issues. Use this tutorial as a starting point, and review other modules and the Drupal handbook and Coding standardsfor more information.

This tutorial assumes the following about you:

- Basic PHP knowledge, including syntax and the concept of PHP objects
- Basic understanding of database tables, fields, records and SQL statements
- A working Drupal installation
- Drupal administration access
- Webserver access

This tutorial does not assume you have any knowledge about the inner workings of a Drupal module. This tutorial will not help you write modules for versions of Drupal earlier than 4.5.

Getting started

To focus this tutorial, we'll start by creating a block module that lists links to content such as blog entries or forum discussions that were created one week ago. The full tutorial will teach us how to create block content, write links, and retrieve information from Drupal nodes.

Start your module by creating a PHP file and save it as 'onthisdate.module' in the modules directory of your Drupal installation.

```
<?php
?>
```

As per the Coding standards, use the longhand <?php tag, and not <? to enclose your PHP code.

All functions in your module are named {modulename}_{hook}, where "hook" is a well defined function name. Drupal will call these functions to get specific data, so having these well defined names means Drupal knows where to look.

Letting Drupal know about the new function

As mentioned above, the function we just wrote isn't a 'hook': it's not a Drupal recognized name. We need to tell Drupal how to access the function when displaying a page. We do this with the menu() hook. The menu() hook defines the association between a URL and the function that creates the content for that url. The hook also does permission checking, if desired.

```
<?php
function onthisdate_menu() {
  $items = array();
  $items[] = array('path' => 'onthisdate',
                                    'title' => t('on this date'),
                                    'callback' => '_onthisdate_all',
                                    'access' => user_access('access
content'),
                                    'type' => MENU_CALLBACK);
  return $items;
}
?>
```

Basically, we're saying if the user goes to "onthisdate" (either via ?q=onthisdate or http://.../onthisdate), the content generated by onthisdate_all will be displayed. The title of the page will be "on this date". The type MENU_CALLBACK Drupal to not display the link in the user's menu, just use this function when the URL is accessed. Use MENU_LOCAL_TASK if you want the user to see the link in the side navigation block.

Navigate to /onthisdate (or ?q=onthisdate) and see what you get.

Telling Drupal about your module

The first function we'll write will tell Drupal information about your module: its name and description. The hook name for this function is 'help', so start with the onthisdate_help function:

```
<?php
function onthisdate_help($section='') {
}
?>
```

The $section variable provides context for the help: where in Drupal or the module are we looking for help. The recommended way to process this variable is with a switch statement. You'll see this code pattern in other modules.

```
<?php
/**
* Display help and module information
* @param section which section of the site we're displaying help
* @return help text for section
```

```
*/
function onthisdate_help($section='') {
  $output = '';
  switch ($section) {
    case "admin/modules#description":
      $output = t("Displays links to nodes created on this date");
      break;
  }
  return $output;
} // function onthisdate_help
?>
```

You will eventually want to add other cases to this switch statement to provide real help messages to the user. In particular, output for "admin/help#onthisdate" will display on the main help page accessed by the admin/help URL for this module (/admin/help or ?q=admin/help).

Telling Drupal who can use your module

The next function to write is the permissions function. The permissions function doesn't grant permission, it just specifies what permissions are available for this module. Access based on these permissions is defined later in the {module}_access function below. At this point, we'll give permission to anyone who can access site content or administrate the module:

```
<?php
/**
* Valid permissions for this module
* @return array An array of valid permissions for the onthisdate module
*/
function onthisdate_perm() {
  return array('access content');
} // function onthisdate_perm()
?>
```

Conversely, if you are going to write a module that needs to have finer control over the permissions, and you're going to do permission control, you should expand this permission set. You can do this by adding strings to the array that is returned. For example:

```
<?php
function onthisdate_perm() {
return array('access content', 'access onthisdate', 'administer
onthisdate');
} // function onthisdate_perm
?>
```

For this tutorial, start with the first one. We'll later move to the second version.

You'll need to adjust who has permission to view your module on the administer Â» accounts Â» permissions page. We'll use the user_access function to check access permissions later (whoa, so many "laters!").

Your permission strings must be unique within your module. If they are not, the permissions page will list the same permission multiple times. They should also contain your module name, to avoid name space conflicts with other modules.

Announce we have block content

There are several types of modules: block modules and node modules are two. Block modules create abbreviated content that is typically (but not always, and not required to be) displayed along the left or right side of a page. Node modules generate full page content (such as blog, forum, or book pages).

We'll create a block content to start, and later discuss node content. A module can generate content for blocks and also for a full page (the blogs module is a good example of this). The hook for a block module is appropriately called "block", so let's start our next function:

```
<?php
/**
* Generate HTML for the onthisdate block
* @param op the operation from the URL
* @param delta offset
* @returns block HTML
*/
function onthisdate_block($op='list', $delta=0) {
} // end function onthisdate_block
?>
```

The block function takes two parameters: the operation and the offset, or delta. We'll just worry about the operation at this point. In particular, we care about the specific case where the block is being listed in the blocks page. In all other situations, we'll display the block content.

When the module will be listed on the blocks page, the $op parameter's value will be 'list':

```
<?php
/**
* Generate HTML for the onthisdate block
* @param op the operation from the URL
* @param delta offset
* @returns block HTML
*/
function onthisdate_block($op='list', $delta=0) {
  // listing of blocks, such as on the admin/block page
  if ($op == "list") {
     $block[0]["info"] = t('On This Date');
```

```
    return $block;
  } else {
    // our block content
  }
} // end onthisdate_block
?>
```

Generate content for a block

Now, we need to generate the 'onthisdate' content for the block. Here we'll demonstrate a basic way to access the database.

Our goal is to get a list of content (stored as "nodes" in the database) created a week ago. Specifically, we want the content created between midnight and 11:59pm on the day one week ago. When a node is first created, the time of creation is stored in the database. We'll use this database field to find our data.

First, we need to calculate the time (in seconds since epoch start, see http://www.php.net/manual/en/function.time.php for more information on time format) for midnight a week ago, and 11:59pm a week ago. This part of the code is Drupal independent, see the PHP website (http://php.net/) for more details.

```
<?php
/**
* Generate HTML for the onthisdate block
* @param op the operation from the URL
* @param delta offset
* @returns block HTML
*/
function onthisdate_block($op='list', $delta=0) {
  // listing of blocks, such as on the admin/block page
  if ($op == "list") {
    $block[0]["info"] = t('On This Date');
    return $block;
  } else {
    // our block content
    // Get today's date
    $today = getdate();
    // calculate midnight one week ago
    $start_time = mktime(0, 0, 0,
                         $today['mon'], ($today['mday'] - 7),
$today['year']);
    // we want items that occur only on the day in question, so
    // calculate 1 day
    $end_time = $start_time + 86400;
    // 60 * 60 * 24 = 86400 seconds in a day
    ...
```

```
  }
}
?>
```

The next step is the SQL statement that will retrieve the content we'd like to display from the database. We're selecting content from the node table, which is the central table for Drupal content. We'll get all sorts of content type with this query: blog entries, forum posts, etc. For this tutorial, this is okay. For a real module, you would adjust the SQL statement to select specific types of content (by adding the 'type' column and a WHERE clause checking the 'type' column).

Note: the table name is enclosed in curly braces: `{node}`. This is necessary so that your module will support database table name prefixes. You can find more information on the Drupal website by reading the Table Prefix (and sharing tables across instances) page in the Drupal handbook.

```
<?php
$query = "SELECT nid, title, created FROM " .
         "{node} WHERE created >= '" . $start_time .
         "' AND created <= '". $end_time . "'";
?>
```

Drupal uses database helper functions to perform database queries. This means that, for the most part, you can write your database SQL statement and not worry about the backend connections.

We'll use db_query() to get the records (i.e. the database rows) that match our SQL query, and db_fetch_object() to look at the individual records:

```
<?php
  // get the links
  $queryResult =  db_query($query);
  // content variable that will be returned for display
  $block_content = '';
  while ($links = db_fetch_object($queryResult)) {
    $block_content .= '<a href="' . url('node/' . $links->nid) . '">' .
                      $links->title . '</a><br />';
  }
  // check to see if there was any content before setting up
  //  the block
  if ($block_content == '') {
    /* No content from a week ago.  If we return nothing, the block
     * doesn't show, which is what we want. */
    return;
  }
  // set up the block
  $block['subject'] = 'On This Date';
  $block['content'] = $block_content;
  return $block;
}
```

```
?>
```

Notice the actual URL is enclosed in the l() function. `l` generates <a href="link"> links, adjust the URL to the installation's URL configuration of either clean URLS: http://(sitename)/node/2 or http://(sitename)/?q=node/2

Also, we return an array that has 'subject' and 'content' elements. This is what Drupal expects from a block function. If you do not include both of these, the block will not render properly.

You may also notice the bad coding practice of combining content with layout. If you are writing a module for others to use, you will want to provide an easy way for others (in particular, non-programmers) to adjust the content's layout. An easy way to do this is to include a class attribute in your link, or surround the HTML with a tag with a module specific CSS class and not necessarily include the
 at the end of the link. Let's ignore this for now, but be aware of this issue when writing modules that others will use.

Putting it all together, our block function at this point looks like this:

```
<?php
function onthisdate_block($op='list', $delta=0) {
  // listing of blocks, such as on the admin/block page
  if ($op == "list") {
    $block[0]["info"] = t("On This Date");
    return $block;
  } else {
  // our block content
    // content variable that will be returned for display
    $block_content = '';
    // Get today's date
    $today = getdate();
    // calculate midnight one week ago
    $start_time = mktime(0, 0, 0,$today['mon'],
                                ($today['mday'] - 7), $today['year']);
    // we want items that occur only on the day in question, so
    //calculate 1 day
    $end_time = $start_time + 86400;
    // 60 * 60 * 24 = 86400 seconds in a day
    $query = "SELECT nid, title, created FROM " .
             "{node} WHERE created >= '" . $start_time .
             "' AND created <= '". $end_time . "'";
    // get the links
    $queryResult =  db_query($query);
    while ($links = db_fetch_object($queryResult)) {
      $block_content .= '<a href="'.url('node/'.$links->nid).'">'.
                        $links->title . '</a><br />';
    }
      // check to see if there was any content before setting up the
```

```
block
    if ($block_content == '') {
      // no content from a week ago, return nothing.
      return;
    }
    // set up the block
    $block['subject'] = 'On This Date';
    $block['content'] = $block_content;
    return $block;
  }
}
?>
```

Installing, enabling and testing the module

At this point, you can install your module and it'll work. Let's do that, and see where we need to improve the module.

To install the module, you'll need to copy your onthisdate.module file to the modules directory of your Drupal installation. The file must be installed in this directory or a subdirectory of the modules directory, and must have the .module name extension.

Log in as your site administrator, and navigate to the modules administration page to get an alphabetical list of modules. In the menus: administer Â» modules, or via URL:

`http://.../admin/modules` **or**
`http://.../?q=admin/modules`

When you scroll down, you'll see the onthisdate module listed with the description next to it.

Enable the module by selecting the checkbox and save your configuration.

Because the module is a blocks module, we'll need to also enable it in the blocks administration menu and specify a location for it to display. Node modules may or may not need further configuration depending on the module. Any module can have settings, which affect the functionality/display of a module. We'll discuss settings later. For now, navigate to the blocks administration page: `admin/block` or administer Â» blocks in the menus.

Enable the module by selecting the enabled checkbox for the 'On This Date' block and save your blocks. Be sure to adjust the location (left/right) if you are using a theme that limits where blocks are displayed.

Now, head to another page, say, select the modules menu. In some themes, the blocks are displayed after the page has rendered the content, and you won't see the change until you go to new page.

If you have content that was created a week ago, the block will display with links to the content. If you don't have content, you'll need to fake some data. You can do this by creating a blog, forum topic or book page, and adjust the "Authored on:" date to be a week ago.

Alternately, if your site has been around for a while, you may have a lot of content created on the day one week ago, and you'll see a large number of links in the block.

Create a module configuration (settings) page

Now that we have a working module, we'd like to make it better. If we have a site that has been around for a while, content from a week ago might not be as interesting as content from a year ago. Similarly, if we have a busy site, we might not want to display all the links to content created last week. So, let's create a configuration page for the administrator to adjust this information.

A module's configuration is set up with the 'settings' hook. We would like only administrators to be able to access this page, so we'll do our first permissions check of the module here:

```
<?php
/**
* Module configuration settings
* @return settings HTML or deny access
*/
function onthisdate_settings() {
  // only administrators can access this module
  if (!user_access("admin onthisdate")) {
    return message_access();
  }
}
?>
```

If you want to tie your modules permissions to the permissions of another module, you can use that module's permission string. The "access content" permission is a good one to check if the user can view the content on your site:

```
<?php
  ...
  // check the user has content access
  if (!user_access("access content")) {
    return message_access();
  }
  ...
?>
```

We'd like to configure how many links display in the block, so we'll create a form for the administrator to set the number of links:

```
<?php
function onthisdate_settings() {
  // only administrators can access this module
  if (!user_access("admin onthisdate")) {
    return message_access();
  }
  $output .= form_textfield(t("Maximum number of links"),
"onthisdate_maxdisp",
            variable_get("onthisdate_maxdisp", "3"), 2, 2,
                                t("The  maximum  number  of  links  to  display  i
block."));  return $output;}
?>
```

This function uses several powerful Drupal form handling features. We don't need to worry about creating an HTML text field or the form, as Drupal will do so for us. We use `variable_get` to retrieve the value of the system configuration variable "onthisdate_maxdisp", which has a default value of 3. We use the form_textfield function to create the form and a text box of size 2, accepting a maximum length of 2 characters. We also use the translate function of t(). There are other form functions that will automatically create the HTML form elements for use. For now, we'll just use the form_textfield function.

Of course, we'll need to use the configuration value in our SQL SELECT, so we'll need to adjust our query statement in the onthisdate_block function:

```
<?php
  $limitnum = variable_get("onthisdate_maxdisp", 3);
  $query = "SELECT nid, title, created FROM " .
           "{node} WHERE created >= '" . $start_time .
           "' AND created <= '". $end_time . "' LIMIT " . $limitnum;
?>
```

You can test the settings page by editing the number of links displayed and noticing the block content adjusts accordingly.

Navigate to the settings page: admin/modules/onthisdate or
administer Â» configuration Â» modules Â» onthisdate. Adjust the number of links and save the configuration. Notice the number of links in the block adjusts accordingly.

Note: We don't have any validation with this input. If you enter "c" in the maximum number of links, you'll break the block.

Adding menu links and creating page content

So far we have our working block and a settings page. The block displays a maximum number of links. However, there may be more links than the maximum we show. So, let's create a page that lists all the content that was created a week ago.

```
<?php
function onthisdate_all() {}
?>
```

We're going to use much of the code from the block function. We'll write this ExtremeProgramming style, and duplicate the code. If we need to use it in a third place, we'll refactor it into a separate function. For now, copy the code to the new function _onthisdate_all(). Contrary to all our other functions, 'all', in this case, is not a Drupal hook. In our code, we can prefix this function with an underscore to help us remember this isn't a hook call. We'll discuss below.

```
<?php
function _onthisdate_all() {
  // content variable that will be returned for display
  $page_content = '';
  // Get today's date
  $today = getdate();
  // calculate midnight one week ago
  $start_time = mktime(0, 0, 0,
                       $today['mon'], ($today['mday'] - 7),
$today['year']);
  // we want items that occur only on the day in question,
  // so calculate 1 day
  $end_time = $start_time + 86400;
  // 60 * 60 * 24 = 86400 seconds in a day
  // NOTE!  No LIMIT clause here!  We want to show all the code
  $query = "SELECT nid, title, created FROM " .
           "{node} WHERE created >= '" . $start_time .
           "' AND created <= '". $end_time . "'";
  // get the links
  $queryResult =  db_query($query);
  while ($links = db_fetch_object($queryResult)) {
    $page_content .= '<a href="'.url('node/'.$links->nid).'">'.
                      $links->title . '</a><br />';
  }
  ...
}
?>
```

We have the page content at this point, but we want to do a little more with it than just return it. When creating pages, we need to send the page content to the theme for proper rendering. We use this with the theme() function. Themes control the look of a site. As noted above, we're including layout in the code. This is bad, and should be avoided. It is, however, the topic of another tutorial, so for now, we'll include the formatting in our content:

```
<?php
print theme("page", $content_string);
```

```
?>
```

The rest of our function checks to see if there is content and lets the user know. This is preferable to showing an empty or blank page, which may confuse the user.

Note that we are responsible for outputting the page content with the 'print theme()' syntax.

```
<?php
function _onthisdate_all() {
  ...
  // check to see if there was any content before
  // setting up the block
  if ($page_content == '') {
    // no content from a week ago, let the user know
    print theme("page",
                              "No events occurred on this site on this date
history.");
    return;
  }
  print theme("page", $page_content);
}
?>
```

Adding a 'more' link and showing all entries

Because we have our function that creates a page with all the content created a week ago, we can link to it from the block with a "more" link.

Add these lines just before that $block['subject'] line, adding this to the $block_content variable before saving it to the $block['content'] variable:

```
<?php
// add a more link to our page that displays all the links
  $block_content .=
         "<div class=\"more-link\">".
           l(t("more"), "onthisdate", array("title" => t("More events on
this day.")))
         ."</div>";
?>
```

This will add the more link.

Conclusion

We now have a working module. It created a block and a page. You should now have enough to get started writing your own modules. We recommend you start with a block module of your own and move onto a node module. Alternately, you can write a filter or theme.

As is, this tutorial's module isn't very useful. However, with a few enhancements, it can be entertaining. Try modifying the select query statement to select only nodes of type 'blog' and see what you get. Alternately, you could get only a particular user's content for a specific week. Instead of using the block function, consider expanding the menu and page functions, adding menus to specific entries or dates, or using the menu callback arguments to adjust what year you look at the content from.

If you start writing modules for others to use, you'll want to provide more details in your code. Comments in the code are incredibly valuable for other developers and users in understanding what's going on in your module. You'll also want to expand the help function, providing better help for the user. Follow the Drupal [Coding standards], especially if you're going to add your module to the project.

Two topics very important in module development are writing themeable pages and writing translatable content. Please check the Drupal Handbook for more details on these two subjects.

Updating your modules

As Drupal develops with each release it becomes necessary to update modules to take advantage of new features and stay functional with Drupal's API.

Converting 3.0 modules to 4.0

Converting modules from version 3.0 to version 4.0 standards requires rewriting the `form()` function, as follows:

Drupal 3.0:

```
function form($action, $form, $method = "post", $options = 0)

// Example

global $REQUEST_URI;
$form = form_hidden("nid", $nid);
print form($REQUEST_URI, $form);
```

Drupal 4.0:

```
function form($form, $method = "post", $action = 0, $options = 0)

// Example

$form = form_hidden("nid", $nid);
print form($form);
```

Converting 4.0 modules to 4.1

Drupal 4.1 changed the block hook function and taxonomy API. To convert a version 4.0 module to 4.1, the following changes must be made. First, the `*_block()` function must be re-written. Next, calls to `taxonomy_get_tree()` must be re-written to supply the parameters required by the new function. Finally, you may wish to take advantage of new functions added to the taxonomy API.

Required changes

Modified block hook:

Drupal 4.0:

```
function *_block() {
  $blocks[0]["info"] = "First block info";
  $blocks[0]["subject"] = "First block subject";
  $blocks[0]["content"] = "First block content";

  $blocks[1]["info"] = "Second block info";
  $blocks[1]["subject"] = "Second block subject";
  $blocks[1]["content"] = "Second block content";

  // return array of blocks
  return $blocks;
  }
}
```

Drupal 4.1:

```
function *_block($op = "list", $delta = 0) {
  if ($op == "list") {
    $blocks[0]["info"] = "First block info";
    $blocks[1]["info"] = "Second block info";
    return $blocks; // return array of block infos
  }
  else {
    switch($delta) {
      case 0:
        $block["subject"] = "First block subject";
        $block["content"] = "First block content";
        return $block;
      case 1:
        $block["subject"] = "Second block subject";
        $block["content"] = "Second block content";
        return $block;
    }
  }
}
```

Modified taxonomy API:

Changes: in function taxonomy_get_tree()

- there is no longer a "parent" property; rather "parents" is an array
- the result tree is now returned instead of being passed by reference

Drupal 4.0:

```
function taxonomy_get_tree($vocabulary_id, &$tree, $parent = 0, $depth = -1, $key = "tid")
```

Drupal 4.1:

```
$tree = taxonomy_get_tree($vocabulary_id, $parents = 0, $depth = -1, $key = "tid")
```

Optional changes

- Take advantage of new taxonomy functions `taxonomy_get_vocabulary_by_name($name)` and `taxonomy_get_term_by_name($name)`
- Take advantage of pager functions
- Move hardcoded markup from modules to themes, using theme_invoke

Converting 4.1 modules to 4.2

Some points posted by Axel on drupal-devel on migrating 4.1.0 modules to CVS [updated and added to by ax]:

- the big "clean URL" patch: Over the weekend, [dries] bit the bullet and converted every single URL in Drupal's code. meaning we'll [can] have clean URLs like http://foo.com/archive/2003/01/06, http://foo.com/user/42, http://foo.com/blog, and so on.. meaning, for the code:
  - `drupal_url(array("mod" => "search", "op" => "bla"), "module"[, $anchor = ""])` became `url("search/bla"),` with the first url part being the module, the second (typically) being the operation ($op); more arguments are handled differently per module convention.
  - `l("view node", array("op" => "view", "id" => $nid), "node"[, $anchor = "", $attributes = array()])` became `l("view node", "node/view/$nid"[,$attributes = array(), $query = NULL])`
  - similar, `lm()`, which meant "module link" and used to be `module.php?mod=bla&op=blub...`, is now `l("title", "bla/blub/...")`; and `la()`, which meant "admin link" and used to be `admin.php?mod=bla&op=blub...`, is now `l("title", "admin/bla/blub/..."`
  - After fixing those functions, you'll need to edit your _page() function and possibly others so that they get their arguments using the arg() function (see includes/common.inc. These arguments used to be globals called "mod", "op", "id" etc. now these same arguments must be accessed as arg(1), arg(3), for example.
- `$theme->function()` became `theme("function")`. see [drupal-devel] renaming 2 functions, [drupal-devel] theme("function") vs $theme->function() and [drupal-devel] [CVS]

theme()

- `<module>_conf_options()` became `<module>_settings()` - see [drupal-devel] renaming 2 functions. note that doesn't get an extra menu entry, but is accessed via "site configuration > modules > modules settings"

- the administration pages got changed quite a lot to use a "database driven link system" and become more logical/intuitive - see [drupal-devel] X-mas commit: administration pages. this first try resulted in poor performance and a not-so-good api, so it got refactored - see [PATCH] menus. this, as of time ax is writing this, isn't really satisfying, neither (you cannot build arbitrary menu-trees, some forms don't work (taxonomy > add term), ...), so it probably will change again. and i won't write more about this here.

  well, this: you use `menu()` to add entries to the admin menu. `menu("admin/node/nodes/0", "new or updated posts", "node_admin", "help", 0);` adds a menu entry "new or updated posts" 1 level below "post overview" (admin/node/nodes) and 2 level below "node management" (admin/node) (ie. at the 3. level), with a weight of 0 in the 3. level, with a line "help" below the main heading. for the callback ("node_admin") ... ask dries or zbynek

  one more note, though: you do **not** add `<module>_settings()` to the menu (they automatically go to "site configuration > modules > module settings" - you only add `<module>_admin...()` ... things.

