# Getting Started

As mentioned in the introduction, in this tutorial we're using the example of a module that lists links to content such as blog entries or forum discussions that were created recently, which we'll currently define as exactly one week ago. This page in the tutorial describes how to create the initial module file and directory.

The first step in creating a module is to choose a "short name" for it. This short name will be used in all file and function names in your module, so it must start with a letter and by Drupal convention it must contain only lower-case letters and underscores. For this example, we'll choose "onthisdate" as the short name.

Given this choice, start your module by creating a folder in your Drupal installation at the path: sites/all/modules/onthisdate. You may need to create the sites/all/modules directory first. Create a PHP file and save it as onthisdate.module in the directory sites/all/modules/onthisdate. As of Drupal 6.x, sites/all/modules is the preferred place for non-core modules (and sites/all/themes for non-core themes), since this places all site-specific files in the sites directory. This allows you to more easily update the core files and modules without erasing your customizations. Alternatively, if you have a multi-site Drupal installation and this module is for only one specific site, you can put it in sites/your-site-folder/modules.

Module files begin with [the opening PHP tag](http://us3.php.net/manual/en/language.basic-syntax.phpmode.php) followed immediately by a CVS ID tag in an inline comment. The CVS ID tag, $Id$, is a token which Drupal.org's version control system will fill out with revision information and authoring information. It is traditional to place this tag in your module regardless of whether you plan on putting it on Drupal.org or not. A similar ID tag is placed in all other files associated with your module.

<?php  
// $Id$

The ID tag is not a special type of PHP syntax and is there purely for version control systems. Thus, it **must** be in an inline comment. Again, this tag will be filled out by automated processes and you can ignore it from here on out.

The module is not operational yet: it hasn't been activated. We'll activate the module later in the tutorial.

**Coding Standards**

As per the [Coding standards](http://drupal.org/node/318), omit the closing ?> tag. Including the closing tag may cause strange runtime issues on certain server setups. (Note that the examples in the handbook will show the closing tag for formatting reasons only and you should not include it in your real code.)

All functions in your module that will be used by Drupal are named {modulename}\_{hook}, where "hook" is a pre-defined function name suffix. Drupal will call these functions to get specific data, so having these well-defined names means Drupal knows where to look. We will come to hooks in a while.

# Telling Drupal about your module

**Main topic described: Let Drupal know the module exists**  
**Drupal hook described: hook\_help**  
All modules must have a modulename.info file, which contains meta information about the module.

The general format is:

; $Id$  
name = Module Name  
description = A description of what your module does.  
core = 6.x

For this example, the file would be named "onthisdate.info". Without this file, the module **will not show up in the module listing**. For this example, it could contain the following:

; $Id$  
name = On this date  
description = A block module that lists links to content such as blog entries or forum discussions that were created one week ago.  
core = 6.x

Add the source above to a file named to onthisdate.info and save it into the module's directory at sites/all/modules/onthisdate.

**Info File Details**

**name (Required)**

The displayed name of your module. It should follow the Drupal 6 capitalization standard: only the first letter of the first word is capitalized ("Example module", not "example module" or "Example Module").  
name = On this date

**description (Required)**

A short, preferably one line description that will tell the administrator what this module does on the module administration page. Remember, overly long descriptions can make this page difficult to work with, so please try to be concise. This field is limited to 255 characters.

description = A block module that lists links to content such as blog entries or forum discussions that were created one week ago.

Note that special characters in this description must be substituted with the HTML entity values. For example, use description = This is my &quot;crazy@email.com&quot; email address instead of description = This is my "crazy@email.com" email address  
If the description has single quotes or apostrophes in it then you can simply put the entire string inside double quotes. For example, description = "Please don't use this unless you know what you are doing."

**core (Required)**

As of version 6.x, Drupal core will refuse to enable or run modules that aren't explicitly written for the right version of core. The .info file must specify which Drupal core compatiblity any module or theme has been ported to. This is accomplished by means of the new core attribute in the .info files.

core = 6.x

**Note:** the drupal.org packaging script automatically sets this value based on the Drupal core compatibility setting on each release node, so users downloading packaged releases from drupal.org will always get the right thing. However, for sites that deploy Drupal directly from CVS, it helps if this value is added to the .info files for the module. This is also a good way to indicate to users of each module what version of core the HEAD of CVS is compatibile with at any given time.

