



Expert in Drupal

**Course in
creation and administration
of Websites with Drupal 7**

EXCLUSIVELY FOR:
Name: Nidhi Badani
Email: nidhitmehta@gmail.com

DOWNLOADED FROM:
Source: www.forcontu.com
Date: 11/05/15 17:11
IP: 125.22.49.35
Verification code:
D7AVZPDFEN00037737007079

Advanced
Fran Gil

www.forcontu.com

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

EXCLUSIVELY FOR:

Name: Nidhi Badani

Email: nidhitmehta@gmail.com

DOWNLOADED FROM:

Source: www.forcontu.com

Date: 11/05/15 17:11

IP: 125.22.49.35

Verification code: D7AVZPDFEN00037737007079

Expert in Drupal 7

Advanced

**Course in creation and administration
of Websites with Drupal 7**

Fran Gil



Expert in Drupal 7. Advanced

Course in creation and administration of Websites with Drupal 7

Learn Drupal with Forcontu Collection

Copyright © 2014 Forcontu S.L.

All rights reserved. The content of this book cannot be wholly or partially reproduced, stored or transmitted in any form or by any means, whether electronic, mechanical, photocopied, recorded or in any other manner, without the prior express written authorisation of Forcontu S.L. In particular, this includes the simple reproduction of this as summaries, reviews or press releases and/or making these available, and for these purposes it will also be necessary to have the relevant authorisation from Forcontu S.L. To obtain further information, please get in touch by means of info@forcontu.com.

ISBN-13 (Printed edition): 978-84-942763-2-3

ISBN-13 (Electronic edition, PDF): 978-84-942763-3-0

Author: Fran Gil Rodríguez

With the assistance of the team of tutors of Forcontu: Iván Méndez López, Miguel Ángel Naranjo Moreno and José Miguel Rodríguez Arias

Cover design: Ateigh Design

Marketing and Distribution Manager: Laura Mª Fornié Alonso

Published by Forcontu S.L.

Although this publication has been prepared with great care, neither Forcontu S.L. nor the authors will be responsible for the possible errors and their consequences that may directly or indirectly arise from the information contained in this work. Please send any recommendation or comment to info@forcontu.com.

For further information about the content of this book or the distribution channels, please write directly to info@forcontu.com or visit the Website www.forcontu.com.

You can find complementary resources to this book at www.forcontu.com.

Presentation

One of the great challenges facing us to use Drupal is, without doubt, **overcome its steep learning curve**. Mastering Drupal is a costly goal and requires effort and dedication. With more than 3 years of experience in Drupal 6 training, Forcontu launches the course **Expert in Drupal 7**, a training plan improved and expanded to train Web development professionals in Drupal 7.

The Course **Expert in Drupal 7** is provided online. It is guided and supervised directly by the team of tutors of Forcontu.

The course **Expert in Drupal 7** is divided into 3 modules or training levels (beginner, intermediate and advanced) and **requires programming knowledge**. As an alternative, for those people who do not have programming skills, there is the possibility of studying only the beginner and intermediate levels.

Many companies and organizations are betting on Drupal as a solution for their Websites development, and on **Forcontu** to train their working team to the highest level. More than 1,000 people have already tested our courses and materials, both Drupal 6 and Drupal 7.

Fran Gil Rodríguez
Co-founder and CEO of Forcontu



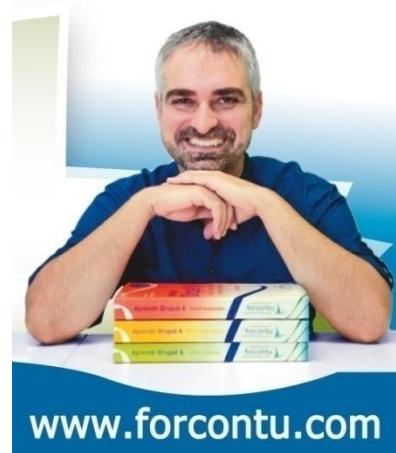
Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

About the author

Fran Gil Rodríguez
Co-founder and CEO of Forcontu

fran.gil@forcontu.com
@frankgil76

Fran Gil Rodríguez is a Technical Telecommunications Engineer from the University of Las Palmas de Gran Canaria (2002), holding a Master's degree in the Management and Administration of Information Technologies and Systems from the UOC (2006) and a University Master's degree in Multidisciplinary Computing, specialising in Electronic Teaching and Learning from the University of Alcalá de Henares, Madrid (2009).



His professional career was initially associated with the University of Las Palmas de Gran Canaria (ULPGC), first as the coordinator of the team for developing the ULPGC Website (2002 to 2006) and then as Head of projects and coordinator of the ITC development department ULPGC S.L. (2006 to 2009). He has participated in numerous projects for developing applications and Websites, as head of Project, an analyst and developer, working with Drupal since 2006.

2009 saw the creation of the Forcontu project, a company specialising in Website development and advanced training in Drupal. Following the success of the Drupal 6 training plan, launched in 2009, the author presents us with an enlarged and improved Drupal 7 training plan: Expert in Drupal 7.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

0 Expert in Drupal 7

0.1 Course description

Teaching method

The Expert in Drupal 7 course is a 100% online course with supervised practice labs. In addition to the virtual classroom, students will be provided with web hosting for their own Drupal installation.

Through the virtual classroom, the course tutor will be in contact with students, guiding them in their learning process and proposing the work necessary for passing the course. Students will hand in their work directly in their assigned web space, **building their own web site** following the proposed activities.

The student will be able perform activities at any time of the day, in order to adapt the training to his/her professional or private needs.



Length and modality

The course is made up of **60 units** and divided into **3 modules**: beginner, intermediate and advanced. After completing all three modules, students will develop and complete a final project. The total course duration is 7 months with 420 hours of certified work.

While this formative experience requires a significant amount of effort, you will save a considerable amount of time when compared to studying Drupal on your own.

Learning objectives

- The main objective of Expert in Drupal 7 is to offer students professional training in design, development, implementation and management of complex websites, using Drupal. The course materials are designed to train students how to complete successful implementation of professional websites using Drupal. However, to reach these objectives, the following secondary objectives must be achieved:
- **Acquire a good knowledge in Drupal.** Having a good base is essential for defining a good strategy for the installation, configuration and programming of features in Drupal. In this course we place special emphasis on the need to study Drupal in an orderly and guided fashion, carrying out work ranging in difficulty from low to high. For this reason we believe that all of the activities proposed in the training plan are important for building a good solid foundation, regardless of the student's background.
- **Knowing Drupal architecture in depth.** Knowing Drupal architecture at the programming level is essential in understanding how the modules interact with the system, as well as developing modules that adapt to the core and the rest of the installed modules on a Website in a proper manner.
- **Essential additional modules.** The Drupal core is only a starting point; it will always be necessary to add additional modules. The most important modules that are offered will be studied in this course. Some of these modules have become essential, because they have been

- used in prior versions to Drupal 7, having enjoyed widespread acceptance in the community for years. Other modules created specifically for Drupal 7 will also be studied.
- **Developers' tools.** We will study some of the programming and system administration tools with the aim of speeding up the work with Drupal.
 - **Adaptation to new versions and modules.** The student will have the necessary skills to adapt to the continuous changes of version and to manage the installation and configuration of new modules contributed by the community successfully.
 - **Modules Programming.** When the available modules are not enough to achieve the functional requirements of our project, we will need to implement new modules. We go into greater depth in the API of Drupal and the Drupal programming style to develop professional modules.
 - **Learn how to contribute to the community.** The Drupal success would not have been possible if it did not have thousands of enthusiasts, users and developers, who have shared their developments, knowledge and experiences with the community. This is why we believe that, wherever possible, we must contribute to the community, whether sharing and maintaining developed modules, reporting bugs and solutions, or helping other users.

What does the course cover?

Through this course you will learn to install, configure and administrate Websites with Drupal 7. You will also be able to develop new functionalities that allow adapting a Website to the required functional requirements. Becoming an expert in Drupal 7 you will be trained to deal with, individually and in team, the design and implementation of complex Websites. The acquired training will be helpful in getting you into the labor market, improving your career or even undertaking your own project based on web technologies.

Who should attend?

The Expert in Drupal 7 course is aimed at web site developers with programming skills that require the development of professional Web projects with Drupal 7, either for third parties or in order to carry out their own business ideas.

Prior Knowledge

No specific academic qualification is required to gain entry to the course. The beginner and intermediate levels can be studied in their entirety with no programming or database management knowledge. Only a minimum knowledge of computer basics and Internet user-level is needed.

Technical requirements

It is necessary to have a computer with Internet connection and web browser to take this course. During the course it will be necessary to install an FTP client (FileZilla or similar).

0.2 Course structure and content

What the course includes

The Expert in Drupal 7 course includes the following services and materials:

- **Materials organized in English.** There is a lot of information on Drupal available in books, blogs, forums, etc. At Forcontu, we have developed organized training materials consisting of different learning subjects to ensure the professional training in Drupal (compulsory work, Practical Case s, enhancement activities, etc). The course includes **3 Expert in Drupal 7 books** in Web page format and an eBook (PDF).
- **Web hosting** for the training plan activities and the final project undertaking.
- **Continuous assessment** of the work carried out during the course, providing feedback on the possible mistakes.
- **Practical examination after each module**
- **Final Project evaluation.**
- **Follow-up on the student's progress.** Tutors will carry out continuous monitoring of the student and will answer the doubts raised in the classroom forums.
- **Certificate for passing part of the module.**
- **Expert in Drupal 7 Certificate.** After successfully passing the training modules and the end of course project.
- **Letter of introduction,** for students who have completed the course Expert in Drupal 7 with a grade of Excellent.
- Consultancy after the course in order to **migrate your Website** (final work) to your personal hosting site.

Module I: Beginner

Beginner level is aimed at those people with or without technical training, who want to start creating Websites. It is also very useful for companies or professionals who wish to create their professional or personal Website without depending on external assistance. At this level, the aim is for the student to acquire sound basic knowledge in order to tackle the following educational levels with guarantees.

This module will be studied for 4 weeks and it will certify 60 hours of work.

1. Introduction to Drupal
 2. Drupal installation
 3. Administration area
 4. Content management
 5. Content types
 6. Menu management
 7. Blocks management
 8. Themes
 9. Extending functionality with modules
 10. Management of users, roles and permissions
 11. Taxonomy
 12. Text formats and WYSIWYG editors
 13. Blogs
 14. Forums
 15. Structured documents: Books
 16. Media Galleries
 17. Searches
 18. Languages and translation
 19. Access statistics
 20. Website management, maintenance and updating
- Final module exam

Module II: Intermediate

The **Intermediate level** complements the beginner level. It is aimed at people who already had previous contact with Drupal and want to understand it in depth and make better use of its full potential. Completing this level is very useful for building professional Websites with features that go far beyond company webs or personal blogs. Programming knowledge is not required to study intermediate level.

This module will be studied for 8 weeks and it will certify 120 hours of work.

21. Creating forms with Webform
 22. Advanced content types
 23. Views
 24. Automatic image processing
 25. Extending menus with additional modules
 26. Extending blocks with additional modules
 27. Panels
 28. Display Suite
 29. Customizing Themes
 30. Actions and Triggers
 31. Rules
 32. Defining contexts
 33. Extending users
 34. Workflow
 35. Multilingual sites
 36. E-commerce: Introduction to Ubercart and Drupal Commerce
 37. SEO Positioning with Drupal
 38. Social tools: voting and sharing content
 39. Newsletter Subscription
 40. Other Features
- Final module exam

Module III: Advanced

Advanced level is intended for PHP and MySQL developers who have also had a prior contact with Drupal. This level is designed for Website developers who work for third parties or for those entrepreneurs with an advanced technical level who wish to launch their own business ideas.

This module will be studied for 12 weeks and it will certify 180 hours of work.

41. Drupal development best practices
 42. Tools for developers
 43. Drupal Architecture
 44. Creating modules
 45. Database access and database schemas
 46. Menu system
 47. Creating forms
 48. Programming blocks
 49. Programming users and permissions
 50. Programming content types
 51. Programming entities and fields
 52. Programming actions and triggers
 53. Working with Files
 54. Search system
 55. Translating modules
 56. Creating themes
 57. Using jQuery and Ajax
 58. Features
 59. Installation Profiles
 60. Contributing to the Drupal Community
- Final module exam

Final Project

The final project may be taken during the course, but there will be 4 weeks available after the three modules to complete and submit it. This Project will certify 60 hours of work.

0.3 Methodology, materials and evaluation

Methodology

The **Expert in Drupal 7** is an online supervised course. The student must carry out the proposed activities on the hosted website, which will be corrected and assessed by the tutor on an ongoing basis, as they are handed in. The e-learning platform will give access to the course contents and the necessary tools for facilitating communication among the participants in the course, the correction of activities and the answering queries.

The schedule includes detailed plans of the work to be carried out, continuous monitoring of the student's progress and evaluations of the student's activities and work completed.

The course is arranged based on a recommended timetable for handing in work, although the students can follow their own pace of work and hand them in throughout the course, but they must always do this before the end of each module. The student can work on his/her web host and in the classrooms used in any schedule, although the questions expressed in the forums and correcting work will generally be done during business hours. Our tutors' full-time commitment ensures flexible communication and corrected activities handed back on a daily basis.

E-learning platform

Once registered for the course, the student will be added onto the educational platform and placed in the classroom corresponding to the module to be studied.

Tutors and students may communicate freely through **forums** (in a way that is public to the entire group), or by **internal e-mail** (privately, only for the stated recipients) for each course. By default, a duplicate of these communications will be sent to the external electronic e-mail address provided by the student.

Students in the same group can contact each other and share their experiences and the results of their work and proposed materials.

The classroom is arranged in the following way:

- **Guide Forum** (Tutor's Forum). Only tutors can publish on this forum, and this will be used to report general issues concerning the course or other issues that may be of interest to the Group.
- **Learning Forum** (Course Forum). Both tutors and students will be able to post on this forum, and it will be used to discuss issues related to Drupal and the course. The students may raise their questions about the work and other general enquiries about Drupal here.
- **Share Forum** (general forum). This is an open forum where students and tutors can share other experiences aside from the course and Drupal.
- **Inbox.** The internal mail enables messages to be sent to other classmates and tutors.
- **Participants.** List of participants on the course, including tutors and students.
- **News.** The latest posts added by tutors in the Forum guide will be published in this block.
- **Upcoming events.** This block shows upcoming events related to course planning. Participants can create private events.

- **Calendar.** Displays events in timetable form.
- **Online Users.** A participants' list shows who is connected at any one time.

Profile (administration). Allows you to administer the user profile: modify the login password, modify the personal data, change the photo, etc.

Educational materials

Materials are written in English and consist of:

- EBooks of the **Learn Drupal with Forcontu set**, corresponding to the registered level. The books will be available in the classroom in digital format (html or PDF).
- **Web Hosting** for doing the work and the student's final project.
- **Links** to external resources.
- **Activities.** Each unit will include activities. That it must be carried out in the time set so the tutor can assess them later on.



Evaluation

During the course each piece of work will be assessed, corrected and rated by the tutors, providing continuous feedback to the student.

At the end of each module the student will get a **continuous assessment rating** based upon the individual marks obtained in their work. To achieve each module it will be also necessary to pass a practical examination. This examination is necessary to validate the mark obtained in the continuous assessment so the student will have only a pass or fail grade. Once the module has been passed, a partial certificate of accomplishment of the module will be issued.

Once the three modules of the course have been passed, the student will have to carry out a Final Project, consisting of the development and documentation of a Website. By passing all the formative modules and the Final Project the student will obtain the **certificate of Expert in Drupal 7**.

Upon completion the course the **certificates of follow-up and accomplishment of each passed module and the certificate of Expert in Drupal 7** will be delivered by ordinary post detailing, among other things, the number of working hours, the content of the course and the final mark obtained.

The **final grade of Expert in Drupal 7** will be obtained by scoring the marks attained in each of the modules and final project, having regarded to the following percentages:

- Module I: Beginner (15%)
- Module II: Intermediate (30%)
- Module III Advanced (40%)
- Final Project (15%)

Drupal Certification with Forcontu

Companies increasingly demand professional Web developers, experts in Drupal, and they usually ask for a training certificate and/or experience in Drupal. Another new development introduced into our training plans is the new model of **Drupal certification with Forcontu**.

All training progress with Forcontu will be available on a unique URL that you will be provided with at the beginning of the course. As you pass the modules or additional activities of the lifelong learning Plan, these achievements will be reflected directly on your **Drupal certification with Forcontu**.

If you would like, you can include this URL in your curriculum vitae so the companies that are in your level of knowledge of Drupal can collate the information provided there in.

Table of contents

Presentation	iii
About the author	v
Expert in Drupal 7	vii
Course description.....	.vii
Course structure and contentix
Methodology, materials and evaluation.....	.xii
Table of contents.....	xv
Unit 41. Drupal development best practices	1
41.1. Coding Standards.....	2
41.2. Code Comments	7
41.3. Code verification with Coder module.....	12
Unit 42. Tools for Developers	15
42.1. Devel Module.....	16
42.2. Debug Drupal in Firefox and Chrome	23
42.3. Drush Module	25
42.4. Module Builder Module	28
42.5. Examples for Developers Module	32
Unit 43. Drupal Architecture	33
43.1. Drupal 7 Requirements	34
43.2. Modules and Hooks	35
43.3. Drupal API.....	37
43.4. File Structure.....	39
43.5. Operation of Drupal.....	41
43.6. Drupal Architecture	43
Unit 44. Creating modules.....	47
44.1. Definition of a module	48
44.2. Hooks	58
44.3. Module configuration.....	76
44.4. Working with the database	82
Unit 45. Database access and database schemas	85
45.1. Database Access	86
45.2. Schema API.....	95
45.3. Other Functions of the Schema API	99
Unit 46. Menu System	103
46.1. Introduction to the menu system.....	104
46.2. Implementation of hook_menu()	105
46.3. URL with parameters.....	109
46.4.Registering URLs and menu elements	111
46.5. Definition of tabs	114
46.6. Access control to the page.....	117
46.7. Menu system functions	119

Unit 47. Creating forms	123
47.1. Defining forms in Drupal	124
47.2. Form elements.....	127
47.3. Validation of forms	142
47.4. Sending forms	144
47.5. Modification of Forms	146
47.6. Forms in the administration area of the module	149
47.7. Functions of Form API	151
Unit 48. Programming Blocks.....	155
48.1. Defining Blocks	156
48.2. Implementing blocks	160
48.3. Automatic Activation of Blocks.....	167
48.4. Modifying Blocks	170
Unit 49. Programming users and permissions	173
49.1. User definition with the object \$user.....	174
49.2. Functions for working with users	177
49.3. Users implementation	183
49.4. Users access.....	192
49.5. Defining permissions	193
Unit 50. Programming content types	195
50.1. Creation of content types.....	196
Practical case 50.1. Create a content type	201
50.2. Node Management.....	205
Practical case 50.2. Managing Nodes.....	209
50.3. Node Presentation.....	213
Practical case 50.3. Node Presentation	215
50.4. Access control.....	217
50.5. Content Filters	225
Practical case 50.4. Create a filter	232
Practical case 50.5. Create a text format	236
50.6. Treatment of nodes from other modules	238
50.7. Other features of content types	242
Unit 51. Programming entities and fields	243
51.1. Definition of Entities	244
Practical case 51.1. Create an entity type	248
51.2. Creation of fields (Field API)	262
51.3. Creation of new field types	267
51.4. Entity API Module	276
51.5. Model Entities module	278
Unit 52. Programming actions and triggers	281
52.1. Implementation of actions	282
52.2. Implementation of triggers	291
52.3. hook_cron() function.....	297
52.4. hook_watchdog() function	300
52.5. Sending email	303
Unit 53. Working with files.....	307
53.1. The file system of Drupal	308
53.2. Files Functions (File API)	310
53.3. Forms with files	318
53.4. Control of file permissions.....	321
Unit 54. Search system.....	323
54.1. Introduction to the search system	324
54.2. Searching nodes	326
54.3. Custom Search	330
54.4. Indexing of content.....	334

Unit 55. Translating modules	339
55.1. Translation of modules	340
55.2. Translation template extractor Module	343
55.3. Translating with Poedit	346
Unit 56. Creating themes	347
56.1. Introduction to the theme system.....	348
56.2. Creating a theme	350
56.3. Template files.....	357
56.4. Element rendering.....	363
56.5. Theme and template functions	367
56.6. Preprocessing Functions	372
56.7. Theme configuration options.....	374
56.8. Creation of module templates	376
Unit 57. Using jQuery and Ajax	381
57.1. Introduction to JavaScript in Drupal	382
57.2. Introduction to jQuery	384
57.3. jQuery Libraries in core.....	393
57.4. Ajax.....	397
Unit 58. Features	407
58.1. Introduction to Features	408
58.2. Create and install Features.....	409
Practical case 58.1. Create a Feature.....	413
58.3. Integration with Drush	420
58.4. Additional Modules	421
Unit 59. Installation profiles	425
59.1. What are installation profiles?	426
59.2. Definition of the installation profile	427
59.3. Additional installation tasks	431
59.4. Actions during the profile installation	440
59.5. Contributed distribution profiles.....	442
Unit 60. Contributing to the Drupal Community	445
60.1. The Drupal Community	446
60.2. Reporting Issues	448
60.3. Sharing a project in Drupal.org.....	451
60.4. Create and apply patches	460
60.5. Create new versions of a project	464
60.6. Assist in Drupal translation.....	466
Annex A. Configuration of the hosting and necessary tools	469
A.1. Hosting dashboard.....	469
A.2. Managing databases with phpMyAdmin	474
A.3. Upload and download files via FTP	476
A.4. Uncompress files with 7zip	480
A.5. Editing text files with Notepad++	481
Annex B. Links and resources of interest	483
B.1. forcontu.com.....	483
B.2. Official Drupal page, drupal.org.....	484
B.3. drupalmodules.com.....	484
B.4. Drupal Association	485
B.5. Drupal Group on Google	486
B.6. Translations of Drupal, localize.drupal.org	486
B.7. Drupal API, api.drupal.org	487
B.8. Drupal paid-for themes	487
B.9. Generic texts for tests	487

*Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079*

41 Drupal development best practices

When developing applications as part of a team or a community, it is especially important that the code created is easy to read and modify, no matter who authored the original code. The first step to achieve this is to follow a set of rules or guidelines to format the code in a standardized way for all the contributors. In this unit we will cover Drupal's coding standards.

Also, keep in mind that the Drupal community uses English as the language for module development and associated documentation.

In this unit we will also learn how to use Coder module to test the code in the modules we create to make sure they meet the Drupal coding standards.

Comparative D7/D6

Coding

Drupal 7 maintains the same coding standards already in use in the module programming of Drupal 6.

Unit contents

41.1 Coding Standards.....	2
41.2 Code Comments	7
41.3 Code verification with Coder module	12



41.1 Coding Standards

Drupal complies with the configuration standards explained in this unit. You may find more information about these coding standards at this website: <http://drupal.org/coding-standards>.

In addition to the examples provided in this unit, it's good practice to study the coding files in the Drupal distribution and in any other modules shared by the Drupal community. Always keep in mind that the modules shared by the community do not pass through quality control prior to publication so they may contain coding errors.

Indentation

Indentation consists of inserting spaces or tabs on certain lines of code to make them easier to read. In coding we use indentation to nest elements.

When coding in Drupal, **indent by 2 spaces, never with tabs**. Also, **you shouldn't leave any blank spaces at the end of each line**. In the following example you can see a code fragment with the indents included, of two spaces each (\rightarrow), and a hard return or **Enter** (\downarrow) at the end of each line (without any spaces).

F41.1

Indentation

When coding in Drupal, indent by 2 spaces, never with tabs. Also, you shouldn't leave any blank spaces at the end of each line.

```
<?php
function forum_block_view($delta = '') {
  $query = db_select('forum_index', 'f')
    ->fields('f')
    ->addTag('node_access');
  switch ($delta) {
    case 'active':
      $title = t('Active forum topics');
      $query
        ->orderBy('f.last_comment_timestamp', 'DESC')
        ->range(0, variable_get('forum_block_num_active', '5'));
      break;
    case 'new':
      $title = t('New forum topics');
      $query
        ->orderBy('f.created', 'DESC')
        ->range(0, variable_get('forum_block_num_new', '5'));
      break;
  }
  $block['subject'] = $title;
  // Cache based on the altered query.
  $block['content'] = drupal_render_cache_by_query($query,
    'forum_block_view');
  $block['content']['#access'] = user_access('access content');
  return $block;
}
?>
```

PHP Code Tags

When writing in PHP, always use the complete tags `<?php y ?>` and never the shorthand `<? y ?>`.

In general the closing tag (`?>`) is omitted in PHP at the end of **.module** and **.inc** files. Removing it helps prevent unwanted white space at the end of the file (after the closing tag`?>`), that can cause HTML validation issues such as "*Cannot modify header information - headers already sent by...*".

Therefore, the closing delimiter at the end of a file (`?>`) is optional in Drupal.

Do not confuse this with the normal use of PHP language in files that also contain HTML (as in, for example, in **.tpl.php** template files), where **each PHP line must contain its corresponding opening and closing tags**, to differentiate it from HTML code.

In the following example you can see how to use the opening and closing PHP tags in **.tpl.php** template files. If the final content of the file is a PHP code fragment, it should have the corresponding closing delimiter. **F41.2**

```
<?php
/**
 * @file
 * Default theme implementation to display a forum which may contain forum
 * containers as well as forum topics.
 *
 * Variables available:
 * - $forums: The forums to display (as processed by forum-list.tpl.php)
 * - $topics: The topics to display (as processed by forum-topic-list.tpl.php)
 * - $forums_defined: A flag to indicate that the forums are configured.
 *
 * @see template_preprocess_forums()
 * @see theme_forums()
 */
?>
<?php if ($forums_defined): ?>
<div id="forum">
  <?php print $forums; ?>
  <?php print $topics; ?>
</div>
<?php endif; ?>
```

F41.2

PHP Code Tags

In files containing PHP and HTML (as `tpl.php`), the opening and closing tags of PHP snippets are mandatory, even at the end of the file (if the last piece of code is PHP).

Operators

Binary operators that are used between two values should be separated from those values on both sides of the operator by a space (e.g., `$number = 3`, instead of `$number=3`). This applies to operators such as `+`, `-`, `*`, `/`, `=`, `==`, `!=`, `>`, `<`, `.` (string concatenation?), `.=`, `+=`, `-=`, etc.

Unary operators such as `++`, shouldn't have spaces. For example, `$number++`.

Use of Quotation Marks

Single quotation marks ('chain') as well as double quotation marks ("chain") can both be used to enclose strings.

Quotation marks are used when variables and text are in the same string. For example, `"<h1>$title</h1>"`. Additionally, one should use quotation marks when the text includes a single quotation mark.

Use of the Semicolon (;) in PHP Code

Although PHP allows one to write individual lines of code without the line closer (;), as in for example <?php print \$title ?>, in Drupal you **always have to include the semicolon** at the end of the line of code: <?php print \$title; ?>.

- Correct: <?php print \$title; ?>
- Incorrect: <?php print \$title ?>

Control Structures

Keep in mind the following guidelines when working with control structures:

F41.3

- There should be a space between the control keyword (if, while, for, etc.) and the opening parenthesis. This is to avoid confusing control structures with function calls, as we will see further along.
- The opening curly { should be placed on the same line as the opening statement, separated by a space.
- It's good practice to always use the curly braces {} even in cases where their use is considered optional (a single "line" of code within the control structure).
- The statements else and elseif are to be written on the line following the closing tag of the previous sentence.

F41.3

Control Structures

Using control structure examples: if, switch and for.

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

//sentence switch
switch (condition) {
    case 1:
        action1;
        break;

    case 2:
        action2;
        break;

    default:
        defaultaction;
}

//sentence for
for ($i = 0; $i < 5; $i++) {
    actions;
}
```

Functions

Functions should be written in lowercase with the words separated by an underscore. You should always include the grouping/module, etc. name as a prefix to avoid duplication.

After the function name, there is **no space** between the opening parenthesis and the first argument. Each subsequent argument should be separated by a space following a comma after the previous argument.

```
function forum_help($path, $arg) {
```

In naming the function you apply the same rules as mentioned previously with respect to parameters, as shown in the following example:

```
$field = field_info_instance('node', 'taxonomy_forums', $node->type);
```

One exception is that it's possible to use more than one space before placing (=) in order to improve the layout, as in when several (=) are placed in a block:

```
$number1      = foo($a, $type);
$first_value  = foo2($b);
$i           = foo3();
```

Arrays

Values within an array (or matrix) should be separated by a space (after the comma that separates them). The operator => should include a space on both sides.

When a line declaring an array exceeds 80 characters, each element should be broken into its own line and indented one level (2 spaces). There should be a comma at the end of the last array element. It prevents parsing errors if another element is placed at the end of the list later. **F41.4**

```
$vector1 = array(1, 2, 'clave' => 'valor');

$vector2 = array(
  'forum' => 'foro1',
  'template' => 'forums',
  'arguments' => array('tid' => NULL, 'topics' => NULL),
  'size' => 128,
);
```

F41.4

Arrays

Examples of array declarations.

Constants

Names of **constants** should be written in **capital letters**, with **underscores** to separate words.

As with functions, constants should always include the module (or theme) name as a prefix to avoid duplication. The prefix should be written in capital letters, too. **F41.5**

F41.5**Constants**

Examples of constants.

```
/**  
 * The current system version.  
 */  
define('VERSION', '7.10');  
  
/**  
 * Core API compatibility.  
 */  
define('DRUPAL_CORE_COMPATIBILITY', '7.x');  
  
/**  
 * Minimum supported version of PHP.  
 */  
define('DRUPAL_MINIMUM_PHP', '5.2.4');
```

Global Variables

Although the use of global variables is not advised, when they are necessary to use, they should be written with an **underscore at the beginning**, followed by the **module or theme name**, followed by another underscore before the variable name.

```
global $_forum_numero_foros;
```

Module Names

As a general rule, module names should not have an underscore, even if several words are included. In this way it is easy to identify the module that belongs to the function, as the module name or prefix will all come before the underscore. For example, it's recommended to use **mimodulo** instead of **mi_modulo**.

This rule is not required, thus it's very common to find, within contributed modules, names with underscores.

File Names

File names should always be written in lowercase letters. The only exception is for documentation files, where the name will be in all capital letters and the extension .txt will be lowercase. For example, README.txt, INSTALL.txt, etc.

Sample URLs

Best practices require that you always use "example.com" for all sample URLs. For example: <http://example.com/node/add/article>.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Language

Drupal uses **English language as the base of the entire system**. For that reason, **all modules would be written in English**, as well as the code (function names, variables, etc) and chains that the user sees. In order for these chains to be translated, you will use the translation function **t()**, which we will cover later on.

Code Comments

41.2

In this section we differentiate between comments to clarify certain lines of code, which are inserted within the code, and **document comments**.

Document comments are usually written at the beginning of a file or of each function and they are used to provide help documents by applications that extract the information from the tags used.

Use the `/* */` tags for comments of several lines and `//` tags for single line comments. You should write in complete sentences, starting with a capital letter and ending with a period. In cases where the text makes reference to a constant, write this work in all caps (e.g. TRUE or FALSE). F41.6

```
/*
 * Implements hook_init().
 */
function system_init() {
  $path = drupal_get_path('module', 'system');
  // Add the CSS for this module. These aren't in system.info, because they
  // need to be in the CSS SYSTEM group rather than the CSS_DEFAULT group.
  drupal_add_css($path . '/system.base.css', array('group' => CSS_SYSTEM,
'every_page' => TRUE));
  if (path_is_admin(current_path())) {
    drupal_add_css($path . '/system.admin.css', array('group' => CSS_SYSTEM));
  }
  drupal_add_css($path . '/system.menus.css', array('group' => CSS_SYSTEM,
'every_page' => TRUE));
  drupal_add_css($path . '/system.messages.css', array('group' => CSS_SYSTEM,
'every_page' => TRUE));
  drupal_add_css($path . '/system.theme.css', array('group' => CSS_SYSTEM,
'every_page' => TRUE));

  // Ignore slave database servers for this request.
  //
  // In Drupal's distributed database structure, new data is written to the
  // master
  // and then propagated to the slave servers. This means there is a lag
  // between when data is written to the master and when it is available on
  // the slave.
  // At these times, we will want to avoid using a slave server temporarily.
  // For example, if a user posts a new node then we want to disable the slave
  // server for that user temporarily to allow the slave server to catch up.
  // That way, that user will see their changes immediately while for other
  // users we still get the benefits of having a slave server, just with
  // slightly
  // stale data. Code that wants to disable the slave server should use the
  // db_set_ignore_slave() function to set $_SESSION['ignore_slave_server'] to
  // the timestamp after which the slave can be re-enabled.
  if (isset($_SESSION['ignore_slave_server'])) {
    if ($_SESSION['ignore_slave_server'] >= REQUEST_TIME) {
      Database::ignoreTarget('default', 'slave');
    }
  }
  else {
    unset($_SESSION['ignore_slave_server']);
  }
}

// Add CSS/JS files from module .info files.
system_add_module_assets();
}
```

F41.6

Code Comments

Example of code comments.

Document comments follow in part the nomenclature of **Doxxygen**, which is a documentation software generator that can be used with various programming languages. **Doxxygen** pulls information directly from the source file, which simplifies the upkeep of the documentation, given that any change in the code source will result in a direct change of the document comments.

Keep in mind that **we do not create the documentation**, rather that is done directly in the Drupal repository with the contributed modules. We should only concern ourselves with correctly using the nomenclature in the developed themes and modules.

Also remember that when contributing to a module, **code comments should be written in English**, given that is the language used with contributing to the Drupal community.

For detailed information about the use of **Doxxygen**, consult the user manual, available at:

<http://www.stack.nl/~dimitri/doxygen/manual.html>

Also, we recommend visiting the Drupal user manual, available here:

<http://drupal.org/node/1354>

General Syntax

To create **documentation comments**, use the following syntax:

```
/**  
 * Documentation comments  
 */
```

Doxxygen uses a series of commands, called **directives**, that allow you to structure the included comments in such a way that it becomes part of a complete and well-structured code. Doxygen automatically generates links to the files and functions referenced in the documentation. Below, we provide an example of using **Doxxygen** commands in Drupal code: **F41.7**

F41.7

Documentation Comments

Comments focused on documentation. Drupal extracts these comments from modules to generate API documentation.