- [from comment_is_new function lost]

```
-  comment_is_new($comment)
+  node_is_new($comment->nid, $comment->timestamp)
```

please add / update / correct!

Converting 4.2 modules to 4.3

Database table prefix

On 2003 Jul 10, Dries committed Slavica's table prefix patch which allows for a configurable "“prefix to each drupal mysql table to easily share one database for multiply applications on server with only one database allowed.”" This patch requires all table names in SQL-queries to be enclosed in {curly brackets}, eg.

```
-  db_query("DELETE FROM book WHERE nid = %d", $node->nid);
+  db_query("DELETE FROM {book} WHERE nid = %d", $node->nid);
```

so that the table prefix can be dynamically prepended to the table name. See the original feature request and the corresponding discussion at the mailing list for details.

New help system

From Michael Frankowski message:

There is a block of text placed at the top of each admin page by
the admin_page function. After 4.3.0 is out the door the function
`menu_get_active_help()` should probably be renamed/moved into the help module and be attached -- somehow -- to every `_page` hook (probably in the node module) so that we can use this system through out Drupal but for now, there is a block of text displayed at the top of every admin page. This is the active help block. (context sensitive help?)

If the URL of the admin page matches a URL in a `_help` hook then the text from that `_help` hook is displayed on the top of the admin page. If there is no match, the block it not displayed. Because Drupal matches URLs in order to stick "other" stuff in the `_help` hook we have taken to sticking descriptors after a "#" sign. So far, the following descriptors are recognised:

Descriptor	Function
admin/system/modules#name	The name of a module (unused, but there)
admin/system/modules#description	The description found on the admin/system/modules page.
admin/help#*modulename*	The module's help text, displayed on the admin/help page and through the module's individual help link.
user/help#*modulename*	The help for a distrbuted authorization module

In the future we will probably recognise #block for the text needed
in a block displayed by the help system.

Creating modules for version 4.3.1

This tutorial describes how to create a module for Drupal-CVS
(i.e. Drupal version > 4.3.1). A module is a collection of functions that link into Drupal, providing
additional functionality to your Drupal installation. After reading this tutorial, you will be able to create a basic block module and use it as a template for
more advanced modules and node modules.

This tutorial will not necessarily prepare you to write modules for
release into the wild. It does not cover caching, nor does it elaborate
on permissions or security issues. Use this tutorial as a starting
point, and review other modules and the [Drupal handbook] and [Coding standards] for more information.

This tutorial assumes the following about you:

- Basic PHP knowledge, including syntax and the concept of PHP objects
- Basic understanding of database tables, fields, records and SQL statements

- A working Drupal installation
- Drupal administration access
- Webserver access

This tutorial does not assume you have any knowledge about the inner workings of a Drupal module. This tutorial will not help you write modules for Drupal 4.3.1 or before.

Getting Started

To focus this tutorial, we'll start by creating a block module that lists links to content such as blog entries or forum discussions that were created one week ago. The full tutorial will teach us how to create block content, write links, and retrieve information from Drupal nodes.

Start your module by creating a PHP file and save it as 'onthisdate.module'.

```
<?php
?>
```

As per the [Coding standards], use the longhand <?php tag, and not <? to enclose your PHP code.

All functions in your module are named {modulename}_{hook}, where "hook" is a well defined function name. Drupal will call these functions to get specific data, so having these well defined names means Drupal knows where to look.

Telling Drupal about your module

The first function we'll write will tell Drupal information about your module: its name and description. The hook name for this function is 'help', so start with the onthisdate_help function:

```
<?php
function onthisdate_help($section) {
}
?>
```

The $section variable provides context for the help: where in Drupal or the module are we looking for help. The recommended way to process this variable is with a switch statement. You'll see this code pattern in other modules.

```
<?php
/* Commented out until bug fixed */
/*
```

```
function onthisdate_help($section) {
  switch($section) {
    case "admin/system/modules#name":
      $output = "onthisdate";
      break;
    case "admin/system/modules#description":
        $output = "Display a list of nodes that were created a week
ago.";
      break;
    default:
      $output = "onthisdate";
      break;
  }
  return $output;
}
*/
?>
```

You will eventually want to add other cases to this switch statement to provide real help messages to the user. In particular, output for "admin/help#onthisdate" will display on the main help page accessed by the admin/help URL for this module (/admin/help or ?q=admin/help).

Note: This function is commented out in the above code. This is on purpose, as the current version of Drupal CVS won't display the module name, and won't enable it properly when installed. Until this bug is fixed, comment out your help function, or your module may not work.

Telling Drupal who can use your module

The next function to write is the permissions function. Here, you can tell Drupal who can access your module. At this point, give permission to anyone who can access site content or administrate the module.

```
<?php
function onthisdate_perm() {
  return array("administer onthisdate");
}
?>
```

If you are going to write a module that needs to have finer control over the permissions, and you're going to do permission control, you may want to define a new permission set. You can do this by adding strings to the array that is returned:

```
<?php
function onthisdate_perm() {
  return array("access onthisdate", "administer onthisdate");
}
?>
```

You'll need to adjust who has permission to view your module on the administer Â» accounts Â» permissions page. We'll use the user_access function to check access permissions later.

Be sure your permission strings must be unique to your module. If they are not, the permissions page will list the same permission multiple times.

Announce we have block content

There are several types of modules: block modules and node modules are two. Block modules create abbreviated content that is typically (but not always, and not required to be) displayed along the left or right side of a page. Node modules generate full page content (such as blog, forum, or book pages).

We'll create a block content to start, and later discuss node content. A module can generate content for blocks and also for a full page (the blogs module is a good example of this). The hook for a block module is appropriately called "block", so let's start our next function:

```
<?php
function onthisdate_block($op='list', $delta=0) {
}
?>
```

The block function takes two parameters: the operation and the offset, or delta. We'll just worry about the operation at this point. In particular, we care about the specific case where the block is being listed in the blocks page. In all other situations, we'll display the block content.

```
<?php
function onthisdate_block($op='list', $delta=0) {
  // listing of blocks, such as on the admin/system/block page
  if ($op == "list") {
    $block[0]["info"] = t("On This Date");
    return $block;
  } else {
  // our block content
  }
}
```

```
?>
```

Generate content for a block

Now, we need to generate the 'onthisdate' content for the block. In here, we'll demonstrate a basic way to access the database.

Our goal is to get a list of content (stored as "nodes" in the database) created a week ago. Specifically, we want the content created between midnight and 11:59pm on the day one week ago. When a node is first created, the time of creation is stored in the database. We'll use this database field to find our data.

First, we need to calculate the time (in seconds since epoch start, see http://www.php.net/manual/en/function.time.php for more information on time format) for midnight a week ago, and 11:59pm a week ago. This part of the code is Drupal independent, see the PHP website (http://php.net/) for more details.

```
<?php
function onthisdate_block($op='list', $delta=0) {
  // listing of blocks, such as on the admin/system/block page
  if ($op == "list") {
    $block[0]["info"] = t("On This Date");
    return $block;
  } else {
  // our block content
    // Get today's date
    $today = getdate();
    // calculate midnight one week ago
    $start_time = mktime(0, 0, 0,
                         $today['mon'], ($today['mday'] - 7),
$today['year']);
     // we want items that occur only on the day in question, so
calculate 1 day
      $end_time = $start_time + 86400;   // 60 * 60 * 24 = 86400
seconds in a day
    ...
  }
}
?>
```

The next step is the SQL statement that will retrieve the content we'd like to display from the database. We're selecting content from the node table, which is the central table for Drupal content. We'll get all sorts of content type with this query: blog entries, forum posts, etc. For this tutorial, this is okay. For a real module, you would

adjust the SQL statement to select specific types of content (by adding the 'type' column and a WHERE clause checking the 'type' column).

Note: the table name is enclosed in curly braces: `{node}`.
This is necessary so that your module will support database table name prefixes. You can find more information on the Drupal website by reading the [Table Prefix (and sharing tables across instances)] page in the Drupal handbook.

```
<?php
  $query = "SELECT nid, title, created FROM " .
           "{node} WHERE created >= '" . $start_time .
           "' AND created <= '". $end_time . "'";
?>
```

Drupal uses database helper functions to perform database queries. This means that, for the most part, you can write your database SQL statement and not worry about the backend connections.

We'll use db_query() to get the records (i.e. the database rows) that match our SQL query, and db_fetch_object() to look at the individual records:

```
<?php
  // get the links
  $queryResult =  db_query($query);
  // content variable that will be returned for display
  $block_content = '';
  while ($links = db_fetch_object($queryResult)) {
    $block_content .= '<a href="' . url('node/view/' . $links->nid )
. '">' .
                      $links->title . '</a><br />';
  }
   // check to see if there was any content before setting up the
block
  if ($block_content == '') {
    /* No content from a week ago.  If we return nothing, the block
     * doesn't show, which is what we want. */
    return;
  }
  // set up the block
  $block['subject'] = 'On This Date';
  $block['content'] = $block_content;
  return $block;
}
?>
```

Notice the actual URL is enclosed in the url() function. This adjusts the URL to the installations URL configuration of either clean URLS: http://sitename/node/view/2 or http://sitename/?q=node/view/2

Also, we return an array that has 'subject' and 'content' elements. This is what Drupal expects from a block function. If you do not include both of these, the block will not render properly.

You may also notice the bad coding practice of combining content with layout. If you are writing a module for others to use, you will want to provide an easy way for others (in particular, non-programmers) to adjust the content's layout. An easy way to do this is to include a class attribute in your link, and not necessarily include the
 at the end of the link. Let's ignore this for now, but be aware of this issue when writing modules that others will use.

Putting it all together, our block function looks like this:

```
<?php
function onthisdate_block($op='list', $delta=0) {
  // listing of blocks, such as on the admin/system/block page
  if ($op == "list") {
    $block[0]["info"] = t("On This Date");
    return $block;
  } else {
  // our block content
    // content variable that will be returned for display
    $block_content = '';
    // Get today's date
    $today = getdate();
    // calculate midnight one week ago
    $start_time = mktime(0, 0, 0,
                         $today['mon'], ($today['mday'] - 7),
$today['year']);
     // we want items that occur only on the day in question, so
calculate 1 day
      $end_time = $start_time + 86400;   // 60 * 60 * 24 = 86400
seconds in a day
    $query = "SELECT nid, title, created FROM " .
             "{node} WHERE created >= '" . $start_time .
             "' AND created <= '". $end_time . "'";
    // get the links
    $queryResult =  db_query($query);
    while ($links = db_fetch_object($queryResult)) {
      $block_content .= '<a
href="'.url('node/view/'.$links->nid).'">'.
                        $links->title . '</a><br />';
```

```
    }
     // check to see if there was any content before setting up the
block
    if ($block_content == '') {
      // no content from a week ago, return nothing.
      return;
    }
    // set up the block
    $block['subject'] = 'On This Date';
    $block['content'] = $block_content;
    return $block;
  }
}
?>
```

Installing, enabling and testing the module

At this point, you can install your module and it'll work. Let's do that, and see where we need to improve the module.

To install the module, you'll need to copy your onthisdate.module file to the modules directory of your Drupal installation. The file must be installed in this directory or a subdirectory of the modules directory, and must have the .module name extension.

Log in as your site administrator, and navigate to the modules administration page to get an alphabetical list of modules. In the menus: administer Â» configuration Â» modules, or via URL:

`http://.../admin/system/modules` **or**
`http://.../?q=admin/system/modules`

Note: You'll see one of three things for the 'onthisdate' module at this point:

- You'll see the 'onthisdate' module name and no description
- You'll see no module name, but the 'onthisdate' description
- You'll see both the module name and the description

Which of these three choices you see is dependent on the state of the CVS tree, your installation and the help function in your module. If you have a description and no module name, and this bothers you, comment out the help function for the moment. You'll then have the module name, but no description. For this tutorial, either is okay, as you will just enable the module, and won't use the help system.

Enable the module by selecting the checkbox and save your configuration.

Because the module is a blocks module, we'll need to also enable it in the blocks administration menu and specify a location for it to display. Navigate to the blocks administration page: admin/system/block or administer Â» configuration Â» blocks in the menus.

Enable the module by selecting the enabled checkbox for the 'On This Date' block and save your blocks. Be sure to adjust the location (left/right) if you are using a theme that limits where blocks are displayed.

Now, head to another page, say select the module. In some themes, the blocks are displayed after the page has rendered the content, and you won't see the change until you go to new page.

If you have content that was created a week ago, the block will display with links to the content. If you don't have content, you'll need to fake some data. You can do this by creating a blog, forum topic or book page, and adjust the "Authored on:" date to be a week ago.

Alternately, if your site has been around for a while, you may have a lot of content created on the day one week ago, and you'll see a large number of links in the block.

Create a module configuration (settings) page

Now that we have a working module, we'd like to make it better. If we have a site that has been around for a while, content from a week ago might not be as interesting as content from a year ago. Similarly, if we have a busy site, we might not want to display all the links to content created last week. So, let's create a configuration page for the administrator to adjust this information.

The configuration page uses the 'settings' hook. We would like only administrators to be able to access this page, so we'll do our first permissions check of the module here:

```
<?php
function onthisdate_settings() {
  // only administrators can access this module
  if (!user_access("admin onthisdate")) {
    return message_access();
  }
}
?>
```

If you want to tie your modules permissions to the permissions of another module, you can use that module's permission string. The "access content" permission is a good one to check if the user can view the content on your site:

```
<?php
  ...
  // check the user has content access
  if (!user_access("access content")) {
    return message_access();
  }
  ...
?>
```

We'd like to configure how many links display in the block, so we'll create a form for the administrator to set the number of links:

```
<?php
function onthisdate_settings() {
  // only administrators can access this module
  if (!user_access("admin onthisdate")) {
    return message_access();
  }
      $output   .=   form_textfield(t("Maximum   number   of   links"),
"onthisdate_maxdisp",
            variable_get("onthisdate_maxdisp", "3"), 2, 2,
                        t("The  maximum  number  of  links  to  display  in  th
block."));
  return $output;
}
?>
```

This function uses several powerful Drupal form handling features. We don't need to worry about creating an HTML text field or the form, as Drupal will do so for us. We use `variable_get` to retrieve the value of the system configuration variable "onthisdate_maxdisp", which has a default value of 3. We use the form_textfield function to create the form and a text box of size 2, accepting a maximum length of 2 characters. We also use the translate function of t(). There are other form functions that will automatically create the HTML form elements for use. For now, we'll just use the form_textfield function.

Of course, we'll need to use the configuration value in our SQL SELECT, so we'll need to adjust our query statement in the onthisdate_block function:

```
<?php
  $limitnum = variable_get("onthisdate_maxdisp", 3);
  $query = "SELECT nid, title, created FROM " .
           "{node} WHERE created >= '" . $start_time .
           "' AND created <= '". $end_time . "' LIMIT " . $limitnum;
?>
```

You can test the settings page by editing the number of links displayed and noticing the block content adjusts accordingly.

Navigate to the settings page: admin/system/modules/onthisdate or administer Â» configuration Â» modules Â» onthisdate. Adjust the number of links and save the configuration. Notice the number of links in the block adjusts accordingly.

Note: We don't have any validation with this input. If you enter "c" in the maximum number of links, you'll break the block.

Adding menu links and creating page content

So far we have our working block and a settings page. The block displays a maximum number of links. However, there may be more links than the maximum we show. So, let's create a page that lists all the content that was created a week ago.

```
<?php
function onthisdate_all() {
}
?>
```

We're going to use much of the code from the block function. We'll write this ExtremeProgramming style, and duplicate the code. If we need to use it in a third place, we'll refactor it into a separate function. For now, copy the code to the new function onthisdate_all(). Contrary to all our other functions, 'all', in this case, is not a Drupal hook. We'll discuss below.

```
<?php
function onthisdate_all() {
  // content variable that will be returned for display
  $page_content = '';
  // Get today's date
  $today = getdate();
  // calculate midnight one week ago
  $start_time = mktime(0, 0, 0,
                       $today['mon'], ($today['mday'] - 7),
$today['year']);
   // we want items that occur only on the day in question, so
```

```
calculate 1 day
  $end_time = $start_time + 86400;  // 60 * 60 * 24 = 86400 seconds
in a day
  // NOTE!  No LIMIT clause here!  We want to show all the code
  $query = "SELECT nid, title, created FROM " .
           "{node} WHERE created >= '" . $start_time .
           "' AND created <= '". $end_time . "'";
  // get the links
  $queryResult =  db_query($query);
  while ($links = db_fetch_object($queryResult)) {
    $page_content .= '<a href="'.url('node/view/'.$links->nid).'">'.
                      $links->title . '</a><br />';
  }
  ...
}
?>
```

We have the page content at this point, but we want to do a little more with it than just return it. When creating pages, we need to send the page content to the theme for proper rendering. We use this with the theme() function. Themes control the look of a site. As noted above, we're including layout in the code. This is bad, and should be avoided. It is, however, the topic of another tutorial, so for now, we'll include the formatting in our content:

```
<?php
    print theme("page", $content_string);
?>
```

The rest of our function checks to see if there is content and lets the user know. This is preferable to showing an empty or blank page, which may confuse the user.

Note that we are responsible for outputting the page content with the 'print theme()' syntax. This is a change from previous 4.3.x themes.

```
<?php
function onthisdate_all() {
  ...
   // check to see if there was any content before setting up the
block
  if ($page_content == '') {
    // no content from a week ago, let the user know
    print theme("page",
                      "No events occurred on this site on this date in
history.");
    return;
```

```
  }
  print theme("page", $page_content);
}
?>
```

Letting Drupal know about the new function

As mentioned above, the function we just wrote isn't a 'hook': it's not a Drupal recognized name. We need to tell Drupal how to access the function when displaying a page. We do this with the _link hook and the menu() function:

```
<?php
function onthisdate_link($type, $node=0) {
}
?>
```

There are many different types, but we're going to use only 'system' in this tutorial.

```
<?php
function onthisdate_link($type, $node=0) {
  if (($type == "system")) {
    // URL, page title, func called for page content, arg, 1 = don't
disp menu
    menu("onthisdate", t("On This Date"), "onthisdate_all", 1, 1);
  }
}
?>
```

Basically, we're saying if the user goes to "onthisdate" (either via ?q=onthisdate or http://.../onthisdate), the content generated by onthisdate_all will be displayed. The title of the page will be "On This Date". The final "1" in the arguments tells Drupal to not display the link in the user's menu. Make this "0" if you want the user to see the link in the side navigation block.

Navigate to /onthisdate (or ?q=onthisdate) and see what you get.

Adding a more link and showing all entries

Because we have our function that creates a page with all the content created a week ago, we can link to it from the block with a "more" link.

Add these lines just before that $block['subject'] line, adding this to the $block_content variable before saving it to the $block['content'] variable:

```
<?php
  // add a more link to our page that displays all the links
        $block_content  .=  "<div  class=\"more-link\">".  l(t("more"),
"onthisdate",  array("title"  =>  t("More  events  on  this  day.")))
."</div>";
?>
```

This will add the more link.

Conclusion

We now have a working module. It created a block and a page. You should now have enough to get started writing your own modules. We recommend you start with a block module of your own and move onto a node module. Alternately, you can write a filter or theme.

As is, this tutorial's module isn't very useful. However, with a few enhancements, it can be entertaining. Try modifying the select query statement to select only nodes of type 'blog' and see what you get. Alternately, you could get only a particular user's content for a specific week. Instead of using the block function, consider expanding the menu and page functions, adding menus to specific entries or dates, or using the menu callback arguments to adjust what year you look at the content from.

If you start writing modules for others to use, you'll want to provide more details in your code. Comments in the code are incredibly valuable for other developers and users in understanding what's going on in your module. You'll also want to expand the help function, providing better help for the user. Follow the Drupal [Coding standards], especially if you're going to add your module to the project.

Two topics very important in module development are writing themeable pages and writing translatable content. Please check the [Drupal Handbook] for more details on these two subject.

How to build up a _help hook

The following template can be used to build a `_help` hook.

```
<?php
function <modulename>_help($section){
  $output = "";
  switch ($section) {
  }
  return $output;
}
?>
```

In the template replace *modulename* with the name of your module.

If you want to add help text to the overall administrative section. (admin/help) stick this inside the switch:

```
<?php
    case 'admin/help#<modulename>':
      $output = t('The text you want displayed');
      break;
?>
```

If you also want this same text displayed for an individual help link in your menu area. You have this kind of tree:

```
+ Administration
|
-> Your area
|  |
|  -> Your configuration
|  -> help
|
-> Overall admin help.
```

Change the function line to this:

```
<?php
function <modulename>_help($section = 'admin/help#<modlename>') {
?>
```

Now that you have the template started place a case statement in for any URL you want a "context sesitive" help message in the admin section. An example, you have a page that individually configures your module, it is at admin/system/modules/, you want to add some text to the top help area.

```
<?php
case 'admin/system/modules/<modulename>':
$output = t('Your new help text');
break;
?>
```

How to convert a _system hook

There are three things that can appear in a _system hook:

Field	Function
$field == "name"	The module name
$field == "description"	The description placed in the module list
$field == "admin-help"	The help text placed at the TOP of this module's individual configuration area.

Take the text for each one and move it into the _help hook. Replace the `$system[<name>]` that is normally at the front of each one with $output, now place a "break;" after the line and a `case '<name>':` before it where *name* is one of the following:

- If `$system` is `$system["name"]` then the case is `case 'admin/system/modules#name'`
- If $system is $system["description"] then case is `case 'admin/system/modules#description'`
- If `$system` is `$system["admin-help"]` then the case is `case 'admin/system/modules/<modulename>'`

Now remove the `_system` function and you are done.