**dependencies (Optional)**

There are a couple of extra options that may appear in the .info file, one of which are module dependencies. If a module requires another module to be enabled, list each module (filename) required in the following syntax:

dependencies[] = taxonomy  
dependencies[] = comment

For the example module, these don't apply and we will simply omit them. If dependencies are assigned for a module, Drupal will not allow it to be activated until the required dependencies are met.

**package (Optional)**

If a module has a package string, on the admin/build/modules page it will be listed with other modules with the same category. If a package string is not assigned, it will simply be listed in the 'Other'. Not assigning a package for your module is perfectly ok; in general packages are best used for modules that are distributed together or are meant to be used together. If in doubt, leave this field blank.

package = "Your arbitrary grouping string"

Suggested examples of appropriate items for the package field:

* Audio
* Bot
* CCK
* Chat
* E-Commerce
* Event
* Feed Parser
* Organic groups
* Station
* Video
* Views
* Voting (if it uses/requires VotingAPI)

The files use the ini format and can include a ; $Id$ to have CVS insert the file ID information.

For more information on ini file formatting, see the [PHP.net parse\_ini\_file documentation](http://php.net/parse_ini_file).

**Help Hook**

We can also provide help and additional information about our module by implementing Drupal's hook\_help(). Because of the use of the .info file described above, this hook is now optional. However, it is a good idea to implement it. To implement any hook in Drupal, replace "hook" in the hook name with your module's short name, and create a function in the .module file with that name. So, to implement hook\_help() in our example module, we create a function called onthisdate\_help() in the onthisdate.module file:

<?php  
function onthisdate\_help($path, $arg) {  
  
}  
?>

The $path parameter provides context for the help: where in Drupal or the module the user is when they are accessing help. The recommended way to process this variable is with a switch statement. This code pattern is common in Drupal modules. Here is a sample implementation of this function:

<?php  
/\*\*  
\* Display help and module information  
\* @param path which path of the site we're displaying help  
\* @param arg array that holds the current path as would be returned from arg() function  
\* @return help text for the path  
\*/  
function onthisdate\_help($path, $arg) {  
  $output = '';  //declare your output variable  
  switch ($path) {  
    case "admin/help#onthisdate":  
      $output = '<p>'.  t("Displays links to nodes created on this date") .'</p>';  
      break;  
  }  
  return $output;  
} // function onthisdate\_help  
?>

(note the closing **?>** should not appear in your code)

The admin/help#modulename case is used by Drupal core to link from the main help page (/admin/help or ?q=admin/help). You will eventually want to add more text to provide a better help message to the user.

# Telling Drupal who can use your module

**Main topic described: *Permissions***  
**Drupal hook described: *hook\_perm***

The next function to write is the permissions function, implementing Drupal's hook\_perm(). This is where you will define the names of the permissions of your module -- it doesn't grant permission or define what the permissions mean, it just specifies what permissions are available for this module. As with the help hook from the previous tutorial page, we implement hook\_perm() by creating a function called onthisdate\_perm() in the onthisdate.module file. This function just needs to return a list of the permission names this module will use; once you have defined permissions in the hook\_perm() implementation in a module, an administrator can define which roles have those permissions on the **Administer » User management » Permissions** page. Here is the function for our example module:

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

(Note for beginners: the opening **<?php** will only appear once in your module file, at the beginning, and the closing **?>** will not appear in your file at all.)

If you are going to write a module that needs to have finer control over the permissions, you can expand this permission set. For example:

<?php  
return array('access onthisdate content', 'administer onthisdate');  
?>

For this tutorial, start with just the one permission. We'll later expand to using additional permissions.

Your permission strings are arbitrary, but each must be unique among all installed modules. Otherwise, one occurrence of the name will take the permissions of the other. For that reason, the permission strings should each usually contain your module name, since this helps avoid name space conflicts with other modules. The suggested naming convention for permissions is "action\_verb modulename". Use the following example as a template for any module you may be developing:

<?php  
function newmodule\_perm() {  
  
return array('access newmodule', 'create newmodule', 'administer newmodule');  
  
} // function newmodule\_perm  
?>

# Declare block content

Modules are created to do all sorts of things: some modules create blocks (abbreviated content that often appears on the right or left side of the page), others create special content types (for full page content - such as the content you are reading right now), others track back-end information, and some do all of the above. You may hear the phrases "Block modules" used to describe modules that primarily create block content (such as the menu module) or "Node modules" used to describe modules that primarily generate full page content (such as the blog and forum modules). At this stage, this module is a "block module", because it generates a block.