```
/**  
 * Summary: a short one-line phrase.  
 *  
 * Longer description.  
 *  
 * A blank line used to separate paragraphs.  
 *  
 * @param $first  
 *   Description of method's or function's input parameter.  
 * @param $second  
 *   Don't skip lines between similar parameter types.  
 *  
 * @return  
 *   "@return" Description of the return value, if one exists.  
 */
```

You can generate (HTML) lists of elements by using the following structure: **F41.8**

```
 * @param $variables
 *   An array of associated content:
 *   - tags: An array of tags:
 *     - first: String for the first element.
 *     - last: String for the first element.
 *   - element: Full value to distinguish various pagers.
 *   More information (belonging to the same parameter).
```

F41.8

Documentation Comments

List of documentation elements.

It's important to correctly **indent** list elements, not to leave blank lines between elements and to always leave a space after element identifiers (e.g. after tags `:>`).

The `@see` command is used to generate links to functions, files or URLs. **F41.9**

```
 /**
 * @see system_settings_form_submit()
 * @see forum-list.tpl.php
 * @see http://drupal.org/node/1354
 */
```

F41.9

Documentation Comments

References to other files, functions or URLs.

Documenting Files

As a general rule, include a description of the file at the beginning of it, using the directive `@file`. For example, the `forum.module` file within the Forum module begins with the following descriptive lines: **F41.10**

```
<?php

/**
 * @file
 * Provides discussion forums.
 */
```

F41.10

Documenting Files

The directive `@file` is used in the header of each file to include a description of the file.

Please note that Drupal no longer uses the **outdated format**, typical of pre-Drupal 7 versions, that places the `$Id` parameter at the beginning of the line. **F41.11**

```
<?php
// $Id: forum.module,v 1.448.2.7 2009/06/03 18:27:48 geba Exp $

/**
 * @file
 * Provides discussion forums.
 */
```

F41.11

\$Id

Drupal 7 does not require a line with the `$Id` parameter that was used in the past for file version control.

Actually, this change is not related to Drupal 7, rather it's a change in Drupal's **Version Control System** that took place in 2011. In older versions, **CVS** was used, necessitating the first line to store information regarding each file iteration. The new repository uses **Git**, which doesn't add that information to the file.

Documenting Functions

All functions that can be called up from other files should include comments (it's recommended to document all functions, regardless of the environment in which they will be used).

The commentary block should go directly above the function declaration, without leaving any blank lines. **F41.12**

F41.12

Documenting Functions

Documenting functions should include a description of each parameter and the return value.

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

The first line contains a short description about the function's operations (no more than 80 characters) and should start with a descriptive verb of what it does. In the example, "Verify the syntax of the given e-mail address." That can be followed by a more-detailed description, which can span several lines.

Those descriptions are to be followed by descriptions of the function parameters (**@param**) and finally use the **@return** directive to describe the return value of the function, if it exists.

If the function can be described in just one line, you can use the following, much shorter structure: **F41.13**

F41.13

Documenting Functions

A shortened version of functions documentation.

```
/** 
 * Retrieve output to be displayed in the HEAD tag of the HTML page.
 */
function drupal_get_html_head() {
  $elements = drupal_add_html_head();
  drupal_alter('html_head', $elements);
  return drupal_render($elements);
}
```

Documenting Hooks

Drupal uses functions called **hooks**, that allow communication between modules and the core. In **Unit 43** we will explore in detail how to utilize hooks.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

F41.14 When a function is the implementation of a hook, it may not be necessary to provide more details in the commentary, as those are already defined in the original hook. In these cases it would suffice to give the following description:

F41.14

Documenting Hooks

Example of documentation when the function implements a hook.

```
/** 
 * Implements hook_help().
 */
function forum_help($path, $arg) {
  ...
}
```

Documenting Theme Functions

Functions that can be utilized by themes are called "themeable". It's standard for these functions to be stored in the **themeable** group, using the **@ingroup** directive. This allows for the documentation to show a list of all the functions that can be used by the theme. [F41.15](#)

```
/** 
 * Returns HTML for the Powered by Drupal text.
 *
 * @ingroup themeable
 */
function theme_system_powered_by() {
  return '<span>' . t('Powered by <a href="@poweredby">Drupal</a>', array('@poweredby' => 'http://drupal.org')) . '</span>';
}
```

F41.15

Documenting Theme Functions

Example of functions documentation for theme functions.

Documenting Template Files

The file template is noted by the **@file** directive and a list of **Variables available**, delivered by the preprocessing functions, which are also referenced by the **@see** directives. For example: [F41.16](#)

```
<?php

/**
 * @file
 * Default theme implementation to display a forum which may contain forum
 * containers as well as forum topics.
 *
 * @Variables available:
 * - $forums: The forums to display (as processed by forum-list.tpl.php)
 * - $topics: The topics to display (as processed by forum-topic-list.tpl.php)
 * - $forums_defined: A flag to indicate that the forums are configured.
 *
 * @see template_preprocess_forums()
 * @see theme_forums()
 */

?>
<?php if ($forums_defined): ?>
<div id="forum">
  <?php print $forums; ?>
  <?php print $topics; ?>
</div>
<?php endif; ?>
```

F41.16

Documenting Files

Example of documentation for tpl.php template files.

To see other real-life code examples, we recommend a review of the Drupal core files and modules.

41.3

Code verification with Coder module

The Coder module allows you to analyze the code of any Drupal file, from core code to modules and themes. In addition to analyzing the code syntax, it also inspects the code commentary, which make it essential to generating solid documentation.

The **Coder** module is available at:

[F41.17](http://drupal.org/project/coder)

<http://drupal.org/project/coder>

Once the included **Coder** and **Coder Review** modules are activated, you can access their configuration at:

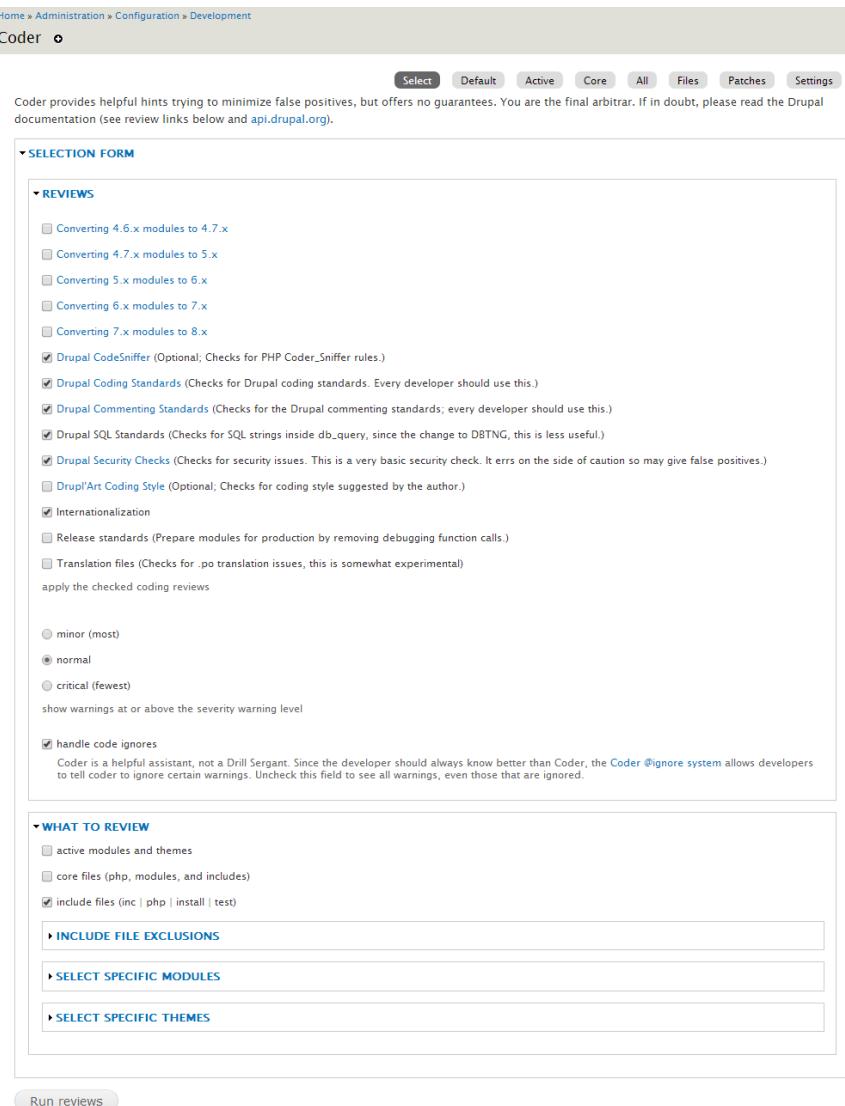
Administration ⇒ **Configuration** ⇒ **Development** ⇒ **Coder**

URL Coder
</admin/config/development/coder>

F41.17

Coder

The Coder module analyzes the code structure and commentary. It's possible to indicate which themes and modules you want inspected.



The screenshot shows the 'SELECTION FORM' section of the Coder configuration. At the top, there are several tabs: 'Select', 'Default', 'Active', 'Core', 'All', 'Files', 'Patches', and 'Settings'. Below the tabs, a note states: 'Coder provides helpful hints trying to minimize false positives, but offers no guarantees. You are the final arbiter. If in doubt, please read the Drupal documentation (see review links below and api.drupal.org).'

REVIEWS

- Converting 4.6.x modules to 4.7.x
- Converting 4.7.x modules to 5.x
- Converting 5.x modules to 6.x
- Converting 6.x modules to 7.x
- Converting 7.x modules to 8.x
- Drupal CodeSniffer (Optional; Checks for PHP Coder_Sniffer rules.)
- Drupal Coding Standards (Checks for Drupal coding standards. Every developer should use this.)
- Drupal Commenting Standards (Checks for the Drupal commenting standards; every developer should use this.)
- Drupal SQL Standards (Checks for SQL strings inside db_query, since the change to DBTNG, this is less useful.)
- Drupal Security Checks (Checks for security issues. This is a very basic security check. It errs on the side of caution so may give false positives.)
- DruplArt Coding Style (Optional; Checks for coding style suggested by the author.)
- Internationalization
- Release standards (Prepare modules for production by removing debugging function calls)
- Translation files (Checks for .po translation issues, this is somewhat experimental)

apply the checked coding reviews

severity level:

- minor (most)
- normal
- critical (fewest)

show warnings at or above the severity warning level

handle code ignores

Coder is a helpful assistant, not a Drill Sergeant. Since the developer should always know better than Coder, the Coder Ignore system allows developers to tell coder to ignore certain warnings. Uncheck this field to see all warnings, even those that are ignored.

WHAT TO REVIEW

- active modules and themes
- core files (php, modules, and includes)
- include files (inc | php | install | test)

INCLUDE FILE EXCLUSIONS

SELECT SPECIFIC MODULES

SELECT SPECIFIC THEMES

Run reviews

From the **Select** tab you can indicate the types of **Reviews** and the elements to check (**What to review**). Of note are the following configuration options:

- **Drupal Coding Standards.** Analyzes the source code syntax.
- **Drupal Commenting Standards.** Analyzes the commentary syntax.
- **Drupal SQL Standards.** Analyzes the syntax of SQL Statements.
- **Select specific modules.** Allows user to select which modules to inspect.
- **Select specific themes.** Allows user to select which themes to inspect.

Run the code analysis in the Select tab by clicking on Run reviews. It will show errors for each file, with caution items in yellow and errors that need correction in red. **F41.18**

▼ MODULES/SIMPLETEST/TESTS/DATABASE_TEST.TEST

database_test.test

- Line -1: @file block missing ([Drupal Docs](#))
- Line 576: missing space after comma

```
>fields('tp', array('name', 'job'))
```

F41.18

Coder Analysis

Example of Coder analysis.

▼ MODULES/SIMPLETEST/TESTS/SCHEMA.TEST

schema.test

- Line 86: table names should be enclosed in {curly_brackets}

```
$this->assertFalse($this->tryInsert(), t('Insert into the old table failed.'));
```
- Line 87: table names should be enclosed in {curly_brackets}

```
$this->assertTrue($this->tryInsert('test_table2'), t('Insert into the new table succeeded.'));
```
- Line 136: The control statement should be on a separate line from the control conditional

```
catch (Exception $e) {}
```
- Line 185: Arrays should be formatted with a space separating each element and assignment operator

▼ MODULES/SIMPLETEST/TESTS/UPDATE.TEST

update.test

- No se hallaron problemas

In addition to the **Select** tab, which allows you to configure the analysis parameters, the module includes other tabs which can run specific, pre-set analyses:

- **Default.** Runs the analysis according to the parameters set in the **Settings** tab.
- **Active.** Inspects the active themes and modules.
- **Core.** Analyzes the modules and themes in the core.
- **All.** Inspects all of the themes and modules, regardless of whether or

not they are active or in the core.

- **Files.** Allows you to choose the files to be reviewed.
- **Patches.** Allows you to analyze patches, either showing the URL or writing the content of the patch.

42 Tools for Developers

In this Unit we will study some modules and tools found in Drupal that are helpful in developing themes and modules. We won't be covering external programming tools like text editors, although those are also useful development tools.

In this Unit we will cover the following modules:

- **Devel**, adds debugging function to modules and themes.
- **Drupal For Firebug**, shows debugging information and SQL scripts directly in Firefox or Chrome web browsers.
- **Drush**, once installed on the server, Drush makes it possible to manage Drupal from the command line.
- **Module Builder**, allows the developer to build a module skeleton or "scaffolding".
- **Examples for Developers**, includes a good number of example modules for developers. We will use some of these examples during the course.

Comparative D7/D6 Tools for Developers

The tools explained in this unit are also available for Drupal 6. Some of these, such as Drush, are new additions to our training plan.

Unit contents

42.1 Devel Module	16
42.2 Debug Drupal in Firefox and Chrome	23
42.3 Drush Module	25
42.4 Module Builder Module	28
42.5 Module Examples for Developers	32



42.1 Devel Module

The **Devel** module adds theme and module debugging functionality for developers. Some of the available functions are:

- Show the database queries run during the page load
- Show the type of page load and memory footprint
- Query and edit site variables
- Execute PHP code
- Direct links to: Execute cron, Rebuild menus, Run the theme registration, Re-install modules, Clear cache, etc.

It's important to keep in mind to **only use Devel during site development**. Once the site is published, it's recommended to de-activate the module.

The Devel module is available at:

<http://drupal.org/project/devel>

It includes the following modules:

- **Devel**, add primary functions.
- **Devel generate**, allows for the creation of dummy users, nodes and taxonomy terminals to use during site development.
- **Devel node access**, shows developer page and block information with relevant node_access records.

In this section we will only review the functions of the main module. Once installed, configure Devel from:

URL Devel

/admin/config/development/devel

Administration ⇒ Configuration ⇒ Development ⇒ Devel settings

Some of the available configuration options are: F42.1

- **Query log**. Shows the database queries run during the current page load. This is useful for detecting **slow page loads**. The list of queries will show at the bottom of the page.
- **XHProf**. Activating the extension PHP XHProf will show information about the slow functions and their memory consumption. We won't be using this in this course.
- **API Site**. By default we use the general Drupal API, available at api.drupal.org.
- **Display page timer**. Shows the page execution time.
- **Display memory usage**. Shows the memory consumed by the current page.
- **Display redirection page**. When the page is redirected to another through drupal_goto(), it shows an intermediate page with the page prior to redirect.

- **Display \$page array.** Shows the contents of the object \$page.
- **Error handlers.** Allows programmer to select the error controls.
- **Krumo display.** Allows for selection of the style to view debugging messages. Krumo is an alternative to the functions print_r() and var_dump(), generally used by developers to create a chart of code variables.
- **Rebuild the theme registry on every page load.** This option is useful when working with templates, whether in themes or modules.
- **Use uncompressed jQuery.** Drupal compresses JavaScript/Jquery contents. Activating this option will show the full, uncompressed code, allowing the reader to debug.

QUERY LOG

Display query log
Display a log of the database queries needed to generate the current page, and the execution time for each. Also, queries which are repeated during a single page view are summed in the # column, and printed in red since they are candidates for caching.

Sort query log
 by source
 by duration
The query table can be sorted in the order that the queries were executed or by descending duration.

Slow query highlighting

Enter an integer in milliseconds. Any query which takes longer than this many milliseconds will be highlighted in the query log. This indicates a possibly inefficient query, or a candidate for caching.

XHPROF
XHProf is a php extension which is essential for profiling your Drupal site. It pinpoints slow functions, and also memory hogging functions.

Enable profiling of all page views and drush requests.
You must enable the [xhprof php extension](#) to use this feature.

API Site

The base URL for your developer documentation links. You might change this if you run [api.module](#) locally.

Display page timer
Display page execution time in the query log box.

Display memory usage
Display how much memory is used to generate the current page. This will show memory usage when devel_init() is called and when devel_exit() is called.

Display redirection page
When a module executes drupal_goto(), the query log and other developer information is lost. Enabling this setting presents an intermediate page to developers so that the log can be examined before continuing to the destination page.

Display \$page array
Display \$page array from hook_page_alter() in the messages area of each page.

Display machine names of permissions and modules
Display the language-independent machine names of the permissions in mouse-over hints on the [Permissions](#) page and the module base file names on the [Permissions](#) and [Modules](#) pages.

Error handlers

None
<input checked="" type="checkbox"/> Standard Drupal
Krumo backtrace in the message area
Krumo backtrace above the rendered page

Select the error handler(s) to use, in case you choose to show errors on screen.

- * None is a good option when stepping through the site in your debugger.
- * Standard Drupal does not display all the information that is often needed to resolve an issue.
- * Krumo backtrace displays nice debug information when any type of error is noticed, but only to users with the [Access developer information](#) permission.

Depending on the situation, the theme, the size of the call stack and the arguments, etc., some handlers may not display their messages, or display them on the subsequent page. Select **Standard Drupal** and **Krumo backtrace above the rendered page** to maximize your chances of not missing any messages.

Demonstrate the current error handler(s): notice+warning, notice+warning+error (The presentation of the error is determined by PHP.)

Krumo display

default

blue

green

orange

white

disabled

Select a skin for your debug messages or select *disabled* to display object and array output in standard PHP format.

Rebuild the theme registry on every page load
While creating new templates and theme_, overrides the theme registry needs to be rebuilt.

Use uncompressed jQuery
Use a human-readable version of jQuery instead of the minified version that ships with Drupal, to make JavaScript debugging easier.

F42.1

Devel

Devel settings.

Sets the default behavior of Devel upon loading the website.

Learn Drupal with Forcontu | Expert in Drupal 7 | Advanced Level

17

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

The **Figure F42.2** shows the functions of the Devel module. In this example the **Query Log**, **Display page timer** and **Display memory usage** are activated. As you'll see, this information is also available to download in the administration area.

Slow queries are highlighted (in red) in the query log. The settings in the Devel configuration determine which functions are considered slow. 5 minutes is the default time.

F42.2

Devel

Example of debugging information shown in Devel.

Page load times and memory consumption and storage are shown. You can also see the list of queries run, and slow queries are highlighted in red.

ms	#	where	ops	query	target
0.14	3	DrupalDatabaseCache::getMultiple	P A E	SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)	default
0.15	3	DrupalDatabaseCache::getMultiple	P A E	SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)	default
0.11	3	DrupalDatabaseCache::getMultiple	P A E	SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)	default
0.35	1	menu_get_item	P A E	SELECT * FROM menu_router WHERE path IN (:ancestors_0, :ancestors_1, :ancestors_2) ORDER BY fit DESC LIMIT 5, 1	default
0.06	2	_update_get_cache_multiple	P A E	SELECT cache_update.cid AS cid, cache_update.data AS data, cache_update.created AS created, cache_update.expire AS expire, cache_update.serialized AS serialized FROM cache_update cache_update WHERE (cache_update.cid LIKE :db_condition_placeholder_0 ESCAPE '\\\'')	default
0.06	1	_update_cache_get	P A E	SELECT data, created, expire, serialized FROM cache_update WHERE cid = :cid	default
0.06	2	_update_get_cache_multiple	P A E	SELECT cache_update.cid AS cid, cache_update.data AS data, cache_update.created AS created, cache_update.expire AS expire, cache_update.serialized AS serialized FROM cache_update cache_update WHERE (cache_update.cid LIKE :db_condition_placeholder_0 ESCAPE '\\\'')	default
0.19	1	system_admin_menu_block	P A E	SELECT mid, menu_name FROM menu_links m1 WHERE m1.router_path = :path AND module = 'system'	default
0.14	3	DrupalDatabaseCache::getMultiple	P A E	SELECT cid, data, created, expire, serialized FROM cache WHERE cid IN (:cids_0)	default

Development Menu and Block

The module creates a **Development** menu and corresponding block. It's recommended to place the block out of the way of the other blocks and site content, like in the **Footer**. The menu is composed of links to various useful tools for site development. Some of these functions come from the Devel module, but many others are available in the Drupal core. **F42.3**

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

F42.3

Menú Desarrollo

Tools included in the Development menu.

The **Development** block has the following options available: **F42.4**

- **Devel settings.** Links to the module's configurations page, reviewed previously.

- Variable editor. Allows for management of the stored variables in the **variable** table or the **\$conf** vector of the configuration file **settings.php**. As we'll see in this level, you can use the `variable_get()` and `variable_set()` variable to create and modify variables directly from the code module.

Home Variable editor

This is a list of the variables and their values currently stored in variables table and the \$conf array of your settings.php file. These variables are usually accessed with `variable_get()` and `variable_set()`. Variables that are too long can slow down your pages.

<input type="checkbox"/>	NAME	VALUE	LENGTH	OPERATIONS
<input type="checkbox"/>	admin_theme	s:5:"seven";	12	Edit
<input type="checkbox"/>	clean_url	s:1:"1";	8	Edit
<input type="checkbox"/>	comment_page	i:0;	4	Edit
<input type="checkbox"/>	cron_key	s:43:"Zi0LoikIE0j9YruqY97RaiU6t40JLCnnnNC_ElbXQs";	51	Edit
<input type="checkbox"/>	cron_last	i:1393062306;	13	Edit
<input type="checkbox"/>	css_js_query_string	s:6:"n1e7ao";	13	Edit
<input type="checkbox"/>	date_default_timezone	s:13:"Europe/Madrid";	21	Edit
<input type="checkbox"/>	devel_api_url	s:14:"api.drupal.org";	22	Edit
<input type="checkbox"/>	devel_error_handlers	a:1:{i:1;s:1:"1";}	18	Edit
<input type="checkbox"/>	devel_execution	s:1:"5";	8	Edit

F42.4

Editor de variables

From the variables editor you can query and edit the registered variables in the table.

- **Run cron.** Direct access to run cron. Once executed, the system redirects to the status report.
- **Execute PHP code.** Allows you to execute PHP code.
- **Menu item.** Shows the options associated with page routing (route, access, arguments, etc.). As we'll see in **Unit 46**, in Drupal the menus system has a dual role: to control the navigation through menus and to route each web page.
- **Function reference.** Shows a list of API functions. From the list we can directly access the available documentation for each API function in Drupal.
- **Hook_elements().** Shows the types of formula elements added by modules and their default settings. We will study the use of **API Forms** in **Unit 47**.
- **Entity info.** Shows information on the available site entities. The initial entities are comment, file, node, taxonomy_term, taxonomy_vocabulary and user. Other entities of the installed modules may appear in this list.
- **PHPInfo().** Shows the configuration of PHP executing the `phpinfo()` function.
- **Rebuild menus.** Rebuilds the menu and page routing system.
- **Theme registry.** Shows the elements that make up the theme registry.
- **Reinstall modules.** Allows re-installation of modules. Keep in mind that when re-installing a module, it should be completely removed, eliminating the associated tables and values. This operation is very useful during module development.

- **Clear cache.** This clears all cache from the site. This operation is similar to the one available through Render.
- **Session viewer.** Shows information related to the session (variable `$_SESSION`).

Debugging Functions

Aside from the seen functions, the Devel module include a collection of **debugging functions** that can be used in the module code or theme code to get the value of certain variables.

In-depth information about Drupal's debugging variables can be found here:
<https://ratatosk.backpackit.com/pub/1836982-debugging-drupal>

Some of these functions are:

- The **dpm()** function prints the value of a variable in a pre-set area of a Drupal message (`drupal_set_message()`). To print the variable internally, use **print_r()**.

```
dpm($input, $name = NULL);
```

When running the function, set the parameters of the variables whose value you want to see, along with an identifying name (optional). It will only print the variable value, not the name, so it's convenient to specify the name in the parameter.

Keep in mind that the variable value matches the point where the functions **dpm()** is called. To run the sequence of one variable at separate points of the same script we can include various **dpm()** function calls. **F42.5**

F42.5

dpm() Function

The **dpm()** debugging function prints the value of select variables in a pre-set area of a Drupal message.

```
$price = "11.25";
dpm($price, "Price value");

$site_name = variable_get("site_name");
dpm($site_name, "Site name");
```



Price value => 11.25

Site name => Expert in Drupal 7 - Advanced

- The **dvm()** function is similar to **dpm()**, but it uses **var_dump()**, instead of **print_r()**, to print the variable value.
- The **dprint_r()** function prints the content of a vector. Although you can input a number of parameters, the only one you must have is the array. **F42.6**

```
dprint_r($input, $return = FALSE, $name = NULL, $function = 'print_r',
        $check = TRUE);
```

```
$countries = array("es" => array("name" => "Spain",
                                    "currency" => "Euro"),
                   "us" => array("name" => "United States",
                                "currency" => "US dollar"),
                   "uk" => array("name" => "United Kingdom",
                                "currency" => "Pound sterling")
                 );
dprint_r($countries);
```

F42.6**dprint_r() Function**

The debugging function `dprint_r()` prints the contents of a vector.

```
Array
(
    [es] => Array
        (
            [name] => Spain
            [currency] => Euro
        )

    [us] => Array
        (
            [name] => United States
            [currency] => US dollar
        )

    [uk] => Array
        (
            [name] => United Kingdom
            [currency] => Pound sterling
        )
)
```

- The **dpr()** function is an alias of **dprint_r()**, and it uses the default values to print (**print_r()**). For that reason, it only accepts these parameters:
`dpr($input, $return = FALSE, $name = NULL);`
- The **dvr()** function is similar to **dpr()**, but it uses **var_dump()**:
`dvr($input, $return = FALSE, $name = NULL);`
- The **kpr()** function uses **Krumo**, instead of **print_r()** or **var_dump()**, to print the variable value. Krumo displays the debugging messages in its particular style.
- The **dargs()** function prints old arguments to the current function. This is very useful to learn which argument the function or hook uses to interact with the site.

To use this function you should make a call from within the function from which you want to know the arguments. It doesn't require input parameters.

- The function **dd()** places all the variables in a file named "drupal_debug.txt" in the temporary site directory.

- The **ddebug_backtrace()** function prints the battery of functions that were run to load the page. This does not require any parameter. **F42.7**

F42.7

ddebug_backtrace() Function

The debugging function **ddebug_backtrace()** shows the battery of functions called to load the page.

```
... (Array, 16 elements)
16: eval (Array, 3 elements)
15: php_eval (Array, 4 elements)
14: check_markup (Array, 4 elements)
13: _text_sanitize (Array, 4 elements)
12: text_field_formatter_view (Array, 4 elements)
11: field_default_view (Array, 4 elements)
10: _field_invoke (Array, 4 elements)
9: _field_invoke_default (Array, 4 elements)
8: field_attach_view (Array, 4 elements)
7: node_build_content (Array, 4 elements)
6: node_view (Array, 4 elements)
5: node_view_multiple (Array, 4 elements)
4: node_show (Array, 4 elements)
3: node_page_view (Array, 2 elements)
2: call_user_func_array (Array, 4 elements)
1: menu_execute_active_handler (Array, 4 elements)
```

Debug Drupal in Firefox and Chrome

42.2

The **Drupal For Firebug** module provides the developer with debugging information and active SQL sequences that show right in the Firefox or Chrome web browser windows.

The Drupal For Firebug module is available at:

<http://drupal.org/project/drupalforfirebug>

In addition to installing and activating the module, you should **install the Firefox or Chrome plugins**. In both cases, open the corresponding web browser and use the URL to access the plugin. Then install it.

- **Firefox plugin (requires Firebug and Drupal for Firebug):**
<https://addons.mozilla.org/es-es/firefox/addon/firebug/>
<https://addons.mozilla.org/en-US/firefox/addon/drupal-for-firebug/>
- **Chrome plugin:**
<https://chrome.google.com/webstore/detail/imlijcpfmhmifofiihbofoamohkdbblc>

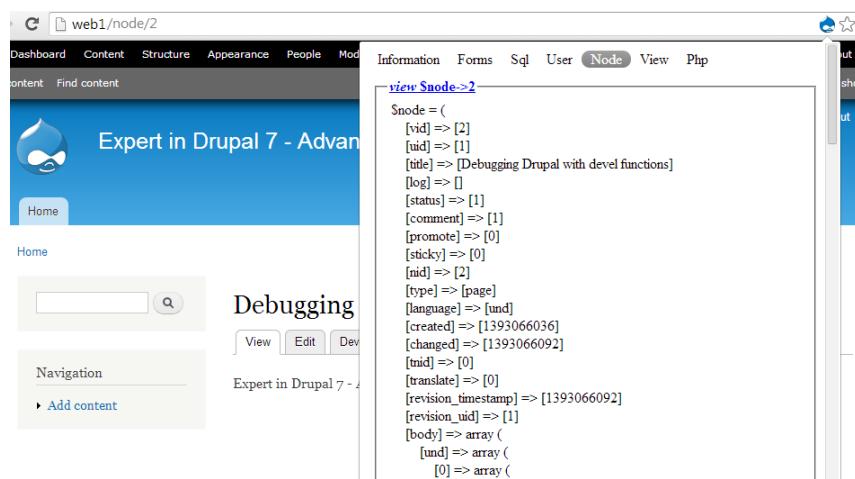
We should also be sure that the site's active theme uses the **\$closure** variable.

Debugging in Chrome

Once installed, the web browser will show a **Drupal icon** whenever you're loading a Drupal page that has access to the debugging options (**Devel** and **Drupal For Firebug** modules)

Clicking on the Drupal icon will open up a window with several options: Information, Forms, Sql, Users, Node, View and Php.

For example, if you're loading a node, the Node tab will show all the information related to the **\$node** variable. **F42.8**



The screenshot shows a Drupal 7 node page for 'Expert in Drupal 7 - Advanced'. The browser window has a Firebug extension open. The title bar says 'C web1/node/2'. The Firebug interface shows the \$closure variable for the \$node object. The code is as follows:

```

$node = (
  [vid] => [2]
  [uid] => [1]
  [title] => [Debugging Drupal with devel functions]
  [log] => []
  [status] => [1]
  [comment] => [1]
  [promote] => [0]
  [sticky] => [0]
  [nid] => [2]
  [type] => [page]
  [language] => [und]
  [created] => [1393066036]
  [changed] => [1393066092]
  [uid] => [0]
  [translate] => [0]
  [revision_timestamp] => [1393066092]
  [revision_uid] => [1]
  [body] => array (
    [und] => array (
      [0] => array (
        ...
      )
    )
  )
)

```

F42.8

Drupal For Firebug in Chrome

The module allows direct debugging with Chrome. The Figure shows the content of **\$node** variable when loading a node.

After activating the SQL Query Log in the configuration panel of Devel, the queries will show on the SQL tab of the web browser plugin instead of at the bottom of the page. **F42.9**

F42.9

Drupal For Firebug with Chrome

Queries executed upon page load are shown in Drupal For Firebug. After activating this option the queries will not show at the bottom of the page.

```

SQL Query Log
Executed 50 queries in 5.12 ms.
ms # where query
0.15 4 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)
0.18 4 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)
0.11 4 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)
SELECT * FROM menu_router WHERE path IN (ancestors_0, ancestors_1, ancestors_2) ORDER BY fit DESC LIMIT 0, 1
0.06 1 menu_get_item SELECT cid, data, created, expire, serialized FROM cache WHERE cid IN (:cids_0)
0.06 5 DrupalDatabaseCache::getMultiple
  
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Debugging in Firefox

In Firefox, activate the **Firebug** plug-in from the Tools tab. Along with the rest of the Firebug options that allow you to analyze content from HTML, CSS, etc., you'll see a new tab for Drupal, with the same options as the Chrome plugin. The only difference is that this options bar appears at the bottom of the page. **F42.10**

F42.10

Drupal For Firebug in Firefox.