An example:

```
<?php
function example_system($field){
   $system["description"] = t("This is my example _system hook to
convert for
the help system I have spent a lot of time with.");
   $system["admin-help"] = t("Can you believe that I would actually
write an
indivdual setup page on an EXAMPLE module??");
  return $system[$field];
}
?>
<?php
function example_help($section) {
  $output = "";
  switch ($section) {
    case 'admin/system/modules#example':
        $output = t("This is my example _system hook to convert for
the help
system I have spent a lot of time with.");
      break;
    case 'admin/system/modules/example':
        $output = t("Can you believe that I would actually write an
indivdual
```

```
setup page on an EXAMPLE module??");
      break;
  }
  return $output;
}
?>
```

How to convert an _auth_help hook

Okay, you have written your Distributed Authorization module, and given us a great help text for it and I had to go and ruin it all by changing the help system. What a terrible thing for me to do. How do you convert it?

It is not that hard. There are two places you have to deal with:

1. The text inside the _auth_help hook needs to be moved inside the _help hook under the section `user/help#<modulename>` and
2. You have to change the _page hook, which normally displays that help text, to find your text in a new location by changing the function call `<modulename>_auth_help()` to `<modulename>_help("user/help#<modulename>")`.

See, it is not THAT terrible.

An example:

```
<?php
function exampleda_page() {
  theme("header");
  theme("box", "Example DA", exampleda_auth_help());
  theme("footer");
}
function exampleda_auth_help() {
  $site = variable_get("site_name", "this web site");
  $html_output = "
   <p>This is my example Distributed Auth help. Using this example
you cannot login to <i>%s</i> because it has no _auth hook.&</p>
<p><u>BUT</u> you should still use Drupal since it is a <b>GREAT</b>
CMS and is only getting better.</p>
<p>To learn about about Drupal you can <a
href=\"www.drupal.org\">visit the site</a></p>";
  return sprintf(t($html_output), $site);
}
?>
<?php
function exampleda_page() {
  theme("header");
```

```
  theme("box", "Example DA", exampleda_help('user/help#exampleda'));
  theme("footer");
}
function exampleda_help($section) {
  $output = "";
  switch ($section) {
    case 'user/help#exampleda':
      $site = variable_get("site_name", "this web site");
      $output .= "<p>This is my example Distributed Auth help. Using
this example you cannot login to %site because it has no _auth
hook.</p>";
      $output .= "<p><u>BUT&</u> you should still use Drupal
since it is a <b>GREAT</b> CMS and is only getting better.</p>";
      $output .= "<p>To learn about about Drupal you can
visit %drupal.</p>";
      $output = t($output, array("%site" => "<i>$site</i>",
"%drupal" => "<a href=\"www.drupal.org\">visit the site</a>"));
      break;
  }
  return $output
}
?>
```

Converting 4.3 modules to 4.4

Since Drupal 4.3, major changes have been made to the theme, menu, and node systems. Most themes and modules will require some changes.

Menu system

The Drupal menu system has been extended to drive all pages, not just administrative pages. This is continuing the work done for Drupal 4.3, which integrated the administrative menu with the user menu. We now have consistency between administrative and "normal" pages; when you learn to create one, you know how to create the other.

The flow of page generation now proceeds as follows:

1. The `_link` hook in all modules is called, so that modules can use `menu()` to add items to the menu. For example, a module could define:
```
<?php
function example_link($type) {
  if ($type == "system") {
    menu("example", t("example"), "example_page");
    menu("example/foo", t("foo"), "example_foo");
  }
}
?>
```

2. The menu system examines the current URL, and finds the "best fit" for the URL in the menu. For example, if the current URL is `example/foo/bar/12`, the above `menu()` calls would cause `example_foo("bar", 12)` to get invoked.
3. The callback may set the title or breadcrumb trail if the defaults are not satisfactory (more on this later).
4. The callback is responsible for printing the requested page. This will usually involve preparing the content, and then printing the return value of `theme("page")`. For example:
```
<?php
function example_foo($theString, $theNumber) {
  $output = $theString. " - " .$theNumber;
  print theme("page", $output);
}
?>
```

The following points should be considered when upgrading modules to use the new menu system:

- The `_page` hook is obsolete. Pages will not be shown unless they are declared with a `menu()` call as discussed above. To convert former `_page` hooks to the new system as simply as possible, just declare that function as a "catchall" callback:
```
<?php
  menu("example", t("example"), "example_page", 0, MENU_HIDE);
?>
```
The trailing MENU_HIDE argument in this call makes the menu item hidden, so the callback functions but the module does not clutter the user menu.
- Old administrative callbacks returned their content. In the new system, administrative and normal callbacks alike are responsible for printing the entire page.
- The title of the page is printed by the theme system, so page content does not need to be wrapped in a `theme("box")` to get a title printed. If the default title is not satisfactory, it can be changed by calling `drupal_set_title($title)` before `theme("page")` gets called, or by passing the title to `theme("page")` as a parameter.
- The breadcrumb trail is also printed by the theme. If the default one needs to be overridden (to present things like forum hierarchies), this can be done by calling `drupal_set_breadcrumb($breadcrumb)` before `theme("page")` gets called, or by passing the breadcrumb to `theme("page")` as a parameter. `$breadcrumb` should be a list of links beginning with "Home" and proceeding up to, but not including, the current page.

Theme system

For full information on theme system changes, see converting 4.3 themes to CVS. The following points are directly relevant to module development:

- All theme functions now return their output instead of printing them to the user. Old `theme()` usage:
```
<?php
theme("box", $title, $output);
?>
```

New usage:

```
<?php
print theme("box", $title, $output);
?>
```

Modules that define their own theme functions should also return their output.

- The naming of theme functions defined by modules has been standardized to `theme_<module>_<name>`. When using a theme function there is no need to include the theme_ part, as `theme()` will do this automatically. Example:

```
<?php
function theme_example_list($list) {
  return implode('<br />', $list);
}
print theme('example_list', array(1,2,3));
?>
```

Theme functions must always be called using `theme()` to allow for the active theme to modify the output if necessary.

- The `theme("header")` and `theme("footer")` functions are not available anymore. Module developers should use the `theme("page")` function which wraps the content in the site theme. The full syntax of this function is

```
<?php
theme("page", $output, $title, $breadcrumb);
?>
```

where `$title` and `$breadcrumb` will override any values set before for these properties.

Node system

The node system has been upgraded to allow a single module to define more than one type of node. This will allow some of the more convoluted code in, for example, project.module to be tidied up.

- The `_node()` hook has been deprecated. In its place, modules that define nodes should use `_node_name()` and `_help()`.
- The `_node_name()` function should return a translated string containing the human-readable name of the node type.
- The `_help()` function, when called with parameter `"node/add#modulename"`, should return a translated string containing the description of the node type.
- Modules wishing to use the new ability to define multiple node types should see the Doxygen documentation for `hook_node_name()` and `hook_node_types()`.

Filter system

- The various filter hooks ('filter', 'conf_filters') have been merged into one 'filter' hook. A module that provides filtering functionality should implement:

```
<?php
function example_filter($op, $text = "") {
  switch ($op) {
```

```
    case "name":
      return t("Name of the filter");
    case "prepare":
      // Do preparing on $text
      return $text;
    case "process":
      // Do processing on $text
      return $text;
    case "settings":
      // Generate $output of settings
      return $output;
  }
}
?>
```

- "name" is new, and should return a friendly name for the filter.
- "prepare" is also new. This is an extra step that is performed before the default HTML processing, if HTML tags are allowed. It is meant to give filters the chance to escape HTML-like data before it can get stripped. This means, to convert meaningful HTML characters like < and > into entities such as < and >.

  Common examples include filtering pieces of PHP code, mathematical formulas, etc. It is not allowed to do anything other than escaping in the "prepare" step.

  If your filter currently performs such a step in the main "process" step, it should be moved into "prepare" instead. If you don't need any escaping, your filter should simply return $text without processing in this case.
- "process" is the equivalent of the old "filter" hook. Normal filtering is performed here, and the changed $text is returned.
- "settings" is the equivalent of the old "conf_filters" hook. If your filter provides configurable options, you should return them here (using the standard form_* functions).

- The filter handling code has been moved to a new required `filter.module`, and thus most of the filter function names changed, although none of those should have been called from modules. The `check_output()` function is still available with the same functionality.
- Node filtering is optimized with the `node_prepare()` function now, which only runs the body through the filters if the node view page is displayed. Otherwise, only the teaser is filtered.
- The `_compose_tips` hook (defined by the contrib `compose_tips.module`) is not supported anymore, but more advanced functionality exists in the core. You can emit extensive compose tips related to the filter you define via the `_help` hook with the `'filter#long-tip'` section identifier. The `compose_tips` URL is thus changed to `filter/tips`. The `form_allowed_tags_text()` function is replaced with `filter_tips_short()`, which now supports short tips to be placed under textareas. Any module can inject short tips about the filter defined via the `_help` hook, with the `'filter#short-tip'` section identifier.

Hook changes

Other than those mentioned above, the following hooks have changed:

- The `_view` hook has been changed to return its content rather than printing it. It also has an extra parameter, `$page`, that indicates whether the node is being viewed as a standalone page or as part of a larger context. This is important because nodes may change the breadcrumb trail if they are being viewed as a page. Old usage:

```
<?php
function example_view($node, $main = 0) {
  if ($main) {
    theme("node", $node, $main);
  }
  else {
    $breadcrumb[] = l(t("Home"), "");
    $breadcrumb[] = l(t("foo"), "foo");
          $node->body   =   theme("breadcrumb",   $breadcrumb)   ."<br   />".
$node->body;
    theme("node", $node, $main);
  }
}
?>
```

  New usage:

```
<?php
function example_view($node, $main = 0, $page = 0) {
  if ($main) {
    return theme("node", $node, $main, $page);
  }
  else {
    if ($page) {
      $breadcrumb[] = l(t("Home"), "");
      $breadcrumb[] = l(t("foo"), "foo");
      drupal_set_breadcrumb($breadcrumb);
    }
    return theme("node", $node, $main, $page);
  }
}
?>
```

- The `_form` hook used by *node modules* no longer takes 3 arguments. The second argument `$help`, typically used to print submission guidelines, has been removed. Instead, the help should be emitted using the module's `_help` hook. For examples, check the story, forum or blog module.
- The `_search` hook was changed to not only return the result set array, but a two element array with the result group title and the result set array. This provides more precise control over result group titles.
- The `_head` hook is eliminated and replaced with the `drupal_set_html_head()` and

drupal_get_html_head() functions. You can add JavaScript code or CSS to the HTML head part with the drupal_set_html_head() function instead.
- See also the description of the _compose_tips hook changes below.

Emitting links

- The functions url() and l() take a new $fragment parameter. Calls to url() or l() that have '#' in the $url parameter need to be updated. If you don't update such calls, Drupal's path aliasing won't work for URLs with # in them.
- Drupal now emits relative URLS instead of absolute URLs. Contributed modules must be updated whenere an absolute url is required. For example:
-
  - Any module that outputs an RSS feed without using node_feed() should be updated. Note: this is discouraged. please use node_feed() instead. Also modules using node_feed() should provide an absolute link in the 'link' key, if any.
  - Any module which send email should be updated so that links in the email have absolute urls instead of relative urls. You do this using a parameter in your call to l() or url()

Status and error messages

- Modules that use theme('error', ...) to print error messages should be updated to use drupal_set_message(..., 'error') unless used to print an error message below a form item.
```
<?php
drupal_set_message(t('failed to update X', 'error'));   // set the
second parameter to 'error'
?>
```
- Modules that print status messages directly to the screen using status() should be updated to use drupal_set_message(). The status() function has been removed.
```
<?php
drupal_set_message(t('updated X'));
?>
```

Converting 4.4 modules to 4.5

Menu system

The Drupal menu system got a complete rewrite. The new features include:

- The administrator may now customize the menu to reorder, remove, and add items.
- Menu items may be classified as "local tasks," which will by default be displayed as tabs on the page content.
- The menu API is much more consistent with the rest of Drupal's API.

The menu() function is no more. In its place, we have hook_menu(). The old hook_link() remains, but will no longer be called with the "system" argument. The hook reference in the Doxygen documentation details all the specifics of this new hook. In short, rather than making many calls to menu() in your hook_link() implementation, you will implement hook_menu() to return an array of the menu items you define.

As an example, the old pattern:

```
<?php
function blog_link($type, $node = 0, $main) {
  global $user;
  if ($type == 'system') {
    menu('node/add/blog', t('blog entry'), user_access('maintain
personal blog') ? MENU_FALLTHROUGH : MENU_DENIED, 0);
    menu('blog', t('blogs'), user_access('access content') ?
'blog_page' : MENU_DENIED, 0, MENU_HIDE);
          menu('blog/'.  $user->uid,  t('my  blog'),  MENU_FALLTHROUGH,  1,
MENU_SHOW, MENU_LOCKED);
       menu('blog/feed',  t('RSS  feed'),  user_access('access  content')  ?
'blog_feed' : MENU_DENIED, 0, MENU_HIDE, MENU_LOCKED);
  }
}
?>
```

becomes:

```
<?php
function blog_menu($may_cache) {
  global $user;
  $items = array();
  if ($may_cache) {
       $items[]  =  array('path'  =>  'node/add/blog',  'title'  =>  t('blog
entry'),
      'access' => user_access('maintain personal blog'));
    $items[] = array('path' => 'blog', 'title' => t('blogs'),
      'callback' => 'blog_page',
      'access' => user_access('access content'),
      'type' => MENU_SUGGESTED_ITEM);
     $items[]  =  array('path'  =>  'blog/'.  $user->uid,  'title'  =>  t('my
blog'),
      'access' => user_access('maintain personal blog'),
      'type' => MENU_DYNAMIC_ITEM);
    $items[] = array('path' => 'blog/feed', 'title' => t('RSS feed'),
      'callback' => 'blog_feed',
      'access' => user_access('access content'),
      'type' => MENU_CALLBACK);
  }
  return $items;
}
```

```
?>
```

Drupal now distinguishes between 404 (Not Found) pages and 403 (Forbidden) pages. To accommodate this, modules should abandon the practice of not declaring menu items when access is denied to them. Instead, they should set the "access" attribute of their newly-declared menu item to FALSE. This will have the effect of the menu item being hidden, and also preventing the callback from being invoked by typing in the URL. Modules may also want to take advantage of the drupal_access_denied() function, which prints a 403 page (the analogue of drupal_not_found(), which prints a 404).

Path changes

Some internal URL paths have changed; check the links printed by your code. Most significant is that paths of the form "node/view/52" are now "node/52" instead, while "node/edit/52" becomes "node/52/edit".

Node changes

- The database field `static` has been renamed to `sticky`.
- Error handling of forms (such as node editing forms) is now done using form_set_error(). It simplifies the forms and validation code; however, it does change the node API slightly:
  - The _validate hook and the _nodeapi('validate') hook of the node API no longer take an "error" parameter, and should no longer return an error array. To set an error, call form_set_error().
  - Node modules' hook_form() implementations no longer take an "error" parameter and should not worry about displaying errors. The same applies to hook_nodeapi('form_post') and hook_nodeapi('form_pre').
  - All of the form_ family of functions can take a parameter that marks the field as required in a standard way. Use this instead of adding that information to the field description.
- In order to allow modules such as book.module to inject HTML elements into the view of nodes safely, hook_nodeapi() was extended to respond to the 'view' operation. This operation needs to be invoked after the filtering of the node, so hook_view() was changed slightly to no longer require a return value. Instead of calling theme('node', $node) and returning the result as before, the hook can just modify $node as it sees fit (including running $node->body and $node->teaser through the filters, as before), and the calling code will take care of sending the result to the theme. Most modules will just work under the new semantics, as the return value from the hook is just discarded, but the $node parameter is now required to be passed by reference (this was common but optional before).

- We have node-level access control now! This means that node modules need to make very small changes to their hook_access() implementations. The check for $node->status should be removed; the node module takes care of this check. A value should only be returned from this hook if the node module needs to override whatever access is granted by the node_access table. See the hook API for details.

Node listing queries need to be changed as well, so that they properly check for whether the user has access to the node before listing it. Queries of the form

```
<?php
db_query('SELECT n.nid, n.title FROM {node} n WHERE n.status = 1 AND
foo');
?>
```

become

```
<?php
db_query('SELECT n.nid, n.title FROM {node} n '.
node_access_join_sql() .' WHERE n.status = 1 AND '.
node_access_where_sql() .' AND foo');
?>
```

See node access rights in the Doxygen reference.

Filtering changes

This change affects non-filter modules as well! Please read on even if your module does not filter.

The filter system was changed to support multiple input formats. Each input format houses an entire filter configuration: which filters to use, in what order and with what settings. The filter system now supports multiple filters per module as well.

Check_output() changes

Because of the multiple input formats, a module which implements content has to take care of managing the format with each item. If your module uses the node system and passes content through `check_output()`, then you need to do two things:

- Pass $node->format as the second parameter to `check_output()` whenever you use it.
- Add a filter format selector to `hook_form` using a snippet like:

```
<?php
$output .= filter_form('format', $node->format);
?>
```

The node system will automatically save/load the format value for you.

If your module provides content outside of the node system, you can decide if you want to support multiple input formats or not. If you don't, the default format will always be used. However, if your module accepts input through the browser, it is strongly advised to support input formats!

To do this, you must:

- Provide a selector for input formats on your forms, using filter_form().
- Validate the chosen input format on submission, using filter_access().
- Store the format ID with each content item (the format ID is a number).
- Pass the format ID to check_output().

Check the API documentation for these functions for more information on how to use them.

Filter hook

The _filter hook was changed significantly. It's best to start with the following framework:

```
<?php
function hook_filter($op, $delta = 0, $format = -1, $text = '') {
  switch ($op) {
    case 'list':
      return array(0 => t('Filter name'));
    case 'description':
      return t("Short description of the filter's actions.");
/*
    case 'no cache':
      return true;
*/
    case 'prepare':
      $text = ...
      return $text;
    case 'process':
      $text = ...
      return $text;
    case 'settings':
      $output = ...;
      return $output;
    default:
      return $text;
  }
}
?>
```

When converting a module to 4.5, you can normally ignore the $delta paramter: it is used to have multiple filters inside one module. The 'prepare', 'process' and 'settings' operations still work the same as before, with only small changes.

However, you should now include the $format parameter in the variable names for filter settings. If your filter has a setting "myfilter_something", it should be changed to "myfilter_something_$format". This allows the setting to be set separately for each input format. To check if it works correctly, add your filter to two different input formats and give each instance different settings. Verify that each input format retains its own settings.

Unlike before, the 'settings' operation should only be used to return actually useful settings, because there is now a separate overview of all enabled filters. A filter does not need its own on/off toggle. If a filter has no configurable settings, it should return nothing for the settings, rather than a message like we did before.

Finally, the filter system now includes caching. If your filter's output is dynamic and should not be cached, uncomment the 'no cache' snippet. Only do this when absolutely necessary, because this turns off caching for any input format your filter is used in. Beware of the filter cache when developing your module: it is advised to uncomment 'no cache' while developing, but be sure to remove it again if it's not needed.

Filter tips

Filter tips are now output through the format selector. Modules no longer need to call filter_tips_short() to display them.

A module's filter tips are returned through the filter_tips hook:

```
<?php
function hook_filter_tips($delta, $format, $long = false) {
  if ($long) {
    return t("Long tip");
  }
  else {
    return t("Short tip");
  }
}
?>
```

As in the filter hook you can ignore the $delta parameter if you're upgrading an existing module. If your filter's tips depend on its settings, make sure you use $format to retrieve the setting for the current input format. $long tells you whether to return long or short tips.

Other changes

In addition to the above mentioned changes:

- hook_user() was changed to allow multiple pages of user profile information. The new syntax of the hook is given in the API reference. Pay particular attention to the "categories", "form", and "view" operations.
- When processing a form submission, you should use drupal_goto() to redirect to the result if the submission was accepted. This prevents a double post when people refresh their browser right after submitting. Messages set with drupal_set_message() will be saved across the redirect. If a submission was rejected, you should not use drupal_goto(), but simply print out the form along with error messages.

Converting 4.5 modules to 4.6

Block system

Every block now has a configuration page to control block-specific options. Modules which have configurations for their blocks should move those into hook_block().

The only required changes to modules implementing hook_block() is to be careful about what is returned. Do not return anything if $op is not 'list' or 'view'. Once this change is made, modules will still be compatible with Drupal 4.5.

If a specific block has configuration options, implement the additional $op options in your module. The implementation of 'configure' should return a string containing the configuration form for the block with the appropriate $delta. 'save' will have an additional $edit argument, which will contain the submitted form data for saving.

Search system

The search system got a significant overhaul.

Node indexing now uses the node's processed and filtered output, which means that any custom node fields will automatically be included in the index, as long as they are visible to normal users who view the node. Modules that implement `hook_search()` and `hook_update_index()` just to have extra node fields indexed no longer need to do this.

If you wish to have additional information indexed that is *not* visible in the node display at *node/id*, then you can do so using nodeapi('update index'). If you want to add extra information to the node results, use nodeapi('search result').

However, the standard search is still limited to a keyword search. Modules that implement custom, specific search forms (like project.module) can still do so. Custom search forms that do not use hook_search() should be located/moved to a local task under the /search page.

If you are unsure of what you need to do, please refer to the complete search documentation.

Module paths

The function `module_get_path` was renamed to `drupal_get_path` which now returns the path for all themes, theme engines and modules. Because of this abstraction you must pass an additional parameter identifying the type of item for which the path is requested. The following example compares retrieving the path to image module between Drupal 4.5 and 4.6.

```
<?php
// Drupal 4.5:
$path = module_get_path('image');
// Drupal 4.6:
$path = drupal_get_path('module', 'image');
```

```
?>
```

All instances of `module_get_path` should be renamed to `drupal_get_path`.

Database backend

The function `check_query` was renamed to `db_escape_string` and now has a database specific implementation. All instances of `check_query` should be renamed to `db_escape_string`.

Theme system

The function theme_page() no longer takes `$title` or `$breadcrumb` arguments. Set page titles using hook_menu() or, if the title must be dynamically determined, use drupal_set_title(). Set breadcrumb trails first using hook_menu(), which can be overridden with menu_set_location() and drupal_set_breadcrumb().

Watchdog messages

The watchdog() function now takes a severity attribute, so `watchdog($type, $message, $link);` becomes `watchdog($type, $message, $severity, $link);`. Specify a severity in case you are reporting a warning or error. Possible severity constants are: `WATCHDOG_NOTICE`, `WATCHDOG_WARNING` and `WATCHDOG_ERROR`. Also make sure that you provide the type as a literal string, so translation extraction can pick it up.

If you are unsure of which severity to use, remember these rules:

- If the problem is caused by a definite fault and should be fixed as soon as possible, use an error message.
- If the problem could point to a fault, but could also be harmless, use a warning message. This type should also be used whenever the problem could be caused by a remote server (example: ping timeout, failed to aggregate a feed, etc).
- Normal messages should be notices.

Node markers

If you have a module calling `theme('mark')`, note that it is now possible to have different markers for different states of a node. The supported states are `MARK_NEW`, `MARK_UPDATED` and `MARK_READ`. You can get the marker state from `node_mark()`, which replaces the `node_new()` function available in previous Drupal versions.