In this page of the tutorial the module will define a block that will eventually (after the next page of the tutorial) display the most recent blog and forum posts. The hook for creating blocks is appropriately called "hook\_block". To implement this hook create a function called onthisdate\_block() in the onthisdate.module file.

Here's the basic format:

<?php  
/\*\*  
\* Implementation of hook\_block  
\*/  
function onthisdate\_block($op = 'list', $delta = 0, $edit = array()) {    
  
  // YOUR MODULE CODE HERE  
  
} // end function onthisdate\_block  
?>

The block function takes three parameters:

* $op (or operation): hook\_block() functions get called to perform 4 different operations, which are "list", "view", "save", and "form". This parameter tells your function which operation is called for. We'll discuss the "list" operation below and the "view" operation on the next tutorial page. The "form" and "save" operations allow your block to have a settings form and save the settings -- we won't be using them in this tutorial.
* $delta: Your module can define more than one block in its "list" operation. Each block has a defined "delta" code, which is normally a number, and for the other operations, Drupal will pass in the "delta" value so you can identify which of a module's blocks to perform the operation on. This example module will only have one block. The core Drupal user module is an example of a module that has multiple blocks: user login block, who's new block and who's online block.
* $edit: used only with the "save" operation and not discussed in this tutorial.

The first operation is the "list" operation, which lists the blocks the module supplies, and defines how they will be seen on the Administer >> Site Building >> Blocks page (the Blocks module will call this function with $op='list' when it is building the list of blocks for the blocks administration page).

Here are the details of the onthisdate\_block() function (the next tutorial page has more):

<?php  
/\*\*  
\* Implementation of hook\_block  
\*/  
function onthisdate\_block($op = 'list', $delta = 0, $edit = array()) {    
if ($op == "list") {  
    // Generate listing of blocks from this module, for the admin/block page  
    $block = array();  
    $block[0]["info"] = t('On This Date');  
    return $block;  
  }   
} // end onthisdate\_block  
?>

(Note for beginners: the opening <?php will only appear once in your module file, at the beginning, and the closing ?> will not appear in your file at all.)

OK, so let's talk about that code....

**$block** - $block is just a variable used to store the necessary data, before returning it.  
**$block[0]** - $block is an array variable, and each item in the array represents one block that your module supplies (our example module only supplies one block). The "0" array index is the $delta value that will be used in later operations (if we were defining multiple blocks, we would need to have a $block[0], $block[1], $block[2], etc.; we would also have to check the $delta value passed into the function for later operations).  
**$block[0]["info"]** Each element (0, 1, 2) representing a block is also an array of key = value pairs. Some of the keys are "info", "cache", "weight", "status". That's a bit much for an intro lesson, so we are only going to consider "info", which is the human-readable name of your block that shows up on the block administration page. So, inside hook\_block() our block is known as **$delta = 0**, but on the admin page it is known as **On This Date**.

The next tutorial page will cover generating the block content, by implementing the 'view' operation.

# Generate the block content

The next step in this tutorial is to generate the content of the block. This will involve accessing the Drupal 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.

To tell Drupal what content we want in the block, we use the 'view' operation of hook\_block(). So, we'll need to add some code to our previously-defined onthisdate\_block() function.

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) {  
  
  if ($op == "list") {  
    // Generate listing of blocks from this module, for the admin/block page  
    $block = array();  
    $block[0]["info"] = t('On This Date');  
    return $block;  
  }   
  else if ($op == 'view') {  
  
    // Generate 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  
      
  
   // more coming...  
  }  
}  
?>

If the site you are running on might not have content for the date exactly one week ago, you might want to change the end time to the following, in order to show something in the block:

$end\_time = time(); // get all posts from one week ago to the present

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 types 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).

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) with our SQL query:

<?php  
  $query = "SELECT nid, title, created FROM " .  
     "{node} WHERE created >= '%d' " .  
     " AND created <= '%d'";  
  
  $queryResult =  db\_query($query, $start\_time, $end\_time);  
?>

This illustrates how to do a "safe" query in Drupal: create your query string with "placeholders" such as %d and %s, and then pass variables into the db\_query() function to fill the "placeholders". This type of practice prevents SQL injection hacks, especially when user-generated content is used in a query. Another Drupal-specific custom to note here is that table names in Drupal database queries are always enclosed in curly braces, such as {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)](http://drupal.org/node/2622) page in the Drupal handbook. Finally, note that in this example, we are not being careful about access permissions for nodes. Normally, all queries on nodes should employ the db\_rewrite\_sql() function, which makes sure the user viewing the page has permission to see each node that is returned, but this is beyond the scope of this tutorial.