Once Firebug is activated, Drupal's debugging window will show at the bottom of the page in the Firefox web browser.

```

SQL Query Log
Executed 299 queries in 439.73 ms.
ms # where query
0.17 4 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)
0.31 1 language_list SELECT * FROM languages ORDER BY weight ASC, name ASC
0.44 4 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)
0.34 4 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache_bootstrap WHERE cid IN (:cids_0)
0.89 12 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache WHERE cid IN (:cids_0)
0.54 1 drupal_lookup_path SELECT source FROM url_alias WHERE alias = :alias AND language IN (language, language_none) ORDER BY language ASC, pid DESC
11.07 4 lock_acquire INSERT INTO semaphore (name, value, expire) VALUES (:db_insert_placeholder_0, :db_insert_placeholder_1, :db_insert_placeholder_2)
0.15 12 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache WHERE cid IN (:cids_0)
0.67 43 locale SELECT s.id, t.translation, s.version FROM locales_source s LEFT JOIN locales_target t ON s.id = t.id AND t.language = :language WHERE s.source = :source AND s.context = :context AND s.textgroup = 'default'
0.55 38 locale UPDATE locales_source SET version = :db_update_placeholder_0 WHERE (id = :db_condition_placeholder_0)
0.62 42 DrupalDatabaseCache::clear DELETE FROM cache WHERE (cid LIKE :db_condition_placeholder_0_ESCAPE '\')
0.54 43 locale SELECT s.id, t.translation, s.version FROM locales_source s LEFT JOIN locales_target t ON s.id = t.id AND t.language = :language WHERE s.source = :source AND s.context = :context AND s.textgroup = 'default'
0.15 38 locale UPDATE locales_source SET version = :db_update_placeholder_0 WHERE (id = :db_condition_placeholder_0)
0.15 42 DrupalDatabaseCache::clear DELETE FROM cache WHERE (cid LIKE :db_condition_placeholder_0_ESCAPE '\')
0.16 12 DrupalDatabaseCache::getMultiple SELECT cid, data, created, expire, serialized FROM cache WHERE cid IN (:cids_0)
0.43 5 node_types_build SELECT nt." FROM node_type nt WHERE (disabled = :db_condition_placeholder_0) ORDER BY nt.type ASC
0.05 6 DrupalDatabaseCache::set SAVEPOINT savepoint_1
0.13 6 DrupalDatabaseCache::set SELECT 1 AS expression FROM cache cache WHERE ((cid = :db_condition_placeholder_0)) FOR UPDATE
0.26 6 DrupalDatabaseCache::set INSERT INTO cache (cid, serialized, created, expire, data) VALUES (:db_insert_placeholder_0, :db_insert_placeholder_1, :db_insert_placeholder_2, :db_insert_placeholder_3, :db_insert_placeholder_4)
0.05 6 MergeQuery::execute RELEASE SAVEPOINT savepoint_1
0.43 43 locale SELECT s.id, t.translation, s.version FROM locales_source s LEFT JOIN locales_target t ON s.id = t.id AND t.language = :language WHERE s.source = :source AND s.context = :context AND s.textgroup = 'default'
0.14 38 locale UPDATE locales_source SET version = :db_update_placeholder_0 WHERE (id = :db_condition_placeholder_0)
  
```

Drush Module

42.3

Drush is not a Drupal module, rather it's a tool that, installed on the server, allows the developer to manage Drupal from the command line.

Drush comes with a collection of commands that conduct specific operations on the web site, and it includes an API that allows other modules to add new commands.

In order to use Drush you have to have access to the server's command line (generally via the SSH). For that reason it's not possible to use just any web hosting. If you don't have access to a server with SSH, check with your web-hosting provider to see if you can have access to SSH.

Drush is available at:

<http://drupal.org/project/drush>

Drush installation varies according to the operating system.

To install Drush on Windows, follow the steps described in: <http://drupal.org/node/594744>.

A Windows installer can be found at: <http://www.drush.org/resources>.

To install Drush on a Linux server, follow these steps:

- Download and unzip **Drush**, uploading the **drush** folder to the root folder in Drupal.
- Compare the **drush** and **drush.php** files within the **drush** folder to ensure that the adequate read and write permissions (755).

With the following steps you can launch drush from within the drush file, executing **./drush commands**.

- To avoid using **./** and be able to use drush from any (user) folder, you will create a symbolic link to the user's bin folder (**/home/user/bin**). From the site root:

In **-s drush/drush /home/user/bin/drush**
- You can also install Drush and make it available across the server by locating the drush folder in **/usr/local/lib** and creating the link:

In **-s /usr/local/lib/drush/drush /usr/local/bin/drush**

To verify that Drush was installed correctly, run the command: **drush core-status**, which shows the installation status. **F42.11**

F42.11**core-status Drush Command**

The Drush command **core-status** shows general information about the Drupal installation.

```
$ drush core-status
Drupal version : 7.10
Site URI : http://default
Database driver : mysql
Database hostname : localhost
Database username : example_user
Database name : example_bd
Database : Connected
Drupal bootstrap : Successful
Drupal user : Anónimo
Default theme : bartik
Administration theme : seven
PHP configuration : /usr/local/lib/php.ini
Drush version : 4.4
Drush configuration :
Drush alias files :
Drupal root : /home/example/public_html
Site path : sites/default
File directory path : sites/default/files
Private file directory path : sites/default/files/private
```

Drush is ready to be used, but you can modify the configuration by adding a **drushrc.php** file.

The configuration file **drushrc.php** is composed of the **\$options** vector with global configuration options, the **\$command_specific** vector for specific commands, and the **\$override** vector to overwrite values in the **variable** table.

Check the **file configurations example** in:

drush/examples/example.drushrc.php

To add a specific configuration, copy and rename the example file to one of the valid locations (e.g. sites/default/).

Drush Commands

The complete list of commands for the Drush core can be found in:

<http://drush.ws/help>

Here are some of the important commands:

- **core-status**. Shows a summary of the Drupal installation.
- **cache-clear**. Empties a specific cache, or all Drupal caches. If the type of cache isn't indicated, the system will show all the available options from which to select.
- **core-cron**. Runs a system cron.
- **drupal-directory**. Return path to the Drupal installation, a module or theme directory.
- **variable-get**, **variable-set** and **variable-delete**. Operations for variables.
- **watchdog-show**. Shows watchdog messages.
- **pm-list**. Shows a list of available modules and themes.

- **sql-query.** Executes a query against the site database.
- **user-information.** Print information about one or more users.
- **user-add-role, user-remove-role.** Add or delete the roles of various users.

Modules that Integrate with Drush

Modules that add new Drush commands use the **hook_drush_command()** hook, typically in the **module.drush.inc** file. One way to find the commands available is to access the function used in each module.

Drupal 7 has more than 50 modules that use the Drush API and add additional commands. To find these modules, search for "Drush" (and "Version 7.x") at drupal.org.

<http://drupal.org/project/modules?filters=tid%3A4654>

Some of these modules are:

- **Backup and Migrate.** Adds commands to manage backups. Commands: bam-backup, bam-restore, bam-destinations, bam-sources, bam-profiles, bam-backups.
- **XML sitemap.** Adds commands to manage XML site maps. Commands: xmlsitemap-regenerate, xmlsitemap-rebuild, xmlsitemap-index.
- **Devel.** Adds debugging commands. Commands: devel-download, devel-reinstall, fn-hook, fn-view, devel-token.
- **Elysia Cron.** Adds specific commands related to cron jobs. Command: elysia-cron.
- **Drush language commands.** Adds specific commands to enable, disable, or specify the default language. Commands: language-enable, language-disable, language-default.
- **Drush Role.** Includes commands to add or delete role permissions. Commands: role-add-perm, role-remove-perm.

42.4

Module Builder Module

The **Module Builder** module allows the developer to build a skeleton or scaffold from which to build a module.

The Module Builder Module is available at:

http://drupal.org/project/module_builder

Although we will make a detailed study of module development in the coming units, we introduce the Module Builder here as it's useful to have from the beginning.

Access the module configuration from:

Administration ⇒ Configuration ⇒ Development ⇒ Module builder

From the Options tab you can configure the following general parameters:

URL Configure Module builder
 /admin/config/development/module_builder

- **Path to hook documentation directory.** The system will download Drupal hooks that will later be used to generate the structure of the modules, which is helpful for the developer.
- **Path to write module files.** The code generated by the Module Builder module is stored in the path indicated, within the **files** folder that does not match the final path of where the modules are stored.
- **Module header and footer.** Allow you to add written code before and after the code in each file.
- **Code detail level:**
 - **Beginner.** If we select Beginner it will add helpful, detailed text to the code. We should select this option, as we are just getting started in module development.
 - **Advanced.** Even for advanced developers, Module Builder can continue to be a good tool to simplify the first steps in module creation. For advanced users, the comments may be disabled by selecting Advanced.

If you click on the **Update Hooks** tab, you will see an opening message that says **The hook documentation has not yet been downloaded**. Click on **Update** to continue. Keep in mind that it will only include the hooks from the installed modules, so that you will have to go back and activate it when you install new modules on the site (this is only necessary if you're going to make use of the hooks from the developed modules).

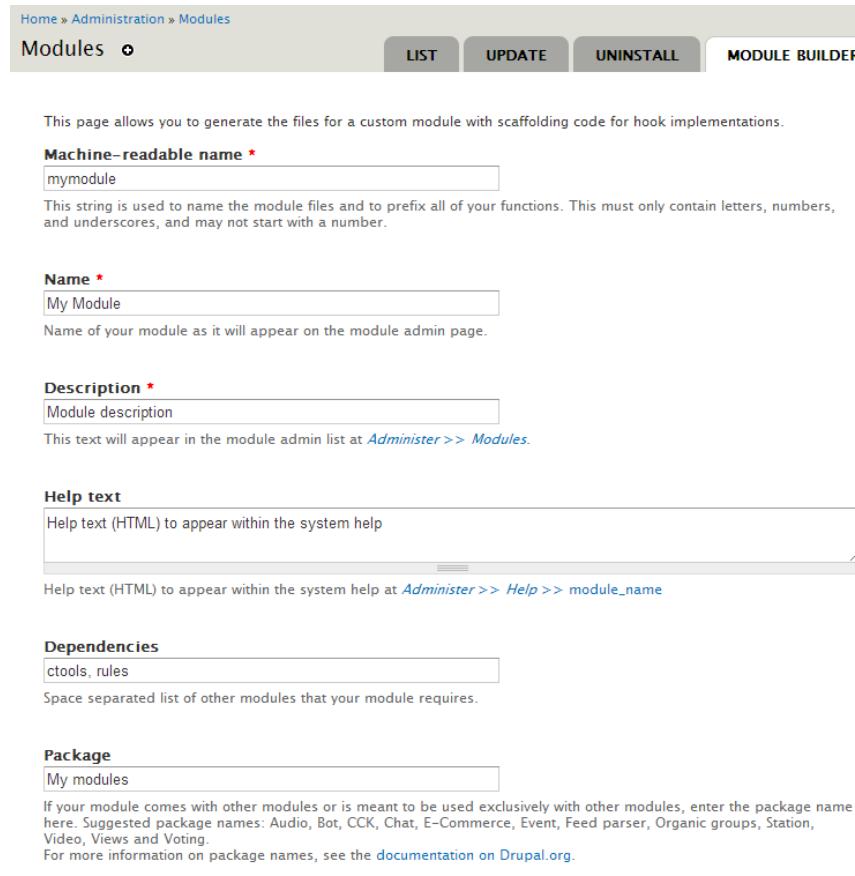
Once the configuration is complete you can create a new module.

Create Module Files with Module Builder

To create a new skeleton or base for a module, go to:

Administration ⇒ Modules [Module Builder tab]

URL Module builder
/admin/modules/module_builder



This page allows you to generate the files for a custom module with scaffolding code for hook implementations.

Machine-readable name *
 This string is used to name the module files and to prefix all of your functions. This must only contain letters, numbers, and underscores, and may not start with a number.

Name *
 Name of your module as it will appear on the module admin page.

Description *
 This text will appear in the module admin list at [Administer >> Modules](#).

Help text
 Help text (HTML) to appear within the system help at [Administer >> Help >> module_name](#)

Dependencies
 Space separated list of other modules that your module requires.

Package
 If your module comes with other modules or is meant to be used exclusively with other modules, enter the package name here. Suggested package names: Audio, Bot, CCK, Chat, E-Commerce, Event, Feed parser, Organic groups, Station, Video, Views and Voting. For more information on package names, see the [documentation on Drupal.org](#).

To generate the skeleton for a new module, set the following parameters:

F42.12

- **Name, Machine-readable name** and **Description**.
- **Help text**. Includes HTML and will show on the help page.
- **Dependencies**. List of other, required modules. Indicate the names of the system of these modules, separated by commas.
- **Package**. Allows the developer to group the modules within the module administration list. Modules can be added to an existing package or have their own.

From the same administration page, you must select the hooks that you want to add to the module skeleton. **F42.13**

You can use presets or groups of presets, in the functionalities function that we will run. For example:

- Define a new type of content.
- Interact with nodes.

F42.12

Module Builder.
Create module

Example of creating a new module with Module Builder. When starting, you must indicate the name, module system name, description, help text, dependent modules and the package where it is located.

- Add blocks.
- Define the field types.

You can also review the hooks for each module, the core as well as contributions, and select particular hooks you want to include. In the following units we will analyze these hooks in detail.

F42.13

Module Builder.

Create module

Before creating the module, select the hooks that you want to add to the skeleton.

HOOK PRESETS

Selecting one or more of these features will automatically select appropriate hooks for you.

Define your own content type (node)

Interact with nodes

Add blocks to your module

Define a field type

ADMIN_MENU HOOKS

AGGREGATOR HOOKS

BLOCK HOOKS

- block_configure**
Define a configuration form for a block.
- block_info**
Define all blocks provided by the module.
- block_info_alter**
Change block definition before saving to the database.
- block_list_alter**
Act on blocks prior to rendering.
- block_save**
Save the configuration options from hook_block_configure().
- block_view**
Return a rendered or renderable view of a block.
- block_view_MODULE_DELTA_alter**
Perform alterations to a specific block.
- block_view_alter**
Perform alterations to the content of a block.

COLORBOX HOOKS

COMMENT HOOKS

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Once the hooks are selected, click on the **Generate** button, and the system will create the module skeleton, made of two files: **F42.14**

- **.module**. Includes all the functions defined in the module.
- **.info**. Includes the definition of the module.

For example, for the module named mymodule, the files **mymodule.module** and **mymodule.info** will be created.

The system creates a folder with the module name and the .module and .info files in the location indicated in the Module Builder configuration. The default location is **/sites/default/files/modules**.

To use this module, copy the folder to the location where the installed modules are stored (e.g. sites/all/modules). Once that is complete the module will appear in the Modules administration area. You can activate it, although initially it won't create any actions.

You can also copy and place the generated code in the corresponding files, for which you will have to manually create the folder for the module and its files.

This page allows you to generate the files for a custom module with scaffolding code for hook implementations. Please copy and paste the following text into a file called mymodule.info.

mymodule.info code

```
name = My Module
description = Module description
dependencies[] = ctools
dependencies[] = rules
package = My modules
core = 7.x
```

Please copy and paste the following text into a file called mymodule.module.

mymodule.module code

```
<?php

/**
 * @file mymodule.module
 * TODO: Enter file description here.
 */

/**
 * Implements hook_help().
 */
function mymodule_help($path, $arg) {
  switch ($path) {
    // Main module help for the block module
    case 'admin/help#block':
      return '

t(Blocks are boxes of content rendered into an area, or region, of a web page. The default theme Bartik, for example, implements the regions "Sidebar first", "Sidebar second", "Featured", "Content", "Header", "Footer", etc., and a block may appear in any one of these areas. The <a href="@blocks">blocks administration page</a> provides a


```

F42.14

Skeleton generated by Module Builder

The skeleton created by Module Builder includes the code files .info and .module.

You can copy this code and paste it in the corresponding empty files, or use the files created in the path indicated in the module configuration.

It's important to keep in mind, especially when using the manual creation process for module files, that the code format should be set to UTF-8 without BOM. Selecting the code format varies according to the type of text editor or code. For example, when using **Notepad++**, you can change the code format from the **Format** tab.

Now that you completed the set up process, the Module Builder will not be needed for further module development. Module Builder is only useful at the beginning of the process for generating the module skeleton, but the rest of the work can be done using the initial files that were created. Before continuing with module development it's necessary to learn some skills from the upcoming Advanced Level units.

Drush Commands

Module Builder includes Drush commands that run processes similar to those carried out through the user interface. Some of the commands include:

- **mb-build.** Generate the code (skeleton) for a new Drupal module, including file headers and hook implementations.
- **mb-download.** Downloads and runs the hook documentation.
- **mb-list.** Shows the list of known hooks from the Module Builder.
- **mb-analyze.** Shows the list of hooks from a select module.

42.5 Examples for Developers Module

The **Examples for Developers** module provides a good selection of example modules for developers. The module is under active development, so the version is unstable. You can download the most-recent version (7.x-1.x-dev) at:

<http://drupal.org/project/examples>

When getting started, it's not necessary to activate the example modules included since we are more interested in revising the code.

If you want to make and test modifications, the best thing to do is to create a new module of your own and copy the code from the example module into it, making code changes as you see fit. Once you have your module, you can activate it and continue working on the implementation.

Some of the sample modules included in **Examples for Developers** are:

- **action_example.** Create actions.
- **ajax_example.** Use of AJAX.
- **block_example.** Define blocks.
- **cron_example.** Use of hook_cron().
- **dbtng_example.** Operations within the database.
- **email_example.** Send email.
- **field_example.** Define fields.
- **file_example.** Manage files.
- **filter_example.** Define input filters.
- **form_example.** API forms examples.
- **image_example.** Image style.
- **js_example.** Javascript examples and use.
- **menu_example.** Manipulate menu links.
- **node_access_example.** Rules for accessing content.
- **node_example.** Create new types of content.
- **nodeapi_example.** Modify existing content.
- **page_example.** Create static pages.
- **pager_example.** Use of pagination.
- **render_example.** Use of the render function.
- **simpltest_example.** Functions to test modules.
- **theming_example.** Functions related to themes and templates.
- **token_example.** Tokens application.
- **trigger_example.** Run actions and triggers.
- **vertical_tabs_example.** Use of vertical tabs.

We will use some of these modules to illustrate the distinct functionality that can be found at the Advanced Level.

43 Drupal Architecture

In the previous levels of this course we reviewed the main components of the Drupal architecture: core, entities, content types, modules, blocks, menus, themes, users and permissions. In order to develop our own modules and themes for Drupal, it's necessary to deepen our understanding of the system functions and architecture.

We already know that **Drupal is a modular framework**. The core of the system is complemented by certain required modules, and some optional ones, that come with the Drupal distribution. We have also learned that one can download and install additional modules that are developed and shared freely by members of the Drupal community of developers and users.

In this level we will learn how to create our own modules using the best practices for Drupal development. To that end, let's learn more about Drupal architecture at the programming level.

Comparative D7/D6

Drupal Architecture

The main change at the architectural level in Drupal 7 is the concept of the entity and the files associated with entities.

In Drupal 7 the node ceases to be the center of attention, becoming instead another entity type, as are users, comment and taxonomy terms. You can assign field to entities and manage their appearance.

Unit contents

43.1 Drupal 7 Requirements	34
43.2 Modules and hooks	35
43.3 Drupal API.....	37
43.4 File Structure	39
43.5 Operation of Drupal	41
43.6 Drupal Architecture	43



43.1 Drupal 7 Requirements

Drupal 7 can be installed and run on any system that supports **PHP** (Windows, Linux, Mac OS X, etc.). PHP is an interpreted programming language designed specifically for creating dynamic web pages. **PHP is integral to the development of Drupal** and the modules that extend the program's functionality, and as we will learn to create in this level, should also be used for development. For that reason, it's necessary to learn or have previous knowledge of PHP to continue with this Advanced Level.

Web Server

To run PHP, and for that matter, Drupal, a **Web server** is needed. The most-used Web servers are **Apache** and **Microsoft IIS**. The Web server takes care of interpreting the PHP code, generating the corresponding HTML page and showing the user the requested information in their web browser.

Drupal 7 can be installed on Apache 1.3, Apache 2.x and Microsoft IIS.

With Apache, we also use the **mod_rewrite** extension to activate clean URLs.

Use PHP version 5.2.5 or higher. Version 5.3 is recommended.

The minimum space recommended for a Drupal installation is 15MB, although this amount will increase as modules are added. The size needed depends on the multimedia content featured on the web site (i.e. images, documents, videos, etc.).

Database Server

Drupal supports several database managers, such as MySQL, SQLite, PostgreSQL and Oracle. This is achieved through a **database extraction layer** that converts the generic instructions given by Drupal to the specific needs of each database. This method makes it possible to change the database manager without needing to change the programming code of the Drupal core or contributed modules.

In this course we will work with the MySQL database, although, as we have already noted, the queries are made through an abstraction layer, so that for our studies, any database would do.

Drupal 7 requires an installation of the PDO of PHP extension, which adds a collection of functions that allow for access to the database.

If you need more information about the requirements for a Drupal 7 installation, please see this URL:

<http://drupal.org/requirements>

Modules and Hooks

43.2

We already know that **Drupal is a modular framework**. The core of the system is complemented by certain required modules, and some optional ones, that come with the Drupal distribution. We have also learned that besides these modules, one can download and install additional modules that are developed and shared freely by members of the Drupal community of developers and users. In this level we will study how to create our own modules using best practices for Drupal development.

The system runs the modules only when needed, and for that it's necessary that each module has a way to **communicate what to do and when to do it**. This is achieved through a group of functions called **hooks**.

To better understand how hooks operate we will look at some examples of available hooks: **F43.1**

Function (hook)	Description
<code>hook_block_info()</code>	This hook defines the blocks generated by the module.
<code>hook_enable()</code>	This hook is active when the module is activated. Therefore, all the actions defined within the function will be executed when the module is activated.
<code>hook_schema()</code>	Defines the database schema implemented by the module (for example, new tables or additional fields in the existing tables). This hook is called during module installation (and de-installation) to indicate which tables and/or fields belong to it to create them in the database (or eliminate them during de-installation of the module).

F43.1

Hooks

Some of the hooks found in Drupal.

But, **how do we use the hooks in a specific module?** It's very simple, we only have to create a function that implements the desired actions and that is named the same as the **hook** but with the module name in place of the prefix **hook_**.

For example, if our module is named **mymodule**, the hooks in the above figure would be named: **mymodule_block_info()**, **mymodule_enable()** and **mymodule_schema()**, respectively.

We can see this with a little code. We learned in previous units that when creating a menu in the website, it automatically generates a block, and later that block can be activated and configured just like any other block.

The **Figure F43.2** shows how the **Menu** module (in the core) implements a `hook_block_info()` (function `menu_block_info()`) to achieve this objective. This code is available in the `/modules/menu/menu.module` file, where you can find other available hooks in the module.

At this point there is no need to study the function's internal code. It's enough to know how to form and use the hook functions within a module.

```
/**
 * Implements hook_block_info().
 */
function menu_block_info() {
  $menus = menu_get_menus(FALSE);
```

F43.2

Implement a Hook

Example of implementation of a function hook.

```

$blocks = array();
foreach ($menus as $name => $title) {
  // Default "Navigation" block is handled by user.module.
  $blocks[$name]['info'] = check_plain($title);
  // Menu blocks can't be cached because each menu item can have
  // a custom access callback. menu.inc manages its own caching.
  $blocks[$name]['cache'] = DRUPAL_NO_CACHE;
}
return $blocks;
}

```

When implementing a function hook in a module, identify it with the comment "Implements hook_...().", ending the line with a period (.). This standard is necessary to generate the module documentation and so that other developers can adequately identify it.

```

/**
 * Implements hook_hookname().
 */

```

Keep in mind that **in Drupal 6 we use the string "Implementation of"**, which has been changed to "Implements" in Drupal 7.

In the menu_block_info() function, the hook_block_info() command is carried out in the following steps:

- The function of the Drupal API **menu_get_menus()**, facilitated by the **Menu** module, returns a list with all the menus created on the site. This list is stored in the \$menus variable.
- Using the name or title of each menu, the vector **\$blocks** is created, which only contains the list of blocks that the module generated (name of the block and if it is cached or not).
- The function returns the list of blocks(**\$blocks**) that the module generates.

When we access the blocks administration area, the system has previously obtained a list of all the blocks on the site. This list is compiled by searching in all the installed modules for the functions that use **hook_block_info()**.

Finally, it's important to mention that **it is not necessary for a module to have all the available hooks defined**. We only need to create those that are needed to provide some new functionality or to modify some function available in another module or in the system.

As we saw in **Unit 42**, the **Module Builder** allows us to build a skeleton for a module using the hooks that we selected.

Drupal API

43.3

In the previous example we make use of the **menu_get_menus()** function, that returns an associated vector with the names of the available menus on the site. This function forms part of a group of functions that make up the **Drupal API**.

An **API (Application programming interface)** is a library of functions that are available to assist developers in accessing certain data from a system in abstract form, without needing to know how the system is organized internally. This also allows the system to make internal modifications without affecting the applications that are using the API function, as long as the entry and exit parameters remain the same.

The Drupal API can be found at <http://api.drupal.org>. The **Figure F43.3** shows the information available in Drupal API for the **menu_get_menus()** function.

The screenshot shows the Drupal API documentation page for the `menu_get_menus()` function. The page has a blue header with the Drupal logo and navigation links for "Drupal Homepage", "Your Dashboard", "Logged in as forcontu", and "Log out". Below the header, there's a "Community Documentation" section with "Docs Home" and "API" links. The main content area starts with the URL "Drupal 7 > menu.module" and the function name "menu_get_menus". A red box highlights the first line of the code: "6 menu.module menu_get_menus(\$all = TRUE)". The code block continues with "7 menu.module menu_get_menus(\$all = TRUE)" and "8 menu.module menu_get_menus(\$all = TRUE)". Below the code, it says "Return an associative array of the custom menus names." The "Parameters" section notes that \$all If FALSE return only user-added menus, or if TRUE also include the menus defined by the system. The "Return value" section states that it's an array with machine-readable names as keys and human-readable titles as values. A blue link "► 9 functions call menu_get_menus()" is shown. The "Code" section displays the PHP code from modules/menu/menu.module, line 784. To the right, there's a sidebar titled "Drupal API" with sections for "Search 7", "API Navigation", and links to "Drupal 7", "Constants", "Classes", "Files", "Functions", "Globals", and "Topics". A note in the sidebar says: "Always be sure to use the documentation that corresponds with the correct version of Drupal (6, 7 or 8)."

The standard documentation for APIs contains the following types of information:

- Function definition for the **different versions of Drupal** (6, 7 and 8). It's important to use the correct version (Drupal 7).
- **Function description.** Text that describes the function and the purpose of the function.
- **Parameters.** Description of the function parameters.
- **Return value.** If the function returns a value, it tells the type and content of the returned value.
- **Functions that use this function** (x calls t...). List of functions in the core that call the function described. Looking at these functions can be very useful to study real examples of function calls.
- **File.** Name of the module and file that use the function.
- **Code.** Function code. We can make use of any function without needing to understand its contents and just by knowing which parameters it requires, how it works and what value it returns. Without a doubt, when we use a function in a module, it's smart to revise and understand the content in order to be sure that the function will carry out the expected actions.
- **Comments.** The API page may have user generated comments at the bottom of the page. Any drupal.org user can publish comments related to the function (always in English).

The column on the right shows the **API search and navigation** block. We can see any element or navigate by the lists:

- **Drupal 7.** Access the API documentation
- **Constants.** List of constants defined by the core modules.
- **Classes.** List of classes.
- **Files.** List of files that make up the core modules.
- **Functions.** List of functions.
- **Globals.** List of global variables.
- **Topics.** Documentation and lists of functions organized by theme.

When navigating through the API elements, be sure to select the corresponding Drupal 7 tab.

File Structure

43.4

We continue by exploring how Drupal folders and files are structured, which is shown in the **Figure:** F43.4

```

📁 includes [files .inc]
📁 misc [various files: js, images, jquery, css, etc.]
📁 modules [core modules]
📁 profiles
  📁 minimal
  📁 standard
  📁 testing
📁 scripts [sh files]
📁 sites
  📁 all
    📁 modules
    📁 themes
  📁 default
    📁 files
    📁 default.settings.php
    📁 settings.php
📁 themes [core themes]
📄 .gitignore
📄 .htaccess
📄 authorize.php
📄 CHANGELOG.txt
📄 cron.php
📄 index.php
📄 INSTALL.mysql.txt
📄 INSTALL.pgsql.txt
📄 install.php
📄 INSTALL.sqlite.txt
📄 INSTALL.txt
📄 LICENSE.txt
📄 MAINTAINERS.txt
📄 README.txt
📄 robots.txt
📄 update.php
📄 UPGRADE.txt
📄 web.config
📄 xmlrpc.php

```

F43.4

File Structure

You should not make changes to the distribution files. All changes should be made in the **profiles** and **sites** folders.

- The **includes** folder contains a group of libraries in the form of PHP files with **.inc** extension, which includes common system functions (ajax.inc, batch.inc, cache.inc, date.inc, form.inc, etc.).
- The **misc** folder contains javascript files and images required by the system (favicon.ico, jquery.js).
- The **modules** folder contains the core modules, each in its own corresponding folder. These modules should never be modified directly, and you should never upload other modules to their respective folders.

As we studied in previous levels, additional modules should be uploaded to **/sites/all/modules** (or **/sites/default/modules**).

- The **profiles** folder contains the installation profiles. Drupal can be installed in standard mode or in minimal mode, with the minimum modules required for Drupal to function. In **Unit 59** of this level we will see how to implement the installation profiles that will be added to this

folder before starting the site installation.

- The **scripts** folder contains additional utilities that Drupal does not use directly, but that can be used with the shell command line. For example, the script `password-hash.sh` allows you to get a coded password in addition to the original password (in plain text).
- The **sites** folder contains extras and modifications that can be added to the original distribution. It contains the additional modules, created by us or downloaded from the Drupal modules repository (for example, in `sites/all/modules`). It also contains the additional themes installed or created (for example, in `sites/all/themes`). The folder also contains the site configuration file (**settings.php**) after installation.
- The **themes** folder contains the themes that come with the Drupal distribution. Any new themes you want to add, create or modify should be uploaded to `/sites/all/themes` or `/sites/default/themes`.
- The **.gitignore** file contains a list of folders and/or files that will be ignored by the GIT version control repository.
- The **.htaccess** file is used by Apache to apply specific Web server configurations for Drupal to run correctly. For example, the **.htaccess** file allows the site to use clean URLs.
- The **cron.php** file executes periodic jobs required by the system.
- The **index.php** file is the front door to the system. When someone loads a website page it calls on **index.php**.
- The **install.php** file is the entry point for the installation.
- The **robots.txt** file tells the search engine robots which site folders or files should not be indexed.
- The **update.php** file activates the database after the system or the installed modules are fired up, as we have seen in previous levels.
- The **xmlrpc.php** file deals with XML-RPC format calls. It uses a protocol that handles incoming calls from XML clients.
- Several **help and information files** are also included in **.txt** format, such as:
 - o **CHANGELOG.txt**, has the history of changes that each version of Drupal has undergone.
 - o **INSTALL.mysql.txt**, **INSTALL.pgsql.txt** and **INSTALL.sqlite** gives information to help with the creation of a database and the necessary permissions, in MySQL and PostgreSQL, respectively.
 - o **INSTALL.txt**, describes how to install Drupal, step-by-step.
 - o **LICENSE.txt**, gives information about the GNU version 2 license, under which Drupal is distributed.
 - o **MAINTAINERS.txt**, lists the people who are in charge of maintaining Drupal.
 - o **UPGRADE.txt**, describes step-by-step how to update Drupal to new versions.

Operation of Drupal

43.5

What does Drupal do when we ask it to load a web page? We will review the process step-by-step:

- Typing `http://www.example.com/node/11` in the web browser asks it to load the page. We want to load the node with identifier 11, assuming that the site uses clean URLs.
- This request goes to the Web server, which analyzes the URL and obtains the Drupal path that it wants to download (`node/11`) and assigns it **parameter q**. The resulting URL will be: `http://www.example.com/index.php?q=node/11`, which corresponds with the URL in case the site isn't using clean URLs.
- The page loading **begins with the index.php script** that calls the `drupal_bootstrap()` function and sets into motion the phases of **booting the system (bootstrap)**.

We will explore in more detail the phases of the bootstrap process initiated by the function `drupal_bootstrap()`, included in the file **includes/bootstrap.inc**. F43.5

```
function drupal_bootstrap($phase = NULL, $new_phase = TRUE) {
  static $phases = array(
    DRUPAL_BOOTSTRAP_CONFIGURATION,
    DRUPAL_BOOTSTRAP_PAGE_CACHE,
    DRUPAL_BOOTSTRAP_DATABASE,
    DRUPAL_BOOTSTRAP_VARIABLES,
    DRUPAL_BOOTSTRAP_SESSION,
    DRUPAL_BOOTSTRAP_PAGE_HEADER,
    DRUPAL_BOOTSTRAP_LANGUAGE,
    DRUPAL_BOOTSTRAP_FULL,
  );
  //...
  switch ($current_phase) {
    case DRUPAL_BOOTSTRAP_CONFIGURATION:
      _drupal_bootstrap_configuration();
      break;
    case DRUPAL_BOOTSTRAP_PAGE_CACHE:
      _drupal_bootstrap_page_cache();
      break;
    //...
  }
}
return $stored_phase;
}
```

F43.5

Bootstrap Process

The `drupal_bootstrap()` function sets into play a series of booting phases.

A good way to understand how Drupal works is to analyze the path made after starting the bootstrap function or the index.php file.

The process of booting the system is comprised of the following phases:

- **DRUPAL_BOOTSTRAP_CONFIGURATION.** System configurations. Sets up the parameters for system configuration and loads `settings.php`.
- **DRUPAL_BOOTSTRAP_PAGE_CACHE.** Previous page caches. Handles special page caches from the `settings.php` file and makes it possible to load a previously loaded page without calling upon the database, thus using less resources.

- **DRUPAL_BOOTSTRAP_DATABASE.** Initializes the database system (without connecting) and runs the autoload functions. If it does not find the definition of the database in the settings.php file, it will redirect the installation process.
- **DRUPAL_BOOTSTRAP_VARIABLES.** Loads system variables and all enabled bootstrap modules (e.g. certain caching for modules).
- **DRUPAL_BOOTSTRAP_SESSION.** Initiates or reestablishes the Drupal session. Keep in mind that Drupal uses its own session system stored in the database. The variable global \$user initializes, with current user data. If the session is already begun, it uses the corresponding \$user data, and if not it uses the \$user data for an anonymous user.
- **DRUPAL_BOOTSTRAP_PAGE_HEADER.** Establishes the page header. The hook_boot() initializes the lock system and sends the pre-set HTTP headers. The lock system allows for predetermined operations to lock the system so that they don't run parallel operations.
- **DRUPAL_BOOTSTRAP_LANGUAGE.** Determines the language in which to show the page.
- **DRUPAL_BOOTSTRAP_FULL.** In this last phase the **common.inc** library loads and_drupal_bootstrap_full() is called, which finished loading the other libraries and active modules (includes/*.inc). It initializes the theme system and loads the predetermined theme to the site.