Control over destination page after form processing

Occasionally a module might want to specify where a user should go after he submits a form. This is now possible by passing a querystring parameter `&destination=<path>`. For example, editing of nodes and comments from within the Admin pages now returns the user to those pages after he is done. For example usage, search drupal_get_destination() which can be found

in path.module, node.module, comment.module, and user.module

Confirmation messages

Confirmations for dangerous actions should now be presented with the `theme('confirm')` function for consistency. Check the function's documentation or look at some of the core modules for examples.

Note that this is a themable function which should be invoked through `theme('confirm')` and not `theme_confirm()`.

Inter module calls

New features are available -- it's not necessary to use them. Now you can really (and should) use module_invoke to call a function from another module. For example, `taxonomy_get_tree` should be called by `module_invoke('taxonomy', 'get_tree')` If you need to loop through the implementations of a hook, please check the new module_implements function.

Node queries

If you have a module which retrieves a list of nodes by issuing its own database query, then the following applies.

The functions node_access_join_sql() and node_access_where_sql() should not be used any more but the SELECT-queries should be wrapped in a db_rewrite_sql() call.

If you have used DISTINCT(nid) -- because of node_access_join_sql() -- you no longer need it, replace it simply with n.nid. If you have SELECT *, please replace it with SELECT n.nid, n.* -- and always make sure that n.nid field comes first in the SELECT statement -- this way the db_rewrite_sql() function can rewrite the query to use DISTINCT(nid) should there be a need for it. If the n.nid field is not first, the query will fail when node access modules are enabled. Also, at the moment db_rewrite_sql can not handle AS -- either leave it out or lowercase it.

Always use table name before the field names, especially before nid because other tables may be JOINed during the rewrite process.

Example:

```
<?php
// Drupal 4.5:
$nodes = db_query_range('SELECT DISTINCT(n.nid) FROM {node} n '.
node_access_join_sql() .' WHERE '. node_access_where_sql() .' AND
n.promote = 1 AND n.status = 1 ORDER BY n.created DESC', 0, 15);
// Drupal 4.6:
$nodes = db_query_range(db_rewrite_sql('SELECT n.nid FROM {node}
n WHERE n.promote = 1 AND n.status = 1 ORDER BY n.created DESC'), 0,
15);
?>
```

If you are not using the node table, then you shall pass the table name from which you SELECTing the nodes. For example

```
<?php
$result = db_query(db_rewrite_sql("SELECT f.nid, f.* from {files} f
WHERE filepath = '%s'", 'f'), $file);
?>
```

note the 'f' parameter of db_rewrite_sql().

Avoid USING because there could be JOINs before it, which will break the USING clause.

Text output

Drupal's text output was audited and several escaping bugs were found. For more info, see the check_plain patch.

You need to pay attention that all user-submitted plain-text in your module is escaped using check_plain() when you output it into HTML. No escaping should be done on data that is going into the database: only escape when outputting to HTML.

Check_plain() replaces drupal_specialchars() and check_form(), so if you are using any of those two, you should use check_plain() instead.

You should also wrap user-submitted text in messages with `theme('placeholder', $text)`. For example for "created term %term".

Pay attention in particular to node and comment titles as their behaviour has been changed. They are now stored as plain-text, like other single-line fields in Drupal and should be escaped when output. However, the function l() now takes plain-text by default instead of HTML, which means that whenever $node->title is used as the caption for a link, it will automatically be escaped. When outputting titles literally, you still have to escape them yourself.

URLs also require attention, as the URL functions (url, request_uri, referer_uri, etc) were changed to output 'real' URLs rather than HTML-escaped URLs. When putting any of them inside an HTML tag attribute (e.g. <a href="...">), you need to pass it through check_url() first. When putting an URL into HTML outside of a tag or attribute, you can use check_url() or check_plain(), it doesn't matter. Don't use check_url() in situations where a real URL is expected (e.g. the HTTP "Location: ..." header).

The best test is to submit forms with HTML tags in the plain-text/single-line fields (e.g. "<u>test</u>"). If the underline tag is not interpreted, but displayed literally, your module is escaping the text correctly.

Nothing has changed for filtered/rich text, which still uses check_output() like before.

Converting 4.6 modules to HEAD

Taxonomy API change

In order to provide more meaningful messages to the user, you are now required to provide your own when using the taxonomy APIs to create or modify terms and vocabularies. This applies to `taxonomy_save_vocabulary()` and `taxonomy_save_term()`.

A status message is returned, which can be either `SAVED_NEW`, `SAVED_UPDATED` or `SAVED_DELETED`.

This snippet shows you an example of handling this:

```
<?php
// Drupal 4.6
taxonomy_save_vocabulary($edit);
// Drupal 4.7
switch (taxonomy_save_vocabulary($edit)) {
  case SAVED_NEW:
     drupal_set_message(t('Created new vocabulary %name.', array('%name'
=> theme('placeholder', $edit['name']))));
    break;
  case SAVED_UPDATED:
     drupal_set_message(t('Updated vocabulary %name.', array('%name' =>
theme('placeholder', $edit['name']))));
    break;
  case SAVED_DELETED:
     drupal_set_message(t('Deleted vocabulary %name.', array('%name' =>
theme('placeholder', $deleted_name))));
    break;
}
?>
```

Table API change

Themes tables sometimes were called with arguments set to NULL or an empty string to indicate that there were either no rows or no header. This has now to be an empty array.

Change

```
<?php
theme('table', '', $rows);
?>
```

to

```
<?php
theme('table', array(), $rows);
?>
```

Check Output change

Due to a security vulnerability discovered earlier in the filter system, we have tightened security around the `check_output()` function. The format passed to `check_output()` is now checked for access by default. If you don't want this check, pass FALSE for the third parameter, `$check`.

```
<?php
function check_output($text, $format = FILTER_FORMAT_DEFAULT, $check =
TRUE) {
?>
```

Note that if you disable the check by passing FALSE, you need to make sure the `$format` value has been checked by `filter_access()` before. `filter_access()` checks the permissions of the current user, so it should be checked on submission, not on output.

Join forces

Too often new modules are contributed that do nothing new, only do it in a different way. We are then stuck with two modules that offer nearly similar functionality, but both do not do it well enough. This leads to confusion, clutter and a lot of innefficiency.

So please consider the following guidelines or ideas:

- Develop and use one central API. do not introduce any new .incs, .modules or other files with APIS, if there are modules that have these already.
- Consult other developers of modules in your domain when you plan to add features, or plan to add a module. try to agree on features, to avoid overlapping. Nothing is more confusing for a user when he has, for example a Spam Queue for comments, and a completely different one for trackbacks, which does not respect the options you set for comments. Even worse, but certainly not unheard of, is that module Foo breaks module Bar, because they want to do the same, or want to use the same database tables.
- Do not try to duplicate functionality because "you do not really like how its done there" That only adds clutter. Rather improve the existing one, then introduce yet another-half-witted-module.

Note that they are not rules or laws. But that respecting them will most often help you and the community better. For only then will we be able to "stand on the shoulders of Giants" as they say in Open Source Land. If you keep reinventing wheels, you will be stuck with lots of incompatible and half finished wheels, in the end. When you use existing wheels and build a car on top of them, you will be able to get somewhere, one day.

Reference

This section is intended as a handy reference, collecting things which you may need to look up as you code to Drupal.

'Status' field values for nodes and comments

Just documenting the **status** field for the following tables

NODES

- 0: not published
- 1: published

COMMENTS

- 0: published
- 1: not published
- 2: deleted (no longer exists in Drupal 4.5 and above)

Values of 'comment' field in node table

Here are the values of the 'comment' field in the node table:

- 0 = comments cannot be added to this node and published comments will not display
- 1 = comments cannot be added to this node, but published comments will display
- 2 = new comments can be added and published comments will display

Module how-to's

This section collects various 'How-to' articles of interest to module writers and hackers.

How to write a node module

This information is superseded by the Doxygen documentation. In particular, its example node module is a good tutorial.

How to write database independent code

In order to ensure that your module works with all compatible database servers (currently Postgres and MySQL), you'll need to remember a few points.

- When you need to LIMIT your result set to certain number of records, you should use the db_query_range() function instead of db_query(). The syntax of the two functions is the same, with the addition of two required parameters at the end of db_query_range(). Those

parameters are $from and then $count. Usually, $from is 0 and $count is the maximum number of records you want returned.
- If possible, provide SQL setup scripts for each supported database platform. The differences between each platform are slight - we hope documentation on these differences will be forthcoming.
- You should test any complex queries for ANSI compatibility using this tool by Mimer
- If you are developing on MySQL, use it's ANSI compatibility mode
- If you can install all database servers in your environment, it is helpful to create shell databases in each and then run sample queries in each platform's query dispatch tool. Once your query succeeds in all tools, congratulate yourself.
- Don't use `''` when you mean `NULL`
- Avoid table and field names that might be reserved words on any platform.
- Don't use auto-increment or SERIAL fields. Instead, use an integer field and leverage Drupal's own sequencing wrapper: `db_next_id(<tablename_fieldname>)`

How to write efficient database JOINs

This page is based on an e-mail posted by Craig Courtney on 6/21/2003 to the drupal-devel mailing list: http://drupal.org/node/view/322.

There are 3 kinds of join: INNER, LEFT OUTER, and RIGHT OUTER. Each requires an ON clause to let the RDBMS know what fields to use joining the tables. For each join there are two tables: the left table and the right table. The syntax is as follows:

{left table} {INNER | LEFT | RIGHT} JOIN {right table} ON {join criteria}

An INNER JOIN returns only those rows from the left table having a matching row in the right table based on the join criteria.

A LEFT JOIN returns ALL rows from the left table even if no matching rows where found in the right table. Any values selected out of the right table will be null for those rows where no matching row is found in the right table.

A RIGHT JOIN works exactly the same as a left join but reversing the direction. So it would return all rows in the right table regardless of matching rows in the left table.

It is recommended that you **not use right joins**, as a query can always be rewritten to use left joins which tend to be more portable and easier to read.

With all of the joins, if there are multiple rows in one table that match one row in the other table, that row will get returned many times.

For example:
Table A
tid, name
1, 'Linux'
2, 'Debian'

Table B
fid, tid, message
1, 1, 'Very Cool'
2, 1, 'What an example'

Query 1:
SELECT a.name, b.message FROM a INNER JOIN b ON a.tid = b.tid
Result 1:
Linux, Very Cool
Linux, What an example

Query 2:
SELECT a.name, b.message FROM a LEFT JOIN b ON a.tid = b.tid
Result 2:
Linux, Very Cool
Linux, What an example
Debian, <null>

Hope that helps in reading some of the queries.

How to connect to multiple databases within Drupal

Drupal can connect to different databases with elegance and ease!

First define the database connections Drupal can use by editing the `$db_url` string in the Drupal configuration file (settings.php for 4.6 and above, otherwise conf.php). By default only a single connection is defined

```
<?php
$db_url = 'mysql://drupal:drupal@localhost/drupal';
?>
```

To allow multiple database connections, convert `$db_url` to an array.

```
<?php
$db_url['default'] = 'mysql://drupal:drupal@localhost/drupal';
$db_url['mydb'] = 'mysql://user:pwd@localhost/anotherdb';
$db_url['db3'] = 'mysql://user:pwd@localhost/yetanotherdb';
?>
```

Note that database storing your Drupal installation should be keyed as the `default` connection.

To query a different database, simply set it as active by referencing the key name.

```
<?php
db_set_active('mydb');
db_query('SELECT * FROM other_db');
//Switch back to the default connection when finished.
```

```
db_set_active('default');
?>
```

Make sure to always switch back to the default connection so Drupal can cleanly finish the request lifecycle and write to its system tables.

How to write themable modules

Note: this page describes Drupal's theming from the code side of things.

Drupal's theme system is very powerful. You can accommodate rather major changes in overall appearance and significant structural changes. Moreover, you control all aspects of your drupal site in terms of colors, mark-up, layout and even the position of most blocks (or boxes). You can leave blocks out, move them from right to left, up and down until it fits your needs.

At the basis of this are Drupal's theme functions. Each theme function takes a particular piece of data and outputs it as HTML. The default theme functions are all named `theme_something()` or `theme_module_something()`, thus allowing any module to add themeable parts to the default set provided by Drupal. Some of the basic theme functions include: `theme_error()` and `theme_table()` which as their name suggest return HTML code for an error message and a table respectively. Theme functions defined by modules include `theme_forum_display()` and `theme_node_list()`.

Custom themes can implement their own version of these theme functions by defining `mytheme_something()` (if the theme is named `mytheme`). For example, functions named: `mytheme_error()`, `mytheme_table()`, `mytheme_forum_display()`, `mytheme_node_list()`, etc. corresponding to the default theme functions described above.

Drupal invokes these functions indirectly using the `theme()` function. For example:

```
<?php
$node = node_load(array('nid' => $nid));
$output .= theme("node", $node);
?>
```

By default, this will call `theme_node($node)`. However, if the currently active theme is "mytheme", and this theme has defined a function `mytheme_node()`, then `mytheme_node($node)` will be invoked instead.

This simple and straight-forward approach has proven to be both flexible and fast.

However, because direct PHP theming is not ideal for everyone, we have implemented mechanisms on top of this: so-called template engines can act as intermediaries between Drupal and the template/theme. The template engine will override the `theme_functions()` and stick the appropriate content into user defined (X)HTML templates.
This way, no PHP knowledge is required and a lot of the complexity is hidden away. More information about this can be found in the Theme developer's guide, specifically the Theming overview.

Theme developer's guide

This section of our handbook documents aspects of our theme system that will be of interest to theme developers.

Theming overview

Note: this page describes the theme system from a themer's perspective. If you are a module coder looking to make your module themable, you should read this page.

As of version 4.5, Drupal's theme system is very flexible. The new structure makes it easy to plug components together to form your theme: templating engines, templates, stylesheets and PHP.

Here's how some existing themes are built:

<table>
<tr><th>Theme</th><th>Engine (PHP)</th><th>Template (XHTML)</th><th>Style (CSS)</th></tr>
<tr><td>Pushbutton</td><td>XTemplate</td><td>.xtmpl</td><td>.css</td></tr>
<tr><td>Box Grey</td><td rowspan="3">PHPTemplate</td><td rowspan="2">.tpl.php</td><td>.css</td></tr>
<tr><td>Box Cleanslate</td><td>.css</td></tr>
<tr><td>Bluebeach</td><td>.tpl.php</td><td>.css</td></tr>
<tr><td>Chameleon</td><td colspan="2" rowspan="2">Chameleon.theme</td><td>.css</td></tr>
<tr><td>Marvin</td><td>.css</td></tr>
</table>

A 'theme' is now an abstract thing, which can be formed in several ways:

- PHP .theme file containing overrides for `theme_functions`: e.g. Chameleon
- Template file (.xtmpl, .tpl.php) for a templating engine (XTemplate, PHPTemplate, ...): e.g. Pushbutton, Bluebeach
- Style sheet for an existing template or theme: e.g. Marvin, Box Cleanslate

The directory structure for the example above looks like this:

```
themes/engines/xtemplate/xtemplate.engine
themes/engines/phptemplate/phptemplate.engine
themes/pushbutton/xtemplate.tmpl
themes/pushbutton/style.css
themes/box_grey/page.tpl.php
themes/box_grey/style.css
themes/box_grey/box_cleanslate/style.css
themes/bluebeach/page.tpl.php
```

```
themes/bluebeach/style.css
themes/chameleon/chameleon.theme
themes/chameleon/style.css
themes/chameleon/marvin/style.css
```

Themes and templates are placed in their own subdirectory in the `themes` directory. The theme engines will scan every subdirectory for template files (.xtmpl, .tpl.php, ...). If a `style.css` file is present, it will also be used.

You can also make CSS-only themes by making a subdirectory in any theme directory and placing a new `style.css` file in it. Drupal will combine the new stylesheet with the template it belongs in, and make it available as a new theme. This is how the Marvin and Box Cleanslate themes work.

Finally, if there is a `screenshot.png` file in the theme directory, Drupal will show it in the theme administration screen.

Creating custom themes

If you want to create a custom theme, you can either customize an existing theme or start from scratch.

To customize an existing theme, just copy it to a new directory in `themes`, and give it a unique name. Themes should not have a name that is the same as any of the default modules in Drupal or any custom modules you might have enabled or configured. Then modify the copy as much as you want. Depending on whether the theme is template or .theme-file based, you can use PHP or XHTML/CSS to modify it. As explained above, if you only want to alter the CSS of a theme, then just place a new `style.css` file in a subdirectory of the theme: it will appear as a new theme in Drupal.

If you want to start from scratch, there are several ways to go. If you're not a programmer, then the easiest solution is to use one of the template engines. By default, Drupal comes with the XTemplate theme engine, which requires you to create an (X)HTML skeleton with special markers. See the XTemplate documentation for more info. There are other template engines available in the contributions repository (e.g. PHPTemplate).

Drupal themes used to be coded directly in PHP. This method is still available, but is harder to use and maintain than template-based themes.

PHPTemplate theme engine

PHPTemplate is a theme engine written by Adrian Rossouw (who is also behind the theme reforms in Drupal 4.5).

It uses individual `something.tpl.php` files to theme Drupal's `theme_something()` functions. Drupal's themeable functions are documented on the Development Plumbing site. Every file contains an HTML skeleton with some simple PHP statements for the dynamic data.

Thus, PHPTemplate is an excellent choice for theming if you know a bit of PHP: with some basic PHP snippets, you can create advanced themes easily.

If you don't know PHP, then PHPTemplate can still be a good choice because only small bits of code are involved. They can just be copy/pasted into your template.

An extended Forum discussion provides some of the reasoning behind the creation of PHPTemplate.

Installing PHPTemplate

The engine that runs PHPTemplate is not included in the default installation of Drupal. To use themes that use PHPTemplate (e.g. Box_grey, Kubrick, Persian) you must have the PHPTemplate engine installed. To set this up:

1. Download the latest release of the PHPTemplate Engine
2. Upload this folder into the *drupal_base/themes/engines* directory on your site.

You will now be able to use and configure PHPTemplate themes.

Creating a new PHPTemplate

To create a new PHPTemplate, create a new directory under your `themes` directory, for example `themes/mytheme`. Then, you need to create a file called `page.tpl.php` in that directory.

This is the only file which is absolutely required. It overrides the `theme('page')` function, which outputs the final page contents, along with all the extra decorations like a header, tabs, breadcrumbs, sidebars and a footer.

You can create files to override the following functions:

- `theme('page')` (page.tpl.php): theme a page
- `theme('block')` (block.tpl.php): theme a block in sidebar
- `theme('box')` (box.tpl.php): theme a generic container for the main area
- `theme('comment')` (comment.tpl.php): theme a comment
- `theme('node')` (node.tpl.php): theme a node

The PHPTemplate package contains example template files for most of these, see box_grey for an example of page.tpl.php. Simply copy them into your `theme/mytheme` directory and edit them. Note that you will need to visit *administer* > *themes* for PHPTemplate to refresh its cache and recognize any new `.tpl.php` files.

If you want to theme a function other than the defaults listed here, you need to provide an override yourself.

Block.tpl.php

Lays out content for blocks (left and/or right side of page). This template is optional, and can be overridden by copying the default template and modifying it.

Available variables

- $block (object)
  - $block->module : The name of the module that generated the block.
  - $block->delta : The number of the block, in the module.
  - $block->subject : The block title.
  - $block->content : The html content for the block.
  - $block->status : Status of block (0, or 1).
  - $block->path : The path that matches whether or not a block is displayed.
  - $block->region : Left (0), or Right(1) column.
  - $block->throttle: Throttle setting.
- $seqid : The sequential id of the block displayed, ie: The first block is 1, the second block is 2 etc.
- $block_seqid : The same as $seqid, but is reset for the left and right sidebars.
- $zebra : Wether or not the block is odd , or even. This is useful for creating 'zebra stripes' with your css. This value will be either 'odd', or 'even'.
- $block_zebra : The same as $zebra, but is reset for the left and right sidebars.

Default template

The default `block.tpl.php`, which can be found at themes/engines/phptemplate/block.tpl.php.

```
<div class="<?php print "block block-$block->module" ?>" id="<?php
print "block-$block->module-$block->delta"; ?>">
  <h2><?php print $block->subject ?></h2>
  <div class="content"><?php print $block->content ?></div>
</div>
```

Box.tpl.php

Prints a simple html box around a page element. For instance: The comment view options are surrounded by `box.tpl.php`.

Available variables

- $title: The title of the box.
- $content: The content of the box.
- $region: Region. main, left or right.

Default template

```
<div class="box">
  <h2><?php print $title ?></h2>
  <div class="content"><?php print $content ?></div>
</div>
```

Comment.tpl.php

Define the HTML for a comment block. This doesn't have anything to do with comment threading, just the actual comment.

Available variables

- $new : Translated text for 'new', if the comment is infact new.
- $comment(object) : Comment object as passed to the theme_comment function.
- $submitted : Translated post information string.
- $title : Link to the comment title.
- $picture : User picture HTML (include <a> tag.) , if display is enabled and picture is set.
- $links : Contextual links below comment.
- $content : Content of link.
- $author : Link to author profile.
- $date : Formatted date for post.

Default template

```
<div class="comment <?php print ($comment->new) ? 'comment-new' : ''
?>">
<?php if ($comment->new) : ?>
  <a id="new"></a>
  <span class="new"><?php print $new ?></span>
<?php endif; ?>
<div class="title"><?php print $title ?></div>
  <?php print $picture ?>
  <div class="author"><?php print $submitted ?></div>
  <div class="content"><?php print $content ?></div>
  <?php if ($picture) : ?>
    <br class="clear" />
  <?php endif; ?>
  <div class="links"><?php print $links ?></div>
</div>
```

Node.tpl.php

This template controls the display of a node, and a node summary.

Available variables

- $title : Title of node.
- $node_url : Link to node.
- $terms : HTML for taxonomy terms.
- $name : Formatted name of author.
- $date : Formatted data.
- $sticky : True if the node is sticky on the front page.
- $picture : HTML for user picture, if enabled.
- $content : Node content, teaser if it is a summary.
- $links : Node links.
- $taxonomy (array) : array of taxonomy terms.
- $node (object) : The node object.
- $main : This variable is set to 1 if the node is being displayed on the main page, 0 otherwise.
- $page : True if on the node view page, and not a summary.
- $submitted : Translated text, if the node info display is enabled for this node type.