Now, we'll use [db\_fetch\_object()](http://api.drupal.org/api/function/db_fetch_object/6) to look at the individual records returned by the query. For each node that we found, we'll generate a link to the node, with the node's title as the link text:

<?php  
  // content variable that will be returned for display      
  $block\_content = '';    
  while ($links = db\_fetch\_object($queryResult)) {   
    $block\_content .=  l($links->title, 'node/'. $links->nid) .'<br />';  
  }  
?>

Notice the actual link is created by the [l()](http://api.drupal.org/api/function/l/6) function. l generates <a href="link"> links from Drupal paths, adjusting the URL to the installation's URL configuration of either clean URLS: [http://(sitename)/node/2](http://drupal.org/node/2) or not [http://(sitename)/?q=node/2](http://drupal.org/?q=node/2) (the path to any node is always "node/#", where # is the ID number of the node).

Finally, we need to return the content we just generated to Drupal for display:

<?php  
  // check to see if there was any content before returning  
  //  the block view  
  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 = array();  
  $block['subject'] = 'On This Date';    
  $block['content'] = $block\_content;  
  return $block;  
}  
?>

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

The example above assumes that if there is no content on the given date one week ago, you want to completely omit the block. Another possibility (especially during the testing phase) would be to display a message:

  if ($block\_content == '') {      
     // no content from a week ago  
     $block['subject'] = 'On This Date';  
     $block['content'] = 'Sorry No Content';  
     return $block;  
  }

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 <div> tag with a module-specific CSS class and not necessarily include the <br /> 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) {  
  
  if ($op == "list") {  
    // Generate listing of blocks from this module, for the admin/block page  
    $block = array();  
    $block[0]["info"] = t('On This Date');  
    return $block;  
  }   
  else if ($op == 'view') {  
  
    // Generate 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 >= '%d' " .  
             " AND created <= '%d'";  
  
    $queryResult =  db\_query($query, $start\_time, $end\_time);  
    while ($links = db\_fetch\_object($queryResult)) {  
      $block\_content .= l($links->title, 'node/'.$links->nid) . '<br />';  
    }  
  
    // check to see if there was any content before returning  
    //  the block view  
    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;  
  }  
}  // end onthisdate\_block  
?>

Our module is now functional - we can install, enable and test it (see next tutorial page).

# Installing, enabling and testing the module

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

### Install

To install the module, you'll need to copy your onthisdate.module and onthisdate.info files to the right directory on your site, if they are not there already (probably sites/all/modules/onthissite -- see the [first page of this tutorial](http://drupal.org/node/206754) for more information).

### Enable

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

* http://example.com/admin/build/modules
* http://example.com/?q=admin/build/modules

When you scroll down, you'll see the onthisdate module listed with the description next to it, in the "Other" section. Enable the module by selecting the checkbox and save your configuration.

### Configure

The purpose of this module is to display a block, but just enabling the module doesn't make the block display. You need to go to the blocks administration page (Admin >> Site building >> Blocks or pate admin/build/block) and enable it.

Enable the block by selecting a region in the drop-down list 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. After saving the block region setting, you might also want to change the configuration (by clicking on the "configure" link) so that the block is only displayed on certain pages of your site or is only visible to certain roles. One thing to be aware of is that if you are logged into your site as User 1 (the original account created when you installed Drupal), you don't necessarily have a role besides "authenticated user".

### Test

To see the block, navigate to another page (it's hard to see them on the Blocks page itself); if you chose to have the block visible only on certain pages, be sure to navigate to one of those pages. Note that our example module is configured to display only posts from the day exactly one week ago, and to be invisible if there are no posts. So if you have no content from that date, you might want to use one of the suggestions on the previous tutorial page to either display more content or display a helpful "no content" message if there is no content.

You can also create some content (or edit some existing content) and adjust the "Authored on:" date to be a week ago, if your site lacks content from that particular date.

### Troubleshoot

If you get a "white screen" or a PHP error on your screen when you enable this module (or after editing the file while it is enabled), it probably means you have a syntax error in your .module file. (In the case of a white screen, you may be able to find out what the PHP error was by looking in your Apache error log.) If you cannot find and fix the syntax error, nothing on your site will display, because Drupal will try to load your module on every page request. The easiest way to get your site working again is to delete the module's folder, in which case Drupal will figure out that it shouldn't load this module after all, and your site should be working again.