This phase ends with the call hook_init(), before processing the page and getting the content. Therefore, all the modules that implement the hook_init() up to this point will be called and the corresponding actions will all run.

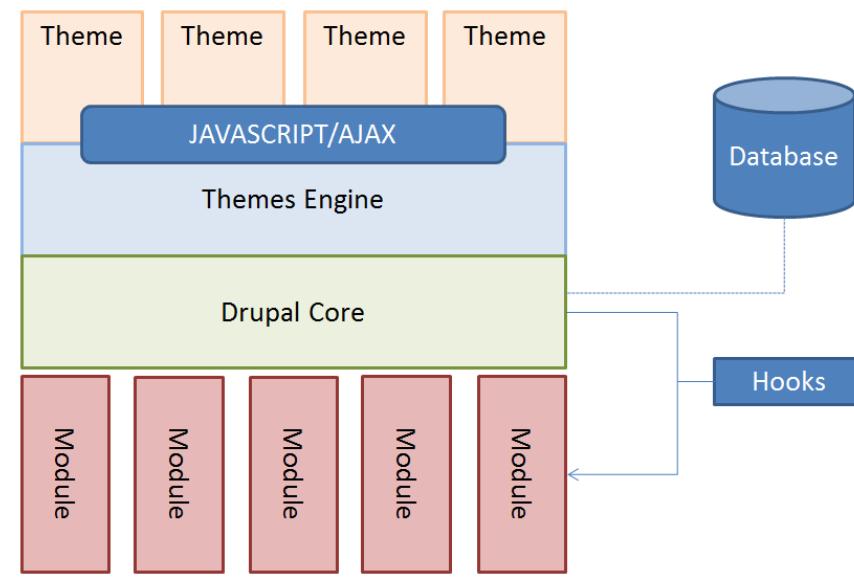
- Once all the Bootstrap phases are complete, the callback function connected to the current path will be executed from index.php through the **menu_execute_active_handler()** call, which will load the page content. In this example, after handling the node, it calls the function node_page_view() from the Node module (node.module).
- Finally, it returns the formatted page (typically in HTML format) to the users web browser.

Drupal Architecture

43.6

As we have learned, Drupal is a **Content Management System (CMS)** running on **PHP** programming logic, following a structured programming model and using a relational database system (for example, **MySQL**).

The schematic of the elements that make up the Drupal system are shown in the **Figure. F43.6**



F43.6
Drupal Architecture
Schematic

Drupal is a modular system. The system includes a group of tools and guidelines to follow in order to develop and integrate new functions through additional modules.

The code that makes up the **core** of Drupal is formed by a collection of libraries that allow for the management of the booting processes of the system. These libraries also offer many services that allow one to integrate the additional functionality of the modules - services like connecting and administering databases, managing mail, image manipulation, internationalization, code support, and a strong **environment of utility integration**. This last service, which we will explain further on, allows for the extension of functionality of the Drupal system in a relatively simple manner.

Drupal is, therefore, a system with **modular architecture** that allows one to extend functionality through uniform methods of development and integration with new modules. Ultimately, a module is made up of files of PHP code that use the Drupal architecture and APIs to add new functional features to a web site.

Before we get into the details of how to develop new Drupal modules, it's important to become familiar with the Drupal environment, its architecture and the functions of the API.

Modules of the Drupal Core

Some modules are indispensable to Drupal, while others are not. Others are optional, but they are needed for the support they provide to an extensive and varied system. In both cases, these modules constitute the **Drupal core** and form part of the basic distribution environment. All the modules of the Drupal core, as well as those developed by the community, follow the same development guidelines, making use of the same APIs.

Hooks

At this point it's important to check back in with **hooks**, a concept that comes up repeatedly in Drupal development. When Drupal responds to a specific user request, it looks through the active modules looking for functions whose names fit certain **patterns**, called Hooks.

For example, if we develop a module called **mymodule**, and want to include a block that features the Forcontu logo, we need to implement the **hook_block_info()**, in which we will define the block, and the **hook_block_view()**, which executes the code for the block.

Functions within the module will call **mymodule_block_info()** and **mymodule_block_view()**. When Drupal is constructing the page and examining what blocks to include, it will look within all the active modules for the functions with the pattern **modulename_block_info()** and **modulename_block_view()**.

The **modules can also define their own hooks** so that other modules can use them. This feature will allow us to incorporate complete functions into our Drupal system.

Fields and Entities

One of the most important changes incorporated in Drupal 7's architecture is **entities** and **fields**. In Drupal 6 and older versions additional fields could be added to content types using CCK. This created a model based on content types and where every element had to be converted to a content type in order to be able to use the CCK fields. One obvious example is the use of the Content Profile module in Drupal 6, that converts the user profiles into a content type to which we can add CCK fields.

Drupal 7 improved in this area, separating the content field types and generating the **entity** concept. In Drupal 7 an entity is a more generic element, one that you can add fields to (in the CCK style). Now the content types (or nodes), users, taxonomy terms and other elements are defined as entities, and as such we can manage the associated fields and their appearance.

From a programming point of view, now we won't have to define an element as a content type so that the associated fields can be managed. In some cases, we can define new entities, treating them separately from the content types.

Entity fields are programmed through the Field API. As we will see, starting with Drupal 7 we will use common functions to manipulate fields, independently from the entity to which they are assigned.

Drupal Nodes

Any content management system should allow for the management and administration of web content. Drupal stores the Web page contents in nodes. Thus, nodes constitute the basic building blocks of a Web site built with Drupal.

Although nodes are usually made up of text (or HTML) content type, Drupal can manage node types whose fundamental elements may be audio, video, etc. files.

The nodes are stored in the database, within the **node** table that stores basic information. Some of the most important table fields are:

- **nid**: unique identifier for the node
- **vid**: indicates the node version, which is used in version control
- **type**: indicates the content type in text mode
- **language**: indicates the language of the node
- **title**: node title
- **uid**: user ID of the node's creator
- **status**: indicates if it's published, yes (1) or no (0).
- **created**: time stamp for the date node was created
- **changed**: time stamp for the date node was modified
- **comment**: indicates if one can comment (yes, no, read-only).
- **promote**: indicates if the node is promoted on the home page
- **sticky**: indicates if the node is attached to the top of lists

The vast majority of nodes come with a title and a body. Developers can decide, ultimately, if they use them or not, but they always appear in the database. The **field_data_body** table stores this information and pertinent details such as the summary, format, etc.

We will work with nodes on many occasions while developing new modules, so it's a good idea to learn more about the **node API**.

As we have already mentioned, beginning with Drupal 7 the nodes and content types are **entities**. Thus, we can assign each field type to a collection of customized fields. In order to work with the entity fields we will use the Field API functions.

Drupal Comments

Another very important element in a Drupal website is comments. In Drupal 7 comments are also considered entities, so that we can manage their fields and appearance as we would with the rest of the entities. The **comment** table stores the information related to comments.

Comments are one example of the flexibility that Drupal provides with the new entity concept.

Drupal Users

Another important element type is users. The information related to users can be found in the database and is used in diverse processes: authentication, preferences, permissions, etc. Using the **user API** allows us to obtain information about users, compare permissions, see their preferences, etc.

As we've studied in the previous levels, Drupal manages permissions by way of roles: users belong to different roles and those roles have certain pre-assigned permissions. Because of this, it can take several steps to find out what permissions a user has: first you have to find out which role is assigned to the user, and then you have to find out what permissions are assigned to those roles. The use of the user API makes this easier for the developer, allowing them to find out in a very simple form whether the user has a certain permission or not.

In addition, as we have mentioned, users are another type of entity. Drupal 7 allows us to add fields to the user profile without needing to convert the user profile to a content type.

Drupal Forms

The main way to send content through the web is by HTML forms. Using the **Forms API** (also known as FAPI) provides powerful development tools for creating web forms due to the fact that Drupal allows one to control the form definition, construction, layout, data capture, validation, etc.

In the upcoming units we will study the distinct elements of Drupal from a programming perspective so that we will be able to construct more complex modules with more functionality as we advance through the course.

44 Creating modules

As has been discussed, Drupal is written in PHP, a programming language that is interpreted and designed for the development of webpages. The expansion of Drupal through modules is also achieved through PHP. Starting with this unit we will begin to study how to develop new modules.

Communication between modules and Drupal core occurs through PHP functions known as hooks. We can also make use of predefined functions that compose the Drupal API. An API (application programming interface) is a library of functions that allow us to access certain data and system functions without needing to know about the system's internal structure. The Drupal API contributes communication functions with the system and with core modules. Details about the API can be found at <http://api.drupal.org/api/drupal>.

To learn to develop modules and understand how existing modules function, we will use these fundamental learning strategies:

- Step by step creation of new modules.
- Analysis of example modules included in the **Examples for Developers** module.
- Analysis of existing modules created and shared by other members of the community.

During module development, the use of tools studied in previous units, such as the Coder, Devel, and Module Builder modules, is recommended.

Comparative D7/D6

Creating modules

The procedure through which modules are created in Drupal 7 is similar to that of Drupal 6.

The main changes are in the hooks. Some hooks in Drupal 6 are not used or have changed in Drupal 7. New hooks have also been introduced in Drupal 7.

In Drupal 6 it was very common to find hooks that achieved various operations (such as the \$op parameter). In Drupal 7, as a general norm, these operations have been separated into multiple hooks.

We can also find changes in the database, since Drupal 7 uses the PDO php extension to communicate with the database. We will introduce these changes in this unit, and they will be explained in more detail in Unit 45.

Unit contents

44.1 Definition of a module	48
44.2 Hooks	58
44.3 Module configuration	76
44.4 Working with the database	82



44.1

Definition of a module

In this section we will begin to learn about creating modules by creating a basic module that will allow us to analyze the minimum necessary structure, both files and file contents. We will study the following points:

- Location of the new module.
- Base language and coding of files
- Main files of a module: .info and .module
- The **t()** function to translate text strings
- Enabling the module

We will create a module called **First example**, which will help us contextualize these 5 points. This first module will generate a block with a piece of welcome text.

Module Location

Each module in Drupal is given its own directory, which generally receives the same name as the module to simplify organization.

As we know, in Drupal modules can be located in:

- **/modules**. This space is **reserved for Drupal core modules**. New modules that we develop should not be stored here.
- **/sites/all/modules**. This is the folder in which we should save modules that should be available to all sites that use this codebase. Multiple sites can be implemented from a single codebase through a multi-site installation. This file location is used by the automatic module installation Drupal system function, and will be the customary place where we will be saving the modules we develop.
- **/sites/default/modules**. This is the typical folder used to install additional modules. It can also be used in place of the previous location.
- **/sites/<site_domain>/modules**. A Drupal installation can serve multiple web sites through a multisite configuration. When modules should only be available to a single specific site, this folder is used.

We will be using the **sites/all/modules** folder to store the new modules we develop.

Now that we know where we will be storing modules, the first step is to create a folder.

Folder names should not contain spaces or special characters. When the name of the module contains more than one word, we can give the folder a name with all words together, or using an underscore to separate words.

For example, the **IMCE Wysiwyg** module uses the **imce_wysiwyg** folder, while the **Language icons** module uses the **language_icons** folder (without a separator).

In cases where the module has a very long name, or is known by an abbreviation, we can use an adequate name. An example of this is the **Chaos tool suite** module, which uses the **ctools** folder.

To continue to create the **First example** module, we will create a folder called **first_example**. We will then have the following system file path: **sites/all/modules/first_example**.

Base language and file encoding

Once we've created a folder for our module, the next step is the creation of the files that will define it. Before beginning to work with files it is important to take into consideration the following notes.

Although Drupal can be installed in different languages, the **base language** always remains the same: **English**. This means that both core modules and contributed modules should be in English. This eases communication and teamwork between community members of various nationalities, and has without a doubt helped Drupal's rapid growth.

One must keep in mind that "translatable" strings that you incorporate in your modules will be incorporated as a part of the English language version of the site (even if you've written them in another language).

For these reasons, we advise that you work from the start in English, especially if you are thinking of contributing your module to the community.

One must also pay attention to file encoding. Although coding is in plain text documents, various different file encodings exist (ANSI, UTF-8, ISO 8859-x, etc.). Drupal works with UTF-8, which means that our first step when we create a file is to convert it to **UTF-8 without BOM** format, this way we avoid problems with special characters such as accents.

Each text editor has its own way of modifying file encoding. For example, if you are working with Notepad++, you can choose encoding formats selecting the Encoding menu, and then **Encode in UTF-8 without BOM**.

Always test that file encoding is correct.

The .info file

Before starting to generate code for the module, it is necessary to create a file with the **.info** file extension. This file is a text file written similarly to the php.ini file, which contains **basic information** about the module (name, description, Drupal version, etc.), minimum requirements (for example a certain version of PHP), possible dependencies on other Drupal modules, the package to which the module belongs, and the files that the module includes.

The file name should be, by convention, similar to that of the module folder. We will follow the **<module_name>.info** pattern, where <module_name> corresponds to the folder created for the module. For this first module that we are developing, the appropriate name would be **first_example.info**.

Remember to encode the file in **UTF-8 without BOM** and whenever possible, work with text in English (we will study in coming units how to incorporate translation in modules).

Figure F44.1 shows the contents of the **First Example** module's information file.

F44.1

.info File

Example of the module definition file (.info).

```
name = "First example"
description = "My first Drupal module"
core = 7.x
files[] = first_example.module
package = My modules
```

The **.info** can contain the following fields, some of which are obligatory.

- **name** (Name, **obligatory** field). Contains the name of the module as it will be displayed to administrators (through the module administration page).
- **description** (description, **obligatory** field). Short description of the module, which will also be displayed on the module administration page.
- **core** (number, **obligatory** field). Indicates the Drupal version for which the module is valid (5.x, 6.x, 7.x, etc.). A specific version, such as 7.2, should not be specified.
- **files** (files, **optional** field). Drupal 7 permits dynamic code registration through the autoload functions. All modules should declare their files that contain classes or interfaces in the .info file. When a module is installed and activated, Drupal scans the module's files and indexes them with the classes and interfaces that it finds. These classes will be automatically loaded by PHP when they will be used.
- **php** (**optional** field). Specifies a minimum required version of PHP. If a PHP version is not indicated in module information, it is assumed that the module functions correctly with the PHP versions accepted by Drupal core.
- **dependencies** (dependencies, **optional** field). Our module could need other modules in order to function. The dependencies field will be a vector with the names of these required modules. The name of each module indicated in the vector should correspond to the system name of the module (name of the .module file), also in lower case. In the following example we will show how to require that the Views and Taxonomy modules be activated in the system.

As shown in the example, **F44.2** we can also specify versions or ranges of versions of a module, applying operators (= or == equal to, > more than, < less than, >= more than or equal to, <= less than or equal to, != different from).

If any of these modules is unavailable, we will not be able to enable our module.

```
dependencies[] = taxonomy
dependencies[] = views

dependencies[] = token (1.0)
dependencies[] = date (>2.0)
dependencies[] = rules (>2.0, <3.1, !=2.2)
```

F44.2**Dependencies**

Dependencies on other modules. These additional modules will be required in order to activate the module.

- **package** (package, **optional** field). When a module is a part of a package of modules, we can include in the information file the name of the package. In this way, the module will be grouped with other related modules in the administration area. For example, if we develop a module that adds a new Views style plugin, we would write:

```
package = Views
```

- **required** (required, **optional** field). Specifies that the current module is necessary and should always be enabled. Modules with this tag will be automatically enabled during installation. In general this should only be used with core modules that Drupal requires (such as Node, User, etc.). The way to use this tag is assigning it the value TRUE (required = TRUE).
- **hidden** (hidden, **optional** field). Specifies if a module or theme should not be visible in the module administration page (hidden = TRUE). This can be used, for example, to test modules on sites where other users have administrator access. In this way, other administrators will not be able to enable or disable the module.
- **configure** (configure, **optional** field). Specifies the path of the module's configuration page. Activating this option a link will be placed next to the module on the module administration page.

The .module file

The **.module** file includes the modules' code, in the form of PHP functions. It is in this file where **hooks** will be defined, creating the functions that they implement.

The standard name of the .module file is similar to that of .info files, following the **<nameofmodule>.module**. For this first module that we are developing, we will create the file **first_example.module**. Remember to encode the text **UTF-8 without BOM** and, wherever possible, work with texts in English.

The **First example** module should show a **block** with a welcome message. For this we will use the **hook_block_info()** function, with which we can declare a new block, and the **hook_block_view()** function, in which we will implement the contents of the block.

We will start to write code in the **first_example.module** file, but first we should remember that Drupal follows a rigorous standard at the moment of writing and documenting code. Our **first_example.module** file starts with the following code.

F44.3

.module file

The .module file includes the code of the module, in the form of PHP functions. It starts with a @file directive describing the module.

```
<?php
/**
 * @file
 * Example module
 * This module presents a welcome message in a block. */
```

As we've already commented, the **.module** file is no more than a PHP script, in that the first line starts with the PHP initiation code "**<?php**". As was explained before, we will not use the PHP closing code "**?>**" at the end of the file, a convention that permits us to avoid possible errors produced by the sending of empty spaces before HTTP headers have been sent.

The code makes available information for automated documentation of Drupal module (through the **API documentation**). It is a block of PHP commentary that starts with **/**** and ends with ***/**. All other lines begin with the ***** character. Not all contents of the code is text, we also have a special identifier that offers additional information. In our example the **@file** identifier indicates that the rest of the information refers to the entire file, not to a specific function, which is why we descriptive information about the module.

We are now ready to write the necessary PHP code for our module. As we've mentioned previously, our module will show a welcome message in a block, which means it will be necessary to use the **hook_block_info()** and **hook_block_view()** functions. These functions will be further explained in **Unit 48**.

We can find the complete description of the **hook_block_info** function in the Drupal API:

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_info/7

The **hook_block_info()** function cannot be passed parameters.

It returns an associative vector with the definition of the block. Some of the parameters of the vector are:

- **info (required)**. Name of the block that will appear in the list of blocks in the site administration.
- **cache (optional)**. Define cache type.
- **status (optional)**. Indicates if the block will be enabled or disabled upon the installation of the module. (1 = enabled, 0 = disabled). If we do not indicate a value, the block will be disabled.

D7/D6 Comparative Block hooks

In Drupal 6 there is a unique `hook_block()` function which use the parameter \$op to distinguish between operations.

In Drupal 7 these operations are separated into different hooks: `hook_block_info()`, `hook_block_view()`, `hook_block_configure()`, etc.

When we implement a hook function, we should follow the following pattern to name the new function: `<modulename>_<hookname>`. In our case, the function that implements the `hook_block_info()` hook for the `first_example` module will be called `first_example_block_info()`.

```
/***
 * Implements hook_block_info().
 */
function first_example_block_info() {
  //Declaring the Block
  $blocks['welcome'] = array(
    'info' => t('My first module: welcome block'),
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}
```

F44.4

Implement a hook

Example of a function that implements the `hook_block_info()` hook.

As we've mentioned, when we implement a hook, the declaration of the function shall be preceded by a block of commentary specifying which hook is implemented "Implements hook_...()".

The line that follows is the declaration of the function itself. During the system's execution, Drupal calls different hook functions to obtain the information it requires in each moment in order to create the website.

For example, at a certain moment during the loading of a site, Drupal needs to know which blocks exist, and looks in all active modules for implementations of the `hook_block_info()` function. Each module indicates to the block system the blocks it creates, and in this way Drupal assembles a complete list of blocks.

In our `first_example_block_info()` function, we only generate one block, for which the returned `$blocks` array only contains one element (the 'welcome' element). As well, we have only included values for the 'info' and 'cache' fields, but we could have included any of the other fields available for `hook_block_info()`, which will be further explained in **Unit 48**.

We could now indicate what will be the contents of the block, implementing the `hook_block_view()` function. The complete description of the function can be found on the Drupal API:

http://api.drupal.org/api/drupal/modules-block-block.api.php/function/hook_block_view/7

The `hook_block_view()` function is passed one parameter `$delta`, a unique identifier for the block within the module . The function returns a vector with the following fields:

- **subject.** We indicate default title of the block. If the block has no default title we should define NULL.
- **content.** The contents of the body of the block. This should preferably be a rendered vector, or a rendered string of HTML content.

In our example, the function that implements the hook_block_view() hook will be called **first_example_block_view()**. F44.5

F44.5**Block content**

The contents of blocks defined in hook_block_info() is indicated in hook_block_view().

```
/***
 * Implements hook_block_view().
 */
function first_example_block_view($delta = '') {
  //Contents of the block 'welcome' defined in hook_block_info().
  $block = array();
  switch ($delta) {
    case 'welcome':
      $block['subject'] = t('My first module');
      $block['content'] = t('Welcome to my Drupal site.');
      break;
  }
  return $block;
}
```

The system will call the **first_example_block_view()** function to obtain the contents of each of the defined blocks. For example, it will call **first_example_block_view("welcome")**, which will return a vector with the values of **subject** and **content** for the block with the "**welcome**" identifier, that we defined previously in **first_example_block_info()**.

It is important to understand that we don't have to worry about these system calls. We just need to adequately implement hooks and the system will take automatically take care of the functions when necessary, and assemble elements of the site.

Within our implementation of **hook_block_view()** we use a sentence `switch($delta)` to differentiate each defined block, and assign them the values of title and content. In our example, we only defined the "welcome" block, but we could have added more blocks. In **Unit 48**, we will study in more depth how to work with multiple blocks generated by the same module.

The **first_example.module** file will be as follows: F44.6

F44.6**first_example.module**

.module file with the implementation of the hooks that generate a block.

```
<?php

/**
 * @file
 * Example module
 * This module presents a welcome message in a block.
 */

/**
 * Implements hook_block_info().
 */
function first_example_block_info() {
  //Declaration of the block
  $blocks['welcome'] = array(
    'info' => t('My first module: welcome block'),
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}

/**
```

```
* Implements hook_block_view().
*/
function first_example_block_view($delta = '') {
  //Implementation of the 'welcome' block defined in
  hook_block_info.
  $block = array();
  switch ($delta) {
    case 'welcome':
      $block['subject'] = t('My first module');
      $block['content'] = t('Welcome to my Drupal site.');
      break;
  }
  return $block;
}
```

The t() function

At this point, we should focus our attention on the **t()** function, which has been used in the assigning of text strings within the **first_example_block_view()** function.

The **t()** function allows the system to translate strings of text into other languages that are active on the system.

As commented, English is the base language, so it's highly recommend that the entire code be written in English. The **t()** function allows us to later add translations to other languages, either manually, through the translation area of the site interface, or automatically through the translation file associated with the module, as we will see in **Unit 55**.

Whenever we include any text in the modules we develop, we should do so with the **t()** function to enable its subsequent translation.

You can find a detailed description of the **t()** function in the Drupal API:

<http://api.drupal.org/api/drupal/includes--bootstrap.inc/function/t/7>

Enabling the module

Once the folder (**/sites/all/modules/first_example**) and module files (**first_example.info** and **first_example.module**) have been created, the next step is enabling the module.

The installation and enabling of the modules is realized in the same way as with any other module, by visiting

Administration ⇒ Modules

URL Modules
</admin/modules>

The **First example** module will appear, disabled, in the **My modules** group. Enable the module and save the configuration. **F44.7**

MY MODULES					
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS	
<input type="checkbox"/>	First example		My first Drupal module		

F44.7

Enable the module

The module is disabled by default.

Once enabled, we will find the block in the block administration area: **F44.8**

URL Blocks
</admin/structure/block>

F44.8

Block available in the list of blocks

The block created by the module will be available for use in the listing of blocks on the site.

The title of the block in the administration area corresponds with the info field in the implementation of **hook_block_info()**, "My first module: welcome block".

Block Title	Visibility	Action
Main menu	- None -	configure
Management	- None -	configure
My first module: welcome block	- None -	configure
My menu	- None - Header Help Highlighted Featured Content Sidebar first Sidebar second	configure
My profile	Triptych first Triptych middle Triptych last	configure
Recent comments		configure
Recent content		configure

We can then access the configuration options that we have studied for blocks, such as setting an alternative title, or visibility options according to pages, content types, roles and/or users. In Unit 48 we also see how to create new block configuration options.

Once the block has been assigned to a region, we can see the module in action.

F44.9**F44.9**

Enabled Block

The block generated by the module has been assigned to the right side region.

Maecenas libero odio, tincidunt sed vehicula ut, aliquet quis sapien.

Submitted by admin on Mon, 02/13/2012 - 12:49

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras eget convallis felis. Duis eu tortor sit amet risus accumsan iaculis at eu elit. Nulla congue aliquam lorem, ornare viverra purus placerat sit amet. Duis placerat euismod turpis sit amet aliquet. Mauris erat lorem, cursus sed congue eu, consectetur eget metus. Sed mauris sapien, facilisis in adipiscing in, consectetur id justo. Donec auctor

My first module
Welcome to my Drupal site.

Translating the module

Lastly, if we are working in another language, we will translate the modules' strings, accessing:

Administration ⇒ **Configuration** ⇒ **Regional and language** ⇒ **Translate interface**

URL Translate interface
</admin/config/regional/translate>

From the **Translate** tab we can find the modules' strings and assign corresponding translations. (Examples are shown translated into Spanish): **F44.10**

- **My first module:** welcome block (Mi primer module: bloque de bienvenida).
- **My first module** (Mi primer bloque).
- **Welcome to my Drupal site.** (Bienvenido a mi sitio Drupal.)
- **My first Drupal module** (Mi primer module en Drupal). This text corresponds to the description of the module in the .info file, and is shown in the list of modules in the administration section.
- **My modules** (Mis módulos). Name of the packet where we will group the modules developed during this course. This is shown in the list of modules in the administration section.

Home » Administration » Configuration » Regional and language » Translate interface

Translate interface OVERVIEW TRANSLATE IMPORT UPDATE EXPORT EXTRACT

This page allows a translator to search for specific translated and untranslated strings, and is used when creating or editing translations. (Note: For translation tasks involving many strings, it may be more convenient to [export](#) strings for offline editing in a desktop Gettext translation editor.) Searches may be limited to strings found within a specific text group or in a specific language.

FILTER TRANSLATABLE STRINGS

String contains
My first
Leave blank to show all strings. The search is case sensitive.

Language All languages **Search in** Both translated and untranslated **Limit search to** All text groups

Filter Reset

TEXT GROUP	STRING	CONTEXT	LANGUAGES	OPERATIONS
Built-in interface	My first module: welcome block /admin/modules/list/confirm		es	edit delete
Built-in interface	My first module /		es	edit delete
Built-in interface	My first Drupal module /admin/modules		es	edit delete

Once the translations are made we will immediately be able to see the changes, both in the block as well as in the block administration area.

F44.11

Artículo

Enviado por admin en Mié, 01/11/2012 - 10:13

Mi primer módulo

Bienvenido a mi sitio
Drupal.

LOREM IPSUM DOLOR SIT AMET, consectetur adipiscing elit. Ut vitae ipsum vel metus lacinia sollicitudin. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Maecenas eget velit sem. Aliquam non pretium neque. Maecenas non lectus

We can also see the finished translations in the module administration area:

F44.12

MIS MÓDULOS

ACTIVADO	NOMBRE	VERSIÓN	DESCRIPCIÓN	OPERACIONES
<input checked="" type="checkbox"/>	First example		Mi primer módulo en Drupal	

F44.12

Translated Module

Translations also affect the administration area (in Spanish).

44.2 Hooks

We have already seen why and how hooks are used in Drupal, and how to implement them. While the site is being loaded, the system will call the various hook function available in the active modules, in order to obtain the required just-in-time information, until the site is completely assembled and shown to the user.

It is difficult to group hooks in order of importance, since that depends on the type of module that we are creating. We can, however, group hooks according to the functionality they provide. The complete list of hooks can be found in the Drupal API:

<http://api.drupal.org/api/drupal/includes--module.inc/group/hooks/7>

In Drupal 7 there are important changes in hook functions, with new functions added and some hooks from Drupal 6 being eliminated.

As a general rule, the hooks that carried out various tasks via the use of the **\$op** parameter in Drupal 6 **have been broken apart into various hooks** which manage the same operations but work independently.

We find one example in the **hook_user()** used in Drupal 6, which has been divided into various Drupal 7 hooks including **hook_user_insert()**, **hook_user_load()**, **hook_user_login()**, **hook_user_logout()**, **hook_user_update()**, etc.

Users and Permissions Hooks

The following hooks are related to users and permissions in general: **F44.13**

F44.13

Hooks

Users and Permissions

In this list we see some hooks that apply to users and permissions.

Name	Description
hook_field_access	Determines whether the user has access to a given field.
hook_node_access	Controls access to a node.
hook_node_grants	Informs the node access system what permissions the user has.
hook_permission	Defines user permissions.
hook_username_alter	Alters the username that is displayed for a user.
hook_user_cancel	Acts on user account cancellations.
hook_user_cancel_methods_alter	Modifies account cancellation methods.
hook_user_categories	Retrieves a list of user setting or profile information categories.
hook_user_delete	Responds to user deletion.
hook_user_insert	Activated when a user account is created
hook_user_load	Acts on user objects when loaded from the database.
hook_user_login	Activated when user just logged in.
hook_user_logout	Activated at the time in which a user logs out.
hook_user_operations	Adds mass user operations.
hook_user_presave	Acts when user account is about to be created or updated.

<code>hook_user_role_delete</code>	Informs other modules that a user role has been deleted.
<code>hook_user_role_insert</code>	Informs other modules that a user role has been added.
<code>hook_user_role_presave</code>	Informs other modules that a user role is about to be saved.
<code>hook_user_role_update</code>	Informs other modules that a user role has been updated.
<code>hook_user_update</code>	Acts when a user account was updated.
<code>hook_user_view</code>	Acts when a user's account information is being displayed.
<code>hook_user_view_alter</code>	Used when a user was built; the module may modify the structured content.

System and Installation Hooks

The following hooks deal with the system (startup, database, logs, cache, installation, etc.): **F44.14**

Name	Description
<code>hook_boot</code>	Perform setup tasks for all page requests. This hook is run at the beginning of the page request
<code>hook_cron</code>	Perform periodic actions using cron system.
<code>hook_cron_queue_info</code>	Declare queues holding items that need to be run periodically.
<code>hook_cron_queue_info_alter</code>	Alter cron queue information before cron runs.
<code>hook_disable</code>	Perform necessary actions before module is disabled.
<code>hook_drupal_goto_alter</code>	Change the page the user is sent to by <code>drupal_goto()</code> .
<code>hook_enable</code>	Perform necessary actions after module is enabled.
<code>hook_exit</code>	Perform cleanup tasks. This hook is run at the end of each page request.
<code>hook_flush_caches</code>	Add a list of cache tables to be cleared.
<code>hook_help</code>	Provide online user help.
<code>hook_html_head_alter</code>	Alter XHTML HEAD tags before they are rendered by <code>drupal_get_html_head()</code> .
<code>hook_init</code>	Perform setup tasks for non-cached page requests.
<code>hook_install</code>	Perform setup tasks when the module is installed.
<code>hook_modules_disabled</code>	Perform necessary actions after modules are disabled.
<code>hook_modules_enabled</code>	Perform necessary actions after modules are enabled.
<code>hook_modules_installed</code>	Perform necessary actions after modules are installed.
<code>hook_modules_uninstalled</code>	Perform necessary actions after modules are uninstalled.
<code>hook_modules_implements_alter</code>	Alter the registry of modules implementing a hook.

F44.14

System and Installation Hooks

This list includes some hooks that relate to the Drupal system and actions within the installation process.

hook_multilingual_settings_changed	Allow modules to react to language settings changes.
hook_requirements	Check installation requirements and do status reporting.
hook_schema	Define the current version of the database schema.
hook_schema_alter	Perform alterations to existing database schemas.
hook_system_info_alter	Alter the information parsed from module and theme .info files
hook_uninstall	Remove any information that the module sets.
hook_update_projects_alter	Alter the list of projects before fetching data and comparing versions.
hook_update_status_alter	Alter the information about available updates for projects.
hook_update_last_removed	Return a number which is no longer available as hook_update_N().
hook_update_N	Perform a single update.
hook_watchdog	Log an event message.
hook_xmlrpc	Register XML-RPC callbacks.
hook_xmlrpc_alter	Alters the definition of XML-RPC methods before they are called.

Block-related Hooks

The following hooks permit interaction with blocks: F44.15

F44.15

Block-related Hooks

This list shows a sampling of some hooks related to blocks.

Name	Description
hook_block_configure	Define a configuration form for a block.
hook_block_info	Define all blocks provided by the module.
hook_block_info_alter	Change block definition before saving to the database.
hook_block_list_alter	Act on blocks prior to rendering.
hook_block_save	Save the configuration options from hook_block_configure().
hook_block_view	Return a rendered or renderable view of a block.
hook_block_view_alter	Perform alterations to the content of a block.
hook_block_view_MOD ULE_DELTA_alter	Perform alterations to a specific block.