Default template

```
<div class="node<?php print ($sticky) ? " sticky" : ""; ?>">
  <?php if ($page == 0): ?>
      <h2><a href="<?php print $node_url ?>" title="<?php print $title
?>"><?php print $title ?></a></h2>
  <?php endif; ?>
  <?php print $picture ?>
   <div class="info"><?php print $submitted ?><span class="terms"><?php
print $terms ?></span></div>
  <div class="content">
    <?php print $content ?>
  </div>
<?php if ($links): ?>
    <?php if ($picture): ?>
      <br class='clear' />
    <?php endif; ?>
    <div class="links"><?php print $links ?></div>
<?php endif; ?>
</div>
```

Theme distinct node types differently

You can easily use PHPTemplate to produce specialized themes for specific node types. For example, to theme forum posts seperatly from your other nodes, use node-forum.tpl.php. This will work for any node type.

Here is the basic template.
`node-<node type>.tpl.php`

Page.tpl.php

This template defines the main skeleton for the page.

Available variables

- head_title: The text to be displayed in the page title.
- language: The language the site is being displayed in.
- site: The name of the site, always filled in.
- head: HTML as generated by drupal_get_html_head() (needed to dynamically add scripts to pages)
- onload_attributes: Onload tags to be added to the head tag, to allow for autoexecution of attached scripts.
- directory: The directory the theme is located in , ie: themes/box_grey or themes/box_grey/box_cleanslate
- logo: The path to the logo image, as defined in theme configuration.
- site_name: The site name of the site, to be used in the header, empty when display has been disabled.
- site_slogan: The slogan of the site, empty when display has been disabled.
- search_box: True(1) if the search box has been enabled.
- search_url: URL the search form is submitted to.
- search_button_text: Translated text on the search button.
- search_description: Translated description for the search button.
- title: Title, different from head_title, as this is just the node title most of the time.
- primary_links (array): An array containing the links as they have been defined in the phptemplate specific configuration block.
- secondary_links (array): An array containing the links as they have been defined in the phptemplate specific configuration block.
- breadcrumb: HTML for displaying the breadcrumbs at the top of the page.
- tabs: HTML for displaying tabs at the top of the page.
- messages: HTML for status and error messages, to be displayed at the top of the page.
- layout: This setting allows you to style different types of layout ('none', 'left', 'right' or 'both') differently, depending on how many sidebars are enabled.
- help: Dynamic help text, mostly for admin pages.
- styles: Required for stylesheet switching to work. This prints out the style tags required.
- mission: The text of the site mission.

- is_front: True if the front page is currently being displayed. Used to toggle the mission.
- sidebar_left: The HTML for the left sidebar.
- content: The HTML content generated by Drupal to be displayed.
- sidebar_right: The HTML for the right sidebar.
- footer_message: The footer message as defined in the admin settings.
- closure: Needs to be displayed at the bottom of the page, for any dynamic javascript that needs to be called once the page has already been displayed.

Default template

Here is the contents of the box_grey template's `page.tpl.php`, to give you an idea of the layout of the file.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <title><?php print $title ?></title>
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <?php print $head ?>
  <?php print $styles ?>
</head>
<body <?php print theme("onload_attribute"); ?>>
<div id="header">
  <?php if ($search_box): ?>
  <form action="<?php print url("search") ?>" method="post">
    <div id="search">
      <input class="form-text" type="text" size="15" value=""
name="keys"  /><input  class="form-submit"  type="submit"  value="<?php
print t("Search")?>" />
    </div>
  </form>
  <?php endif; ?>
  <?php if ($logo) : ?>
   <a  href="<?php  print  url()  ?>"  title="Index  Page"><img  src="<?php
print($logo) ?>" alt="Logo" /></a>
  <?php endif; ?>
  <?php if ($site_name) : ?>
       <h1  id="site-name"><a  href="<?php  print  url()  ?>"  title="Index
Page"><?php print($site_name) ?></a></h1>
  <?php endif;?>
  <?php if ($site_slogan) : ?>
    <span id="site-slogan"><?php print($site_slogan) ?></span>
  <?php endif;?>
  <br class="clear" />
</div>
```

```
<div id="top-nav">
  <?php if (is_array($secondary_links)) : ?>
    <ul id="secondary">
    <?php foreach ($secondary_links as $link): ?>
      <li><?php print $link?></li>
    <?php endforeach; ?>
    </ul>
  <?php endif; ?>
  <?php if (is_array($primary_links)) : ?>
    <ul id="primary">
    <?php foreach ($primary_links as $link): ?>
      <li><?php print $link?></li>
    <?php endforeach; ?>
    </ul>
  <?php endif; ?>
</div>
<table id="content">
  <tr>
    <?php if ($sidebar_left != ""): ?>
      <td class="sidebar" id="sidebar-left">
        <?php print $sidebar_left ?>
      </td>
    <?php endif; ?>
        <td class="main-content" id="content-<?php print $layout ?>">
        <?php if ($title != ""): ?>
          <h2 class="content-title"><?php print $title ?></h2>
        <?php endif; ?>
        <?php if ($tabs != ""): ?>
          <?php print $tabs ?>
        <?php endif; ?>
        <?php if ($mission != ""): ?>
          <p id="mission"><?php print $mission ?></p>
        <?php endif; ?>
        <?php if ($help != ""): ?>
          <p id="help"><?php print $help ?></p>
        <?php endif; ?>
        <?php if ($messages != ""): ?>
          <div id="message"><?php print $messages ?></div>
        <?php endif; ?>
        <!-- start main content -->
        <?php print($content) ?>
        <!-- end main content -->
        </td><!-- mainContent -->
    <?php if ($sidebar_right != ""): ?>
    <td class="sidebar" id="sidebar-right">
        <?php print $sidebar_right ?>
    </td>
```

```
    <?php endif; ?>
  </tr>
</table>
<?php if ($breadcrumb != ""): ?>
  <?php print $breadcrumb ?>
<?php endif; ?>
<div id="footer">
  <?php if ($footer_message) : ?>
    <p><?php print $footer_message;?></p>
  <?php endif; ?>
Validate  <a  href="http://validator.w3.org/check/referer">XHTML</a>  or
<a href="http://jigsaw.w3.org/css-validator/check/referer">CSS</a>.
</div><!-- footer -->
 <?php print $closure;?>
  </body>
</html>
```

Alternative templates for different node types

There are times when you may want to create a static page within Drupal such as an "about" page, help pages and the like. Obviously for these, you don't want the title, author links, or indeed anything except the page content.

To accomplish this, copy your node.tpl.php to node-$type.tpl.php. In the case of a static page, you would copy your node.tpl.php file to node-page.tpl.php.

The path module will then allow you to type in "clean URLs" like "about", "bio", etc.

This feature was added to the last release of the 4.5 release of PHPTemplate, so make sure that your phptemplate is current if this doesn't work for you.

Example - Theming flexinode

Since it took me a while to make sense of this I thought I would post an example to help others along the way. It's actually quite simple, just not very intuitive.

This example is for changing the way that the flexinode 'date/time' field will display on a page. (I only wanted month and year to show). But could very easily be adapted to other things.

My theme is called 'licc' - it is a phptemplate theme.

The Quick Version

In the directory for your phptemplate theme (this could be an existing or custom theme), create the following 2 files. *For me the files were put in 'themes/licc'.*

Create template.php

Something like this:

```
<?php
/**
* Override theme_flexinode_timestamp() from
modules/flexinode/field_timestamp.inc
*/
function phptemplate_flexinode_timestamp($field_id, $label, $value,
$formatted_value) {
  // nothing happens here.
   return _phptemplate_callback('flexinode_timestamp', array('field_id'
=> $field_id, 'label' => $label, 'value' => $value, 'formatted_value'
=> $formatted_value));
}
?>
```

Create flexinode_timestamp.tpl.php

Something like this:

```
<?php
$formatted_value = strftime ("%B %Y", $value); // format as Month and
Year, eg. 'July 2004'
?>
<div class="flexinode-timestamp-<?php print $field_id; ?>">
<strong><?php print $label; ?>: </strong><br />
<?php print $formatted_value; ?>
</div>
```

That's it! Just modify the second file so that the field is displayed the way you would like.

Note: that you will need to visit *administer* > *themes* for PHPTemplate to refresh its cache and recognize any new .tpl.php files.

Below is the long-winded version, read on if you are interested...

The Long Version

1. find the theme function for the flexinode field

Found in modules/flexinode/field_timestamp.inc

```
<?php
function theme_flexinode_timestamp($field_id, $label, $value,
$formatted_value) {
```

```
  $output = theme('form_element', $label, $formatted_value);
     $output   =   '<div   class="flexinode-timestamp-'.   $field_id   .'">'.
$output .'</div>';
  return $output;
}
?>
```

This is just for reference, you could just as easily look in the API documentation. Core documentation is here:
http://drupaldocs.org/api/head/group/themeable
(only core modules seem to be online at the moment, so you will need to search through the code for any add-on modules like flexinode)

If you wanted to theme flexinode 'image' fields, you would need to look for the theme function in modules/flexinode/field_image.inc

2. Create template.php and add override function

For my theme I created themes/licc/template.php and then copied the function declaration from above replacing the word 'theme' with 'phptemplate'.

```
<?php
function   phptemplate_flexinode_timestamp($field_id,   $label,   $value,
$formatted_value) {
}
?>
```

add in the phptemplate callback:

```
<?php
return _phptemplate_callback('flexinode_timestamp', array('field_id' =>
$field_id, 'label' =>
$label, 'value' => $value, 'formatted_value' => $formatted_value));
?>
```

This function doesn't really _do_ anything except give phptemplate control over the display of this field, the next step looks after the actual formatting. Note how the variables are passed on to the _phptemplate_callback() in an associative array.

Note: do not use the key 'file' in the callback array, as it causes problems for phptemplate. This is a value used for the image field in particular. This is what I did (for the image field) to get around this problem (see 'imgfile' used instead of 'file')

```
<?php
/**
* Override theme_flexinode_image() from
modules/flexinode/field_image.inc
*/
```

```
function phptemplate_flexinode_image($field_id, $label, $file,
$formatted_value) {
  // empty 'stub' function
    return  _phptemplate_callback('flexinode_image',  array('field_id'  =>
$field_id, 'label' => $la
bel, 'imgfile' => $file, 'formatted_value' => $formatted_value));
}
?>
```

3. Create flexinode_timestamp.tpl.php to do formatting

This goes in your theme directory (for me themes/licc/flexinode_timestamp.tpl.php). As you can see the name matches the bit after 'phptemplate_' in the theme override function, and the first argument of the _phptemplate_callback().

Put the HTML/PHP that you want in this file for the display of all date/time (timestamp) fields in all flexinode pages.

Something like this:

```
<div class="flexinode-timestamp-<?php print $field_id; ?>">
<strong><?php print $label; ?>: </strong><br />
<?php print $formatted_value; ?>
</div>
```

Example Files

template.php

```
<?php
/***
* template.php
*
*  This  file  contains  functions  for  over-riding  the  default  theme
functions
* in Drupal core and modules (look at the API documentation for more
info).
* The functions don't actually _do_ anything, except pass the variables
* available to phptemplate for use in the *.tpl.php files.
*
*  Add  similar  'stub'  functions  to  override  other  default  theme
functions.
*/
/**
* Override theme_flexinode_timestamp() from
modules/flexinode/field_timestamp.inc
*/
```

```
function phptemplate_flexinode_timestamp($field_id, $label, $value,
$formatted_value) {
  // like I said, nothing happens here.
   return _phptemplate_callback('flexinode_timestamp', array('field_id'
=> $field_id, 'label' => $label, 'value' => $value, 'formatted_value'
=> $formatted_value));
}
?>
```

flexinode_timestamp.tpl.php

```
<?php
/***
* Customised formatting of flexinode timestamp data in nodes.
* These fields are available:
* $field_id, $label, $value, $formatted_value
**/
// Change the default $formatted_value so that it suits me (no time or
day)
$formatted_value = strftime ("%B %Y", $value); // format as Month and
Year, eg. 'July 2004'
?>
<div class="flexinode-timestamp-<?php print $field_id; ?>">
<strong><?php print $label; ?>: </strong><br />
<?php print $formatted_value; ?>
</div>
```

Making additional variables available to your templates

Examples from this forum discussion. The $hook refers to the area the variable is to be used in (e.g. for comment.tpl.php, it would be "comment").

This function needs to be defined in a `template.php` file, which is placed inside the template directory (for instance : `themes/box_cleanslate/template.php`)

Note: For these changes to take effect, you need to load the admin/themes page first.

```
<?php
function _phptemplate_variables($hook, $vars) {
   switch($hook) {
     case 'comment' :
        $vars['newvar'] = 'new variable';
        $vars['title'] = 'new title';
        break;
   }
   return $vars;
}
```

```
?>
```

The output of this function is merged with the variables returned from phptemplate_comment, so you can easily adjust whichever variables you feel necessary.

Your `comment.tpl.php` file will now have a new variable available in it called $newvar. Similarly the $title variable will be overridden with the value specified in the function.

A neat trick is to count how many times each of the hooks is called, so you can pass an extra variable. re :

```
<?php
function _phptemplate_variables($hook, $vars) {
    static $count;
    $count = is_array($count) ? $count : array();
    $count[$hook] = is_int($count[$hook]) ? $count[$hook] : 1;
    $vars['zebra'] = ($count[$hook] % 2) ?'odd' : 'odd';
    $vars['seqid'] = $count[$hook]++;
    return $vars;
  }
?>
```

That is 'even' if it is an even number, and 'odd' if it is odd. This means you do zebra striping (ie: alternating colors) for each of your nodes / blocks / comments / whatever.

Then you can set up a some styles for class='$zebra' , which handle the alternating colors.

Another example is a flag to show us if we are looking at a node. Might be handy for rendering items different, when someone is looking at an article.

```
<?php
function _phptemplate_variables($hook, $vars) {
  switch ($hook) {
    case 'page':
      if (arg(0) == 'node' && is_numeric(arg(1)) && arg(2) == '') {
        $vars['content_is_node'] = TRUE;
      }
    break;
  }
  return $vars;
}
?>
```

Note that the switch is kind of obsolete here, but i leave it here, because you might want to add more variables. In that case you need them.

The args() checks will see if you have an url like /node/NID/ and not like /node/NID/edit or /node. If that is found, we set the flag TRUE.

Overriding other theme functions

If you want to override a theme function not included in the basic list (block, box, comment, node, page), you need to tell PHPTemplate about it.

To do this, you need to create a `template.php` file in your theme's directory. This file should contain the required <?php ?> tags, along with *stubs* for the theme overrides. These stubs instruct the engine what template file to use and which variables to pass to it.

First, you need to locate the appropriate theme function to override. You can find a list of these in the API documentation. We will use `theme_item_list()` as an example.

The function definition for `theme_item_list()` looks like this:

```
<?php
function theme_item_list($items = array(), $title = NULL) {
?>
```

Now you need to place a stub in your theme's `template.php`, like this:

```
<?php
/**
* Catch the theme_item_list function, and redirect through the template
api
*/
function phptemplate_item_list($items = array(), $title = NULL) {
   // Pass to phptemplate, including translating the parameters to an
associative array. The element names are the names that the variables
  // will be assigned within your template.
    return _phptemplate_callback('item_list', array('items' => $items,
'title' => $title));
}
?>
```

We replaced the word `theme` in the function name with `phptemplate` and used a call to `_phptemplate_callback()` to pass the parameters (`$items` and `$title`) to PHPTemplate.

Now, you can create a `item_list.tpl.php` file in your theme's directory, which will be used to theme item lists. This function should follow the same logic as the original `theme_item_list()`.

Note that you will need to visit *admininster* > *themes* for PHPTemplate to refresh its cache and recognize the new file. Beginning with version 4.6, this is not necessary anymore.

Example - Overriding the user profile pages using PHPTemplate

This description Illustrates how easy it is to override theme functions. I'm very very new to php and sql, but even I managed to work out how to customize how user profile pages appear by overriding the theme function.

Before

This is how the out-of-the-box user profile looks like, with extra profile fields, such as City, Country, Postcode, Position etc. added in. (please note that i couldn't fit the whole page into the one screenshot..there is an extra "background/more info." field that doesn't show in the BEFORE screen shot.

click to view the BEFORE screenshot in a new window

After

This is how the exact same user profile looks after overriding the theme and applying a simple `user_profile.tpl.php` file in my theme directory.

click to view the AFTER screenshot in a new window

More details & discussion on this is in the original forum post.

Not including drupal.css

You can overridde the `theme_stylesheet_import` function and omit drupal.css. Simply add this to your theme's template.php file:

```
<?php
function phptemplate_stylesheet_import($stylesheet, $media = 'all') {
  if ($stylesheet != 'misc/drupal.css') {
    return theme_stylesheet_import($stylesheet, $media);
  }
}
?>
```

of course, you may include your own version of drupal.css here but that's the basic idea.

Protecting content from anonymous users when using overrides

This is a useful tip, especially for designers or newbies to php who want to unleash the power of drupal & the power of CSS, layouts while protecting content intended for Logged In users only.

Using PHPTemplate Overrides with protected content

PHPtemplate is superb, in my opinion. I particularly like the ability to override specific layouts, but, when you override the theme function you also override/bypass permission settings -- it doesn't pass through any user access layer in Drupal. **Which is important to know if your drupal site has content that is intended for logged in users only.**

Example

In the example below, I wanted to override the way a User Profile is displayed.

Access to view User Profiles was set under ADMINISTER -> USERS -> CONFIGURE -> PERMISSIONS so that only logged in Users could view a user profile.

After the page layout override was implemented, anyone could see profile pages by guessing a link like `?=user/989` for example.

Solution

To get around that problem (and after a lot of playing around) I came up with the following solution, i.e. check to see if the user is logged in BEFORE invoking your phptemplate override. It's remarkably simple, now that I have worked it out, but, I thought it would be worth sharing on here as there maybe other drupal site administrators like me who are as thick as a plank of wood and do not have a lot of experiece in PHP programming.

I have pasted example code below that goes in the template.php file in the themes folder which invokes an override and loads a custom user_profile.tpl.php which overrides the way a user profile is displayed.

```
<?php
/**
* check if the user is logged in before invoking the template override
*/
global $user;
if($user->uid) // check to see if the user is logged in
{
function phptemplate_user_profile($user, $fields = array()) {
   // Pass to phptemplate, including translating the parameters to an
associative array. The element names are the names that the variables
  // will be assigned within your template.
  /* potential need for other code to extract field info */
return  _phptemplate_callback('user_profile',  array('user'  =>  $user,
'fields' => $fields));
  }
}
?>
```

Themeing front page and others

One weakness of many Drupal sites is the *sameness* of the pages. The sections.module allows the admin to assign different themes to different areas of the site, but for a site with custom themes this requires duplication of a lot of resources such as css and image files into separate theme directories.

FactoryJoe recently turned me onto a great trick for getting much of the sections.module functionality into a single phptemplate theme.

Probably the easiest and most common use is to give the home (a.k.a. front) page an entirely different layout. It turns out that this is really easy to do. Here's how:

1. Create your home page theme template using the techniques described elsewhere in this manual. Call this file "home.tpl.php" and place it into the directory for the theme you're developing.

2. Create your second-level page template and call it "page.tpl.php". But, here's the trick, at the very top of the file, before the doctype declaration or anything, add this code:

```
<?php
if ($is_front) {
  include('home.tpl.php');
  return;
}
?>
```

This checks phptemplate's $is_front variable and if it is true, the home.tpl.php file is imported and the 'return' line keeps the rest of the page.tpl.php code from executing.

This technique can be extended to other areas of the site by substituting other tests for $is_front. For instance you could retheme the entire administration area, by using the following:

```
<?php
if ($is_front) {
  include('home.tpl.php');
  return;
}
elseif (arg(0)=="admin") {
  include('admin.tpl.php');
  return;
}
?>
```

Find more information about Drupal's arg() function here.

Note that there shouldn't be a line break before the DOCTYPE declaration, so you should immediately follow both examples with the doctype line. Example:

<?php
// code //
return;
} ?><!DOCTYPE HTML PUBLIC "-//W3C//DTD HT... etc...

XTemplate to PHPTemplate conversion

Firstly rename the xtemplate.xtmpl file to original.xtmpl so that the theme is no longer a xtemplate theme.

Creating page.tpl.php

1. copy original.xtmpl to page.tpl.php
2. Remove the node, comment, box, and block sections. In the place of these sections add the following code
   <?php echo $content ?>
3. Change all the "{" characters to "<?php print $"
4. Change all the "}" characters to "; ?>"
5. Change the "$footer" to "$closure"
6. Change "$message" to "$messages"
7. As the primary and secondary links in phptemplate are arrays you will need to change them from "echo $primary_links;" to "echo theme('links', $primary_links);". Also the same needs to be done for secondary links.
8. In the blocks section we need to change the $block to either $sidebar_left or $sidebar_right depending on which side of the content it is on.

Creating node.tpl.php

1. Copy the node section from the original.xtmpl to a new file node.tpl.php
2. As before change all the "{" and "}" characters to "<?php print $" and "; ?>" respectively.
3. change $link to $node_url.
4. Change "$taxonomy" to "$terms"
5. Change "print $sticky;" to "if ($sticky) { print " sticky"; }"
6. Change "print $picture;" to "if ($picture) { print $picture; }"

Creating comment.tpl.php

1. Copy the comment section from the original.xtmpl to a new file comment.tpl.php
2. As before change all the "{" and "}" characters to "<?php print $" and "; ?>" respectively.
3. Change "print $picture;" to "if ($picture) { print $picture; }"

Also just to be clean, you may want to change the displaying of the new so it will only show when the $new != ''

Create block.tpl.php

1. Copy the block section from the original.xtmpl to a new file comment.tpl.php
2. As before change all the "{" and "}" characters to "<?php print $block->" and "; ?>" respectively.
3. Then change the $block->title to $block->subject.

Create box.tpl.php

1. Copy the box section from the original.xtmpl to a new file comment.tpl.php
2. As before change all the "{" and "}" characters to "<?php print $" and "; ?>" respectively.

XTemplate theme engine

The XTemplate theme system uses templates to layout and style Web pages. It separates **logic** (PHP), **structure** (XHTML/HTML), and **style** (CSS), making it easy for designers to create or modify templates by working on XHTML/HTML and CSS without having to worry about any PHP coding.

XTemplate templates are directories, which contain all the XHTML/HTML, CSS, image and JavaScript files that a template uses. Templates are located in the themes directory of a Drupal installation:

```
/themes/
```

Once a template exists in the themes directory, XTemplate auto-detects it, and makes it available for selection to administrators:

```
administer -> themes
```

Drupal is distributed with two XTemplate templates included - **Bluemarine** and **Pushbutton**.

Although XTemplate is still supported as part of the core, it may not be in the future, for several reasons. This will not necessarily mean the end of XTemplate since it may be maintained as an alternative contributed engine like PHPTemplate.

Creating a new XTemplate

To make a new XTemplate template, create a directory in your Drupal installation at this location:

```
/themes/
```

Whatever you name the new directory will be used as the name of your new template, for instance:

```
/themes/rembrant
```

Once you create a template in this directory, it will appear on the theme selection page as the "rembrant" template.

The easiest way to create a new template is to make a copy of an existing template, such as Default or Pushbutton, and start making changes to the files.

The only file required in a template directory is **xtemplate.xtmpl**, which is a regular HTML or XHTML file containing some XTemplate tags that Drupal substitutes with content when a page is served. The xtemplate.xtmpl file can be edited in DreamWeaver, GoLive, BBEdit or any other application you use to work on HTML/XHTML.

All other files in the template are optional, and are linked to from the xtemplate.xtmpl file. These can include CSS, image or JavaScript files, and should all be included in the template directory to make the template easy to maintain and portable between Drupal installations.

Note that if you name your stylesheet `style.css`, it will automatically be picked up by Drupal, and you will not need to add an explicit `@import` or `<link />` for it. If you make a subdirectory within your template, containing another `style.css` file, then the subdirectory becomes a new theme, using the XHTML from the first template, but with a different stylesheet.

Template basics

xTemplate creates Web pages by substituting place holder tags in a template, the xtemplate.xtmpl file, with content from the database.

There are two kinds of template place holder tags, section tags and item tags.

Section Tags

Section tags deal with the structure of a Web page, marking areas of the page, and are XHTML/HTML comment tags which look like this:

```
<!-- BEGIN: title -->

<!-- END: title -->
```

Some section tags mark areas were the content, and it's structure, will be repeated. For instance the comment section may be repeated more than once depending on how many comments are on a page:

```
<!-- BEGIN: comment -->
```

```
<!-- END: comment -->
```

Section tags can be nested, so that one set of section tags can be contained by annother:

```
<!-- BEGIN: node -->

  <!-- BEGIN: title -->

  <!-- END: title -->

<!-- END: node -->
```

Item Tags

Item tags are place holders for content items, such as the title of a page, who the page was submitted by, or the main content of a page. Item tags look like this:

```
{title}

{submitted}

{content}
```

Item tags are associated with the section tag that surrounds them, for instance:

```
<!-- BEGIN: node -->
{title}
<!-- END: node -->
```

The {title} tag above is the main title of a page, while the {title} tag below is the title for the comments on a page.