# Create a module configuration (settings) page

**Main topic described: Module settings**  
**Drupal hook used:** [***hook\_menu***](http://api.drupal.org/api/function/hook_menu/6)

Now that we have a working module, we'd like to make it more flexible. 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 how many links to display, and leave it at that for this tutorial.

### Create the configuration function

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. The first step in doing that is to define a "system settings" form page, using the Drupal Forms API. Since this is our "administer" page, we'll call the function that generates the function onthisdate\_admin() (or we could choose another name, but it should start with onthisdate\_ ) and put it into our onthisdate.module file:

<?php  
function onthisdate\_admin() {  
  $form = array();  
  
  $form['onthisdate\_maxdisp'] = array(  
    '#type' => 'textfield',  
    '#title' => t('Maximum number of links'),  
    '#default\_value' => variable\_get('onthisdate\_maxdisp', 3),  
    '#size' => 2,  
    '#maxlength' => 2,  
    '#description' => t("The maximum number of links to display in the block."),  
    '#required' => TRUE,  
  );  
  
  return system\_settings\_form($form);  
}  
?>

There are several things to notice about this function:

* $form is an array defining form elements in the Drupal Forms API. Each element of the array corresponds to a form element; in this case, we have one required two-character text field whose label is "Maximum number of links", with help text "The maximum number of links to display in the block".
* We only have to define form elements for the actual settings elements -- the system\_settings\_form() function will take care of creating the form page, adding a submit button, and saving the settings.
* Drupal maintains a database of "variables", or settings; each setting must have a unique name, so customarily the module name is used as a prefix -- our setting is called 'onthisdate\_maxdisp'.
* The Drupal function [variable\_get()](http://api.drupal.org/api/function/variable_get/6) is used to retrieve the previously-stored value of the setting, and we've given it a default value of 3 if there was no previously-stored value.
* The setting name, 'onthisdate\_maxdisp', is used in the call to variable\_get() and is also the array key for the form element in the $form array. This is important, because the Drupal system\_settings\_form() function will use the array key as the name of the setting to save when the form is submitted.
* All of the text our form will display is passed through the translate function of t(), so that sites in other languages can use our module.
* Refer to Drupal [Forms API Reference](http://api.drupal.org/api/file/developer/topics/forms_api_reference.html/6) and Drupal [Forms API Quickstart Guide](http://api.drupal.org/api/file/developer/topics/forms_api.html/6) for more detailed information on what more you can do with the Drupal Forms API.

We'll also need to modify our hook\_block() implementation to use this setting. The best way to do that in Drupal is to use the [db\_query\_range](http://api.drupal.org/api/function/db_query_range/6) function:

<?php  
  $limitnum = variable\_get("onthisdate\_maxdisp", 3);  
  
  $query = "SELECT nid, title, created FROM " .  
           "{node} WHERE created >= %d " .  
           "AND created <= %d";  
  
  $queryResult = db\_query\_range($query, $start\_time, $end\_time, 0, $limitnum);  
?>

You'll need to replace the corresponding $query and $queryResult lines in your existing onthisdate\_block() function with these three lines.

### Add the page to [hook\_menu](http://api.drupal.org/api/function/hook_menu/6)

Once you have created the function with your settings form, you need to define a URL within Drupal for your settings page. This is done by implementing Drupal's [hook\_menu](http://api.drupal.org/api/function/hook_menu/6). In our [hook\_menu](http://api.drupal.org/api/function/hook_menu/6) implementation, we will return an array which describes to Drupal which URL path to use, the title to display, the function to call to generate the page, and the permissions required.

We would like only administrators to be able to access this page, so we'll place the permissions check for the module here in [hook\_menu](http://api.drupal.org/api/function/hook_menu/6) so that Drupal can itself check the appropriate permission. To minimize the number of permissions an administrator has to deal with, we're going to use the global administration permission for administrating our module instead of creating a new custom permission.