Menu and Page Routing Hooks

The following hooks allow for actions on site menus and page routing: **F44.16**

Name	Description
hook_menu	Define menu items and page callbacks.
hook_menu_alter	Alter the data being saved to the {menu_router} table after hook_menu is invoked.
hook_menu_breadcrumb_alter	Alter links in the active trail before it is rendered as the breadcrumb.
hook_menu_contextual_links_alter	Alter contextual links before they are rendered.
hook_menu_delete	Informs modules that a custom menu was deleted.
hook_menu_get_item_alter	Alter a menu router item right after it has been retrieved from the database or cache.
hook_menu_insert	Informs modules that a custom menu was created.
hook_menu_link_alter	Alter the data being saved to the {menu_links} table by menu_link_save().
hook_menu_link_delete	Inform modules that a menu link has been deleted.
hook_menu_link_insert	Inform modules that a menu link has been created.
hook_menu_link_update	Inform modules that a menu link has been updated.
hook_menu_local_tasks_alter	Alter tabs and actions displayed on the page before they are rendered.
hook_menu_site_status_alter	Control site status before menu dispatching.
hook_menu_update	Informs modules that a custom menu was updated.
hook_page_alter	Perform alterations before a page is rendered.
hook_page_build	Add elements to a page before it is rendered.
hook_page_delivery_callback_alter	Alters the delivery callback used to send the result of the page callback to the browser.
hook_translated_menu_link_alter	Alter a menu link after it has been translated and before it is rendered.

F44.16

Menu and routing hooks

This list includes hooks that deal with menus and the page routing system.

Node and Form Hooks

The following hooks allow for interaction with nodes and forms: **F44.17**

F44.17

Node and Form Hooks

In this list we see some of the hooks that deal with nodes, comments, and forms.

Name	Description
hook_comment_delete	The comment is being deleted by the moderator.
hook_comment_insert	The comment is being inserted.
hook_comment_load	Comments are being loaded from the database.
hook_comment_presave	The comment passed validation and is about to be saved.
hook_comment_publish	The comment is being published by the moderator.
hook_comment_unpublish	The comment is being unpublished by the moderator.
hook_comment_update	The comment is being updated.
hook_comment_view	The comment is being viewed. This hook can be used to add additional data to the comment before theming.
hook_comment_view_alter	The comment was built; the module may modify the structured content.
hook_delete	Respond to node deletion.
hook_element_info	Allows modules to declare their own Forms API element types and specify their default values.
hook_element_info_alter	Alter the element type information returned from modules.
hook_filter_format_disable	Perform actions when a text format has been disabled.
hook_filter_format_insert	Perform actions when a new text format has been created.
hook_filter_format_update	Perform actions when a text format has been updated.
hook_filter_info	Define content filters.
hook_filter_info_alter	Perform alterations on filter definitions.
hook_form	Display a node editing form.
hook_forms	Map form_ids to form builder functions.
hook_form_alter	Perform alterations before a form is rendered.
hook_form_FORM_ID_alter	Provide a form-specific alteration instead of the global hook_form_alter().
hook_form_BASE_FORM_ID_alter	Provide a form-specific alteration for shared ('base') forms.
hook_insert	Respond to creation of a new node.
hook_load	Act on nodes being loaded from the database.
hook_node_access	Control access to a node.
hook_node_access_records	Set permissions for a node to be written to the database.

<code>hook_node_access_records_alter</code>	Alter permissions for a node before it is written to the database.
<code>hook_node_delete</code>	Respond to node deletion.
<code>hook_node_grants</code>	Inform the node access system what permissions the user has.
<code>hook_node_grants_alter</code>	Alter user access rules when trying to view, edit or delete a node.
<code>hook_node_info</code>	Define module-provided node types.
<code>hook_node_insert</code>	Respond to creation of a new node.
<code>hook_node_load</code>	Act on nodes being loaded from the database.
<code>hook_node_operations</code>	Add mass node operations.
<code>hook_node_prepare</code>	Act on a node object about to be shown on the add/edit form.
<code>hook_node_presave</code>	Act on a node being inserted or updated.
<code>hook_node_revision_delete</code>	Respond to deletion of a node revision.
<code>hook_node_search_result</code>	Act on a node being displayed as a search result.
<code>hook_node_submit</code>	Act on a node after validated form values have been copied to it.
<code>hook_node_type_delete</code>	Respond to node type deletion.
<code>hook_node_type_insert</code>	Respond to node type creation.
<code>hook_node_type_update</code>	Respond to node type updates.
<code>hook_node_update</code>	Respond to updates to a node.
<code>hook_node_update_index</code>	Act on a node being indexed for searching.
<code>hook_node_validate</code>	Perform node validation before a node is created or updated.
<code>hook_node_view</code>	Act on a node that is being assembled before rendering.
<code>hook_node_view_alter</code>	Alter the results of node_view().
<code>hook_prepare</code>	Act on a node object about to be shown on the add/edit form.
<code>hook_update</code>	Respond to updates to a node.
<code>hook_validate</code>	Perform node validation before a node is created or updated.
<code>hook_view</code>	Display a node.

Entity Hooks

The following hooks allow for performing actions on entities: **F44.18**

F44.18

Entity Hooks

In this list we have some entity-related hooks.

Name	Description
hook_entity_delete	Act on entities when deleted.
hook_entity_info	Inform the base system and the Field API about one or more entity types.
hook_entity_info_alter	Alter the entity info.
hook_entity_insert	Act on entities when inserted.
hook_entity_load	Act on entities when loaded.
hook_entity_prepare_view	Act on entities as they are being prepared for view.
hook_entity_presave	Act on an entity before it is about to be created or updated.
hook_entity_query_alter	Alter or execute an EntityFieldQuery.
hook_entity_update	Act on entities when updated.
hook_entity_view	Act on entities being assembled before rendering.
hook_entity_view_alter	Alter the results of ENTITY_view().

Field hooks

The following hooks allow for interaction with fields:

F44.19**F44.19**

Field Hooks

In this list we see examples of hooks that work with fields that relate to entities.

Name	Description
Field Attach API	Set of hooks that operate on Field API data attached to Drupal entities. http://api.drupal.org/api/drupal/modules--field--field.attach.inc/group/field_attach/7
hook_field_create_field	Act on a field being created.
hook_field_create_instance	Act on a field instance being created.
hook_field_delete	Define custom delete behavior for this module's field data.
hook_field_delete_field	Act on a field being deleted.
hook_field_delete_instance	Act on a field instance being deleted.
hook_field_delete_revision	Define custom revision delete behavior for this module's field types.
hook_field_display_alter	Alters the display settings of a field before it gets displayed.
hook_field_display_ENTITY_TYPE_alter	Alters the display settings of a field on a given entity type before it gets displayed.
hook_field_formatter_info	Expose Field API formatter types.
hook_field_formatter_info_alter	Perform alterations on Field API formatter types.
hook_field_formatter_prepare_view	Allow formatters to load information for field values being displayed.
hook_field_formatter_view	Build a renderable array for a field value.
hook_field_info	Define Field API field types.

<code>hook_field_info_alter</code>	Perform alterations on Field API field types.
<code>hook_field_info_max_weight</code>	Returns the maximum weight for the entity components handled by the module.
<code>hook_field_insert</code>	Define custom insert behavior for this module's field data.
<code>hook_field_is_empty</code>	Define what constitutes an empty item for a field type.
<code>hook_field_load</code>	Define custom load behavior for this module's field types.
<code>hook_field_prepare_view</code>	Prepare field values prior to display.
<code>hook_field_presave</code>	Define custom presave behavior for this module's field types.
<code>hook_field_read_field</code>	Act on field records being read from the database.
<code>hook_field_read_instance</code>	Act on a field record being read from the database.
<code>hook_field_schema</code>	Define the Field API schema for a field structure.
<code>Field Storage API</code>	Implement a storage engine for Field API data. http://api.drupal.org/api/drupal/modules--field--field.attach.inc/group/field_storage/7
<code>hook_field_update</code>	Define custom update behavior for this module's field data.
<code>hook_field_update_field</code>	Act on a field being updated.
<code>hook_field_update_forbid</code>	Forbid a field update from occurring.
<code>hook_field_update_instance</code>	Act on a field instance being updated.
<code>hook_field_validate</code>	Validate this module's field data.
<code>hook_field_widget_error</code>	Flag a field-level validation error.
<code>hook_field_widget_form</code>	Return the form for a single field widget.
<code>hook_field_widget_form_alter</code>	Alter forms for field widgets provided by other modules.
<code>hook_field_widget_info</code>	Expose Field API widget types.
<code>hook_field_widget_info_alter</code>	Perform alterations on Field API widget types.
<code>hook_field_widget_properties_alter</code>	Alters the widget properties of a field instance before it gets displayed.
<code>hook_field_widget_properties_ENTITY_TYPE_alter</code>	Alters the widget properties of a field instance on a given entity type before it gets displayed.
<code>hook_field_widget_WIDGET_TYPE_form_alter</code>	Alter widget forms for a specific widget provided by another module.

Action and Trigger Hooks

The following hooks allow for interaction with system actions and triggers: [F44.20](#)

F44.20

Action and Trigger Hooks

In this list we see some hooks related to actions and triggers.

Name	Description
hook_actions_delete	Executes code after an action is deleted.
hook_action_info	Declares information about actions.
hook_action_info_alter	Alters the actions declared by another module.
hook_trigger_info	Declare triggers (events) for users to assign actions to.
hook_trigger_info_alter	Alter triggers declared by hook_trigger_info().

Hook Definition Hooks

The following hooks allows for the definition of new hooks within a module: [F44.21](#)

F44.21

Hooks New hooks

In this list are hooks related to the creation of new hooks.

Name	Description
hook_hook_info	Defines one or more hooks that are exposed by a module.
hook_hook_info_alter	Alter information from hook_hook_info().

Theme Hooks

The following hooks permits interaction with site themes: [F44.22](#)

F44.22

Theme Hooks

This list includes hooks that deal with themes.

Name	Description
hook_theme	Register a module (or theme's) theme implementations.
hook_theme_registry_alter	Alter the theme registry information returned from hook_theme().
hook_custom_theme	Return the machine-readable name of the theme to use for the current page.
hooks_system_theme_info	Return additional themes provided by modules.

File Hooks

The following hooks give access to site files: [F44.23](#)

F44.23

File Hooks

In this list we see some hooks related to the site's file management.

Name	Description
hook_archiver_info	Declare archivers to the system.
hook_archiver_info_alter	Alter archiver information declared by other modules.
hook_filetransfer_info	Register information about FileTransfer classes provided by a module.
hook_filetransfer_info_alter	Alter the FileTransfer class registry.
hook_file_copy	Respond to a file that has been copied.

<code>hook_file_delete</code>	Respond to a file being deleted.
<code>hook_file_download</code>	Control access to private file downloads and specify HTTP headers.
<code>hook_file_download_access</code>	Control download access to files.
<code>hook_file_download_access_alter</code>	Alter the access rules applied to a file download
<code>hook_file_insert</code>	Respond to a file being added.
<code>hook_file_load</code>	Load additional information into file objects.
<code>hook_file_mimetype_mapping_alter</code>	Alter MIME type mappings used to determine MIME type from a file extension.
<code>hook_file_move</code>	Respond to a file that has been moved.
<code>hook_file_presave</code>	Act on a file being inserted or updated.
<code>hook_file_update</code>	Respond to a file being updated.
<code>hook_file_url_alter</code>	Alter the URL to a file.
<code>hook_file_validate</code>	Check that files meet a given criteria.
<code>hook_registry_files_alter</code>	Perform necessary alterations to the list of files parsed by the registry.

Taxonomy Hooks

The following hooks allow interaction with taxonomy terms and vocabularies: **F44.24**

Name	Description	F44.24
<code>hook_taxonomy_term_delete</code>	Respond to the deletion of taxonomy terms.	
<code>hook_taxonomy_term_insert</code>	Act on taxonomy terms when inserted.	
<code>hook_taxonomy_term_load</code>	Act on taxonomy terms when loaded.	
<code>hook_taxonomy_term_presave</code>	Act on taxonomy terms before they are saved.	
<code>hook_taxonomy_term_update</code>	Act on taxonomy terms when updated.	
<code>hook_taxonomy_term_view_alter</code>	Alter the results of taxonomy_term_view().	
<code>hook_taxonomy_vocabulary_delete</code>	Respond to the deletion of taxonomy vocabularies.	
<code>hook_taxonomy_vocabulary_insert</code>	Act on taxonomy vocabularies when inserted.	
<code>hook_taxonomy_vocabulary_load</code>	Act on taxonomy vocabularies when loaded.	
<code>hook_taxonomy_vocabulary_presave</code>	Act on taxonomy vocabularies before they are saved.	
<code>hook_taxonomy_vocabulary_update</code>	Act on taxonomy vocabularies when updated.	

F44.24

Taxonomy Hooks

This list includes some hooks that deal with taxonomy terms and vocabularies.

Search Hooks

The following hooks allow interaction with Drupal's search engine:

F44.25

Search Hooks

In this list, some examples of hooks related to Drupal's search engine are given.

Name	Description
hook_search_access	Define access to a custom search routine.
hook_search_admin	Add elements to the search settings form.
hook_search_execute	Execute a search for a set of key words.
hook_search_info	Define a custom search type.
hook_search_page	Override the rendering of search results.
hook_search_preprocess	Preprocess text for search.
hook_search_reset	Take action when the search index is going to be rebuilt.
hook_search_status	Report the status of indexing.
hook_update_index	Update the search index for this module.
hook_ranking	Provide additional methods of scoring for core search results for nodes.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Alias Hooks

The following hooks allow for actions over URL aliases:

F44.26

URL Alias Hooks

Name	Description
hook_path_delete	Respond to a path being deleted.
hook_path_insert	Respond to a path being inserted.
hook_path_update	Respond to a path being updated.

Administrative Path Hooks

The following hooks allow for interaction with site administrative paths.

F44.27

Administrative Path Hooks

Name	Description
hook_admin_paths	Define administrative paths.
hook_admin_paths_alter	Redefine administrative paths defined by other modules.

Date Format Hooks

The following hooks permit access to date format settings.

F44.28

Date Hooks

This list includes hooks that handle date formats.

Name	Description
hook_date_formats	Define additional date formats.
hook_date_formats_alter	Alter date formats declared by another module.
hook_date_format_types	Define additional date types.
hook_date_format_types_alter	Modify existing date types.

Image Style Hooks

The following hooks allow access to image styles: **F44.29**

Name	Description
hook_image_default_styles	Provide module-based image styles for reuse throughout Drupal.
hook_image_effect_info	Define information about image effects provided by a module.
hook_image_effect_info_alter	Alter the information provided in hook_image_effect_info().
hook_image_styles_alter	Modify any image styles provided by other modules or the user.
hook_image_style_delete	Respond to image style deletion.
hook_image_style_flush	Respond to image style flushing.
hook_image_style_save	Respond to image style updating.
hook_image_toolkits	Define image toolkits provided by this module.

F44.29

Image Style Hooks

In this list, some hooks that deal with image style are given.

Language Hooks

The following hooks allow for site language interaction: **F44.30**

Name	Description
hook_language_fallback_candidates_alter	Perform alterations on the language fallback candidates.
hook_language_init	Allows modules to act after language initialization has been performed.
hook_language_negotiation_info	Allow modules to define their own language providers.
hook_language_negotiation_info_alter	Perform alterations on language providers.
hook_language_switch_links_alter	Perform alterations on language switcher links.
hook_language_types_info	Allow modules to define their own language types.
hook_language_types_info_alter	Perform alterations on language types.
hook_locale	Allows modules to define their own text groups that can be translated.

F44.30

Language Hooks

In this list, some language-related hooks are listed.

Token Hooks

The following hooks allow for token interaction: **F44.31**

Name	Description
hook_tokens	Provide replacement values for placeholder tokens.
hook_tokens_alter	Alter replacement values for placeholder tokens.
hook_tokens_info	Provide information about available placeholder tokens and token types.
hook_tokens_info_alter	Alter the metadata about available placeholder tokens and token types.

F44.31

Token Hooks

Some hooks related to tokens and replacement values are listed.

JavaScript, CSS, and Ajax Hooks

The following hooks allow access to JavaScript, CSS, and Ajax files: [F44.32](#)

F44.32

Javascript, CSS, and Ajax Hooks

This list includes some hooks related to the interaction with Javascript, CSS and Ajax files.

Name	Description
hook_ajax_render_alter	Alter the commands that are sent to the user through the Ajax framework.
hook_css_alter	Alter CSS files before they are output on the page.
hook_js_alter	Perform necessary alterations to the JavaScript before it is presented on the page.
hook_library	Registers JavaScript/CSS libraries associated with a module.
hook_library_alter	Alters the JavaScript/CSS library registry.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Email Hooks

The following hooks permit action on email messages: [F44.33](#)

F44.33

Email Hooks

In this list we display hooks that relate to email transmissions. .

Name	Description
hook_mail	Prepare an email message based on parameters; called from drupal_mail().
hook_mail_alter	Alter an email message created with the drupal_mail() function.

Overlay Hooks

The following hooks allow for interaction with the superimposed administration layers generated by the Overlay module: [F44.34](#)

F44.34

Overlay Hooks

Here some hooks are listed which deal with administration overlays.

Name	Description
hook_overlay_child_initialize	Allow modules to act when an overlay child window is initialized.
hook_overlay_parent_initialize	Allow modules to act when an overlay parent window is initialized.

Database Query Hooks

The following hooks permit interaction with database queries: [F44.35](#)

F44.35

Database Query Hooks

Name	Description
hook_query_alter	Perform alterations to a structured query.
hook_query_TAG_alter	Perform alterations to a structured query for a given tag.

Installation Profile Hooks

The following hooks act during the installation process of an installation profile: [F44.36](#)

Name	Description
<code>hook_install_tasks</code>	Return an array of tasks to be performed by an installation profile.
<code>hook_install_tasks_alter</code>	Alter the full list of installation tasks.

[F44.36](#)

Installation Profile hooks

Other functions

Below are other functions which are not themselves hooks, but are related to them. [F44.37](#)

Name	Description
<code>module_hook</code>	Determine whether a module implements a hook.
<code>module_hook_info</code>	Retrieve a list of what hooks are explicitly declared.
<code>module_implements</code>	Determine which modules are implementing a hook.
<code>module_invoke</code>	Invoke a hook in a particular module.
<code>module_invoke_all</code>	Invoke a hook in all enabled modules that implement it.

[F44.37](#)

Other functions

These API functions are not hooks, which means they can be called directly, without needing to implement them in the context of a module.

The complete list of hooks available in Drupal 7 core is available at:

<http://api.drupal.org/api/drupal/includes--module.inc/group/hooks/7>

By consulting Drupal's API we can see and analyze the source code of each of the hook functions.

Implementing an example hook

In order to familiarize ourselves with hooks, we will implement a new hook function in the **First example** module.

It is a helpful practice to always include help text in contributed modules. To do so, we will use the `hook_help()` function, whose description and source code can be found at: [F44.38](#)

http://api.drupal.org/api/drupal/modules--help--help.api.php/function/hook_help/7

The `hook_help()` function allows a module to include general help that applies to the module as a whole, or, by making use of specific paths, to provide user help in various appropriate points in the application.

As stated in the Drupal API documentation, the function must be passed two parameters: `$path` and `$arg`.

`hook_help($path, $arg)`

Via the `$path` parameter, the hook function will know the path to the page being loaded at the moment, in the form in which it was defined in its corresponding

hook_menu() (admin/node, user/edit, node/%, node/%/edit, admin/help#modulename, etc.). Within the function we will define which paths the module's help function should display.

The **\$arg** parameter allows access to the rest of the URL parameters, in case we want to use them in order to show a more specific or personalized help text. For example, if we are dealing with the \$path equal to node/%, we might display different help texts according to the nid in question. In the case of node/76, with \$path == 'node/%', we know that \$arg[1] =='76'.

Generally, we will only use the value of the **\$path** parameter to show the corresponding help for the paths which the module affects.

F44.38

hook_help() function

In api.drupal.org we find all the information necessary to correctly use each hook function. It is especially important to know which parameters the function accepts, and what value or values it returns.

We should always check that we are consulting the documentation for the correct Drupal version.

[Drupal 7](#) » [help.api.php](#)

hook_help

```
5 core.php hook_help($section)
6 core.php hook_help($path, $arg)
7 help.api.php hook_help($path, $arg)
8 help.api.php hook_help($path, $arg)
```

Provide online user help.

By implementing **hook_help()**, a module can make documentation available to the user for the module as a whole, or for specific paths. Help for developers should usually be provided via function header comments in the code, or in special API example files.

For a detailed usage example, see [page_example.module](#).

Parameters

\$path The router menu path, as defined in **hook_menu()**, for the help that is being requested; e.g., 'admin/people' or 'user/register'. If the router path includes a wildcard, then this will appear in \$path as %, even if it is a named %autoloader wildcard in the **hook_menu()** implementation; for example, node pages would have \$path equal to 'node/%' or 'node/%/view'. To provide a help page for a whole module with a listing on admin/help, your hook implementation should match a path with a special descriptor after a "#" sign: 'admin/help#modulename' The main module help text, displayed on the admin/help/modulename page and linked to from the admin/help page.

\$arg An array that corresponds to the return value of the **arg()** function, for modules that want to provide help that is specific to certain values of wildcards in \$path. For example, you could provide help for the path 'user/1' by looking for the path 'user/%' and \$arg[1] == '1'. This given array should always be used rather than directly invoking **arg()**, because your hook implementation may be called for other purposes besides building the current page's help. Note that depending on which module is invoking **hook_help**, \$arg may contain only empty strings. Regardless, \$arg[0] to \$arg[11] will always be set.

Return value

A localized string containing the help text.

The value returned by the hook should be a "translatable" help text. As we have already seen, in order for a text to be translatable, it is necessary to use the translation function **t()**, as we will see in the following examples.

Below, we are going to implement **hook_help()** for the **First example** module. We will add the function to the **first_example.module** file, following the naming convention for hook functions already covered, **<modulename>_<hookname>**, such that the function to be created will be called **first_example_help()**. [F44.39](#)

```
/***
 * Implements hook_help().
 */
function first_example_help($path, $arg) {
  switch ($path) {
    // General help for First example module
    case 'admin/help#first_example':
      return '<p>' . t('This module shows a welcome message into a
block.') . '</p>';
    // Specific help which will be shown in the block
    administration area
    case 'admin/structure/block':
      return '<p>' . t('Use the block Welcome block to show a
welcome message.') . '</p>';
  }
}
```

F44.39**Implementation of
hook_help()**

We will add an implementation of hook_help() to the module First example, creating the first_example_help() function.

In the hook function help text strings have been defined for two cases: general help for the module (URL /admin/help#first_example) and more specific help which will be shown in the block administration area (URL /admin/structure/block).

Once we have made these changes, we will upload the updated file to the server, replacing the existing files. Some changes to modules in development will have immediate and direct effects on the site. In other cases, these changes require a database update, the emptying of cache, etc. As a general rule, it is a good practice to execute update.php after making changes in a module or theme:

<http://www.example.com/update.php>

We can also use the **Reinstall modules** option of the **Devel** module. Keep in mind that when a module is reinstalled, it is first completely uninstalled, eliminating its database tables and all the data within them. When the changes made to the module do not affect the installation, the recommended procedure is to execute only update.php and in certain cases clear all site caches. It is also a good idea to clear the browser cache using Control+F5.

In order to understand the logic behind the first URL (admin/help/first_example), load the help page for the administration area, available at: **F44.40**

Administration ⇒ Help

URL Help
</admin/help>

In the help page, whose URL is **admin/help**, a list with of all modules with help texts available will be displayed. For each module, a URL is generated of the form **admin/help/modulename**. The special # symbol is necessary so that the link to the module's help will be displayed in the help page for the administration area. Therefore, for the help for our **First example** module, we will use the URL admin/help#first_example.

F44.40**hook_help() function**

List of modules with available help. In the list the First example module is displayed.

Follow these steps to set up and start using your website:

- Configure your website** Once logged in, visit the [administration section](#), where you can [customize and configure all aspects of your website](#).
- Enable additional functionality** Next, visit the [module list](#) and enable features which suit your specific needs. You can find additional modules in the [Drupal modules download section](#).
- Customize your website design** To change the "look and feel" of your website, visit the [themes section](#). You may choose from one of the included themes or download additional themes from the [Drupal themes download section](#).
- Start posting content** Finally, you can [add new content](#) for your website.

For more information, refer to the specific topics listed in the next section or to the [online Drupal handbooks](#). You may also post at the [Drupal forum](#) or view the wide range of [other support options](#) available.

Help topics

Help is available on the following items:

- Block
- Color
- Comment
- Contextual links
- Dashboard
- Database logging
- Date
- Field
- Field Permissions
- Field SQL storage
- Field UI
- File
- Filter
- Forum
- Help
- Image
- List
- Locale
- Masquerade
- Media
- Menu
- Module Builder
- Node
- Node clone
- Number
- Options
- PHP filter
- Path
- Pathauto
- RDF
- Search
- Shortcut
- System
- Taxonomy
- Taxonomy Access Control
- Text
- Token
- Toolbar
- Update manager
- User
- Views Accordion
- Webform

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

F44.41**Module help page**

The general help for the module is available through the administration area help.

Once again we will translate the texts which have been passed through the `t()` function, and which will be available from the translate interface. **F44.41** **F44.42**

[Home](#) » [Administration](#) » [Help](#)**First example**

This module shows a welcome message into a block.

F44.42**Module help page**

The help added to the block administration area will be displayed along with other help texts added by the system or by other modules.

[Home](#) » [Administration](#) » [Structure](#)**Blocks** **BARTIK****SEVEN**

This page provides a drag-and-drop interface for assigning a block to a region, and for controlling the order of blocks within regions. Since not all themes implement the same regions, or display regions in the same way, blocks are positioned on a per-theme basis. Remember that your changes will not be saved until you click the [Save blocks](#) button at the bottom of the page. Click the [configure](#) link next to each block to configure its specific title and visibility settings.

[Demonstrate block regions \(Bartik\)](#)

Use the block Welcome block to show a welcome message.

[+ Add block](#)

In the module administration area, a link to **Help** for the module will be displayed, thanks to our implementation of `hook_help()`. **F44.43**

MY MODULES				
ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input checked="" type="checkbox"/>	First example		My first Drupal module	Help

F44.43**Module Help Page**

The general help for the module is available through the Help link displayed in the module list.

Internally, the functioning is quite simple. Let's see what happens when we load **admin/structure/block**:

1. The user asks to load **admin/structure/block**. Let us assume that the user has sufficient permissions to view this page.
2. During page loading, the system calls all the **hook_help()** functions of installed and active modules. Some of these functions will return help messages for the page that's loading, **admin/structure/block**, but others will simply return nothing.
3. Once the system arrives at **first_example_help()**, the values of \$path will be evaluated. In this case, \$path has the value of the page that is requested: \$path == admin/structure/block.

For the first case in our switch, **case 'admin/help#first_example'**, no value will be returned, but for the second, **case 'admin/structure/block'**, the corresponding help text will be returned.

4. When page loading is complete, the system will display all the pertinent help texts in the appropriate area.

44.3 Module configuration

In order to introduce the concepts behind **module configuration** in this section, we will begin developing a new module. We will create a personalized module called **Node Expiration Date**, which will allow us to add an expiration date to any node. When the system detects, through a cron job, that the expiration date of a node has arrived, the node will stop being published in the site.

The module will have a configuration page from which we will be able to select which types of content will include a node expiration date.

Keep in mind that in Drupal's node repository you may find modules with similar functionality to the one we are about to implement. The implementation of this module has only didactic goals, and should be followed step by step in order to understand how to construct a module from scratch.

Step 1. Creation of the module directory

The first step consists of creating a folder where we will add the various files that make up the module. We will create a directory called **node_expiration_date** in /sites/all/modules.

Step 2. Creation of the .info file

Within the directory /sites/all/modules/node_expiration_date, we will create the information file for the module, **node_expiration_date.info**: F44.44

F44.44

.info file

Module configuration file
for Node Expiration Date.

```
name = "Node Expiration Date"
description = "Add an expiration date to content"
core = 7.x
files[] = node_expiration_date.module
package = My modules
```

It is important to save the file with the **UTF8 without BOM** format. This will allow us to add special characters such as accented letters.

Step 3. Creation of the .module file

The next step will be to begin work on the module itself, creating the main module file **node_expiration_date.module**, also in the **UTF8 without BOM** format.

Step 4. Add online help with hook_help()

The **hook_help()** help allows us to add online help for module users. In order to implement the hook, we will create the function **node_expiration_date_help()** within the main module file (node_expiration_date.module): F44.45

```
<?php

/**
 * @file
 * Module Node Expiration Date
 * This module allows the addition of an expiration date to nodes.
 */

/**
 * Implements hook_help().
 */
function node_expiration_date_help($path, $arg) {
  switch ($path) {
    // General help for the Node Expiration Date module
    case 'admin/help#node_expiration_date':
      return '<p>' . t('This module permits addition of an expiration date to any content in the site. The administrator can configure which content types could have an expiration date.') . '</p>';
  }
}
```

F44.45**Implementation of hook_help()**

We use the `hook_help()` function for the Node Expiration Date module in the `node_expiration_date_help()` function.

Step 5. Prepare the module configuration page

The module will have a configuration page from which we will be able to select which content types will include a node expiration date.

In order to make this page available, we must first register the URL, which is accomplished through Drupal's menu system, implementing the `hook_menu()` function. The function is described at:

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_menu/7

Drupal's menu system, which we will study more in **Unit 46**, not only serves to create site menus and menu elements, but also is used for site pathing. For this reason, any path within the site (such as `node/12`) will pass through the menu system in order to determine what contents the site should load when that URL is requested. **F44.46**

```
/**
 * Implements hook_menu().
 */
function node_expiration_date_menu() {
  $items['admin/config/workflow/node_expiration_date'] = array(
    'title' => 'Node Expiration Date settings',
    'description' => 'Settings for module Node Expiration Date',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('node_expiration_date_admin_settings'),
    'access callback' => 'user_access',
    'access arguments' => array('administer site configuration'),
    'type' => MENU_NORMAL_ITEM,
    'file' => 'node_expiration_date.admin.inc',
  );
  return $items;
}
```

F44.46**Implementation of hook_menu()**

In the implementation of `hook_menu()` we register the URL of the module configuration page.

The implementation of `hook_menu()` returns an array `$items` with the new URLs which will be available to the system. In our function we add the path: '`admin/config/workflow/node_expiration_date`', which indicates that the configuration option will be shown in the administration area, specifically within:

Administration ⇒ Configuration ⇒ Workflow ⇒ Node Expiration Date

URL Node Expiration Date

/admin/config/workflow/node_expiration_date

The defined path will have the following parameters:

- **title.** The title of the menu option
- **description.** Description of the menu option
- **page callback.** The name of the callback function which will be called by the system when the URL defined in the menu is accessed.
- **page arguments.** An array with the arguments to be passed to the function defined in the "page callback" parameter.
- **access callback.** Function used to control page access.
- **access arguments.** A vector with the arguments to be passed to the access validation function indicated by "access callback".
- **type.** Describes the properties of the menu element.
- **file.** The file in which the callback function is defined.

The configuration area of our module will be a page with a list of the different content types used in our web site, each one with a corresponding checkbox, such that the user can select which content types will use the expiration date.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Step 6. Module configuration page

It is not necessary to load the module administration page with all of its functions, since the system only needs to load it when a user attempts to access the module configuration URL (`admin/config/workflow/node_expiration_date`). For this reason, a common practice is to separate the callback function generated by the administration page into an independent file, which we will call:

node_expiration_date.admin.inc

When we add a new file to the module implementation, we will need to reference it in the **files[]** parameter of the module definition file (.info). **F44.47** We must also use the **configure** directive to indicate the path that the configuration area will use.

F44.47

Adding files to the module

We must register the new files in the .info file, using `files[]`.

```
name = "Node Expiration Date"
description = "Add an expiration date to content"
core = 7.x
files[] = node_expiration_date.module
files[] = node_expiration_date.admin.inc
configure = admin/config/workflow/node_expiration_date
package = My modules
```

The "page callback" parameter has the value **drupal_get_form**, which is a function of the Drupal API which allows for web form construction. In **Unit 47** the various Drupal API functions related to form construction will be analyzed more closely. The definition of the **drupal_get_form()** function can be consulted here:

http://api.drupal.org/api/drupal/includes--form.inc/function/drupal_get_form/7

Drupal will call the **drupal_get_form()** function, which will in turn call the function given in the "page arguments" parameter (`node_expiration_date_admin_settings`). This second function will contain the structure of the form to be generated, and is the function we will implement in the file **node_expiration_date.admin.inc**. **F44.48**

```

<?php

/**
 * @file
 * Result of calls to module administration pages for
 * Node Expiration Date.
 */

/**
 * Form constructor for module configuration
 *
 */
function node_expiration_date_admin_settings() {

/**
 * With node_type_get_types() we obtain information about all content
 * types, each one in the form of an object, so that one only has to
 * select the name of the content type in order to show it in the options.
 */

$content_types_list = node_type_get_types();
foreach ($content_types_list as $key => $type) {
    $list[$key] = $type->name;
}
$form['node_expiration_date_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Add an expiration date to these content types'),
    '#options' => $list,
    //the variable used to store the configuration;
    //it is best practice to name it the same as the form field
    //used, in this case 'node_expiration_date_node_types'.
    '#default_value'=> variable_get('node_expiration_date_node_types',
        array('page')),
    '#description' => t('The selected content types will have an expiration date.'),
);
return system_settings_form($form);
}

```

F44.48**Administration file**

We have separated the functions that present the administration area of the module into the .admin.inc file.

Although we will go more deeply into the **Form API** in **Unit 47**, it is necessary at this point to offer a first, brief treatment. Forms in Drupal are represented as a nested tree, via an array of arrays. We can see how in the code for our module, we only have access to one element of the form, `$form['node_expiration_date_node_types']`, with the following characteristics:

- **#type**. The type of form element. In our case, this is a checkbox.
- **#title**. The title that appears in the form element. The title and other text elements will be translatable thanks to our use of the `t()` function.
- **#options**. Contains an array with the key/value elements necessary for the checkbox content type. In our example we have constructed the array based on the `node_type_get_types()` function, which we can find in the Drupal API here:

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_type_get_types/7

This function returns a list of all the types of nodes available in the site.