```
<!-- BEGIN: comment -->
{title}
<!-- END: comment -->
```

Header section

The Section

The xTemplate Header section starts and ends with these tags

```
<!-- BEGIN: header -->

<!-- END: header -->
```

Don't confuse the Header section with the XHTML/HTML <head> element. Although the <head> element is included in the Header section, it also holds the top part of the Web page - the area designers usualy refer to as the "Header", which usually consists of a horizontal bar with the site's logo and some navigation links.

Prolog

The WC3 recommends that all XHTML documents should start with an XML prolog specifying the encoding of the document, for instance:

```
<?xml version="1.0" encoding="utf-8"?>
```

Unfortunately there are many browsers that handle the XML prolog badly, and either crash, fail to display the page, or display it incorrectly. It is therefore recommended to leave out the XML prolog, and specify encoding in a Content-Type element in the <head> of your template (which Drupal does automatically).

DOCTYPE

The DOCTYPE element tells a browser two things, which XML language the document is using, and where the DTD (Document Type Declaration) of that language is located.

This is an example of a DOCTYPE element:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

There should be absolutely nothing in your document before the DOCTYPE or XML prolog. The xTemplate tag <!-- BEGIN: header --> is OK, as it will be removed by Drupal before sending the page to the browser, but make sure to **remove spaces or line breaks** between this and the DOCTYPE or XML prolog elements, or you may get unexpected results in some browsers.

To learn more about the DOCTYPE element, and which version would suit your needs best, read:

Fix Your Site With the Right DOCTYPE!
by Jeffrey Zeldman

{head_title}

Content of the <title> element. Used as the window title by browsers, and as the page title in search engine listings.

{head}

Filled in with the following:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<base href="http://yoursite.com/" />
<style type="text/css" media="all">
@import url(misc/drupal.css);
</style>
```

{styles}

Declarations for the current style:

```
<style type="text/css" media="all">@import "themes/bluemarine/style.css";</style>
```

Add this tag to allow your template to take advantage of the Drupal theme system's style-switching ability. Note that, if you have a default stylesheet, it should be named `style.css` and be located in the same directory as your `xtemplate.xtmpl` file.

{onload_attributes}

The page attributes for the <body> tag.

{logo}

The logo section begins and ends with these tags:

```
<!-- BEGIN: logo -->

<!-- END: logo -->
```

The filename for the site logo, configurable by the Administrator in the text box in the Drupal theme administration section. (Display of this item is optional.)

{site_name}

The site name section begins and ends with these tags:

```
<!-- BEGIN: site_name -->

<!-- END: site_name -->
```

The current site name, configured by the Administrator in the text box "Name" on Drupal page:

```
administer->settings
```

(Display of this item is optional.)

{site_slogan}

The site slogan section begins and ends with these tags:

```
<!-- BEGIN: site_slogan -->

<!-- END: site_slogan -->
```

The current site slogan, configured by the Administrator in the text box "Slogan" on Drupal page:

```
administer->settings
```

(Display of this item is optional.)

{secondary_links} {primary_links}

These tags hold whatever the Administrator inputs into the text boxes "Secondary links:" and "Primary links" in the Drupal theme administration section. If the Administrator does not specify any "Primary links", Drupal will automatically generate a set of links based on the currently-enabled modules.

The Administrator could use these tags to input links to the main sections of the site, the title of the site, a site message, an image or anything else they require.

Search Box

The Search Box section begins and ends with these tags:

```
<!-- BEGIN: search_box -->

<!-- END: search_box -->
```

{search_url}

The form action: "search"

{search_description}

The alt text description of the search text box: "Enter the terms you wish to search for."

{search_button_text}

The value of the search submit button: "Search"

Mission

The Mission section begins and ends with these tags:

<!-- BEGIN: mission -->

<!-- END: mission -->

{mission}

The text of the site mission statement, appears only on the Home Page, and is configured by the Administrator in the text box "Mission" on Drupal page:

```
administer->settings
```

Title

The Title section begins and ends with these tags:

<!-- BEGIN: title -->

<!-- END: title -->

{title}

The title of the node

Tabs

The Tabs section begins and ends with these tags:

```
<!-- BEGIN: tabs -->
```

```
<!-- END: tabs -->
```

{tabs}

Draws the Drupal "local tasks" for the current page.

{breadcrumb}

The breadcrumb trail of the page, the path from Home Page to the current page.

Help

The Help section begins and ends with these tags:

```
<!-- BEGIN: help -->
```

```
<!-- END: help -->
```

{help}

Contains any help information which exists for a particular page.

Message

The Message section begins and ends with these tags:

```
<!-- BEGIN: message -->
```

```
<!-- END: message -->
```

Message appears when Drupal confirms the results of an action by the user, for instance after updating or deleting a page.

{message}

The text of the message.

Node section

The Node Section

The node section (xtemplate.xtmpl) contains the main content of the page, and begins and ends with these tags:

```
<!-- BEGIN: node -->

<!-- END: node -->
```

{sticky}

Sets the class to "node sticky" if a node is "stickied" at the top of lists. (i.e. if a teaser for the page is always to be displayed on the home page) If the node has not been set to be sticky, the class is set to "node ".

Picture

Picture contains an image representing the user who posted the content of a node, the image is linked to the poster's profile. This is also sometimes called an "avatar". Picture begins and ends with these tags:

```
<!-- BEGIN: picture -->

<!-- END: picture -->
```

{picture}

Outputs the following:

```
<a href="user/1" title="View user profile.">

<img src="http://www.yoursite/files/pictures/picture-1.gif"
alt="Username's picture" /></a>
```

Title

The title of the main content of the page (node), tags begin and end:

```
<!-- BEGIN: title -->

<!-- END: title -->
```

On a node page, the title is output as:

```
<h1 class="title">Node Title</h1>
```

On the Home Page, each node title is output as:

```
<h2 class="title"><a href="node/31"
>Node Title</a></h2>
```

{link}

Outputs the link to the node , "node/31" in the example above.

{title}

Outputs the text of the node title, "Node Title" in the example above.

{submitted}

The username of the person who submitted the node content, outputs:

```
Submitted by <a href="user/1" title="View user profile."
>Username</a> on 16 February, 2004 - 23:46.
```

Taxonomy

A list of links to taxonomies which the node belongs to, tags begin and end:

```
<!-- BEGIN: taxonomy -->

<!-- END: taxonomy -->
```

{taxonomy}

Outputs a taxonomy term that the node belonds to:

Taxonomy Term

{content}

The main content of the node.

Links

The control options for the node: "printer-friendly version", "add new comment", and the visitor history of the node. Tags begin and end:

```
<!-- BEGIN: links -->

<!-- END: links -->
```

{links}

Outputs the following (depending on the viewer's permisions):

```
<a href="book/print/8"
title="Show a printer-friendly version of this book page
and its sub-pages.">printer-friendly version</a> |
<a href="comment/reply/8#comment"
title="Share your thoughts and opinions related to this posting."
>add new comment</a> |
<a href="admin/statistics/log/node/8">662 reads</a>
```

Comment

The Comment Section

The comment section (xtemplate.xtmpl) contains all the comments associated with a node, and begins and ends with these tags:

```
<!-- BEGIN: comment -->

<!-- END: comment -->
```

The content of this section creates the code for a single comment, and is automaticaly repeated for as many times are there are comments.

Avatar

Avatar contains an image representing the user who posted the content of a node, the image is linked to the poster's profile. Avatar begins and ends with these tags:

```
<!-- BEGIN: avatar -->

<!-- END: avatar -->
```

{avatar}

Outputs the following:

```
<div class="avatar">
<a href="user/1" title="View user profile.">
<img src="http://www.drupal.site/files/avatars/avatar-1.jpg" alt="username's avatar" />
</a>
</div>
```

Title

The title of a comment. Tags begin and end:

```
<!-- BEGIN: title -->

<!-- END: title -->
```

{link}

If required, changes the comment title into a link to the comment. Used when displaying comments in certain views.

{title}

The text of the comment title.

Submitted

{submitted}

Displays the username of the comment poster, linked to their profile, and the date and time the comment was posted. This is the output:

Submitted by username on Mon, 04/19/2008 - 11:56.

New

Indicates if a comment is new. Tags begin and end:

```
<!-- BEGIN: new -->

<!-- END: new -->
```

{new}

Adds the word "new" to a comment.

Content

Displays the content of a comment.

{content}

The comment text.

Links

Displays control links for comment, such as "reply", "delete", and "edit". Tags begin and end:

```
<!-- BEGIN: links -->

<!-- END: links -->
```

{links}

Displays the control links.

Blocks

The Section

The blocks section contains the column of boxes which can be used to display various navigation and feature options, such as **Forum Topics**, **Blogs**, **Who's Online**, and **Syndicate**. Blocks sections can be configured to appear on the left or right of a page, or on both sides. The section begins and ends with this code:

```
<!-- BEGIN: blocks -->
<!-- END: blocks -->
```

{blocks}

This tag is replaced by whatever blocks have been switched on in the Administration page (admin/system/block).

Block

The block section defines the structure of each block, note the 's' in block/blocks.

```
<!-- BEGIN: block -->
```

```
<!-- END: block -->
```

{module}

The name of the module who's block is being displayed, this is added to a CSS class and ID which can be used customise the look of the block.

{delta}

Adds a number to the ID of a block, so that each block has a unique ID even if a module displays more than one block.

{title}

The title of the block.

{content}

The content of a block.

Footer

The Footer Section

The footer section appears at the very bottom of each page, it's content can be specified by the Administrator (admin/settings). The section begins and ends with this code:

```
<!-- BEGIN: footer -->
<!-- END: footer -->
```

Message

This area holds the mark-up around the message posted by the Administrator. The section begins and ends with this code:

```
<!-- BEGIN: message -->
<!-- END: message -->
```

{footer_message}

Displays the actual content defined through the field "Footer message" in the "Settings" Administration page (admin/settings).

{footer}

Outputs footer messages generated by Drupal modules. (i.e. performance statistics from devel.module)

Editing with Golive

Set Up

To edit xTemplate template files (xtemplate.xtmpl) in Adobe GoLive, follow these simple steps:

1. In the GoLive menu select "GoLive" then "Web Settings"
2. The Web Settings window will appear, click on the "File Mappings" tag.
3. In the File Mappings window open the "text/" directory
4. Scroll down until you see "html" in the Suffix column.
5. Click on "html" to select it, then click on the "+" button to create a duplicate.
6. Change the suffix of the duplicate html to "xtmpl"
7. That's it you're done!

Editing

If when opening a template file GoLive asks you which encoding to use, select "UTF-8".

If all you see after opening a template is "body onload-attributes", go into source mode and delete "{onload_attributes}" from:

```
<body{onload_attributes}>
```

Remember to add "{onload-attributes}" back once you are finished editing.

In xtemplate.xtmpl, you may wish to add the following line temporarily:

```
<link type="text/css" rel="stylesheet" href="style.css" />
```

Remember to remove this line when completing work on the template, however. If you do not, Drupal will not be able to switch between various styles for your theme. Drupal will automatically load your style.css, if one exists, in the {styles} tag.

Plain PHP themes

PHP themes are the most direct way of themeing Drupal. A PHP theme consists of overrides for Drupal's built-in theme functions. You will most likely only override the basic theme hooks (pages, nodes, blocks, ...), but you can theme anything from lists to links if you desire.

To create a PHP theme, create a directory in your `themes` directory (we will assume `themes/mytheme` in this document), and inside that directory create a `mytheme.theme` file. This file is a regular PHP file, so make sure it contains <?php ?> tags.

The default theme functions in Drupal are all named `theme_something()` or `theme_module_something()`, thus allowing any module to add themeable parts to the default set provided by Drupal. Some of the basic theme functions include: `theme_error()` and `theme_table()` which as their name suggests, return HTML code for an error message and a table respectively. Theme functions defined by modules include `theme_forum_display()` and `theme_node_list()`.

In your .theme file, you can override any of these functions. To override the function `theme_something()`, define the function `mytheme_something()` in your .theme file. This function should have the same definition as the original. It is easiest to start with Drupal's function, and apply your changes there: many theme functions contain code logic within them. To avoid problems when upgrading Drupal in the future, it is best to mark the changes between the original Drupal function and your customized version. That way, you can reapply to your customizations if the original was changed.

Aside from theme functions, there is one function that you need to include, called `mytheme_features()`. This function should return an array of strings, marking the features your theme supports (e.g. search box, logo, mission statement, ...). The theme system will provide toggles and settings for these features in the administration section. In your code, you can retrieve the value of these settings though `theme_get_setting()`. If you are planning on releasing your theme to the public, it is advised to implement all Drupal features, so others can customize your theme.

Available features are:

logo	A logo can be used. The theme should check the settings `default_logo` (boolean) and `logo_path` (string).
toggle_logo	The logo can be turned on/off
toggle_name	The site name can be turned on/off
toggle_search	The search box can be turned on/off
toggle_slogan	The site slogan can be turned on/off
toggle_mission	The mission statement can be turned on/off
toggle_primary_links	The primary navigation bar can be turned on/off/
toggle_secondary_links	The secondary navigation bar can be turned on/off
toggle_node_user_picture	The theme can optionally display user pictures next to nodes
toggle_comment_user_picture	The theme can optionally display user pictures next to comments

Here's the `_features()` function from the standard `chameleon.theme`:

```
<?php
function chameleon_features() {
  return array(
       'logo',
       'toggle_name',
       'toggle_slogan',
       'toggle_primary_links',
       'toggle_secondary_links');
}
?>
```

Note that unlike templates and styles, themes are tied to their directory name. If you want to clone a PHP theme, you need to rename its directory, its .theme file and its functions inside the .theme file.

Theme coding conventions

This theme coding style guide is based on the cvs log message of a developer sick of fixing strange spacing and indentation.

Theme authors should take care to consistently treat spacing and indentation in their code. Just as we have rules for indenting code - because this makes it easier to understand and maintain - there are basic rules for themes and included HTML files:

- Each level of indentation adds 2 spaces
- Match the indentation of (long) opening and closing block html tags
- Distinguish between PHP and HTML indentation. Not

```
  function header($title = "") {
    <?PHP
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
    <html>
    ...
```

  but

```
  function header($title = "") {
<?PHP
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
...
```

This not only saves the superfluous leading spaces, but also makes it much easier to find matching opening and closing tags defined in functions with different indentation.

- Prefer PHP in HTML to HTML in PHP.

  Example: not

```
<?php
  function node($node, $main = 0) {
    print "\n<!-- node: \$node->title\ -->\n";
    print "<div class=\"nodetitle\">$node->title</div>";
           print   "<div   class=\"nodebody\"><span   class=\nodedate\>"   .
$this->links(
      array(format_name($node), format_date($node->created,
"small"), "&amp;nbsp;")
    ) . "</span>";
?>
```

  but

```
  function node($node, $main = 0) {
<?PHP
      <!-- node: "<?php print $node->title; ?>" -->;
      <div class="nodetitle"><?php print $node->title; ?></div>
      <div class="nodebody">
      <span class="nodedate"><?php print $this->links(
array(format_name($node), format_date($node->created, "small"),
"&amp;nbsp;") ); ?></span>
```

  After all, PHP is a HTML embedded scripting language - and not the other way around.

Updating your themes

As Drupal develops with each release it becomes necessary to update themes to take advantage of new features and stay functional with Drupal's theme system.

Converting 3.0 themes to 4.0

Required changes

Changes in class definition

Theme class definition uses now a different syntax:
Instead

```
class Theme extends BaseTheme {
```

you should use

```
class Theme_themename extends BaseTheme {
```

where *themename* is name of your theme in lowercase.

Changes in function header()

- Function header() takes now an optional parameter $title.
  Instead

  ```
  function header() {
  ```

  you should use

  ```
  function header($title = "") {
  ```

- Previously all pages in Drupal site had the fixed page title: *sitename - site slogan*. Now the page title can be dynamic - for example when displaying single node, the page title can be *note title - sitename*. So, instead

  ```
  print variable_get("site_name", "drupal") ." - ". variable_get("site_slogan", "");
  ```

  you should use a more complex syntax:

  ```
  if ($title) {
    print $title ." - ". variable_get("site_name", "drupal");
  }
  else {
    print variable_get("site_name", "drupal") ." - ". variable_get("site_slogan", "");
  }
  ```

  of if you want to use compact version of the same construction:

  ```
  print $title ? $title." - ". variable_get("site_name", "drupal") :
variable_get("site_name", "drupal") ."  ". variable_get("site_slogan", "");
  ```

  This piece of code checks if $title is present. If yes, it outputs $title and site name, if not, it outputs site name and slogan.

- If you used theme_account() function (what outputs login/membership box) in header(), please remove it. Login box placement is controlled in *Administration* > *blocks* page from now on and theme_account() is no longer used.

Changes in function node()

- format_name() accepts now parameter $node, not $node->name. Also $node->timestamp is replaced with $node->created. So, instead

```
print strtr(t("Submitted by %a on %b"), array("%a" => format_name($node->name), "%b" => format_date($node->timestamp)));
```

you should use

```
print strtr(t("Submitted by %a on %b"), array("%a" => format_name($node), "%b" => format_date($node->created)));
```

- node_index() is no longer used because Drupal 4.0 has more sophisticated classification system than Drupal 3.0 meta tags. So instead plain simple

```
print node_index($node);
```

you have to use

```
$terms = array();

if (function_exists("taxonomy_node_get_terms")) {
  foreach (taxonomy_node_get_terms($node->nid) as $term) {
    $terms[] = l($term->name, array("or" => $term->tid), "index");
  }
}

print $this->links($terms);
```

- Function link_node() accepts an optional parameter $main. Instead

```
if ($main) {
  print $this->links(link_node($node));
}
```

you should use

```
if ($links = link_node($node, $main)) {
  print $this->links($links);
}
```

Changes in function comment()

- format_name() accepts now parameter $comment, not $comment->name. Instead

```
print strtr(t("Submitted by %a on %b"), array("%a" => format_name($comment->name), "%b" => format_date($comment->timestamp)));
```

you should use

```
print strtr(t("Submitted by %a on %b"), array("%a" => format_name($comment), "%b" => format_date($comment->timestamp)));
```

Changes in function footer()

- If you used theme_account() function (what outputs login/membership box) in footer() function, please remove it. Login box placement is controlled in *Administration* > *blocks* page from now on and theme_account() is no longer used.

Optional changes

New function: system()

- Optionally theme can have a system() function what provides info about theme and its author:

```
function system($field) {
    $system["name"] = "theme name";
    $system["author"] = "author name";
    $system["description"] = "description of the theme";
    return $system[$field];
  }
```

Converting 4.0 themes to 4.1

Required changes

There is no required changes, all Drupal 4.0 themes should also work in Drupal 4.1

Optional changes

theme_head

Insert a function `theme_head()` inside your theme, right after the HTML's `<head>` tag:

```
<html>
  <head>
  <?php print theme_head(); ?>
  ...
```

This change allows modules to incorporate custom markup inside `<head>` `</head>` tags such as Javascript, `<meta>` tags, CSS and more.

Converting 4.1 themes to 4.2

Required changes

Add a theme_onload_attribute() to a <body> tag:

```
<body <?php print theme_onload_attribute(); ?> >
```

Optional changes

Take advantage of settings() hook

Themes can now populate settings to adminstration pages using the function themename_settings(). Example:

```
function mytheme_settings() {
  $output  = form_select("Sidebar placement", "mytheme_sidebar",
             variable_get("mytheme_sidebar", "right"),
             array(
             "none" => t("No sidebars"),
             "left" => t("Sidebar on the left"),
             "right" => t("Sidebar on the right"));
}
```

Direct you site logo to index.php

If you theme has the logo and you have made it to link <a href=""> or even <a href="<?php print path_uri();>"> then please replace these instances with a simple <a href="index.php" alt="">

One additional change may be needed. Using a custom theme adapted from a generic one, the original node function has the following code:

```
<?php
$terms = array();
if (function_exists("taxonomy_node_get_terms")) {
if ($terms = taxonomy_node_get_terms($node->nid)) {
$taxlinks = array();
foreach ($terms as $term) {
$taxlinks[] = l($term->name, array("or" => $term->tid), "index");
}
$taxo = $this->links($taxlinks);
}
}
?>
```

Which gaves an error on the index page after upgrading from 4.1 to 4.2 and contains invalid URLs. Replacing the above code with this fixes the error.

```
<?php
$terms = array();
if (module_exist("taxonomy")) {
$terms = taxonomy_link("taxonomy terms", $node);
}
$taxo = $this->links($terms);
?>
```

Converting 4.2 themes to 4.3

No changes are required :)

A few more CSS classes are available to you if you wish to use them. A non-exhaustive list is

- **read-more**: affects the formatting of the 'read more' link
- **cell-highlight:** affects the cell in the table header which is currently the sort key. this cell also has an image which you can override in your theme->image directory (most images are overridable in this way).