To implement hook\_menu(), create a function called onthisdate\_menu() and put it in your onthisdate.module file:

<?php  
function onthisdate\_menu() {  
  
  $items = array();  
  
  $items['admin/settings/onthisdate'] = array(  
    'title' => 'On this date module settings',  
    'description' => 'Description of your On this date settings page',  
    'page callback' => 'drupal\_get\_form',  
    'page arguments' => array('onthisdate\_admin'),  
    'access arguments' => array('access administration pages'),  
    'type' => MENU\_NORMAL\_ITEM,  
   );  
  
  return $items;  
}  
?>

Note that the array key 'admin/settings/onthisdate' is the URL that we are defining, and the array elements give the menu link title ("On this date module settings" -- it should always start with a capital letter and otherwise be lower-case), a longer description, the name of the function to call that will return the settings form ('onthisdate\_admin'), and the access permission. You can check out the [hook\_menu documentation](http://api.drupal.org/api/function/hook_menu/6) for more details.

After adding this to the module, you will need to clear the menu cache, so that Drupal will recognize the new URL (Drupal caches a lot of data, including a list of all the URLs it recognizes). To clear the cache, go to Administer >> Settings >> Performance, scroll to the foot of the page, and click the "Clear cached data" button.

Now you can test the settings page by editing the number of links displayed and noticing that the block content adjusts accordingly. Navigate to the settings page: admin/settings/onthisdate or Administer » Site configuration » On this date. Adjust the number of links and save the configuration. The maximum number of links in the block should adjust accordingly.

### Validate the user input

Although we aren't required to validate the user input, it is nice to do so. We can do this by writing a onthisdate\_admin\_validate function (in onthisdate.module) that checks whether the value the user entered is a number greater than 0. Because the validation function has the same name as the form generation function, with a "\_validate" suffix, Drupal will use our validation function automatically when the form is submitted.

<?php  
function onthisdate\_admin\_validate($form, &$form\_state) {  
  $maxdisp = $form\_state['values']['onthisdate\_maxdisp'];  
  if (!is\_int($maxdisp)) {  
    form\_set\_error('onthisdate\_maxdisp', t('You must enter an integer for the maximum number of links.'));  
  }  
  else if ($maxdisp <= 0) {  
    form\_set\_error('onthisdate\_maxdisp', t('Maximum number of links must be positive.'));  
  }  
}  
?>

Now if you try to enter something that it doesn't like (a word, or a negative number), it will tell you to enter a correct value.

# Generate page content

**Main topic described: *Displaying 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. This is a three-step process, which we'll describe on this tutorial page and the next two pages.

The first step, described on this page, is to add a new function to our onthisdate.module file that will output a complete list of all the content that was created a week ago. We'll name this function onthisdate\_all() (we could choose another name, but it must start with "onthisdate\_").

A quick note on function naming conventions in Drupal: If you are creating a function that you intend to be strictly private (i.e. no other module should rely on it being a stable function or call it), start the function name with "\_your\_module\_name\_". If your function is public (i.e. it would be OK for another module to call it, and you don't intend to change its argument signature or behavior often), start the function name with "your\_module\_name\_". If you are implementing a Drupal hook, you must always name the function "your\_module\_name\_hookname".

So, back to the new onthisdate\_all() function in the onthisdate.module file. It is basically going to reproduce the original version of our block, from before we put in the option to limit the block to a certain number of links (see [page 5 of the tutorial](http://drupal.org/node/206759)), and we want the function to return the HTML content for the page. Note that we don't have to worry about HTML headers, page titles, menus, footers, etc. We only have to generate the content section of the page, and Drupal and our theme will take care of the rest. Here's the part of the function that is essentially copied from the block function:

<?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  
  
  $query = "SELECT nid, title, created FROM " .  
           "{node} WHERE created >= '%d' " .  
           " AND created <= '%d'";  
  
  // get the links (no range limit here)  
  $queryResult =  db\_query($query, $start\_time, $end\_time);  
  while ($links = db\_fetch\_object($queryResult)) {  
    $page\_content .= l($links->title, 'node/'.$links->nid) . '<br />';  
  }  
  
  // More coming....  
}  
?>

As noted before, 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.

Also note that, as in the case of a block, you might want to incorporate a different time range if your site has no content on that particular date one week ago.

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.

<?php  
function onthisdate\_all() {  
  
  // Be sure to include the previous piece of this function!  
  
  // check to see if there was any content before  
  // returning the page  
  if ($page\_content == '') {  
    // no content from a week ago, let the user know  
    $page\_content = "No events occurred on this site on this date in history.";  
  }  
  return $page\_content;  
}  
?>

Even though we have this function that will output links to the content generated a week ago, we haven't specified what URL will display this page. We'll do that next.