- **#default_value**. Defines the default value for the form element. In our example we have defined the default value as the Page content type, which we obtain from the `variable_get()` function, which gives access to the variables stored in the table variable. You can consult the function in the API :

http://api.drupal.org/api/drupal/includes--bootstrap.inc/function/variable_get/7

- **#description.** Description of the field, also translatable by using the t() function.

Step 7. Testing administration area functionality

Having arrived at this point, we are ready to install the module and access its administration area from:

Administration ⇒ Configuration ⇒ Workflow ⇒ Node Expiration Date

By default the **Basic Page** option will be checked (the internal value is known as "page"). By selecting other options, we can test that the changes are saved correctly. **F44.49**

[Home](#) » [Administration](#) » [Configuration](#) » [Workflow](#)

Node Expiration Date settings

Add an expiration date to these content types

Article

Basic page

The selected content types will have an expiration date.

[Save configuration](#)

In order to save the selected values and retrieve them each time we access the module configuration area, we use the **node_expiration_date_node_types** variable from the variable table.

Note that we are dealing as well with the name of the form field, so that when these values are changed and the form is submitted (save configuration), the new values will also be stored in the same variable.

In order to confirm that these values are stored in the database, access the database using phpMyAdmin and check the value of the variable, or use the **Variable editor** tool offered by the **Devel** module.

Step 8. String translation

From the **Translate interface** area we should find the strings from our module and assign the corresponding translations: **F44.50**

- **Add an expiration date to content** (for example, in Spanish: Añadir una fecha de caducidad al contenido).
- **This module permits to add an expiration date to any content in the site. The administrator can configure which content types could have an expiration date.** (for example, in Spanish: Este módulo permite añadir una fecha de caducidad a cualquier contenido del sitio. El administrador puede configurar qué tipos de contenido podrán tener una fecha de caducidad).
- **Node Expiration Date settings** (for example, in Spanish: Configuración de Node Expiration Date)
- **Settings for module Node Expiration Date** (for example, in Spanish: Opciones de configuración del módulo Node Expiration Date).
- **Add an expiration date to these content types** (for example, in

- Spanish: Añadir una fecha de caducidad a estos tipos de contenido).
- **The selected content types will have an expiration date** (for example, in Spanish: Los tipos de contenido seleccionados tendrán una fecha de caducidad).

Inicio » Administración » Configuración » Flujo de trabajo

Configuración de Node Expiration Date

Añadir una fecha de caducidad a estos tipos de contenido

Artículo

Evento

Página básica

Los tipos de contenido seleccionados tendrán una fecha de caducidad.

F44.50
Module configuration
 Translated module configuration area
 (translated into Spanish).

Guardar configuración

We will set aside the **Node Expiration Date** module for now. We will return to do more work in this module, in upcoming sections and units, in order to complete its functionality.

44.4

Working with the database

In this section we will introduce certain concepts related to manipulating the database, concepts which will be covered more fully later, in **Unit 45**. We will continue working with the **Node Expiration Date** module.

In order to store the expiration date for each node, we will create a new table in the database and call it **node_expiration_date**. This way, we will keep the module information separated from the tables belonging to Drupal core.

The **node_expiration_date** table will have the following fields:

- The node identifier (nid).
- The expiration date of the node (expiration_date).

The primary key of the table will be the node identifier. Because of this, each node will have one and only one expiration date.

The SQL code to generate the table will be the following:

```
CREATE TABLE node_expiration_date (
    nid int(10) NOT NULL,
    expiration_date int(11) NOT NULL default '0',
    PRIMARY KEY (nid),
);
```

As we will see, **we do not create the table manually**, nor do we do so with the SQL code above. Drupal offers a database abstraction layer which allows, among other things, the definition of table structure in a unique way. This structured information, known as the **database schema**, is used during module installation in order to create the table, and during its uninstallation, in order to remove the table.

Module installation file

The module installation file should follow the pattern **<modulename>.install**. For the Node Expiration Date module, we will create the file named **node_expiration_date.install**, which will be placed within the module directory, alongside the rest of the files already created.

This file will contain the function that implements **hook_schema()**, which is responsible for defining the database schema:

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_schema/7

We will study the database schema and its related topics in greater depth in **Unit 45**.

The file **node_expiration_date.install** will have the following content: F44.51

```
<?php

/**
 * Implements hook_schema()
 */
function node_expiration_date_schema() {
  $schema['node_expiration_date'] = array(
    'description' => t('Table to store node expiration dates'),
    'fields' => array(
      'nid' => array(
        'type' => 'int',
        'unsigned' => TRUE,
        'not null' => TRUE,
        'description' => t('The node id {node}.nid with an expiration date'),
      ),
      'expiration_date' => array(
        'description' => t('Expiration date for the node: Unix timestamp'),
        'type' => 'int',
        'not null' => TRUE,
      ),
    ),
    'primary key' => array('nid'),
  );
  // The defined schema is returned.
  return $schema;
}
```

F44.51**.install file**

In the module's .install file are included the actions which will be carried out during the installation and uninstallation processes.

We must implement i hook_schema() to indicate the structure of the database elements added by the module (new tables or additional fields in existing tables).

The database schema installation is carried out when the module is enabled for the first time. The **hook_install()** function can be used to carry out actions on database tables. The uninstallation of the database schema will occur after the disabling and uninstallation of the module, and the **hook_uninstall()** can in turn be used to act on data.

At this point, we can uninstall the module (deleting the module directory and updating the list of modules), upload the latest **node_expiration_date** folder with all of its files (including **node_expiration_date.install**), and continue with a new installation, checking to ensure that the **node_expiration_date** table is generated within the database.

In order to uninstall the module we should go to the **Uninstall** tab which we can find in the module administration area. In order for the module to appear listed among the "uninstallable" modules, we must first disable it from the module list. If the uninstall executes correctly, the table generated by the module, as well as its associated variables, will be deleted.

Keep in mind that disabling the module or simply deleting its folder does not really eliminate all the information stored by the module in the database, and it is therefore very advisable to provide modules a corresponding uninstall function.

In order to **uninstall/install** the module, we can also use the **Reinstall modules** utility provided by the **Devel** module.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

As we already know, Drupal needs a database management system to be able to function. In order to avoid any dependency on a particular database management system, many applications include a **database abstraction layer**, which acts as an intermediary between the application and the database.

In this model, used as well by Drupal, direct queries to the database are not used. Instead, they are replaced by generic functions defined by the **database abstraction layer**. These functions will always be identical, regardless of the database management system used (MySQL, PostgreSQL, Oracle, etc.), which allows us to change the type of database being used without affecting the application itself.

Drupal can be installed with various database management systems, such as MySQL, SQLite, PostgreSQL, Oracle, etc.

Comparative D7/D6

Drupal 7 has important differences in database access, compared to what was used in Drupal 6.

An important change in the database abstraction layer of Drupal 7 is the use of the **PDO library** to access the database.

Drupal 7 also differentiates between static queries (which can use db_query()) and dynamic queries, which use new API functions: db_select(), db_insert(), db_update() and db_delete().

Unit contents

45.1 Database Access	86
45.2 The Schema API	95
45.3 Other Schema API Functions	99



45.1 Database Access

As we already know, Drupal requires a database management system to be able to function. In order to avoid any dependency on a particular database management system, many applications include a database abstraction layer, which acts as an intermediary between the application and the database.

In this model, which is used by Drupal as well, direct queries to the database are not used. Instead, they are replaced by generic functions defined by the **database abstraction layer**. These functions will always be identical, regardless of the database management system used (MySQL, PostgreSQL, Oracle, etc.), which allows us to change the type of database being used without affecting the application itself.

Selecting the database during installation

During the Drupal installation process, database parameters are configured:

- **Database type:** MySQL, SQLite, PostgreSQL, etc.
- **Database name.**
- **Database username** used by Drupal to connect to the database
- **Database password** used by Drupal for user authentication during the connection process.

The installation process is carried out using the browser, and includes changing the file **settings.php**, which is located by default in **sites/default/settings.php**, where we find the following code: **F45.1**

F45.1

settings.php

Database definition in the configuration file settings.php.

```
$databases = array (
  'default' =>
    array (
      'default' =>
        array (
          'database' => 'db_name',
          'username' => 'db_username',
          'password' => 'db_password',
          'host' => 'localhost',
          'port' => '',
          'driver' => 'mysql',
          'prefix' => '',
        ),
      ),
    );
);
```

These variable assignments in the **\$databases** configuration array indicate that there is a connection to a MySQL database named "db_name" which is accessed using the username "db_username" and the password "db_password".

In the **host** field, the server location of the database is specified. The "localhost" value indicates that the database can be found in the same server where Drupal has been installed. Although this is the most common scenario, it may happen that the database management system is located in a different server, in which case we would substitute "localhost" with the IP address or domain where the database is hosted.

In the **port** port field we can indicate the port number that corresponds to the database connection. If no value is included, the default value is used.

The **prefix** field allows us to assign a prefix to the tables in the database. This option is useful when we need to share the same database with several different applications.

Drupal handles database connections automatically upon system startup, which means that we do not need to handle connectivity at the level of each module.

Query Execution

Module development that makes use of a database abstraction layer will allow us to communicate with the database independently of its type, as long as its queries follow **SQL** standards.

At the same time, using the database abstraction layer helps us protect our database from direct accesses which may contain errors and cause undesired behavior.

In Drupal 6 the SELECT, INSERT, UPDATE and DELETE commands were all made using the API function **db_query()**. In order to achieve compatibility with all database types, in Drupal 7, new functions have been added to the **database abstraction layer**:

- **db_query()**. This function can still be used to make simple SELECT queries contained in a single line of text. For other SELECT queries we will use **db_select()**.
- db_query()** should not be used with INSERT, UPDATE or DELETE commands.
- **db_query_range()**. Executes a SELECT query that is limited to a given range.
- **db_select()**. Selects records from the database (SELECT).
- **db_insert()**. Inserts records into the database (INSERT).
- **db_update()**. Updates records in the database (UPDATE).
- **db_delete()**. Removes records from the database (DELETE).

In order to execute a SQL query we will call Drupal's generic functions, instead of calling database-specific functions like **mysql_query()** or **pg_query()**.

PDO

An important change in the database abstraction layer in Drupal 7 is the use of PHP's **PDO library** to access the database.

Although Drupal offers a set of high-level functions, we will also need to use certain PDO functions, so it's useful to know its syntax. You can find more information at: <http://php.net/manual/en/book pdo.php>

db_query() query

As already discussed, we can use the **db_query()** function to make simple or static SELECT queries.

We can find all the information and use case examples in the API:

http://api.drupal.org/api/drupal/includes--database.inc/function/db_query/7

When using db_query() we should keep in mind the following aspects of its syntax:

- Table names should be placed between curly braces: {tablename}. During Drupal installation, we can indicate whether table names will include a prefix. By using the {tablename} convention to refer to tables, Drupal takes on the responsibility of replacing the text with the real name of the table.
- The use of substitution parameters will use PDO nomenclature, which is analyzed below.

In Drupal 7, the method of adding substitution parameters in SQL queries uses PDO syntax, which is described below: **F45.2**

```
//the query returns the nodes created by a given user ($uid)

// Drupal 7
$result = db_query('SELECT nid, title FROM {node} WHERE uid = :uid',
array(':uid' => $uid));

// Drupal 6
$result = db_query("SELECT nid, title FROM {node} WHERE uid = %d", $uid);
```

The first parameter passed to the **db_query()** function is always the SQL query to be executed. The rest of the parameters will be values or variables which will undergo substitution in order to generate the final executable query.

In our example we have a WHERE clause which specifies the value of the user ID (\$uid). The :uid parameter will be exchanged for the value of \$uid in the final query which will be executed on the database. The {node} term will also be replaced by the name of the table (with or without a prefix).

Accessing data returned by the query

In order to recover the data returned by queries, we will use PDO functions. We will now take a look at different ways to obtain the returned records.

To test query functioning in our site, we have various quick alternatives (when creating a module):

- Create a node with a PHP code text format
- Use the Execute PHP Code offered by the Devel module.

In either case we will use Devel's filtering functions (**dpm()**, **dprint_r()**, etc.) to display the results.

Without the need for additional functions, we can access the contents of each record with a **foreach** command on the **\$result** variable. In this case each record will be returned as an object (stdClass) in which the properties are the fields specified in the query (\$record->nid, \$record->title, etc.): **F45.3**

```
$uid = 1;
$result = db_query('SELECT nid, title FROM {node} WHERE uid = :uid', array(':uid' => $uid));

foreach ($result as $record) {
  //Access each record as an object with the
  //following properties:
  // $record->nid, $record->title
  dpm($record);
}
```

F45.3**Query Results**

Example of how to access the results of a query.

An equivalent format which also returns each record as a stdClass object is:

```
// Return the following record as a stdClass object.
while ($record = $result->fetchObject()) {
  dpm($record);
};
```

If we want the results returned as an associative array, we can user:

```
// Return the following record as an associative array.
while ($record = $result->fetchAssoc()) {
  dpm($record);
};
```

We can also obtain all the records in a single call, as an object array:

```
// Return the following records as a stdClass object array.
$records = $result->fetchAll();

dpm($records);
```

Here are other methods used to obtain values:

```
// Return an object array where the array key
// will be the value of the indicated field.

$records = $result->fetchAllAssoc('nid');

// Get an entire column in a single array.
$records = $result->fetchCol();

// Return the number of rows.
$contador = $result->rowCount();
```

db_query_range() query

The **db_query_range()** command allows us to specify a range for the values returned by a SELECT query. It functions like a LIMIT used within the query itself.

We can find complete information and use case examples in the API:

http://api.drupal.org/api/drupal/includes--database--database.inc/function/db_query_range/7

```
db_query_range($query, $from, $count, array $args = array(),
               array $options = array())
```

The parameters of the function are:

- **\$query.** SELECT query which follows the same conventions as in db_query().
- **\$from.** Indicates the numerical value or position of the first record which the query will return. The first position is 0.
- **\$count.** Indicates the limit, or number of records to return.
- **\$args.** Arguments which will be replaced in the query.
- **\$options.** Additional options.

In the following example, the query returns five records (\$count = 5), beginning with the first returned value (\$from = 0).

```
$uid = 1;
$result = db_query_range('SELECT nid, title FROM {node} WHERE uid
= :uid', 0, 5, array(':uid' => $uid));
```

The methods for accessing the returned values are similar to those described for db_query().

Dynamic queries

Dynamic queries are dynamically constructed by Drupal, instead of being based on a text string. As already discussed, in Drupal 7, the **db_query()** command should only be used to execute static SELECT queries. New Drupal functions will be used to construct dynamic queries: **db_select()**, **db_update()**, **db_insert()** and **db_delete()**. All UPDATE, INSERT, and DELETE commands are considered dynamic.

The new functions for generating dynamic queries return an **object** upon which we apply **methods** to complete the query dynamically. Before examining the available methods, let's analyze an initial example: **F45.4**

F45.4

Dynamic Queries

An example of dynamic query construction. All INSERT, UPDATE and DELETE queries are considered dynamic and should use the new API functions provided by Drupal 7.

```
//SELECT query on the 'node' table, with the table alias 'n'.
//We add the nid and title fields.
//We add the condition that nid > 1.
$query = db_select('node', 'n')
  ->fields('n', array('nid', 'title'))
  ->condition('nid', 1, '>');

//We can continue adding methods to the object to construct
//the query dynamically.
//Results are ordered according to the 'title' field, ascending.
$query->orderBy('title', 'ASC');

//Execute the query
$result = $query->execute();

//We obtain the values returned by the query
while($record = $result->fetchAssoc()) {
  dpm($record);
}
```

When the function (for example, db_select()) is called, we are constructing the object with the query. During the construction we can apply whatever methods are necessary in order to configure the query.

In the example we use the **fields()** method to add additional fields to the 'node' table, to which we have assigned the alias 'n'.

We also use the **condition()** method to add conditions. These are equivalent to the typical conditions normally added in a WHERE clause. The SQL equivalent of the condition added here is (WHERE nid > 1).

Once the \$query object is built, we can continue to alter it using the available methods. As an example, we have added an ordering condition, using the **orderBy()** method.

Finally, we execute the query using the **execute()** method. Once the query is executed, we arrive at the values returned by the query using the methods applicable to the \$result object, which we have already seen (for example, \$result->**fetchAssoc()**).

Additional information about the use of dynamic queries and their associated methods is available at: <http://drupal.org/node/310075>.

Some of these methods include:

- **addField()**. Add a field to the query. We indicate the name of the table (or the alias), the name of the field, and, optionally, an alias for the field being added. In order to add more than one field, we will call the method for each field.

```
//Add the 'title' field to the table with the alias 'n'
//Also, give the field the alias of 'my-title'
```

```
$query->addField('n', 'title', 'my-title');
```

- **fields()**. Allows the addition of multiple fields via the use of an array.

```
$query->fields('n', array('nid', 'title', 'created', 'uid'));
$query->fields('n'); //add all fields (*)
```

- **distinct()**. Apply the DISTINCT parameter to the query, so that no duplicate results will be returned.

```
$query->distinct();
```

- **addExpression()**. Allows the addition of other expressions (COUNT(), CONCAT(), LENGTH(), SUM(), etc.).

```
$query->addExpression('COUNT(uid)', 'uid_count');
```

- **orderBy()**. Add an ordering clause to the query (ORDER BY).

```
$query->orderBy('title', 'ASC');
```

- **groupBy()**. Allows results to be grouped by a given field (GROUP BY).

```
$query->groupBy('uid');
```

- **range()**. Allows the specification of a range, such that the query will return only the results between the indicated limits. This method corresponds to LIMIT, where the first parameter indicates the position of the first element and the second parameter indicates the number of records that the query will return. We can also use the **limit()** method.

```
//return 10 elements starting with the fifth
$query->range(5, 10);

//return the first 10 results
$query->range(0, 10);
```

- **join()**, **innerJoin()**, **leftJoin()** and **rightJoin()**. Allows two or more tables to be joined in order to obtain combined results.

```
$query->join('user', 'u', 'n.uid = u.uid AND u.uid = :uid',
array(':uid' => 5));
```

- **condition()**. Allows for the addition of conditions, using the (field-value-operator) structure. It is also possible to create additional conditions with AND, OR and XOR operators. The conditions are discussed in detail here: <http://drupal.org/node/310086>

```
//the value of nid>1
$query->condition('nid', 1, '>');
//the value of nid is between 5 and 10
$query->condition('nid', array(5, 10), 'BETWEEN');
```

- **isNull()**, **isNotNull()**. These conditions test whether a field is null or not null.

```
$query->isNull('title');
$query->isNotNull('uid');
```

- **execute()**. Once we've completed the query, we call the **execute()** method to compile and execute the command.

```
//Execute the query
$result = $query->execute();

foreach ($result as $record) {
    //actions to be done to each record ($record)
}
```

db_select() query

The **db_select()** function is new in Drupal 7, and replaces **db_query()** in dynamic SELECT queries.

http://api.drupal.org/api/drupal/includes--database.inc/function/db_select/7

db_select(\$table, \$alias = NULL, array \$options = array())

The **db_select()** function generates an object of the **SelectQuery** type, which we can configure by using the aforementioned methods.

The parameters of the table are:

- **\$table.** The name of the table. Keep in mind that it is no longer necessary to indicate the name of the table within curly braces {node}. The system takes on the role of finding the table with the right prefix.
- **\$alias.** We can optionally set an alias for the table. Alias use is particularly recommended for long table names.
- **\$options.** Additional options.

Let's consider some examples: F45.5

```
// Return all fields of the table named 'node' where 'type'='article'.
// Return the first 30 results.
$query = db_select('node', 'n')
  ->condition('type', 'article', '=')
  ->fields('n')
  ->range(0, 30);

$result = $query->execute();
```

F45.5

db_select() query

Example of how to build SELECT queries with db_select().

```
// Return all fields of the table named 'node' where
// the value of nid matches $node->nid, status > 0
// and uid is 1, 5 or 7.
$result = db_select('node', 'n')
  ->fields('n')
  ->condition('nid', $node->nid, '=')
  ->condition('status', 0, '>')
  ->condition('uid', array(1,5,7), 'IN')
  ->execute();
```

db_insert() query

The **db_insert()** function allows records to be inserted in a database table. The function creates an insertion object which we can complete dynamically using certain methods.

http://api.drupal.org/api/drupal/includes--database.inc/function/db_insert/7

db_insert(\$table, array \$options = array())

The parameters of the function are:

- **\$table.** The name of the table where the insertion will be done. This does not require {}.
- **\$options.** Additional options.

Some typical methods include: **fields()**, for adding fields and their values, and **execute()**, which executes the command and makes the insertion. F45.6

F45.6

db_insert() command
Example of how to build INSERT commands with db_insert().

```
//Insert in the poll table the values of the nid, runtime,
//and active fields obtained from the variable $node
$result = db_insert('poll')
  ->fields(array(
    'nid' => $node->nid,
    'runtime' => $node->runtime,
    'active' => $node->active,
  ))
  ->execute();

//Insert in the contact table two fixed values (string and
//number) and a variable from the variable table (variable_get()).
$result = db_insert('contact')
  ->fields(array(
    'category' => 'Website feedback',
    'recipients' => variable_get('site_mail', ini_get('sendmail_from')),
    'selected' => 1,
    'reply' => '',
  ))
  ->execute();
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

db_update() query

The **db_update()** function is used to update existing records in a database table. The function creates an object which we can complete dynamically by using certain methods.

http://api.drupal.org/api/drupal/includes--database--database.inc/function/db_update/7

db_update(\$table, array \$options = array())

Typical methods include: **fields()**, which indicate which fields will be updated; **condition()**, to add conditions which will determine which records will be updated; and **execute()** in order to execute the command and update the records. F45.7

F45.7

db_update() command
An example of how to build UPDATE commands with db_update().

```
db_update('node')
  ->fields(array('vid' => $node->vid))
  ->condition('nid', $node->nid)
  ->execute();
```

db_delete() query

The **db_delete()** function eliminates records from a table which meet given conditions. The function creates an object which we can complete dynamically by using certain methods.

http://api.drupal.org/api/drupal/includes--database--database.inc/function/db_delete/7

db_delete(\$table, array \$options = array())

Typical methods include: **condition()**, where we specify which records will be removed as a function of the conditions given; and **execute()** to execute the command. F45.8

F45.8

db_delete() command
Example of how to build DELETE commands with db_delete().

```
db_delete('history')
  ->condition('uid', $account->uid)
  ->execute();
```

Schema API

45.2

We already saw in **Unit 44** how to define the structure of a table using the **hook_schema()** function of the **Schema API**. The use of the Schema API is directly related to the database abstraction layer. In order to remove dependency on any specific database management system, the **Schema API** provides a way to define table structure which will later be translated to the corresponding command to create tables in each database management system (MySQL, PostgreSQL, etc.). In the install files (**.install**) of modules which require the creation of new tables or fields, it is necessary to use these table definition schemas.

Comprehensive information about **Schema API**, including the structure and types of data accepted in the definition of database schemas, can be found in Drupal's documentation: <http://drupal.org/node/146843>

In a module's installation file, we can implement the following hooks:

- **hook_schema()**. Defines the structure of the additional tables added by the module. For each table, its associated fields, keys, and indexes will be defined.
- **hook_install()**. Carries out the indicated tasks during the module installation process. Some of the tasks which are often done within this hook include:
 - o Creating and assigning values to variables within the variable table, via **variable_set()**.
 - o Inserting new records in the tables belonging to the module, or in existing tables.
 - o Creating new content types, vocabularies, and taxonomy terms, defining new entities and fields, etc.
- **hook_uninstall()**. Carries out tasks during the uninstallation of the module. This is generally used to:
 - o Remove tables or fields added by the module.
 - o Remove variables created by the module, via **variable_del()**.

As an example of the **.install** file, we will analyze the example module **DBTNG example (dbtng_example)**, which is included in the module **Examples for developers**. The **DBTNG example** module provides use case examples of the new Drupal 7 database access functions.

We begin with the implementation of **hook_install()**: F45.9

```
/** 
 * Implements hook_install().
 */
function dbtng_example_install() {
  // Add an entry to the dbtng_example table
  $fields = array(
    'name'      => 'John',
    'surname'   => 'Doe',
    'age'        => 0,
  );
  db_insert('dbtng_example')
    ->fields($fields)
    ->execute();
}
```

F45.9

hook_install()

The **hook_install()** function is called during module installation.

Within the implementation of **hook_install()**, a record is added to the **dbtng_example** table. This table is defined in **hook_schema()**, and will be created before **hook_install()** is entered.

In **hook_uninstall()** implementation, F45.10 additional actions can be included which will be carried out during the uninstall of the module, such as the deletion of variables, content types, vocabularies, taxonomy terms, etc.

F45.10**hook_uninstall()**

The **hook_uninstall()** function is called during module uninstallation.

```
/***
 * Implements hook_uninstall().
 */
function dbtng_example_uninstall() {
    // does not perform actions
}
```

Finally, in **hook_schema()** implementation the structure of the tables created by the module is described: F45.11

F45.11**hook_schema()**

The **hook_schema()** function defines the database schema with the new elements added by the module.

```
/***
 * Implements hook_schema().
 */
function dbtng_example_schema() {
    $schema['dbtng_example'] = array(
        'description' => 'Stores example person entries.',
        'fields' => array(
            'pid' => array(
                'type' => 'serial',
                'not null' => TRUE,
                'description' => 'Primary Key: Unique person ID.',
            ),
            'uid' => array(
                'type' => 'int',
                'not null' => TRUE,
                'default' => 0,
                'description' => "Creator user's {users}.uid",
            ),
            'name' => array(
                'type' => 'varchar',
                'length' => 255,
                'not null' => TRUE,
                'default' => '',
                'description' => 'Name of the person.',
            ),
            'surname' => array(
                'type' => 'varchar',
                'length' => 255,
                'not null' => TRUE,
                'default' => '',
                'description' => 'Surname of the person.',
            ),
            'age' => array(
                'type' => 'int',
                'not null' => TRUE,
                'default' => 0,
                'size' => 'tiny',
                'description' => 'The age of the person in years.',
            ),
        ),
        'primary key' => array('pid'),
        'indexes' => array(
            'name' => array('name'),
            'surname' => array('surname'),
            'age' => array('age'),
        ),
    );
    return $schema;
}
```

The **hook_schema()** function returns the **\$schema** array, in which each element is the definition of a table, expressed via an associative array. In the example module, the table dbtng_example is defined with the fields pid, uid, name, surname, and age. The primary key (pid) is also defined, as well as the name, surname and age indexes.

It is important to point out that, if there are references in table descriptions or fields to other database elements that do not form part of the module, these references should be enclosed within curly braces {}. In the schema example, a reference is made to the {users} table. As we've already noted, this allows for converting Drupal table names to their real names, if table prefixing is used.

Below, we describe the fields of the **\$schema** array, which defines the schema of the new tables in **hook_schema()**:

- **description:** String which describes the table and its function.
- **fields:** An array that defines the field structure of the table. Every array element is identified with a field name. Each field, in turn, is an array with the following elements:
 - o **description:** String which describes the field and its function.
 - o **Type:** Defines the field's data type: 'varchar', 'int', 'serial', 'float', 'numeric', 'text', 'blob' or 'datetime'. The use of the 'serial' type defines autoincrementing fields.
 - o **size:** Sets the maximum size that the data type can store: 'tiny', 'small', 'medium', 'normal', 'big'.
 - o **not null:** Accepts values of TRUE and FALSE, with FALSE as the default value. If this is set to TRUE, NULL values are not permitted in the field.
 - o **default:** Sets the default value of the field.
 - o **length:** Gives the maximum length of 'varchar', 'text' or 'int' types, ignored in the case of other data types.
 - o **unsigned:** Can be TRUE or FALSE, with FALSE as the default value. If set to TRUE, this indicates that the 'int', 'float', and 'numeric' types are not signed.
 - o **precision, scale:** For 'numeric' types, this indicates the decimal precision.
 - o **serialize:** Accepts values of TRUE and FALSE, indicating whether the field is stored as a serialized string.
- **primary key:** Array with one or more table fields which will form its primary key.
- **unique keys:** Array which defines the unique keys of the table.
- **indexes:** Array which defines table indexes.

The tables defined in **hook_schema()** will be created automatically during the installation process, without the need to add any action in the **hook_install()** implementation. In the same way, the system will take on the responsibility for removing the module's tables without the need to include specific actions in **hook_uninstall()**.

It is important to keep these points in mind:

- Table creation corresponding to the schema defined in **hook_schema()** and the additional operations of **hook_install()** will only be executed when the module is enabled for the first time. That means that if the module has been installed and enabled once, and later disabled and reenabled, the installation process will not be repeated.
- Disabling a module will not execute the uninstalation process. For this

reason, if we re-enable the module, its configuration options and the additional contained in its tables will remain intact.

- To completely eliminate a module and execute **hook_uninstall()**, we must uninstall it from the Uninstall tab in module administration. In this list, only the modules that have been previously disabled will be displayed. Keep in mind that all associated database data will be deleted and that this operation cannot be reversed.
- If, having uninstalled a module, we once again enable it, the module will be reinstalled from the ground up. The system will create the tables anew and carry out the actions defined in **hook_install()**.

Keep in mind that in order for a module to appear in this list, it must fulfill these conditions:

- The module has been previously installed within the site.
- The module is disabled.
- The module has an installation file named **.install**.

Schema Module

The **Schema** module facilitates the work of developers by generating the database schema, which we can later use in the implementation of **hook_schema()**. The **Schema** module is available at:

<http://drupal.org/project/schema>

Configuration options for the module can be found at:

Administration ⇒ **Structure** ⇒ **Schema**

We will only describe the functionality that applies to this section, although the module includes other functionality related to the **Schema API**, which we recommend studying.

From the **Inspect** tab we can view the schema of **all the tables of the database**, including those which were not defined by any module. Therefore, the procedure for generating the content of the **\$schema** variable is as simple as creating the table from the database management interface (for example, phpMyAdmin), and then accessing the code generated by the **Schema module**, copying the code and adding the implementation of **hook_schema()**. F45.12

URL Schema
</admin/structure/schema/inspect>

F45.12

Schema Module

The Schema module makes the task of defining database schemas easier. The module generates the code for the schema using the tables created in the database.

This page shows the live database schema as it currently exists on this system. Known tables are grouped by the module that defines them; unknown tables are all grouped together.

To implement hook_schema() for a module that has existing tables, copy the schema structure for those tables directly into the module's hook_schema() and return \$schema.

BLOCK

```
$schema['block'] = array(
  'description' => 'Stores block settings, such as region and visibility...',
  'fields' => array(
    'bid' => array(
      'description' => 'Primary Key: Unique block ID.',
      'type' => 'serial',
      'not null' => TRUE,
    ),
  ),
  'module' => array(
    'description' => 'The module from which the block originates; for example, "user" for the Who's Online block, and "block" for any
  ),
);
```

Other Functions of the Schema API

45.3

Now we'll take a look at some functions of interest which belong to the Schema API. All the available functions can be viewed at:

<http://api.drupal.org/api/drupal/includes--database--schema.inc/group/schemaapi/7>

hook_schema_alter()

As we've already seen, the implementation of the **hook_schema()** function, added to the **.install** file, allows the module to create new tables during the installation process.

When the module needs to modify the structure of a table created by a different module, generally in order to add additional fields, we have to use two steps:

- Add the fields to the table from **hook_install()**, using the **db_add_field()** function.
- Register the changes in the database schema using the **hook_schema_alter()** function.

The function **hook_schema_alter()** alerts the general schema of the database about changes that have been made. These changes are normally carried out during the installation of a module which adds fields to tables that already exist in the system.

We can find more information about this function in the Drupal API at:

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_schema_alter/7

In the code of the example displayed below we add a new field to the definition of the **node** table from an example module: **F45.13**

```
/** 
 * Implements hook_schema_alter().
 */
function example_module_schema_alter(&$schema) {
  // Definition of a new field in the node table.
  $schema['node']['fields']['new_field'] = array(
    'type' => 'int',
    'unsigned' => TRUE,
    'not null' => TRUE,
    'default' => 0,
    'description' => t('New field in table: {node}'),
  );
}
```

F45.13

hook_schema_alter()

Informs the general schema about the changes made in the database.

Note that the **\$schema** global is passed to the function by reference (**&\$schema**), which allows any module to carry out, through its implementation of **hook_schema_alter()**, modifications of the schema and of any elements which have already been included. Therefore, the fields and database definitions of any previously loaded module can be altered, whether that means additions, changes, or deletions of fields, keys, tables, etc.

It is **very important** to remember that the modification of the schema through the **hook_schema_alter()** function **is not the same as the modification of tables**. With **hook_schema_alter()** we are only informing the system of how the table structure will look after the change is completed. The change in the affected table will take place in **hook_install()**. For example, in order to add the field '**new_field**' that was defined previously:

F45.14**F45.14**

Table Modification

An example of the modification of a table and the corresponding call to **hook_schema_alter()** which updates the database schema.