Converting 4.3 themes to 4.4

For more information on how the interaction between themes and modules has changed, see converting 4.3 modules to 4.4.

- The theme system is no longer built on PHP's object model. The BaseTheme class is no more and, as such, you no longer have to use a class for your theme. Instead, a theme is a collection of functions. This will make Drupal theme development feel much the same as Drupal module development. Prefix your theme function with your theme's name. Examples:
  `mytheme_page()`,`mytheme_comment()`,`mytheme_node()`.
- `mytheme::system()` (or in the new parlance, `mytheme_system()`) is no longer used. The theme description used on the theme administration page should instead be returned by a new function called
  `mytheme_help()`. This function follows the same semantics as the regular module `_help` hook:
  ```
  <?php
  function mytheme_help($section) {
    switch ($section) {
      case 'admin/system/themes#description':
        return t("A description of mytheme");
    }
  }
  ?>
  ```
- All theme functions now return their output instead of printing them to the user. There should be no print or echo statements in your theme.
- The `mytheme_header()` and `mytheme_footer()` functions and no longer used, a `mytheme_page()` function is introduced instead.
  ```
  <?php
  function mytheme_page($content, $title = NULL, $breadcrumb = NULL) {
    if (isset($title)) {
      drupal_set_title($title);
    }
    if (isset($breadcrumb)) {
      drupal_set_breadcrumb($breadcrumb);
  ```

```
  }
  ...
}
?>
```

This function should return the HTML code for the full page, including the header, footer and sidebars (if any). Note that it is important to set the title and the breadcrumbs for Drupal with the setter functions as suggested above, instead of just using the values provided as parameters. This way modules acting on the title or breadcrumb values can use the real value when generating blocks for example.

- Themes now have the responsibility of placing the title, breadcrumb trail, status messages, and help text for each page. This gives them the flexibility to, for example, place the breadcrumb trail above the title or in the footer. It is now expected that `mytheme_page()` will return these elements. The page theme function should override the title and breadcrumb trail retrieved from Drupal, in case some explicit value is provided in the function parameters (see above). A theme can obtain the values set before by calling the functions `drupal_get_title()`, `drupal_get_messages()`, `menu_get_active_help()`, and `drupal_get_breadcrumb()`. The breadcrumb trail is returned from the latter function as an array of links; it can be formatted into a string by using `theme("breadcrumb", drupal_get_breadcrumb())`. Most themes use the following new code-snippet in their `page` function:

```
<?php
if ($title = drupal_get_title()) {
  $output .= theme("breadcrumb", drupal_get_breadcrumb());
  $output .= "<h2>$title</h2>";
}
if ($help = menu_get_active_help()) {
  $output .= "<div class=\"help\">$help</div><hr />";
}
foreach (drupal_get_messages() as $message) {
  list($message, $type) = $message;
  $output .= "<strong>". t("Status") ."</strong>: $message<hr />";
}
?>
```

- The `_head()` hook is eliminated and replaced with the `drupal_set_html_head()` and `drupal_get_html_head()` functions, therefore the HTML head part should include the return value of `drupal_get_html_head()` instead of the return value of `theme("head")`.
- The `theme_node()` function takes an extra parameter now, `$page`, that indicates to the theme whether to display the node as a standalone page or not. If `$page` is true, then the title of the node should not be printed, as it will already have been printed by `theme_page`. Also note that the node body will only be filtered with the configured filters if the node page is displayed. Otherwise only the teaser will be filtered for performance reasons. Example:

```
<?php
function mytheme_node($node, $main = 0, $page = 0) {
  if (!$page) {
    $output = "<h2>" . $node->title . "</h2>";
```

```
  }
  if ($main && $node->teaser) {
    $output .= "<div>". $node->teaser . "</div>";
  }
  else {
    $output .= "<div>". $node->body . "</div>";
  }
  return $output;
}
?>
```

- To improve block themeability, `theme_block()` has been changed. The old

  ```
  function theme_block($subject, $content, $region = "main")
  ```

  has become

  ```
  function theme_block($block)
  ```

  with `$block` being an object containing `$block->subject`, `$block->content`, etc. See the doxygen doc for details and for how you can style blocks with CSS.

- Also, `theme_blocks()` has been improved to allow themes to hook into (change) the blocks before outputting them. See this cvs log message for details.

fooo bar

Converting 4.4 themes to 4.5

Note: the theme system changed significantly in 4.5. Make sure you read through this entire guide, as an outdated theme will prevent you from accessing vital parts of your site.

Directory structure

Templates are now seen as themes unto themselves, rather than hiding behind their template engine. Template engines now reside in subdirectories of `themes/engines`, while templates simply are placed in subdirectories of `themes`. Template engines compatible with Drupal 4.5 will identify templates based on their filename and send the appropriate listings to the theme system.

For xtemplate templates, your template must be named `xtemplate.xtmpl`, and your default stylesheet must be named `style.css` (as mentioned below in the "Styles" section).

For example, the old Xtemplate pushbutton template has moved from `themes/xtemplate/pushbutton` to `themes/pushbutton`.

Tabs (a.k.a. Local Tasks)

Drupal now separates out menu items that are "local tasks"; functions to be performed on the current location. By default, these are rendered as a set of tabs. Themes are responsible for printing these. A typical location is below the page title, so that

```
<?php
if ($title = drupal_get_title()) {
  $output .= theme("breadcrumb", drupal_get_breadcrumb());
  $output .= "<h2>$title</h2>";
}
if ($help = menu_get_active_help()) {
  $output .= "<small>$help</small><hr />";
}
?>
```

becomes

```
<?php
if ($title = drupal_get_title()) {
  $output .= theme("breadcrumb", drupal_get_breadcrumb());
  $output .= "<h2>$title</h2>";
}
if ($tabs = theme('menu_local_tasks')) {
  $output .= $tabs;
}
if ($help = menu_get_active_help()) {
  $output .= "<small>$help</small><hr />";
}
?>
```

For xtemplate templates, *Before:*

```
<!-- BEGIN: title -->
{breadcrumb}
<h1 class="title">{title}</h1>
<!-- END: title -->
```

After:

```
<!-- BEGIN: title -->
{breadcrumb}
<h1 class="title">{title}</h1>
<!-- BEGIN: tabs -->
<div class="tabs">{tabs}</div>
<!-- END: tabs -->
<!-- END: title -->
```

Status Messages

The `theme_page` function is no longer responsible for rendering each status message. Instead, we now use the `theme_status_messages()` function. *Before:*

```
<?php
foreach (drupal_get_messages() as $message) {
list($message, $type) = $message;
$output .= "<strong>". t("Status") ."</strong>: $message<hr />";
}
?>
```

After:

```
<?php
$output .= theme_status_messages();
?>
```

Static vs. Sticky

In Drupal 4.5, "static" posts have been renamed as "sticky" posts. If your theme uses special styling for this type of post, you'll want to change any references from "static" to "sticky".

Avatar vs. User Picture

In Drupal 4.5, "avatars" have been renamed to "user pictures". Additionally, the method by which themes display avatars has changed. Themes now call `theme_user_picture`, which returns the appropriate image and link HTML. *Before:*

```
<?php
if (module_exist("profile") && variable_get("theme_avatar_node", 0)) {
$avatar = $node->profile_avatar;
if (empty($avatar) || !file_exists($avatar)) {
$avatar = variable_get("theme_avatar_default", "");
}
else {
$avatar = file_create_url($avatar);
}
if ($avatar) {
$avatar = "<img src=\"$avatar\" alt=\"" . t("%user's avatar",
array("%user" => $node->name ? $node->name :
t(variable_get("anonymous", "Anonymous")))) . "\" />";
if ($node->uid) {
$avatar = l($avatar, "user/view/$node->uid", array("title" => t("View
user profile.")));
}
$output .= $avatar;
}
}
```

```
?>
```
After:
```
<?php
$output .= theme('user_picture', $node);
?>
```
For xtemplate templates, simply replace:

```
<!-- BEGIN: avatar -->
<div class="avatar">{avatar}</div>
<!-- END: avatar -->
```

with:

```
<!-- BEGIN: picture -->
{picture}
<!-- END: picture -->
```

Theme Screenshots

The new theme selector looks for a screenshot of each theme with the filename `screenshot.png` in each directory. Screenshots are optional and themes without screenshots will simply display "no screenshot" on theme selection pages. To create a screenshot which matches those in core, follow these instructions:

1. Log in as administrator user.
2. Enable the following modules, for some extra menu items:
   aggregator, blog, node, page, story, tracker
3. Create the following story node:
   - **title:** Donec felis eros, blandit non.
   - **body:** Morbi id lacus. Etiam malesuada diam ut libero. Sed blandit, justo nec euismod laoreet, nunc nulla iaculis elit, vitae. Donec dolor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus vestibulum felis nec libero. Duis lobortis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nunc venenatis pretium magna. Donec dictum ultrices massa. Donec vestibulum porttitor purus. Mauris nibh ligula, porta non, porttitor sed, fermentum id, dolor. Donec eu lectus et elit porttitor rutrum. Aenean justo. Phasellus augue tortor, mattis nonummy, aliquam euismod, cursus eget, ipsum. Sed ultricies bibendum ante. Maecenas rhoncus tincidunt eros.
4. Look at the node, and make sure the tabs are visible. Take a screenshot.
5. Cut out a piece about 420x254 resized to 150x90 (35% zoom). Try to show useful page elements (menu, tabs, title, links).
6. Applied a plain 'sharpen' filter to the thumbnail.
7. Save as "screenshot.png" in theme (or style) directory.

Centralized Theme Configuration

The theme system now has the ability to store certain common configuration items for each theme. However, some themes may not wish to utilize all of these settings, so a `theme_features` hook has been introduced. In each theme / theme engine, this function should return an array of settings which the theme supports. To implement each of these functions, themes / theme engines should call the `theme_get_setting` function, which will return data regarding the administrator's setting for this particluar theme. If there are no settings for the current theme, global values will be returned. Below is a table of values for the `_features` hook, a description of their function, and a code snippet of the appropriate theme_get_settings call.

_features hook value	**Description**	**theme_get_settings call**

'logo'	theme allows customization of site logo	`<?php`
		`if ($logo = theme_get_setting('logo')) {`
		`$output .= " <a href=\"./\" title=\"Home\"><img src=\"$logo\" alt=\"Home\" /></a>";`
		`}`
		`?>`

'toggle_name'	theme allows site name to be switched on/off	`<?php`
		`if (theme_get_setting('toggle_name')) {`
		`$output .= " <h1 class=\"site-name title\">".`
		`l(variable_get('site_name', 'drupal'), "").`
		`"</h1>";`
		`}`
		`?>`

'toggle_search'	theme allows search box to be switched on/off	`<?php`
		`if (theme_get_setting('toggle_search')) {`
		`$output .= search_form();`
		`}`
		`?>`

'toggle_slogan'	theme allows site slogan to be switched on/off	`<?php`
		`if (theme_get_setting('toggle_slogan')) {`
		`$output .= " <div class=\"site-slogan\">".`
		`variable_get('site_slogan', '') ."</div>";`
		`}`
		`?>`

'toggle_mission'	theme allows site mission to be switched on/off	`<?php`
		`if ($mission = theme_get_setting('mission')) {`
		`$output .= $mission;`
		`}`
		`?>`

'toggle_primary_links'	theme allows primary links to be customized	`<?php`
		`$output .= theme_get_setting('primary_links');`
		`?>`

'toggle_secondary_links'	theme allows secondary links to be customized	`<?php $output .= theme_get_setting('secondary_links'); ?>`
'toggle_node_user_picture'	theme allows node user pictures to be switched on/off	`<?php if (theme_get_setting('toggle_node_user_picture') && $picture = theme('user_picture', $node)) { $output .= $picture; } ?>`
'toggle_comment_user_picture'	theme allows comment user pictures to be switched on/off	`<?php if (theme_get_setting('toggle_comment_user_picture') && $picture = theme('user_picture', $comment)) { $output .= $picture; } ?>`
N/A (Global Setting)	Allow admin to specify which node types should display "Submitted by..." message	`<?php $output .= theme_get_setting("toggle_node_info_$node->type") ? t("Submitted by %a on %b.", array("%a" => format_name($node), "%b" => format_date($node->created))) : ''; ?>`

Note that all of these settings are optional, but recommended.

Theme-specific settings are still possible as well. They are still read from the `theme_settings`, but are now placed in a group on the appropriate theme's tab, rather than on a separate page.

Styles

The theme system now allows for switching between different "styles" for each theme. Each "style" is defined by a `style.css` file in a subdirectory of the theme. In order to accomplish this style switching, themes should add a call to `theme_get_styles()` within their <head> block. For example:

```
<?php
$output .= drupal_get_html_head();
$output .= " &lt;link rel=\"stylesheet\" type=\"text/css\"
href=\"themes/chameleon/common.css\" /&gt;\n";
$output .= theme_get_styles();
$output .= "&lt;/head&gt;";
?>
```

Notice how the reference to `common.css` is listed before `theme_get_styles()`. This allows individual styles to override your common CSS rules (if you use any).

The "default" style for each theme (the stylesheet in which you define color scheme and other general presentation items) should be renamed to `style.css` and placed in your theme directory. You should also remove any references to it from your theme or template. Drupal will reference it in `theme_get_styles()`. (If the default style is selected)

For xtemplate themes, you need to add the `{styles}` tag add the end of your <head> section.

_help hook

The `theme_help` hook is no longer used. It can be removed if desired.

References to comment_referer_load() should be switched to comment_node_url()

Converting 4.5 themes to 4.6

Search form

If your theme implements a search form, it needs to be altered. The search box `<input>` tag should have the `name` attribute set to `edit[keys]` rather than `keys`.

Node links

Node links no longer use the `link_node()` function, but instead are passed as an array in `$node->links`. PHP-based themes will need to be updated to pass this array through `theme('links')`. Template-based themes shouldn't need any changes.

Pages

The function theme_page() no longer takes `$title` or `$breadcrumb` arguments. Remove the two arguments and any special handling of them. All page titles and breadcrumbs are now retieved using drupal_get_title() and drupal_get_breadcrumb().

Node and comment markers

Node and comment markers are not restricted anymore to signal that something is new or that a form element is required. The required form element marker was moved to `theme_form_element()`, while `theme_mark()` was kept to generate content markers. New constants help in deciding on the marker to display: `MARK_NEW` signals new content, `MARK_UPDATED` is for changed or extended content and `MARK_READ` is for read or too old content. Now it is possible to output markers for read content too, and distinguish between new and updated content.

Pager and menu item themeing

Parts of the pager are now themeable themselfs. The menu theming was also reorganized, to be easier to add wrappers and theme menu links. If you override any of the theme_menu...() functions in your theme or template, compare them to the current versions in theme.inc and menu.inc to update them.

Text validation changes

Due to some changes in plain-text processing, some parameters which were HTML are now plain-text and vice-versa. If you use a theme engine, you shouldn't need any changes, except if you override extra theme_ functions yourself.

If you are seeing problems, the best approach is to compare every theme_ functions that you override with the one from Drupal core. In particular, the menu theme functions (theme_menu_*) require changes in the way l() is used.

You should also try submitting a node and a comment with HTML tags in the subject. The tags should come out escaped, and should be shown on screen rather than interpreted. If this is not the case, you need to check your theme_node() and theme_comment() functions.

Finally, page titles should be run through strip_tags() when put into the html <title> tag in <head>. This should only be a problem if you have a .theme theme, as the theme engines have all be updated to accomodate this change.

Converting 4.6 themes to HEAD

Table row coloring

The class names for alternating table rows have been changed from `light` and `dark` to `odd` and `even`.

Theme screenshot guidelines

Every theme for 4.5+ needs a screenshot in the form of a screenshot.png placed in the theme/template/style directory. It is best that screenshots are consistent. The guidelines for core theme screenshots are (starting from a blank Drupal site):

1. Log in as the first user.
2. Enable the following modules, for some extra menu items: *aggregator*, *blog*, *node*, *page*, *search*, *story* and *tracker*.
3. Turn on the features that the theme supports (logo, site name, slogan, search box). Add some primary and secondary links if needed. We suggest "Link 1" "Link 2" "Link 3", you can link them to e.g. "user/1".
4. Set the site name to *Drupal* and slogan to *Community Plumbing*.
5. Create the following story node:

Donec felis eros, blandit non

Morbi id lacus. Etiam malesuada diam ut libero. Sed blandit, justo nec euismod laoreet, nunc nulla iaculis elit, vitae. Donec dolor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus vestibulum felis nec libero. Duis lobortis. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nunc venenatis pretium magna. Donec dictum ultrices massa. Donec vestibulum porttitor purus. Mauris nibh ligula, porta non, porttitor sed, fermentum id, dolor. Donec eu lectus et elit porttitor rutrum. Aenean justo. Phasellus augue tortor, mattis nonummy, aliquam euismod, cursus eget, ipsum. Sed ultricies bibendum ante. Maecenas rhoncus tincidunt eros.

6. Look at the node, and make sure the tabs are visible. Take a screenshot.
7. Cut out a piece about ~420x254 resized to exactly 150x90 (~35% of the original size). Try to show only useful page elements (menu, tabs, title, links). Don't include browser chrome (toolbar, status bar, scrollbar, etc).
8. Apply a standard 'sharpen' filter to the thumbnail for clarity.
9. Save as a PNG, in paletted colorspace to cut down on size.

Example:

For Drupal.org project thumbnails, use the guidelines above except that:

- You should fill up the site more. For example, add a comment to the story node or add some blocks.
- The screenshot should show the entire page, though still without browser chrome (toolbar, status bar, scrollbar, etc).
- The thumbnail should be 320x200 pixels large. It is best to resize to a width of 320 pixels first, then to crop off the bottom to a height of 200 pixels.
- Try to make your original screenshot have a width of about 1000 pixels, so that the thumbnail is about 30% of the original size.
- Name the screenshot `screenshot-drupal.org.png` or `screenshot-drupal.org.jpg`.
- Try to keep the image small: save as paletted PNG or as a JPEG with 10-20% compression. This reduces the load time of the theme list on Drupal.org. Aim for 15-20KB.
- Include the image in your project's description with class picture:
  `<img src="http://cvs.drupal.org/...../screenshot-drupal.org.png" class="picture" />`
  If you don't have permission to use the `<img>` tag, ask a site maintainer to add your screenshot.
- If your theme is available on the Drupal Theme Garden, turn the thumbnail into link to it:

```
<a href="http://themes.drupal.org/?theme=yourtheme"><img ...></a>
```

Example:

Theme how-to's

This section collects 'How-to' articles on subjects relevant to theme developers.

Tips for designing themes in Dreamweaver, GoLive etc.

- An Xtemplate-in-GoLive how-to is available at http://drupal.org/node/6634

Dreamweaver

Dreamweaver can edit your PHP, template and CSS files just fine, and in some cases (XTemplate, PHPTAL) with a graphical preview.

Find the Dreamweaver configuration files, `MMDocumentTypes.xml` and `Extensions.txt`. where they are depends on what platform is being used - the Find file function is your friend here.

In Extensions.txt

Change the line ending in

`: All Documents`
to include
`TAL,XTMPL,MODULE,THEME,ENGINE`
Likewise, add `TAL,XTMPL` to the line ending in `: HTML files`
and
`MODULE,THEME,ENGINE` to the `: PHP files` line.

In MMDocumentTypes.xml

you likewise want to add tal and xtmpl file types to the file type descriptions, e.g. change the line beginging `documenttype id="HTML"` to be

```
<documenttype id="HTML" internaltype="HTML"
winfileextension="htm,html,shtml,shtm,stm,tpl,lasso,xhtml,tal,xtmpl"
macfileextension="htm,html,shtml,shtm,tpl,lasso,xhtml,tal,xtmpl"
file="Default.html" writebyteordermark="false">
```

and likewise add .theme, .module and .engine to the appropriate section

```
<documenttype id="PHP_MySQL" servermodel="PHP MySQL"
internaltype="Dynamic"
winfileextension="php,php3,php4,theme,module,engine"
macfileextension="php,php3,php4,theme,module,engine"  file="Default.php"
writebyteordermark="false">
```

The local copy of your site can now be edited in Dreamweaver.

Adding your theme to Drupal.org

To add your theme to Drupal.org, it must be GPL. Do not include images or other copyrighted works that you do not want to see re-used or otherwise altered.

Themes are tracked the same way that code is, in the CVS repository. You will need to apply for a CVS account. Once you are approved, you will be able to check your theme into the Drupal CVS repository. Create a project and the download will be created for it automatically.

If you do add your theme, users will likely post suggestions, file bugs, and generally desire that you keep the theme up to date with current versions of Drupal.

Theme snippets repository

Did you write a nice custom template function? Please add a book page below this one. Please remember to document your code well.

Custom login

This creates a little custom login area. If you are logged in, it displays your username and a link to your profile, otherwise it includes a Register and minimal Login area. It was developed for OurMedia.

```
<?php if ($user->uid) : ?>
  Logged in as: <?php print l($user->name,'user/'.$user->uid); ?> |
<?php print l("logout","logout"); ?>
  <?php else : ?>
  <?php print l("Register","user/register"); ?> | Login: <form
```

```
action="user/login" method="post"><input type="hidden"
name="edit[destination]" value="user" /><input type="text"
maxlength="64" class="form-text" name="edit[name]" id="edit-name"
size="15" value="" /><input type="password" class="form-password"
maxlength="64" name="edit[pass]" id="edit-pass" size="15" value=""
/><input type="submit" name="op" value="Log in" /></form>
  <?php endif; ?>
```

Temporarily on the dev wiki until someone can get it posted here.

Customize display of submission information based on node type

You can display different author information and still respect the global "display post information" settings. In this example I wanted flexinode-2 items to be displayed with the usual submission info, but have every other type respect the "display post information" setting. Note that the content type I created, "news", goes by its flexinode name, and not by what I named it. Here is an example snippet for a node.tpl.php:

```
<?php   if (theme_get_setting('toggle_node_info_' . $node->type)) : ?>
<?php   if ($node->type == 'flexinode-2'):?>
<div class="info"><?php print $submitted; ?></div>
<?php elseif($node->type): ?>
<div class="info"> by <?php print $name; ?></div>
<?php endif; ?>
<?php endif; ?>
```

For my settings this displays:

NEWS NODE TITLE HERE
submitted by me on Tue, 04/26/2005 - 10:55am.

STORY NODE TITLE HERE
by me

PAGE NODE TITLE HERE

because I have set news and story nodes to display the submitted info while page is set to display no submission info at all.

get an contextual array for your node-links

Often I am very annoyed by the fact the links under the nodes are only availble as plain text strings. For example when I only wnat the read more link, or only the "comments" links. (example of usage)

This PHPtemplate-specific snipped creates an additional variable: an array with much nicer link data.

```
function _phptemplate_variables($hook, $vars) {
  switch ($hook) {
    case 'node':
      foreach ($vars[node]->links as $link) {
       preg_match("/<a\s*.*?href\s*=\s*['\"]([^\"'>]*).*?>(.*?)<\/a>/i",$link,
$matches);
        $vars[nodelinks][]['url'] = $matches[1];
        $vars[nodelinks][]['text'] = $matches[2];
      }
    break;
  }
  return $vars;
}
```

Please add improved regular expressions for the decomposition of anchors in this post directly, or in a comment.

How to display mission on every page?

This seems an easy one, but I can't figure out how to display the site mission on every page, it only shows in the home page... I am using a self-made php template, and I have the following line in page.tpl.php:
Hope you can help me, TIA!

Make images square

Often images are not square -landscape or portrait- which causes rendering problems or just ugly pages. Tables might have different sized rows or columns, inline images look differnt in each post, or rows of images appear horribly cluttered and inconsistent.
This function will add padding to your images, to make them appear in a square. But remember that this will override any padding applied to img in your stylesheets.

```
<?php
function themename_image($path, $alt = '', $title = '', $attr = '',
$getsize = true) {
  //always do getimagesize
  list($width, $height, $type, $attr) = @getimagesize($path);
  //get the biggest value.
  if ($width > $height) {
    $padding = round(($width - $height)/2);
    $style_str = ' style="padding:'.$padding.'px 0;"';
  }
  elseif ($width < $height) {
    $padding = round(($height - $width)/2);
```

```
    $style_str = ' style="padding:0 '.$padding.'px;"';
  }
    return "<img src=\"$path\" $attr alt=\"$alt\" title=\"$title\"
$style_str />";
}
?>
```

Overriding drupal.css; two approaches

There are two methods to removing drupal.css from your theme in phptemplate.

The first cuts out the link from the $head variable. In page.tpl.php, replace your

```
<?php
print $head
?>
```

with (remove the space in 'style'):

```
<?php
print str_replace('<st yle type="text/css" media="all">@import
"misc/drupal.css";</style>', '', $head);
?>
```

The other method requires overriding the stylesheet import themable function. Simply add this to your theme's template.php file:

```
<?php
function phptemplate_stylesheet_import($stylesheet, $media = 'all') {
  if ($stylesheet != 'drupal/misc.css') return
theme_stylesheet_import($stylesheet, $media);
}
?>
```

Drupal.org site maintainers

Below is an alphabetical list of users who have additional permissions to help maintain the drupal.org website:

1. adrian
2. ahoppin
3. aldon@deanspace.org
4. Amazon
5. andremolnar
6. ax
7. BÃ¨r Kessels
8. bertboerland@ww...
9. blogdiva@www.cu...
10. Boris Mann
11. bryan kennedy
12. cel4145
13. chx
14. Development Seed
15. dggreenberg
16. Dries
17. DriesK
18. drumm
19. Dublin Drupaller
20. ericgundersen
21. FactoryJoe@civi...
22. Gerhard Killesreiter
23. Goba
24. JonBob
25. Junyor
26. jvandyk
27. kbahey
28. Kieran Huggins
29. kika
30. killes@www.drop.org
31. Kjartan
32. Kobus
33. mathias
34. Morbus Iff
35. moshe weitzman
36. nysus
37. puregin
38. Richard Eriksson

39. rivena
40. Robert Castelo
41. robertDouglass
42. Robin Monks
43. Roland Tanglao@...
44. sepeck
45. Steven
46. TDobes
47. Uwe Hermann
48. walkah
49. wnorrix

If you have been around for a while, and you want to help maintain Drupal.org, get in touch with Dries.

Site maintainer's guide

This page lists some guidelines for Site Maintainers on Drupal.org.

Unpublishing vs deleting of content

You should only unpublish a post if you can conceivably imagine it being re-published in the future. This should only be for very rare cases. A good example is an unmaintained project (someone might take over development later): unpublish, don't delete (*). On the other hand, spam can be deleted immediately.

(*) Actually, old projects are automatically unpublished by the cvs scripts, so this is not something you need to do.

Blocking vs deleting of users

Deleting users is a very destructive action, as it makes all their content inaccessible in most places, even to administrators. It should not be done. If a user is a troublemaker, just block their account (click username -> edit -> status: blocked). Of course, you should not block people just because they say unfavorable things about Drupal. Here are good reasons to block someone:

- Spamming (even once)
- Repetitive flaming
- Repetitive posting of trash content (test posts, inappropriate book pages, ...)

Suggested Workflow

When you spot something out of the ordinary, we suggest these steps:

1. Take a look at the user's post history on the "user -> track -> track posts" page. This will possibily show more bad posts by the same person.
2. Take a look at the user's page visit history on the "user -> track -> track page visits" page. That way, you can easily tell if a user just registered to spam or if they made only one bad post in a series of good ones.
3. If the content is spam, you can delete it immediately and block the person's account. Otherwise, send them a note through their contact tab about it:

   > Your post Foobar on http://drupal.org/node/1234 was inappropriate because it contained flaming. Please be nice to your fellow visitors on Drupal.org, or your account may be blocked.

4. If you know someone to be a troublemaker who has been warned before, block their account.

Badly formatted posts

If you see a post with bad formatting which messes up the page's layout, please edit it. A common mistake for newbies is to use two opening tags rather than an opening/closing pair. Tags like bold and italic can 'bleed through' beyond the post, while unclosed block-level tags can mess up the positioning of the sidebar.

If someone made a serious mistake while posting a forum topic and posted a correction in a comment below, try to update the original post and delete the correction.

Translator's guide

This is the Drupal translator's guide. It will cover most aspects of translating Drupal's user interface. It will not cover the use of the various programs that can be used to do a translation. These programs are usually quite well documented.

As of version 4.5.0, Drupal includes an extended locale.module that enables you to share translations through the use of PO files. PO files are files containing translations as used by the GNU gettext program.

User contributed PO files for various languages can be found on the download page.

If your language is not present, you might want to start a translation yourself. If this is the case, please download the Drupal POT translation templates. You can get a PO file editor and start translating.

You should translate the individual PO files (per module) rather than one big file. The individual files are automatically packaged into one large file per language in the CVS repository, which is what others will download from this site.

Once you have completed a reasonable part of the translation, create an issue on the *Translation templates* project and upload your PO files. Some helpful developer will then come by and put them in CVS for you. If you have write access to the contrib CVS you can commit your files yourself. In any case a project for your translation will be created, you will be made the maintainer, and your translation becomes available on the download page

Translation templates

Translators should start by downloading the tarball and translating the files to their language of choice.

The translated files should be stored in contrib-cvs/translations/*id* where *id* is the ISO 639 language code. If you don't know your code, ask in drupal-devel.

You should only put the individual translated files in this directory. A script will generate a merged *id*.po file. Make sure to fill out the header section of each file and rename them from .pot to .po.

If you do not have a CVS account, create an issue for this project and attach your files to it.

Note that the Drupal team will not check contributed translations for accuracy or errors.

Programs to use for translation

Recommended PO file editors are (in no particular order):

- XEmacs (with po-mode): runs on Unices with X
- GNU Emacs (with po-mode): runs on Unices
- KBabel: runs on KDE
- poEdit: linux and windows
  poEdit does support multiple plural forms since version 1.3.

For Mac OS X there is AquaEmacs and a port of GNU Emacs available using carbon for OS X:
http://www.apple.com/downloads/macosx/unix_open_source/carbonemacspackage.html
also see the Emacs wiki for more usage help and tips:
http://www.emacswiki.org/cgi-bin/emacs-en/CarbonEmacsPackage

po-mode is not included, but is easy to add. get it from the GNU gettext distribution.

Be sure to get a recent version for all editors, multiple plural forms are a recent addition to the gettext standard.

Issues using poEdit

poEdit for windows, version 1.3.1 (latest at the moment) doesn't seem to recognize plural forms (if you try to edit a term which has plurals, even if you translate it, it doesn't appear in poedit when you move to an other term, as usual, and even if you save, it doesn't).

Plurals Solution #1

So, if you find a plural term, close poedit, open the file you were translating with a normal text editor (no, not Word...), and search for "plural" in it, you find something similar to this:

```
#: modules/comment.module:187 modules/node.module:89
msgid "1 comment"
msgid_plural "%count comments"
msgstr[0] "1 commento"
msgstr[1] "%count commenti"
```

simply tranlate the text in **msgid** (singular form) into **msgstr[0]**, and the text in **msgid_plural** (plural form) into **msgstr[1]**, save the file, close the editor and return to poedit. Even better, you can do this BEFORE start translating the rest of the file with poedit, translating every occurrance of plural in the same way, in every file, and THEN start using poedit: this way, you will find those strings already translated in poedit, and they don't bother you.

Plurals Solution #2

To use plurals in PO edit you can start with the catalog setting for english and then modify to suit. The syntax is:

nplurals=2; plural=(n != 1);

which gave me what I needed in Swedish translation of:

#: modules/aggregator.module:100;711;722
msgid "1 item"
msgid_plural "items"
msgstr[0] "1 inlÃ¤gg"
msgstr[1] "%count inlÃ¤gg"

I tested this in PO Edit 1.3.1 and got the proper GUI response and saved withut error.

Plurals Solution #3

The plural forms to use in PO edit under catalog-settings where you see

nplural=INTEGER; plural=EXPRESSION

Only one form:
Some languages only require one single form. There is no
distinction between the singular and plural form. An appropriate
header entry would look like this:

Plural-Forms: nplurals=1; plural=0;

Languages with this property include:

Finno-Ugric family
Hungarian

Asian family
Japanese, Korean

Turkic/Altaic family
Turkish

Two forms, singular used for one only
This is the form used in most existing programs since it is what
English is using. A header entry would look like this:

```
Plural-Forms: nplurals=2; plural=n != 1;
```

(Note: this uses the feature of C expressions that boolean expressions have to value zero or one.)

Languages with this property include:

Germanic family
Danish, Dutch, English, German, Norwegian, Swedish

Finno-Ugric family
Estonian, Finnish

Latin/Greek family
Greek

Semitic family
Hebrew

Romanic family
Italian, Portuguese, Spanish

Artificial
Esperanto

Two forms, singular used for zero and one
Exceptional case in the language family. The header entry would be:

```
Plural-Forms: nplurals=2; plural=n>1;
```

Languages with this property include:

Romanic family
French, Brazilian Portuguese

Three forms, special case for zero
The header entry would be:

```
Plural-Forms: nplurals=3; plural=n%10==1 && n%100!=11 ? 0 : n != 0 ? 1
: 2;
```

Languages with this property include:

Baltic family
Latvian

Three forms, special cases for one and two
The header entry would be:

```
Plural-Forms: nplurals=3; plural=n==1 ? 0 : n==2 ? 1 : 2;
```

Languages with this property include:

Celtic
Gaeilge (Irish)

Three forms, special case for numbers ending in 1[2-9]
The header entry would look like this:

```
Plural-Forms: nplurals=3; \
               plural=n%10==1 && n%100!=11 ? 0 : \
                      n%10>=2 && (n%100<10 || n%100>=20) ? 1 : 2;
```

Languages with this property include:

Baltic family
Lithuanian

Three forms, special cases for numbers ending in 1 and 2, 3, 4, except those ending in 1[1-4]
The header entry would look like this:

```
Plural-Forms: nplurals=3; \
               plural=n%10==1 && n%100!=11 ? 0 : \
                       n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1
: 2;
```

Languages with this property include:

Slavic family
Croatian, Czech, Russian, Slovak, Ukrainian

Three forms, special case for one and some numbers ending in 2, 3, or 4
The header entry would look like this:

Plural-Forms: nplurals=3; \

```
plural=n==1 ? 0 : \
                       n%10>=2 && n%10<=4 && (n%100<10 || n%100>=20) ? 1
: 2;
```

Languages with this property include:

Slavic family
Polish

Four forms, special case for one and all numbers ending in 02, 03, or 04
The header entry would look like this:

Plural-Forms: nplurals=4; \
`plural=n%100==1 ? 0 : n%100==2 ? 1 : n%100==3 || n%100==4 ? 2 : 3;`

Languages with this property include:

Slavic family
Slovenian

Long plural formulae: Those should not be broken into several lines in the header of the PO file. Drupal expects the formula to be on one line. One could consider this a bug.

I don't think that you can use line breaks in POedit either. The text is fixed to keep from breaking the site layout. But this:

nplurals=1; plural=ar;

produces an error. The plural form "ar" is not recognized.

Setting up XEmacs with po-mode on Windows

XEmacs has been supported on Windows for a long time and can be downloaded from here: http://www.xemacs.org/Download/win32. The po-mode is bundled with XEmacs (no need to get the GNU gettext distribution).

However, you need a MULE-enabled XEmacs binary to edit the UTF-8-encoded PO files and I could not find such a binary for Windows on www.xemacs.org. What I did is this:

1. Install XEmacs 21.4.13 using the Netinstall (http://www.xemacs.org/Download/win32/setup.exe)
2. Replace the files installed in the C:\Program Files\XEmacs\XEmacs-21.4.13 directory with those from http://www.suiyokai.org/tomonori/xemacs/xemacs-i586-pc-win32-21.4.13-mule.tar.gz (as the filename implies, this is a MULE-enabled XEmacs 21.4.13 binary for Windows)
3. Install the MULE packages in the C:\Program Files\XEmacs\mule-packages directory (these can be downloaded from ftp://ftp.xemacs.org/xemacs/packages/xemacs-all-mule-packages.tar.gz).
4. Set the environment variable EMACSPACKAGEPATH with this value:
   `C:\Program Files\XEmacs\site-packages;C:\Program Files\XEmacs\mule-packages;C:\Program Files\XEmacs\xemacs-packages`
5. To ensure automatic Unicode detection when opening files, add these lines to your init file (init.el):
   ```
   (require 'un-define)
   (set-coding-priority-list '(utf-8))
   (set-coding-category-system 'utf-8 'utf-8)
   ```
6. And finally, add this to automatically enable the po-mode:
   `(require 'po-mode)`

Yes, this is a bit complicated... Welcome to the wonderful world of XEmacs! :-) If you never used XEmacs before, prepare yourself for a steep learning curve.

BTW, your installation directory does not have to be C:\Program Files\XEmacs. I used this for simplicity in the above instructions.

Translated Drupal information

Some documentation about Drupal (outside of the Drupal interface itself) has also been translated into other languages. Links to such 3rd party translations of external Drupal documentation should live here.

African

This page is for the translation of Drupal Core into Afrikaans. The rest of this text will be in Afrikaans, as that is the purpose of this document.

Vir diegene wat betrokke wil raak by die vertaling:

Stuur 'n e-pos aan Kobus en spesifiseer waarmee jy betrokke sou wou raak. Kobus sal dan met jou in verbinding tree indien jy die nodige besonderhede verskaf het.

Om 'n fout met die vertaling te rapporteer:

Besoek asb. die foutrapporteringsblad

Vir enige ander verwante redes waaroor jy wil kontak:

Stuur 'n e-pos aan Kobus en spesifiseer die presiese rede vir u skakeling.

Russian

Translations and original documentation for Russian Drupal users (still very incomplete, but work is in progress):

- Read about Drupal features in Russian
- Drupal Administrator's Guide - a translation of the original English version
- Drupal Handbook - a translation of the original English version with some rewrites and additions

Visit drupal.ru/docs for a complete list of links to Russian documentation.

Join our efforts to translate Drupal docs into Russian!

Spanish

- Drupal: Manejador de Contenidos y Comunidad Virtual
- Drupal: CaracterÃ-sticas
- Drupal: Manuales

Translation guidelines

To achieve translations that are consistent throughout a whole Drupal site, certain guidelines need to be agreed upon by the translator community for a particular language.

Such guidelines should include a wordlist for words that occur in Drupal's strings. The Ankur Bangla Developer's Guide provides a good example of how this is done on a project unrelated to Drupal. It will be helpfull to not set up a new word lists, but re-use existing ones from an existing translation project.

Other areas which need guidelines will differ from language to language. Please add those guidelines as child pages to this book page.

Translation of contributed modules

Translatable strings from contributed modules are not included in the Drupal core POT files. Module authors can use the *extractor.php* script which comes with the core PO files to generate a POT file on their own. Instructions for running the script can be found in the *README* that comes with the core POT files. The generated POT file should be named as the module, but with a *.pot* extension, e.g. *event.module* gets an *event.pot* file. This file should be placed in a suddirectory *po*. Translations should be added to the same directory. E.g. the *po* subdirectory of event.module currently contains the following files: de.po, es.po, event.pot, he.po, hu.po.

Translators should take care to populate their started translation with the strings from the general.po file for their language using msgmerge. In this way they can avoid using different translations for terms that occur in both files.

Distributing the translation effort

To facilitate easier handling of a community translation effort, the Drupal POT file is split up into small files that do not contain doubly occurring strings.

All strings that occur more than once in the Drupal core distribution are put into the general.pot file. This ensures that those strings are translated to the same string. Also, files that have ten or less translatable strings will not get their own POT file, but those strings will be appended to the general.pot file.

Of course, some coordination among the project members is still needed to ensure the quality of the translation.

If a language has several options on how to translate some strings, then it is possible to create PO files that only change those strings. An example would be German where your can translate *you* either as *Du* or *Sie* depending on the audience of your site.

Status of the translations

The table below presents an overview of the status of each translation project. This page is updated daily by the package script: it was last updated 1 hour 22 min ago.

Status overview

Language	cvs	4.6.0	4.5.0
af	100% (complete)	100% (complete)	
ar	77% (236 missing)		77% (236 missing)
bg	88% (197 missing)		
bn	3% (1764 missing)		
ca	64% (409 missing)	96% (58 missing)	92% (130 missing)
cs	89% (190 missing)	89% (190 missing)	63% (564 missing)
da	100% (complete)	100% (complete)	100% (complete)
de	76% (380 missing)	76% (380 missing)	99% (5 missing)
eo			49% (116 missing)
es	100% (complete)	100% (complete)	100% (complete)
es-la	0% (256 missing)		
eu	67% (496 missing)		14% (1163 missing)
fi	98% (4 missing)		
fr	95% (90 missing)	100% (complete)	93% (97 missing)
gu	0% (1825 missing)		
hu	100% (complete)	100% (complete)	100% (complete)
id	96% (56 missing)		96% (55 missing)
it	94% (91 missing)	94% (91 missing)	100% (complete)

ja	100% (complete)	100% (complete)	100% (complete)
lt	67% (543 missing)		
mk	25% (1148 missing)		
nb	22% (943 missing)		
nl	76% (380 missing)	80% (320 missing)	84% (228 missing)
nno	26% (1015 missing)		28% (979 missing)
pl	17% (1384 missing)	60% (723 missing)	6% (1306 missing)
pt-br	100% (complete)	100% (complete)	23% (862 missing)
pt-pt	100% (complete)	100% (complete)	100% (complete)
ro	98% (19 missing)		98% (19 missing)
ru	71% (358 missing)	64% (597 missing)	88% (155 missing)
sk	9% (1436 missing)	9% (1417 missing)	
sq	39% (687 missing)		39% (687 missing)
sv	translation broken		translation broken
tl-ph	translation broken		
uk	100% (complete)		
zh-hans	translation broken	translation broken	62% (514 missing)
zh-hant	99% (1 missing)	99% (3 missing)	translation broken

Checking your translation status

To see how many of the strings in the PO files you already translated you can try this:

```
for i in *.po; do echo -n "$i: " ; msgfmt --statistics $i ; done
```

Some PO editors already include this feature.

Make a single file from the loose .po files from CVS

If you want to make a single po file from a CVS folder containing all the small po files, the following commands will do (*nix only). You should execute this, while being in the folder with the .po files.

```
$ msgcat --use-first general.po [^g]*.po | msgattrib --no-fuzzy -o
nl.po
```

Off course you should change nl into your own language code.

Recycling old translations

Drupal users with existing translations might want to add those to the translations download page. To do this they first need to export their translation from the localization *manage languages* screen (*export* subtab). Let us assume you have an Italian translation. The above mentioned process will create an it.po file for you. To use this file as a basis for a new translation, you treat it as a PO compendium, i.e. a library of pre-translated strings.

This guide assumes a Unix/Linux environment. If you use Windows, check if your PO editor doesn't have a function for this.

We will split the single, large PO file into the smaller files that the Drupal translation Project requires.

First, put the small PO files into a subdirectory *drupal-pot* and your it.po file into another one. Then create an empty directory where you want to keep your new small PO files.

Then go to the empty directory and execute the following command from the command line:

```
for i in /path/to/drupal-pot/*.pot ; do msgmerge --compendium
/path/to/it.po -o 'basename $i .pot'.po /dev/null $i ; done
```

After a while (yes this will take a few minutes) you should have a directory of small PO files that have the matching strings inserted.

Troubleshooting

When doing translations or importing them, several problems can occur. If you think you found a bug in either a translation or in Drupal's locale module, please file bug reports against the project in question.

If you have a more general question you can ask it in the translations forum.

Here we collect some of the more common issues found.

Weird characters or question marks

Symptom: After importing a translation you find all kind of weird characters or question marks on your site.

Solution 1: The translator did not use UTF-8. Drupal is fully UTF-8 aware and expects translations to be supplied in that character set as well. You can change the charset of a PO file using GNU msgconv. Please file a bug against the translation in question.

Solution 2: You do not have the correct font installed to display the language in question.

Drupal test suite

Drupal is currently lacking some test suite to be run by developers before submitting important patches. The following setup isn't really a test suite but it is a start to avoid the most embarrassing errors. A more complete solution would be unit tests as proposed by Moshe Weitzman. but they'd also be a lot more work.

Ok, here is what I will do in the future:

1. Enable the menu module and disable the 'log out' link.
2. Run
   ```
   wget --mirror --delete-after http://killes.drupaldevs.org/
   ```
   where killes.drupaldevs.org is my development site. You can add --wait=5 to the options if you don't want a free stress test.
3. If I want to test as an authenticated user I do
   ```
   wget --mirror --delete-after --load-cookies=/path/to/cookies.txt http://killes.drupaldevs.org/
   ```
   where /path/to/cookies.txt is the cookie inside my .mozilla directory.

Note that this can take some time. wget will access every Drupal page linked from the frontpage. You can later have a look at the error logs and find out if any errors where caused.

FAQ

This FAQ (Frequently Asked Questions) collects questions of interest to Drupal Contributors.

PHP Debugger

What do you folks use for debugging PHP? I'm getting $conf and header errors on my Drupal installation and would like to track them down, learning Drupal in the process. Is there a debugger for PHP where I can set breakpoints, see values change, etc.? Thanks for your help.