```
/***
 * Implements hook_install().
 */
function example_module_install() {
  // The schema is updated
  $schema['node'] = array();
  example_module_schema_alter($schema);

  // A new field is added to the node table
  db_add_field('node', 'new_field', array('type' => 'int',
    'unsigned' => TRUE, 'not null' => TRUE,
    'default' => 0));
}

/***
 * Implements hook_uninstall().
 */
function example_module_uninstall() {
  //the field created by the module is removed
  db_drop_field('node', 'new_field');
}
```

The **db_add_field()** function allows us to add a new field to a table that already exists. More information can be found at:

http://api.drupal.org/api/drupal/includes--database--database.inc/function/db_add_field/7

In the same way, when a module is uninstalled, we should remove the extra field that was added. We can do so using the **db_drop_field()** function.

drupal_get_schema()

The **drupal_get_schema()** function returns the schema which defines either a table or the complete database schema. The schema, according to the parameters of the function, will include the new tables created by any module using **hook_schema()** and the modifications made by other modules using the **hook_schema_alter()** function:

http://api.drupal.org/api/drupal/includes--bootstrap.inc/function/drupal_get_schema/7

The parameters which should be passed to the function are:

drupal_get_schema(\$table = NULL, \$rebuild = FALSE)

- **\$table**. This is the table name, which should exist in the schema API. If no table name is given, the complete database schema will be returned.
- **\$rebuild**. If the value TRUE is passed, the schema will be returned starting from a rebuild, instead of being obtained directly from the cache.

drupal_write_record()

The **drupal_write_record()** function allows for the insertion or updating of records in a table based on the database schema definition.

The only required parameter is an array, passed by reference, which contains the values of the record to be inserted in the table.

If the record exists (update), only the values indicated in the array will be updated. If this is a new record (insert), all the values in the vector will be used to populate the record fields, and in the case of an absent value, the default value as given in the database schema will be used.

The function uses the Schema API in order to access the structure of the table, which means that it can only be used with tables for which a corresponding schema has been defined. More information about this function can be found at:

http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_write_record/7

The function requires the following parameters:

drupal_write_record(\$table, &\$record, \$primary_keys = array())

- **\$table:** This is the table name, which should exist in the schema API.
- **\$record:** Object (or array) with the values to be inserted in the table.
- **\$primary_keys:** If the requested operation is the insertion of a new record, this field is omitted. In the case of an update, this argument specifies the primary keys. If there is only one primary key, a string can be passed, but if there are multiple primary keys, an array must be indicated.

The function may return the following values:

- FALSE. This occurs when the function has not been able to carry out the request.
- SAVED_NEW. A new record has been inserted.
- SAVED_UPDATED. An existing record has been updated.

Let's take a look at an example, the **user_role_save()** function from the core User module, which stores an object **\$role** in the **role** table: **F45.15**

```
<?php

function user_role_save($role) {
  //...
  if (!empty($role->rid) && $role->name) {
    // Do an update: the primary key is rid
    $status = drupal_write_record('role', $role, 'rid');
    //...
  }
  else {
    // Add a new record
    $status = drupal_write_record('role', $role);
    //...
  }
  //...
}
?>
```

The **\$role** object has the **rid**, **name**, and **weight** fields, just like the **role** table where it will be stored, using **rid** as the primary key.

F45.15

drupal_write_record()

Example of the use of the **drupal_write_record()** function. The function carries out insertions and updates in a table based on the definition from the database schema.

The **drupal_write_record()** function is used twice:

- In the first case it is used to **update** an existing record, and must therefore be passed the **\$primary_keys** parameter, which includes the primary key of the table ('**rid**'). As we've already discussed, since this is a case of the primary key consisting of a single field, it is passed here as a text string. If the key had been formed by more than one field, it would have had to be passed as an array, where each element of the array was a string with the field name.
- In the second case the function is used to **insert** the complete record as a new record, and therefore the **\$primary_keys** parameter is not used.

The **drupal_write_record()** function carries out the role of inserting and updating records according to the schema of the table as defined in the Schema API.

46 Menu System

In this unit we will look at how the Drupal menu system works and the capabilities it offers from the programming point of view.

The Drupal menu system has a dual functionality. On the one hand, it controls the navigation hierarchically and allows you to organize the contents of the site through the menus and their elements. On the other hand, it is responsible for routing to any page of the website, by calling the callback function corresponding to each URL you are trying to load from the browser. These callback functions are responsible for establishing the content and functionality of each page. Therefore, without the menu system you would not be able to create and reference pages on the site.

Comparative D7/D6

The menu system has not changed from Drupal 6.

Unit contents

46.1 Introduction to the menu system	104
46.2 Implementation of hook_menu().....	105
46.3 URL with parameters.....	109
46.4 Registering URLs and menu elements.....	111
46.5 Definition of tabs	114
46.6 Access control to the page.....	117
46.7 Menu system functions.....	119



46.1 Introduction to the menu system

In this unit we will look at how the Drupal menu system works and the capabilities it offers.

The Drupal menu system has a dual functionality:

- Controls the navigation hierarchically and allows you to organize the contents of the site through the **menus and their elements**.
- **Allows you to record and route any URL of the site**, by calling to the callback function corresponding to each URL, which will be responsible for generating the content of each page.

When the browser makes a request to a web site with Drupal, it's always done through a particular URL, for example <http://www.example.com/node/1234>.

Drupal captures part of this chain which we call a path. Discarding the location of the site or domain (www.example.com). The path that Drupal will analyze will be "**node/1234**". This path corresponds to the string passed through the parameter 'q'.

Remember, if you are not using clean URLs, the corresponding path would be of the type:

<http://www.example.com/index.php?q=node/1234>

Although a node has assigned an alias URL, the system will search for the original URL, and internally will work with this instead of with your alias.

Drupal will check this route on all active modules of the site to find the module that implements it, checking their implementations of **hook_menu()**. Once the module is found, it will return to the system a **callback function**, which is the PHP function that is executed every time it receives a similar URL (in this case node/4, node/23, etc.). This callback function is responsible for generating the content of the page.

For example, to display a node, is the module **Node** in the core responsible for providing, through **node_menu()**, the function **node_page_view()**, that is used to display any page that matches the URL **node/%**, where % represents the value of the node identifier (nid).

Implementation of hook_menu()

46.2

When the user visits a page, the system must determine, from the URL, which module is responsible for its management and presentation.

For example, if it's a node, it will be in the Node module, but if it is a view, will be covered by the Views module.

Each module registers their URLs by implementing the function **hook_menu()**. The system will cycle through all these hooks up to identify the module "owner" of the URL, surrendering control to the presentation.

We will see a basic example of the implementation of the **hook_menu()** function for the module **menu_forcontu**, where we implement the function **menu_forcontu_menu()** in **menu_forcontu.module**.

In the **menu_forcontu_menu()** the URL **url1** is registered and its corresponding function is returned (page callback), called **menu_forcontu_url1()**: F46.1

```
<?php

/**
 * @file
 * Menu Forcontu
 * URLs and menu items examples
 */

/**
 * Implements hook_menu().
 */
function menu_forcontu_menu() {
  $items['url1'] = array(
    'title' => 'Example URL 1',
    'page callback' => 'menu_forcontu_url1',
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}

/**
 * Page callback
 * Function called when loading the URL www.example.com/url1
 */
function menu_forcontu_url1() {
  $output = t('Example Menu Module!');
  return $output;
}
```

F46.1

hook_menu()

Example of the implementation of the **hook_menu()** function and the callback function.

The callback function is responsible for displaying the contents of the page.

When a user requests, through the browser, the function of the URL **www.example.com/url1**, the system **will execute the page callback function** (Declared in the parameter **page callback**) **menu_forcontu_url1()**.

If we analyze the function **menu_forcontu_url1()**, we see that it returns the text "Example Menu Module!" (which we can translate). Therefore, the result of the url **/url1**, will be the "Example Menu Module!". The page will appear with the title "Example URL 1", defined by the parameter 'title' which we can also translate. F46.2

F46.2**Result /url1**

Example of displayed content through the callback function.

Example URL 1

Example Menu Module!

Within the implementation of **hook_menu()**, we will define as many items (in the vector **\$items**) as URLs we want register in the system. **F46.3**

F46.3**Array \$items**

Through the implementation of the **hook_menu()** function, we will define all the URLs generated by the module.

```
<?php

/**
 * Implements hook_menu().
 */
function menu_forcontu_menu() {
  $items['url1'] = array(
    // Definition of url1
    ...
  );
  $items['url2'] = array(
    // Definition of url2
    ...
  );
  $items['url3'] = array(
    // Definition of url3
    ...
  );
  return $items;
}
```

Structure of each hook_menu() item

Before continuing to analyze the most complete examples and uses of **hook_menu()**, we'll review some of the parameters that we will be able to use in the array **\$items**, for each URL registered. The complete listing can be found at:

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_menu/7

- **'title'**: Required value with the title of the menu item, without a translation.
- **'title callback'**: Callback function used to generate the title. By default the function that is used is **t()**, which allows the translation of the title.
- **'title arguments'**: Arguments that are passed to the callback function of the title, that can be **t()** or any custom function indicated in "title callback".
- **'description'**: Description of the menu items, not translated.
- **'page callback'**: Callback function responsible for displaying the page when a user requests to load the URL from browser.
- **'page arguments'**: Array of arguments passed to the callback function defined in "page callback". Integer values indicating we are referencing

the arguments given by the URL, according to the order indicated.

- **'access callback'**: Callback function to determine if the user has access or not to the page. The function returns a Boolean value (TRUE, FALSE). By default it will use the **user_access()** function, except to indicate a particular function or inherits from a parent element.
- **'access arguments'**: Array of arguments passed to the callback function defined in "access callback". Integer values indicating we are referencing the arguments passed by the URL, according to the order indicated.
- **'theme callback'**: A callback function that includes the system name of the item that will be used to display the page. If not provided, the value is inherited from a top-level element.
- **'theme arguments'**: A vector of arguments that are passed to the callback function theme.
- **'file'**: Name of file to be included before you access the callback function. This allows you to separate the callback functions of the remaining files of the module. The file must be located in the module folder, unless you specify another path in "file path".
- **'file path'**: Path to the folder where the file is located. The default route is that of the module.
- **'weight'**: Integer value (positive or negative) that allows you to indicate the position of the menu item with respect to other elements. The default value is 0. In the event that multiple elements have the same weight, these are presented in alphabetical order.
- **'menu_name'**: By default, the menu item will be created in the navigation menu. In "menu_name" we can indicate any other menu created on the site, by referring to the name of the menu system.
- **'type'**: Determines the type of element, according to the following values:
 - o **MENU_NORMAL_ITEM**: This is the default value. In addition, the indicated page will create a menu item, which may be dealt with from the administration area as will any other menu item.
 - o **MENU_CALLBACK**: With this option you will only register the URL, but will not create any menu item.
 - o **MENU_SUGGESTED_ITEM**: With this option a menu item will be created but disabled. In this way, the site administrator may turn it on if they consider it necessary.
 - o **MENU_LOCAL_ACTION**: Allows you to define actions related to a top-level element. These actions are generally supplied as links. For example, the module User adds a link to add user within the user management page. This route is defined as MENU_LOCAL_ACTION and is presented in the user management page as a link.

- **MENU_LOCAL_TASK:** Allows you to define tabs inside a page.
- **MENU_DEFAULT_LOCAL_TASK:** When we define several tabs, one of them must be set as default tab, which will be loaded using the same route as the parent element.
- **'options':** A vector of options that is passed by the function `I()` when a link is generated for the menu item.

URL with parameters

46.3

In the previous section we define a page callback function in very simple terms, but it is also possible to create more complex functions, even with input parameters.

F46.4

```
/***
 * Implements hook_menu().
 */
function menu_forcontu_menu() {
  // Code from previous examples
  // URL with parameters
  $items['url2/%/%'] = array(
    'title' => 'Example URL 2',
    'page callback' => 'menu_forcontu_url2',
    'page arguments' => array(1,2),
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}

/***
 * Page callback url2/%/%
 * Function called when loading the URL:
 * www.example.com/url2/%/%
 */
function menu_forcontu_url2($firstname='', $surname='') {
  $output = t('Welcome, @firstname @surname!', [
    '@firstname' => $firstname,
    '@surname' => $surname));
  return $output;
}
```

F46.4

Callback Function with parameters

Example of the declaration of the page callback function with parameters.

In this example the callback function receives two parameters that are passed by the URL. When loading the page <http://www.example.com/url2/Fran/Gil>, the result will be a page with the contents: 'Welcome, Fran Gil!'.

By analysing the previous example we discovered the generic form of passing parameters to a callback function, which consists of the following elements:

1. Set the URL and the corresponding parameters. In the example we have defined \$items['url2/%/%'], and we are expecting that after the alias, the page will receive two additional arguments.
2. Indicate in the '**page arguments**' the arguments that you want to go to the callback function. At this point it is important to have the order in which they receive the arguments through the URL. The URL above consists of 3 arguments. To indicate that we want to pass arguments to the callback function, we need to understand the list of arguments available as an array of items ordered, taking into account that in PHP the first element of an array is referenced with index 0:
 - o array[0]: url2
 - o array[1]: %
 - o array[2]: %

To indicate that we will pass the second and third element of the list of arguments passed by URL to the callback function, we will have to reference elements 1 and 2 of the array:

```
'page arguments' => array(1,2),
```

3. Finally, we define the callback function with its parameters (giving them a name). In this example, we declare the parameters **\$name** and **\$surname**, which we will use only as textual information to display a dynamic text:

```
function menu_forcontu_url2($name="", $surname="")
```

Let's look at other variants that will help us to better understand the possible combinations in passing parameters to the callback function: **F46.5**

F46.5

Internal Parameters

In this example internal parameters are not defined by URL parameters, but are passed to the function directly through 'page arguments'.

```
/***
 * Implements hook_menu() .
 */
function menu_forcontu_menu() {
  // Code from previous examples
  // Example of URL with parameters passed internally
  $items['url3'] = array(
    'title' => 'Example URL 3',
    'page callback' => 'menu_forcontu_url2',
    'page arguments' => array('Fran', 'Gil'),
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

In this example we do not pass arguments through the URL, but they are allocated directly through 'page arguments'. Just as you have assigned these parameters directly, they could have also been obtained by other means: by accessing the database, the table variables, or through the callback function.

Even though we have modified the means to get the arguments, we have not needed to modify the callback function, **menu_forcontu_url2()**.

In another example we have complicated the URL that was generated. Now the arguments passed by the URL are interspersed, in third and fifth place. Characters % indicate the position of the arguments, and dynamic values to be passed by URL, while the rest of the elements must be static. We can successfully use this URL: /url4/firstname/**Fran**/surname/**Gil**. Again we have reused the same callback function, **menu_forcontu_url2()**. **F46.6**

F46.6

Interspersed parameters

Examples of interspersing parameters in the URL.

```
/***
 * Implements hook_menu() .
 */
function menu_forcontu_menu() {
  // Code from previous examples
  // Example of URL with interspersed parameters
  $items['url4/firstname/%/surname/%'] = array(
    'title' => 'Example URL 4',
    'page callback' => 'menu_forcontu_url2',
    'page arguments' => array(2, 4),
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

Registering URLs and menu elements

46.4

As we already discussed in this unit, the Drupal menu system has a dual functionality: to register URLs and define menu items.

In the previous examples we have defined this type of URLs 'type' => **MENU_CALLBACK**. In this way we are indicating that we only want to record the URL, linking it with its corresponding callback function, but without creating a menu item.

In addition to registering the URL, you can include it in a menu, using: **MENU_NORMAL_ITEM**.

F46.7

```
/***
 * Implements hook_menu().
 */
function menu_forcontu_menu() {
  // Code from previous examples
  // Example of menu item
  $items['admin/config/development/url5'] = array(
    'title' => 'New menu item',
    'page callback' => 'menu_forcontu_url5',
    'description' => 'link description',
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
    'weight' => 0,
  );
  return $items;
}

/**
 * Page callback url5
 * Function called when loading the URL
 * www.example.com/admin/config/development/url5
 */
function menu_forcontu_url5() {
  $output = t('Here we can implement the code for module configuration. For now, this page only displays this text string.');
  return $output;
}
```

F46.7

Menu Items

Register a URL like an item from a menu.

Figure F46.8 shows the result of applying the previous configuration. The new menu item has been created within the administration menu, in:

Administration ⇒ **Configuration** ⇒ **Development** ⇒ **New menu item**

URL for a new menu element

/admin/config/development/url5

But, how have we indicated this route? Well that is very simple, through the URL registered for the page: **admin/config/development/url5**. In Drupal the URLs are linked to the menu items, so if we want to create a new one we have to register the correct URL from the URL of the parent element.

In this example:

- The main Administration page corresponds to the URL **admin**.
- The **Configuration** element, together with the previous one, corresponds to the URL **admin/config**.
- The **Development** element, together with the previous one,

corresponds to the URL **admin/config/development**.

- The new element, that we want this nested within development, will have the URL **admin/config/development/url5**.

F46.8**MENU_NORMAL_ITEM**

Page and menu item created from the `hook_menu()`. With the type `MENU_NORMAL_ITEM` we are indicating that you register the URL and to create the corresponding menu item.

DEVELOPMENT**Performance**

Enable or disable page caching for anonymous users and set CSS and JS bandwidth optimization options.

Logging and errors

Settings for logging and alerts modules. Various modules can route Drupal's system events to different destinations, such as syslog, database, email, etc.

Maintenance mode

Take the site offline for maintenance or bring it back online.

Coder

Select code review plugins and modules, and upgrade files.

Devel settings

Helper functions, pages, and blocks to assist Drupal developers. The devel blocks can be managed via the [block administration](#) page.

Module builder

Set default header and footer, api download location, defaults for detail and download and force the api to be re-downloaded.

New menu item

Item description

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

F46.9**MENU_NORMAL_ITEM**

Page and menu item created from the `hook_menu()`.

MENU LINK	ENABLED	OPERATIONS
+ Development	<input checked="" type="checkbox"/>	edit
+ Performance	<input checked="" type="checkbox"/>	edit
+ Logging and errors	<input checked="" type="checkbox"/>	edit
+ Maintenance mode	<input checked="" type="checkbox"/>	edit
+ Coder	<input checked="" type="checkbox"/>	edit
+ Devel settings	<input checked="" type="checkbox"/>	edit
+ Module builder	<input checked="" type="checkbox"/>	edit
+ New menu item	<input checked="" type="checkbox"/>	edit

We have also indicated the order in which we want to show the element ('weight' => 0). Remember that the weight makes elements that are less important show first, and the more important elements show later. However, when multiple elements have the same weight, these will be displayed in alphabetical order. In our example, the elements **Devel settings**, **Module builder** and **New menu item** have the same weight 0, so they are listed in **F46.9** alphabetical order.

Menu Selection

By default the menu items declared using MENU_NORMAL_ITEM, will be created in the **Navigation** menu of the site, unless they are a registered item in another menu, as we saw in the previous example.

For the menu item created within another menu, we'll use the directive "menu_name", inside of **hook_menu()**. The value assigned to **menu_name** should correspond with the name of the menu system. For example, we will use **main-menu** to refer to the Main Menu. **F46.10**

```
/***
 * Implements hook_menu().
 */
function menu_forcontu_menu() {
  // Code from previous examples
  // Example of menu item in Main menu
  $items['url6'] = array(
    'title' => 'Menu item in Main menu',
    'page callback' => 'menu_forcontu_url6',
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
    'menu_name' => 'main-menu',
    'weight' => 1,
  );
  return $items;
}
```

F46.10

Menu Selection

We can indicate in which menu the added URL is registered.

The result of running the module with the above code will be the following: **F46.11**

MENU LINK	ENABLED	OPERATIONS
+ Home	<input checked="" type="checkbox"/>	edit delete
+ Menu item in Main menu	<input checked="" type="checkbox"/>	edit

F46.11

Menu Item

Menu item created in a specific menu.

46.5 Definition of tabs

It is possible to create groups of pages, grouped into tabs. In the Drupal administration area, we found many examples of grouping in tabs. The options to **View** and **Edit** any node in the site are also presented as tabs.

In a grouping of pages, the relationship between parents and children is established on the basis of the path, as with other menu items, with the difference that the item is created with the type **MENU_LOCAL_TASK**.

For example, in **Administration ⇒ Structure ⇒ Menus**, it initially shows two tabs: List Menus and Settings.

The main tab, **List Menus**, is found in the URL `admin/structure/menu`, and it is the tab defined, by default, as **MENU_DEFAULT_LOCAL_TASK**. In reality, the main tab is registered to the URL `admin/structure/menu/list`, but it is pointed to the URL of its parent element for being the main tab.

The **Settings** tab is under the URL `admin/structure/menu/settings`, being a child element of **List Menus**.

Below is a fragment of the function `menu_menu()`, that implements the `hook_menu()` within the **Menu** of the core module. **F46.12**

F46.12

Tabs

Definition of Tabs tabs from `hook_menu()`.

```
/**
 * Implements hook_menu().
 */
function menu_menu() {
  $items['admin/structure/menu'] = array(
    'title' => 'Menus',
    'description' => 'Add new menus to your site...',
    'page callback' => 'menu_overview_page',
    'access callback' => 'user_access',
    'access arguments' => array('administer menu'),
    'file' => 'menu.admin.inc',
  );
  $items['admin/structure/menu/list'] = array(
    'title' => 'List menus',
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'weight' => -10,
  );
  $items['admin/structure/menu/settings'] = array(
    'title' => 'Settings',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('menu_configure'),
    'access arguments' => array('administer menu'),
    'type' => MENU_LOCAL_TASK,
    'weight' => 5,
    'file' => 'menu.admin.inc',
  );
  //...
  return $items;
}
```

Note that when using the element type **MENU_DEFAULT_LOCAL_TASK**, you do not have to declare some of the attributes, such as **page callback** or **access callback**, since it is automatically inherited from the parent element. If you do, these values will overwrite the values defined by the parent element.

In the previous example, the URLs have been created as tabs, but you could also have included them in a menu, menu items as normal (nested below the parent item or main element). To achieve this dual functionality enough to combine the element types, as shown below: **F46.13**

```

$items['set/url1'] = array(
  'title' => 'URL1',
  'type' => MENU_NORMAL_ITEM | MENU_DEFAULT_LOCAL_TASK,
  //...
);
$items['set/url2'] = array(
  'title' => 'URL2',
  'type' => MENU_NORMAL_ITEM | MENU_LOCAL_TASK,
  //...
);

//...

return $items;
}

```

F46.13**URL with several types**

We can register a URL to generate several types of elements (menu items and tabs, for example).

As an example, creating tabs will add a page with tabs in the example module **Menu Forcontu**. **F46.14**

```

/**
 * Implements hook_menu()
 */
function menu_forcontu_menu() {
  // Code from previous examples
  // Example of tabs
  $items['tabs'] = array(
    'title' => 'Example of tabs',
    'page callback' => 'menu_forcontu_tab1',
    'access callback' => TRUE,
    'type' => MENU_NORMAL_ITEM,
  );
  $items['tabs/tab1'] = array(
    'title' => 'Tab 1',
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'weight' => 1,
  );
  $items['tabs/tab2'] = array(
    'title' => 'Tab 2',
    'page callback' => 'menu_forcontu_tab2',
    'access callback' => TRUE,
    'type' => MENU_LOCAL_TASK,
    'weight' => 2,
  );
  $items['tabs/tab3'] = array(
    'title' => 'Tab 3',
    'page callback' => 'menu_forcontu_tab3',
    'access callback' => TRUE,
    'type' => MENU_LOCAL_TASK,
    'weight' => 3,
  );
  $items['tabs/tab3/api'] = array(
    'title' => 'Go to Drupal API',
    'page callback' => 'drupal_goto',
    'page arguments' => array('http://api.drupal.org'),
    'access callback' => TRUE,
    'type' => MENU_LOCAL_ACTION,
  );
}

return $items;
}

```

F46.14**Tabs**

Example of definition of tabs in the module Menu Forcontu.

```
/***
 * Page callback - Tab 1
 */
function menu_forcontu_tab1() {
  $output = t('Tab 1 content');
  return $output;
}

/***
 * Page callback - Tab 2
 */
function menu_forcontu_tab2() {
  $output = t('Tab 2 content');
  return $output;
}

/***
 * Page callback - Tab 3
 */
function menu_forcontu_tab3() {
  $output = t('Tab 3 content');
  return $output;
}
```

The result of this code is shown in **Figure F46.15**

The pages **tabs/tab1**, **tabs/tab2** and **tabs/tab3** constitute three tabs on a page.

F46.15

Tabs

Example of defined tab from hook_menu()

The screenshot shows a Drupal page with three tabs at the top: "Tab 1", "Tab 2", and "Tab 3". "Tab 3" is highlighted with a thicker border. Below the tabs, the content area displays the text "Tab 3 content". At the bottom left of the content area, there is a link labeled "+ Go to Drupal API".

The page **tabs/tab3/api** is a **MENU_LOCAL_ACTION** link, which defines an action within an upper element, in this case, within **Tab 3**.

In addition, we have added a link to an external page, defining a callback function of **drupal_goto()**, to which we pass the external URL as an argument (**page arguments**).

Access control to the page

46.6

In the examples seen so far (except for the `menu_menu()`) , the parameter "access callback" has always been true, allowing access to the page to any user, without any restrictions.

As we have studied in previous levels of course, Drupal controls access to certain pages of the site through the use of roles and permissions. The operation is very simple: a page will be associated to a specific permit, so that will be accessible only to those users who have been assigned that permission (that is checked from the roles of the user).

Although we could use an already existing permission, we are going to create a new permission using the hook `hook_permission()`, described in:

http://api.drupal.org/api/function/hook_permission/7

The `hook_permission()` function returns an array with permissions created by the module. The key to the array is the name of the permissions system, and for each permission we will be able to define, among others, the fields; title and description). **F46.16**

```
/***
 * Implements hook_permission().
 */
function menu_forcontu_permission() {
  return array(
    'access to menu forcontu' => array(
      'title' => t('Access to Menu Forcontu restricted pages'),
      'description' => t('Allows access to restricted pages of the menu items defined in Menu Forcontu'),
    ),
  );
}
```

F46.16

hook_permission()

From `hook_permission()` we define new permissions added by the module.

The created permission through the deployment of `hook_permission()` will be available for allocation to the roles of the site from: **F46.17**

Administration ⇒ People [Permissions]

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
Menu Forcontu			
Access to Menu Forcontu restricted pages Allows access to restricted pages of the menu items defined in Menu Forcontu	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

URL Permissions

/admin/people/permissions

F46.17

Module permissions

Listing permissions defined by the module.

The next step is to modify the registry of the URL so that only users with this permission have access to it.

The callback function used to check the access is `user_access()`, which is defined in:

http://api.drupal.org/api/drupal/modules--user--user.module/function/user_access/7

Of course, we could use a custom function. But, only in a test case you are required to validate access. It cannot be obtained from the use of roles and permissions, and requires a specific schedule.

In 'access callback' we will tell you what will be done using the `user_access`

function. As arguments, through 'access arguments', passing an array with the permission or the permissions that will validate access to the page.

The value of access arguments should be an array, regardless of if it passes a single value or multiple permissions. If we use integer values, we are referring to the elements of the loaded URL. **F46.18**

F46.18

user_access() function

With user_access() we check to see if the current user has specific permissions.

```
/**  
 * Implementación del hook_menu()  
 */  
function menu_forcontu_menu() {  
    // Code from previous examples  
    // Example of URL with control access  
    $items['intranet'] = array(  
        'title' => 'Intranet',  
        'page callback' => 'menu_forcontu_intranet',  
        'access callback' => 'user_access',  
        'access arguments' => array('access to menu forcontu'),  
        'type' => MENU_CALLBACK,  
    );  
  
    return $items;  
}  
  
/**  
 * Page callback - Intranet  
 */  
function menu_forcontu_intranet() {  
    $output = t('This page is restricted to users with the appropriate permission.');//  
    return $output;  
}
```

Only when the user has assigned the permission 'access to menu forcontu', you will be able to access the page /intranet. Otherwise the system will return an **Access Denied** error (error 403).

Menu system functions

46.7

Within the Drupal API, the menu system provides its own set of functions that we will be able to use in the development of modules. This is not only for hook features, so to use them we should not modify the name. We only need to know what they are used for, the input parameters required and the value returned.

The complete listing of available functions can be found at <http://api.drupal.org/api/group/menu/7>, from where we can access the detailed description of each one of them.

In this section we revisit some of these functions.

Caching Features and menu reconstruction

The caching functions allow you to clean the cache of menu items and vpsns for page. Will be useful when we direct actions on these elements that do not directly involve a cache refresh. **F46.19**

Function	Description
<code>menu_cache_clear()</code>	It lets you clean the cache of a particular menu.
<code>menu_cache_clear_all()</code>	Allows you to clean the entire cache of all the menus and URLs registered.
<code>menu_rebuild()</code>	Reconstructs the related tables with the routes and the menu items. This function also includes full cleaning of the cache of menus through a call to the function <code>menu_cache_clear_all()</code> .
<code>menu_reset_static_cache()</code>	Resets the static cache menu system.

F46.19

Cache functions

Functions list related to the cache and reconstruction of the menu.

As an example of using one of these functions, let's look at the code for the implementation of `hook_uninstall()` of the **Book** of the core module of Drupal: **F46.20**

```
/**
 * Implements hook_uninstall().
 */
function book_uninstall() {
  variable_del('book_allowed_types');
  variable_del('book_child_type');
  variable_del('book_block_mode');

  // Delete menu items
  db_delete('menu_links')
    ->condition('module', 'book')
    ->execute();
  // Clean menu cache
  menu_cache_clear_all();
}
```

F46.20

Clear cache menu

Example for using the feature

`menu_cache_clear_all()`.

The process of uninstalling the module includes the elimination of menu items in the database, using the `db_delete()` function. If performing this action does not clean the cache, and these items are removed from the database, they will remain visible to the visitors of the site, as it is information that is obtained in the first instance of the cache.

An important note to keep in mind is that the call to the cleaning functions of cache do not reconstruct the cache. This will rely directly on the system when

you need to get the information of routes and menus not located in the cache.

Routing Functions

Within functions of routing functions include those that work in one way or another with the menu items or the routes page. **F46.21**

F46.21

Routing Features

List of features related to the elements or the routing page.

Function	Description
<code>menu_execute_active_handler()</code>	Executes the callback function associated with the specified path as a parameter. As we have already seen, this callback function is defined in the parameter "page callback" and "page arguments") the registered item through <code>hook_menu()</code> .
<code>menu_get_item()</code> <code>menu_set_item()</code>	Returns the information related to a URL, from its path.
<code>menu_get_object()</code>	Returns the information related to a URL, in the form of object. For example, we can recover an object of node or user type.
<code>menu_get_names()</code>	Returns an array with the name of the site menu.

Functions of breadcrumbs, and active menu items

These functions work with the breadcrumbs, and with the menu items, including the active elements (element that corresponds to the page loaded). **F46.22**

F46.22

Functions of bread crumbs

Listing of related functions with bread crumbs and the menu items assets.

Function	Description
<code>menu_get_active_breadcrumb()</code>	Returns an array with the breadcrumbs for the current route. Each array element is a link already formatted in HTML.
<code>menu_get_active_trail()</code> <code>menu_set_active_trail()</code>	Returns an array with the elements that make up the path to the current page. Each element of the array is composed of several fields with information about the element (title, href, localized_options, type).
<code>menu_get_active_title()</code>	Returns the title of the current page (active page). The function gets this information from <code>menu_get_active_trail()</code> .
<code>menu_get_active_menu_names()</code> <code>menu_set_active_menu_names()</code>	Returns the active menu for the current page.
<code>menu_get_ancestors()</code>	Returns an array with the routes predecessors of the given path, including wildcards (%s). For example, the ancestors of node/13/edit are: node/13/edit, node/13/%, node/%/edit, node/%/%, node/13, node/% and node.
<code>menu_links_clone()</code>	Clones a vector of menu items.
<code>menu_link_save()</code>	Saves a menu link.
<code>menu_link_delete()</code>	Removes one or more menu links.
<code>menu_link_load()</code>	Gets a menu link.

Functions of predefined menus: navigation, main menu, and secondary links

The following are some features of the API related to the predefined menus of the site: [F46.23](#)

Function	Description
<code>menu_navigation_links()</code>	Returns an array with elements of the navigation menu.
<code>menu_main_menu()</code>	Returns an array with the main menu items.
<code>menu_secondary_menu()</code>	Returns an array with elements of the sub-menu.

[F46.23](#)

Functions of predefined menus

List of functions related to the predefined menus.

Functions of menu presentation (theming)

The following functions return a formatted output of the menus and their elements, ready to print depending on the theme of the site. [F46.24](#)

Function	Description
<code>theme_menu_link()</code>	Returns the HTML output for a link menu and submenu.
<code>theme_menu_local_action()</code>	Returns the HTML output for a link that defines an action.
<code>theme_menu_local_task()</code>	Returns the HTML output for a link that defines a tab.
<code>theme_menu_local_tasks()</code>	Returns the HTML output for the main tabs and secondary.
<code>theme_menu_tree()</code>	Returns the HTML output with the complete menu tree.

[F46.24](#)

Functions of menu presentation Theming

List of functions related to the menu presentation.

*Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079*

47 Creating forms

HTML forms are a fundamental element on any website for user interaction, hence their importance in module development in Drupal.

Drupal facilitates form generation, validation and processing through a comprehensive set of functions that make up the forms API (Form API).

In this unit we will study how to create forms and apply them from the administration module and to create an administration form inside a Drupal module. We will also see how to modify the forms created by other site modules.

Comparative D7/D6 Forms (Form API)

Programming forms in Drupal 7 follows the same structure as in Drupal 6. We must consider only little changes in Form API functions.

Unit contents

47.1 Defining forms in Drupal	124
47.2 Form elements	127
47.3 Validation of forms	142
47.4 Sending forms	144
47.5 Modification of Forms	146
47.6 Forms in the administration area of the module	149
47.7 Functions of Form API	151



47.1 Defining forms in Drupal

HTML forms are a critical element in any web site for interacting with users. Forms let you collect information from the user, sending it to the system for further processing (save the information in the database, send it by email, etc.).

Drupal facilitates form generation, validation and processing through a comprehensive set of functions that make up the forms API (Form API).

The way you define Drupal forms is very similar to the way you define the schema of the database (via the Schema API). Forms are defined by a structured array, and the system takes care of building the HTML form to display to the user at the time of building the page based on that definition.

To show how the API forms operate, we'll output some of the **examples available in the module**.

A Basic Form

We'll build a new module called **Forms Forcontu** (`forms_forcontu`) in which we will be adding pages and forms created in this unit. We start with a basic example that will help us understand the minimum structure necessary to build a form.

F47.1

F47.2

F47.1

Page with a form

With `hook_menu()` we register a page, the content of which will be a Drupal form. The page will be processed by the `drupal_get_form()` function, which receives as a parameter the name of the function that defines the form (`forms_forcontu_form1`).

```
/***
 * Implements hook_menu().
 */
function forms_forcontu_menu() {
  $items['forms_forcontu/form1'] = array(
    'title' => t('Example Form 1'),
    'page callback' => 'drupal_get_form',
    'page arguments' => array('forms_forcontu_form1'),
    'access callback' => TRUE,
    'description' => t('Example Form 1'),
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

F47.2

Form definition

Function that defines the structure of a form.

```
/***
 * Return function for the page forms_forcontu/form1.
 */
function forms_forcontu_form1($form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A simple form'),
  );

  $form['full_name'] = array(
    '#type' => 'textfield',
    '#title' => t('Full Name'),
  );
  return $form;
}
```

In this example we have registered the URL `forms_forcontu/form1` through the implementation of the corresponding `hook_menu()`. Loading this URL displays the first form we are building.

We look at the callback function associated with this URL, through the variable "page callback" in **drupal_get_form**, to which we will pass the argument "**forms_forcontu_form1**" through "page arguments".

The **drupal_get_form(\$form_id)** function is part of the Drupal API and is responsible for building the HTML form according to its definition. You can review the complete definition of the function at:

http://api.drupal.org/api/drupal/includes--form.inc/function/drupal_get_form/7

It is required that we provide a **\$form_id**, which is simply the unique identifier of the form. In this first example the value of \$form_id is the argument initiated through "page arguments": **forms_forcontu_form1**. This will be, therefore, the form identifier.

The **drupal_get_form()** function will find the form definition to build a function with the same name as the form, which is known as the constructor function of the form:

```
function forms_forcontu_form1($form_state)
```

The form's constructor function receives as a parameter the variable **\$form_state**, which will be discussed later. Usually this variable is passed by reference, **&\$form_state**, so that it may modify its content and share these changes with the system and with other modules.

Defining forms

In the form's constructor function we create a structured array called \$form that will also be the value returned by the function.

In **\$form["description"]** we include the form description. Each element of the form will be defined separately, creating an entry in the array with the form element. For example, in **\$form ["full_name"]** we are defining a field called "Full name", a textfield (which is an input field for a single line of text).

```
$form['full_name'] = array(
  '#type' => 'textfield',
  '#title' => t('Full Name'),
);
```

Form submit button

Adding a submit button to the form is as simple as adding an element **\$form['submit']** in the form's constructor function: **F47.3**

F47.3

Submit button

Defining a button to send or Submit the form.

```
/***
 * Callback function for the page forms_forcontu/form1.
 */
function forms_forcontu_form1($form_state) {
  $form['description'] = array(
    '#type' => 'item',
    '#title' => t('A simple form'),
  );

  $form['full_name'] = array(
    '#type' => 'textfield',
    '#title' => t('Full Name'),
  );

  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => 'Enviar',
  );
}

return $form;
}
```

Before proceeding with completion of the form, let's look at the result of installing and activating the module.

This form will be displayed on loading the URL **forms_forcontu/form1**. **F47.4**

F47.4

Example of a form

Example form defined in `forms_forcontu_form1()` function.

Example Form 1

A simple form

Full Name

Enviar

Form elements

47.2

In the previous section we created a form with a textfield type element, but other types of elements exist (button, checkbox, date, fieldset, etc.). Each form element has a number of features that can be defined through a specific array for that element. In this section we will study some of these items. You can see the full list at:

http://api.drupal.org/api/drupal/developer--topics--forms_api_reference.html/7

actions

Adding a wrapper element to group one or more buttons on a form. We will use it, for example, to group the buttons Save, Delete, Cancel, etc. **F47.5**

```
$form['actions'] = array('#type' => 'actions');
$form['actions']['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Save'),
);
$form['actions']['delete'] = array(
  '#type' => 'button',
  '#value' => t('Delete'),
);
$form['actions']['cancel'] = array(
  '#markup' => l(t('Cancel'), 'foo/bar'),
);
```

F47.5

Action elements

Example using action elements.

Properties: #access, #after_build, **#attributes**, #children, #id, #parents, #post_render, #pre_render, #prefix, #process, #states, #suffix, #theme, #theme_wrappers, #tree, **#type**, #weight

button

Adding a button to the form. When you click on the button, the form is submitted to the system, which will validate it and perform the appropriate actions, as defined. For example, it can include a button for creating a preview button, as in the Node element module for each node: **F47.6**

```
$form['preview'] = array(
  '#type' => 'button',
  '#value' => t('Preview'),
  '#weight' => 19,
);
```

F47.6

Button element

Example using a button element that adds a button to a form.

Properties: #access, #after_build, #ajax, #attributes, #button_type (default: submit), #disabled, #element_validate, #executes_submit_callback (default: FALSE), #limit_validation_errors, #name (default: op), #parents, #post_render, #prefix, #pre_render, #process, #submit, #states, #suffix, #theme, #theme_wrappers, #tree, **#type**, #validate, **#value**, #weight

checkbox

Create a single item checkbox (selection box). **F47.7**

F47.7

Checkbox element

Example using the checkbox element or a single selection box.

```
$form['legal_notice'] = array(
  '#type' => 'checkbox',
  '#title' => t('Accept terms and conditions.'),
);
```

Properties: #access, #after_build, #ajax, #attributes, **#default_value**, #description, #disabled, #element_validate, #field_prefix, #field_suffix, #parents, #post_render, #prefix, #pre_render, #process, #required, **#return_value** (default: 1), #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display (default: after), #tree, **#type**, #weight

checkboxes

Create a set of checkboxes. An associative array will be defined with key/value pairs where the key will be used internally by the form, and the value will display on the form (and may also be translated). **F47.8**

F47.8

Checkbox element

Example using the checkboxes element, which adds a set of checkboxes.

```
$form['colours'] = array(
  '#type' => 'checkboxes',
  '#title' => t('Colours'),
  '#default_value' => array('red', 'green'),
  '#options' => array(
    'red' => t('Red'),
    'green' => t('Green'),
    'blue' => t('Blue'),
    'yellow' => t('Yellow'),
  ),
  '#description' => t('Select your preferred colours.'),
);
```

Properties: #access, #after_build, #ajax, #attributes, **#default_value**, #description, #disabled, #element_validate, **#options**, #parents, #post_render, #prefix, #pre_render, #process, #required, #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, **#tree** (default: TRUE), **#type**, #weight

date

Creates a date picker. If no default value is found (#default_value), the default value will be the current date. **F47.9**

F47.9

Date element

Example using the date element that adds a date picker.

```
$form['start_date'] = array(
  '#type' => 'date',
  '#title' => t('Start Date'),
  '#default_value' => array('day' => 1, 'month' => 1, 'year' => 2012),
);
```

Properties: #access, #after_build, #attributes, **#default_value**, #description, #disabled, #element_validate, #parents, #post_render, #prefix, #pre_render, #process, #required, #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

fieldset

A fieldset allows the grouping of other form elements. For example, we can create a fieldset to group the first and last names from other fields. **F47.10**

```
$form['personal_data'] = array(
  '#type' => 'fieldset',
  '#title' => t('Personal Data'),
  '#collapsible' => TRUE,
  '#collapsed' => FALSE,
);

$form['personal_data']['first_name'] = array(
  '#type' => 'textfield',
  '#title' => t('First name'),
  '#required' => TRUE,
  '#default_value' => "First Name",
  '#description' => "Enter your first name",
  '#size' => 20,
  '#maxlength' => 20,
);

$form['personal_data']['last_name'] = array(
  '#type' => 'textfield',
  '#title' => t('Last name'),
  '#required' => TRUE,
  '#default_value' => "Last Name",
  '#description' => "Enter your last name",
  '#size' => 40,
  '#maxlength' => 40,
);
```

F47.10

Fieldset element

Example using the fieldset element, which allows you to group several form elements.

The elements grouped inside a fieldset must be declared within the same associative array:

- **\$form['personal_data']**, defines the fieldset 'personal_data'.
- **\$form['personal_data']['first_name']**, defines an element 'first_name' inside the fieldset 'personal_data'.
- **\$form['personal_data']['last_name']**, defines an element 'last_name' inside the fieldset 'personal_data'.

Properties: #access, #after_build, #attributes, **#collapsed** (default: FALSE), **#collapsible** (default: FALSE), #description, #element_validate, #parents, #post_render, #prefix, #pre_render, #process, #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

file

Attaches a file to the form. To attach files to forms in Drupal 6 it was necessary to modify a form's general attributes with the line: \$form ['#attributes'] = array('enctype' => "multipart/form-data");

However, in Drupal 7 the system will add the form code automatically, so it is not necessary to indicate it while constructing the form. **F47.11**

```
$form['new_upload'] = array(
  '#type' => 'file',
  '#title' => t('Attach new file'),
  '#size' => 40,
);
```

F47.11

File element

Example using the file element, which allows the attachment of files to the form.

Properties: #access, #after_build, #array_parents, #attached, #attributes, #description, #disabled, #element_validate, #parents, #post_render, #prefix, #pre_render, #process, #required, **#size** (default: 60), #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

hidden

Adds hidden information to the form. F47.12

F47.12

Hidden element

Example using the hidden element for adding hidden information.

```
$form['nid'] = array(
  '#type' => 'hidden',
  '#value' => $nid,
);
```

Properties: #access, #after_build, #ajax, #default_value, #element_validate, #parents, #post_render, #prefix, #pre_render, #process, #states, #suffix, #theme, #theme_wrappers, #tree, **#type**, **#value**, #weight

image_button

Creates a send form button with an image. F47.13

F47.13

Image_button element

Example using the image_button element to create a form button with an image.

```
$form['send'] = array(
  '#type' => 'image_button',
  '#value' => 'Send',
  '#src' => 'sites/default/files/images/send.png',
  '#name' => 'send',
  '#submit' => array('test_form_submit'),
);
```

Properties: #access, #after_build, #ajax, #attributes, #button_type (default: 'submit'), #disabled, #element_validate, #executes_submit_callback (default: TRUE), #limit_validation_errors, #parents, #post_render, #prefix, #pre_render, #process, #return_value (default: TRUE), #src, #submit, #states, #suffix, #theme, #theme_wrappers, #tree, **#type**, #validate, **#value**, #weight

item

Add a display-only element. F47.14

F47.14

Item element

Example using the item element, which adds a read-only element.

```
$form['from'] = array(
  '#type' => 'item',
  '#title' => t('From'),
  '#markup' => $user->name . ' <'. $user->mail .'>',
);
```

Properties: #access, #after_build, #description, #element_validate, #markup, #parents, #post_render, #prefix, #pre_render, #process, #required, #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

machine_name

Add a text field to enter a system name. The field checks that the name is valid, replacing invalid characters with a designated wildcard. F47.15

```
$form['machine_name'] = array(
  '#type' => 'machine_name',
  '#default_value' => $vocabulary->machine_name,
  '#maxlength' => 21,
  '#machine_name' => array(
    'exists' => 'menu_edit_menu_name_exists',
  ),
);
```

F47.15**Machine_name element**

Example using the machine_name element, which allows you to enter a system name.

#Machine_name property can include these additional attributes:

- **exists**. Name of the function that checks if the value is unique.
- **source**. Element containing the field name associated with the system name.
- **label**. Text to display as the field label. The default value is "Machine name".
- **replace_pattern**. Regular expression indicating illegal characters. By default only lowercase letters, numbers and underscores are allowed.
- **replace**. Replacement character for illegal characters. By default '_'.

Properties: #access, #after_build, #ajax, #attributes, #autocomplete_path (default: FALSE), #default_value, #description (default: 'A unique machine-readable name. Can only contain lowercase letters, numbers, and underscores.'), #disabled, #element_validate, #field_prefix, #field_suffix, #maxlength (default: 64), #parents, #post_render, #prefix, #pre_render, #process, #required (default: TRUE), #size (default: 60), #states, #suffix, #theme (default: 'textfield'), #theme_wrappers (default: 'form_element'), #title (default: 'Machine-readable name'), #title_display #tree, #type, #weight

markup

Allows the adding of HTML text inside the form. It is the default entity type, so it is not necessary to indicate #type = 'markup'.

```
$form['contact_information'] = array(
  '#value' => t('Text inside the form.'),
);
```

F47.16**Markup element**

Example using the markup element, which allows you to add HTML.

Properties: #access, #after_build, #element_validate, #markup, #parents, #post_render, #prefix, #pre_render, #process, #states, #suffix, #theme, #theme_wrappers, #tree, #type, #weight

password

Creates a text field for entering passwords. Like any username and password entry, the text entered by the user in the password field is not displayed. **F47.17**

```
$form['pass'] = array(
  '#type' => 'password',
  '#title' => t('Password'),
  '#maxlength' => 64,
  '#size' => 15,
);
```

F47.17**Password element**

Example using the password element, which allows you to enter a password.

Properties: #access, #after_build, #ajax, #attributes, #description, #disabled, #element_validate, #field_prefix, #field_suffix, #maxlength (default: 128), #parents, #post_render, #prefix, #pre_render, #process, #required, #size (default: 60), #states, #suffix, #theme, #theme_wrappers, #title, #title_display, #tree, #type, #weight

password_confirm

Creates a pair of password fields so that the form validates only when the contents of both match. **F47.18**

F47.18

password_confirm element

Example using the password_confirm elements, which adds a password field and an additional confirmation field.

```
$form['pass'] = array(
  '#type' => 'password_confirm',
  '#title' => t('Password'),
  '#size' => 15,
);
```

When we want the form to include the password confirmation we will include only a single field type called PASSWORD_CONFIRM that already includes the two password fields. A common mistake is to use a password field type and an additional field for PASSWORD_CONFIRM.

Properties: #access, #after_build, #array_parents, #attached, #description, #disabled, #element_validate, #field_prefix, #field_suffix, #parents, #post_render, #prefix, #pre_render, #process, #required, #size (default: 60), #states, #suffix, #theme, #theme_wrappers, #title, #title_display, #tree, #type, #value_callback, #weight

radio

Creates a single radio button. **F47.19**

F47.19

Radio element

Example using the radio element, which adds a single selection button.

```
$form['accept'] = array(
  '#type' => 'radio',
  '#title' => t('Accept the agreement'),
  '#default_value' => 'accept',
);
```

Properties: #access, #after_build, #ajax, #attributes, #default_value, #description, #disabled, #element_validate, #field_prefix, #field_suffix, #parents, #post_render, #prefix, #pre_render, #process, #required, #return_value, #states, #suffix, #theme, #theme_wrappers, #title, #title_display (default: after), #tree, #type, #weight

radios

Creates a set of radio buttons. **F47.20**

F47.20

Radios element

Example using the radios element, which adds a set of related radio buttons.

```
$form['day_week'] = array(
  '#type' => 'radios',
  '#title' => t('Day of the week'),
  '#default_value' => 'Monday',
  '#options' => array(
    t('Monday'),
    t('Tuesday'),
    t('Wednesday'),
    t('Thursday'),
    t('Friday'),
    t('Saturday'),
    t('Sunday'),
  ),
);
```

Properties: #access, #after_build, #ajax, #attributes, #default_value, #description, #disabled, #element_validate, #options, #parents, #post_render, #prefix, #pre_render, #process, #required, #states, #suffix, #theme, #theme_wrappers, #title, #title_display, #tree, #type, #weight

select

Create a drop-down list.

F47.21

```
$form['months'] = array(
  '#type' => 'select',
  '#title' => t('Month'),
  '#default_value' => 'january',
  '#options' => array(
    'january' => t('January'),
    'february' => t('February'),
    'march' => t('March'),
    'april' => t('April'),
    'may' => t('May'),
    'june' => t('June'),
    'july' => t('July'),
    'august' => t('August'),
    'september' => t('September'),
    'october' => t('October'),
    'november' => t('November'),
    'december' => t('December'),
  ),
  '#description' => t('Select month'),
);
```

F47.21

Select element

Example using the select element, which adds a selection list.

Properties: #access, #after_build, #ajax, #attributes, **#default_value**, #description, #disabled, #element_validate, #empty_option, #empty_value, #field_prefix, #field_suffix, **#multiple**, **#options**, #parents, #post_render, #prefix, #pre_render, #process, #required, #size, #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

submit

Creates a submit send form button.

F47.22

```
$form['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Save and continue'),
);
```

F47.22

Submit element

Example using the submit element, which adds a send button.

Properties: #access, #after_build, #ajax, #attributes, #button_type (default: 'submit'), #disabled, #element_validate, #executes_submit_callback (default: TRUE), #limit_validation_errors, #name (default: 'op'), #parents, #post_render, #prefix, #pre_render, #process, #submit, #states, #suffix, #theme, #theme_wrappers, #tree, **#type**, #validate, **#value**, #weight

tableselect

Creates a selection table with radio buttons or checkboxes. Columns are defined with the #header property, and rows, with the #options property.

F47.23

The default display is checkboxes. Setting #multiple to TRUE displays the options as radio buttons. Setting the #js_select property to TRUE (the default) allows all the formatting of all table elements to be selected as checkboxes.

For more information on this item, we recommend visiting the following URL:
<http://drupal.org/node/945102#drupal7>

F47.23**Tableselect element**

Example using the tableselect element, which creates a table of selected elements.

```
// Builds the table header (#header).
$header = array(
  'title' => array('data' => t('Title'), 'field' => 'n.title'),
  'type' => array('data' => t('Type'), 'field' => 'n.type'),
  'author' => t('Author'),
);
// Get data from the database (table node).
$nids = $query
  ->fields('n',array('nid'))
  ->limit(50)
  ->orderByHeader($header)
  ->execute()
  ->fetchCol();
$nodes = node_load_multiple($nids);

// Buils each row (#options).
$options = array();
foreach ($nodes as $node) {
  $options[$node->nid] = array(
    'title' => array(
      'data' => array(
        '#type' => 'link',
        '#title' => $node->title,
        '#href' => 'node/' . $node->nid,
        '#options' => $l_options,
        '#suffix' => ' ' . theme('mark', array('type' =>
node_mark($node->nid, $node->changed))),
      ),
    ),
    'type' => check_plain(node_type_get_name($node)),
    'author' => theme('username', array('account' => $node)),
  );
}

// Build the form element tableselect.
$form['nodes'] = array(
  '#type' => 'tableselect',
  '#header' => $header,
  '#options' => $options,
  '#empty' => t('No content available.'),
);

```

Properties: #access, #after_build, #ajax, #attributes, #**default_value**, #element_validate, #empty, #header, #js_select, #**multiple**, #**options**, #parents, #post_render, #prefix, #pre_render, #process, #states, #suffix, #theme, #theme_wrappers, #tree, #**type**, #weight

text_format

Text area (textarea) with text format activated. **F47.24**

F47.24**Text_format element**

Example using the text_format element, for adding a text area with formatting.

```
$form['description'] = array(
  '#type' => 'text_format',
  '#title' => t('Description'),
  '#default_value' => $term->description,
  '#format' => $term->format,
  '#weight' => 0,
);
```

This element has these specific properties:

- **#format**. Lets you specify the text format to be applied to the text area.
- **#base_type**. "Textarea" is used by default as the base element. This item can also be applied to other field types, such as single-line text fields (textfield).

Properties: #access, #after_build, #ajax, #attributes, **#cols** (default: 60), **#default_value**, #description, #disabled, #element_validate, #parents, #post_render, #prefix, #pre_render, #process, #required, #resizable (default: TRUE), **#rows** (default: 5), #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

textarea

Add a multiple line text area (textarea). **F47.25**

```
$form['body'] = array(
  '#type' => 'textarea',
  '#title' => t('Body'),
  '#default_value' => $node->body,
  '#required' => TRUE,
  '#cols' => 60,
  '#rows' => 5,
);
```

Properties: #access, #after_build, #ajax, #attributes, **#cols** (default: 60), **#default_value**, #description, #disabled, #element_validate, #field_prefix, #field_suffix, #parents, #post_render, #prefix, #pre_render, #process, #required, #resizable (default: TRUE), #rows (default: 5), #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

textfield

Adds a single line text field. **F47.26**

```
$form['title'] = array(
  '#type' => 'textfield',
  '#title' => t('Subject'),
  '#default_value' => $node->title,
  '#size' => 60,
  '#maxlength' => 128,
  '#required' => TRUE,
);
```

Properties: #access, #after_build, #ajax, #attributes, #autocomplete_path (default: FALSE), **#default_value**, #description, #disabled, #element_validate, #field_prefix, #field_suffix, **#maxlength** (default: 128), #parents, #post_render, #prefix, #pre_render, #process, #required, **#size** (default: 60), #states, #suffix, #text_format, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

vertical_tabs

Creates a vertical_tabs group so that we can add multiple sets of fields (fieldset), which are displayed as vertical tabs. **F47.27**

We first define the element type 'vertical_tabs'. Then each 'fieldset' element that we want to add to the group has to bear a '#group' property with the name of the 'vertical_tabs' element.

In the following example we have created a 'vertical_tabs' group called '**additional_settings**'. The 'fieldset' group of fields takes the property '#group' =>'additional_settings' to indicate that they belong to the vertical tabs group.

F47.25

Textarea element

Example using the textarea element, which adds a text area.

NOTE

Textarea and texfield

Drupal themes apply CSS styles through these form elements. This means that in order to change the width it is not enough to modify the properties #cols or #size. Rather, we have to make changes to the CSS stylesheet.

F47.26

Textfield element

Example using the textfield element, which adds a single line text field.

F47.27**Vertical_tabs element**

Example using the vertical_tabs element, which creates a group of vertical tabs. An example of this type of grouping of tabs is shown in the editing or creating of a node.

```
$form['additional_settings'] = array(
  '#type' => 'vertical_tabs',
  '#weight' => 99,
);

$form['revision_information'] = array(
  '#type' => 'fieldset',
  '#title' => t('Revision information'),
  '#collapsible' => TRUE,
  '#group' => 'additional_settings',
  '#weight' => 100,
);

// Form fields included in the fieldset.

$form['author'] = array(
  '#type' => 'fieldset',
  '#title' => t('Authoring information'),
  '#collapsible' => TRUE,
  '#group' => 'additional_settings',
  '#weight' => 90,
);

// Form fields included in the fieldset.
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

We can also indicate that the fieldset should display as nested vertical tabs within the vertical_tabs element. In this case it's not necessary to use the #group property. **F47.28**

F47.28**Vertical_tabs element**

This displays another way of defining the content of vertical tabs.

```
$form['additional_settings'] = array(
  '#type' => 'vertical_tabs',
  '#weight' => 99,
);

$form['additional_settings']['revision_information'] = array(
  '#type' => 'fieldset',
  //...
);
$form['additional_settings']['author'] = array(
  '#type' => 'fieldset',
  //...
);
```

Properties: #access, #after_build, #default_tab, #element_validate, #parents, #post_render, #prefix, #pre_render, #process, #states, #suffix, #theme, #theme_wrappers, #tree, **#type**, #weight

weight

Adds a weight selector for menu items. By writing '#delta' => 10, we are indicating that a selector between -10 and 10 should be created. **F47.29**

F47.29**Weight element**

Example using the weight element, which adds a weight selector.

```
$form['weight'] = array(
  '#type' => 'weight',
  '#title' => t('Weight'),
  '#default_value' => $edit['weight'],
  '#delta' => 10,
  '#description' => t('Select item order.'),
);
```

Properties: #access, #after_build, #attributes, **#default_value**, **#delta** (default: 10), #description, #disabled, #element_validate, #parents, #post_render, #prefix, #pre_render, #process, #required, #states, #suffix, #theme, #theme_wrappers, **#title**, #title_display, #tree, **#type**, #weight

Form fields example

A sample form showing some of the studied elements. We'll create a new form in the same module created earlier, **Forms Forcontu**. F47.30

The form consists of three groups of options shown as vertical tabs: Account information, personal information and professional information.

As we will see in later sections, once the form is sent, the **forms_forcontu_form2_submit()** function is processed. In our example, the function displays a message and redirects the user to the site's home page.

```
/*
 * Implements hook_menu().
 */
function forms_forcontu_menu() {
//...
$items['forms_forcontu/form2'] = array(
  'title' => 'Example Form 2',
  'description' => 'Form elements.',
  'page callback' => 'drupal_get_form',
  'page arguments' => array('forms_forcontu_form2'),
  'access callback' => TRUE,
  'type' => MENU_CALLBACK,
);

return $items;
}

/*
 * Callback function for the page forms_forcontu/form2.
 */
function forms_forcontu_form2(&$form_state) {
  // Container for vertical tabs
  $form['tabs'] = array (
    '#type' => 'vertical_tabs',
  );
  // Container for vertical tabs
  $form['tabs']['account_info'] = array(
    '#type' => 'fieldset',
    '#title' => t('Account information'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
  );
  $form['tabs']['account_info']['email'] = array(
    '#type' => 'textfield',
    '#title' => t('E-mail'),
    '#default_value' => 'User email',
    '#size' => 30,
    '#maxlength' => 30,
    '#required' => TRUE,
  );
  $form['tabs']['account_info']['pass'] = array(
    '#type' => 'password_confirm',
    '#title' => t('Password'),
    '#size' => 15,
  );
  $form['tabs']['account_info']['signature'] = array(
    '#type' => 'textfield',
    '#title' => 'Your signature',
    '#default_value' => 'Signature',
    '#size' => 30,
    '#maxlength' => 30,
  );
}
```

F47.30

Example form

Implementation of a complete application with some of the studied elements.

```
// Block personal information
$form['tabs']['personal_info'] = array(
  '#type' => 'fieldset',
  '#title' => 'Personal information',
  '#collapsible' => TRUE,
  '#collapsed' => FALSE,
);
$form['tabs']['personal_info']['first_name'] = array(
  '#type' => 'textfield',
  '#title' => 'First name',
  '#required' => TRUE,
  '#default_value' => 'First name',
  '#description' => 'Input your first name',
  '#size' => 20,
  '#maxlength' => 20,
);
$form['tabs']['personal_info']['last_name'] = array(
  '#type' => 'textfield',
  '#title' => 'Last name',
  '#required' => TRUE,
  '#default_value' => 'Last name',
  '#description' => 'Input your last name',
  '#size' => 40,
  '#maxlength' => 40,
);
$form['tabs']['personal_info']['example_image_fid'] = array(
  '#title' => 'Your image',
  '#type' => 'managed_file',
  '#description' => t('The uploaded image will be shown in your profile.'),
  '#default_value' => variable_get('image_demo_form', ''),
  '#upload_location' => '/sites/default/files',
);
$form['tabs']['personal_info']['months'] = array(
  '#type' => 'select',
  '#title' => t('Month'),
  '#default_value' => 'january',
  '#options' => array(
    'january' => t('January'),
    'february' => t('February'),
    'march' => t('March'),
    'april' => t('April'),
    'may' => t('May'),
    'june' => t('June'),
    'july' => t('July'),
    'august' => t('August'),
    'september' => t('September'),
    'october' => t('October'),
    'november' => t('November'),
    'december' => t('December'),
  ),
  '#description' => t('Select your month of birth'),
);
// Block professional information
$form['tabs']['professional_info'] = array(
  '#type' => 'fieldset',
  '#title' => 'Professional information',
  '#collapsible' => TRUE,
  '#collapsed' => FALSE,
);
$form['tabs']['professional_info']['new_upload'] = array(
  '#type' => 'file',
  '#title' => 'Attach your resume',
  '#size' => 40,
  '#attributes' => array('enctype' => 'multipart/form-data'),
);

```

```
// Send the form
$form['submit'] = array(
  '#type' => 'submit',
  '#value' => t('Send'),
);

return $form;
}

/**
 * Function to send the form
 */
function forms_forcontu_form2_submit($form, &$form_state) {
  drupal_set_message ('<strong>Your user account has been created correctly.</strong>');
  drupal_goto('');
}
```

Then we display the general form.

F47.31

Example Form 2

The screenshot shows a web form with three vertical tabs on the left: "Account information", "Personal information", and "Profesional information". The "Personal information" tab is currently selected. Inside the form, there are fields for "First name" (with placeholder "Your first name"), "Last name" (with placeholder "Your last name"), and "Your picture". The "Your picture" section includes a "Choose File" button, a "No file chosen" message, and an "Upload" button. Below these is a note: "The uploaded image will be shown in your profile page." There is also a "Month" dropdown menu set to "January", with a placeholder "Select your birth month". At the bottom of the form is a "Send" button.

F47.31

Example form

This shows the form that results from the above definition. The fields are grouped into 3 vertical tabs.

Properties

As indicated above, each type of element can have different properties. Here we describe the properties available for form elements.

All these properties are available at:

F47.32

http://api.drupal.org/api/drupal/developer--topics--forms_api_reference.html/7

Property	Description
#access	Indicates whether the item is accessible or not. If FALSE, the element is not taken into account.
#action	The path to which the form will be sent for processing.
#after_build	Array of functions that will be called after constructing the element or form.
#ajax	Allows communication via AJAX, so that we can update information without reloading the entire page. Will learn

F47.32

Properties

List of properties applied to the form element. In each element the properties that can be applied are specified.

F47.32**Properties (cont.)**

List of properties applicable to the form elements. In each element the properties that can be applied are specified.

	more about using AJAX in unit 57 .
#array_parents	Returns an array with the names of the parent elements of an element (including the element itself).
#attached	Allows the loading of CSS, JavaScript, libraries or custom types, when the form is built.
#attributes	Allows adding additional attributes to the form (For example, class and encytype. \$form['#attributes'] = array('class' => 'styles-form');
#button_type	Adds a CSS class to a submit button.
#collapsed	Indicates whether a fieldset will display collapsed by default.
#collapsible	Indicates if a fieldset can be collapsed or not.
#cols	Number of columns of a textarea element.
#default_tab	Tabs that open by default.
#default_value	Default value for an element.
#delta	Number used to indicate the possible weight values of a weight element. With #delta => 10, the selector will be amounts between -10 and 10.
#description	Element description. Always use the t() function to enable translation.
#disabled	The element will be displayed to the user, but it will be disabled, so its value cannot be changed.
#element_validate	Lists specific validation functions for the element.
#empty	In tableselect type elements, the text that displays if the #option property is empty.
#empty_option	In select type elements, the label that displays when no item is selected. The defaults are '-Select-' for a required field and '-None-' for an optional field.
#empty_value	In select type elements, the internal value of the item to be sent with the form if the user does not select any value. It is the value of the option #empty_option .
#executes_submit_callback	Indicates if the button should submit the form or not (TRUE/FALSE).
#field_prefix	Attaches text as a prefix before an entry in a textfield type field.
#field_suffix	Attaches text as a suffix after an entry in a textfield type field.
#group	Specifies a "vertical_tabs" type element to group different fieldset elements in vertical tab format.
#header	Column header for tableselect type elements.
#id	The system assigns a unique identifier to the form. Through this property you can override the value of the system value of the form id.
#js_select	Adds a 'select all elements' functionality to tableselect elements.
#markup	Dispaly HTML text in the form.
#maxlength	Maximum number of input characters (in textfield and password fields).
#method	Form submission method: GET or POST. The default method is POST.
#multiple	In a select or tableSelect element type, indicates whether the user can select more than one item.
#name	In the submit or send buttons, the value of the element is, by default, 'op'. With #name we can change this value. For other items this value cannot be changed.
#options	In the list of components (checkboxes, radios, select and tableSelect), indicates the possible options. It can be an

	array of values or an array with key/value pairs.
#parents	Identifies the parent elements of a form.
#post_render	Function that executes after building an item. This allows subsequent modifications before submitting the form.
#prefix	Allows text or HTML code to be included before an element. Combined with #suffix, allows a form field to be enclosed between two HTML tags in order to apply the corresponding CSS styles.
#pre_render	Functions to run before building an item. This makes it possible to modify the definition of the element before construction.
#process	An array with functions that will be executed when an element is processed.
#required	Indicates if an element is required or mandatory. The form will validate the required fields in the first place that is left empty.
#resizable	In textarea elements, allows the user to resize the text area
#return_value	Element to be returned when an element of the type checkbox, image_button or radio is selected.
#rows	Number of rows in a text field type textarea.
#size	Width of a text element, in characters.
#src	In a image_button element type, indicates the URL of the image.
#submit	Allows you to specify a list of callback functions to execute when the form is submitted or certain buttons have been used
#suffix	Allows text or HTML code after an element. Combined with #prefix allows a form field to be enclosed between two HTML tags in order to apply the corresponding CSS styles.
#theme	Theme function to call for each element. You should indicate the name without including the prefix theme_.
#title	Element title.
#title_display	Indicates how to display with the tab with the title of the element (before, after, invisible and attribute).
#tree	Defines a collection of elements.
#type	type of form element.
#validate	List of additional validation functions.
#value	Values that cannot be edited by users.
#value_callback	Specifies a function to get the values of the attribute #value.
#weight	Used to indicate the weight or order of the element relative to other elements of the form.

F47.32**Properties (cont.)**

List of properties applicable to the form elements. In each element the properties that can be applied are specified.

47.3

Validation of forms

Drupal API includes a general validation of forms that checks, for example, that the user has entered a value in the required fields.

In addition to this general validation, we can add additional validation by creating a form validation function. We can simply create a function that follows the following naming pattern: `formID_validate()`, `formID` being the ID or name of the form.

The validation function has two arguments: `$form` and `$form_state`. `$form` is the array we already studied while defining forms, and `$form_state['values']` contain the values entered by the user for each form element.

NOTE**valid_email_address() function**

With this feature you can validate the syntax of the email address that is passed as a parameter. Returns TRUE if the format is valid.

Therefore, to check and validate if the values entered by the user are valid, we access `$form_state['values']`. To access a form field of a generic form: `$form_state['values']['field_name']`.

Remember that we can use the debugging features provided by the **Devel** module for all values of these variables.

Let's create a validation function for the form from the previous example, whose `formID` is `forms_forcontu_form2`. As mentioned, the name of the validation function will be `forms_forcontu_form2_validate()`. **F47.33**

This function will add a validation of the 'email' field defined in:

```
$form['tabs']['account_info']['email']
```

To check if the email is valid, we use the function from the Drupal API `valid_email_address()`. We can find the definition of the function at:

http://api.drupal.org/api/drupal/includes--common.inc/function/valid_email_address/

F47.33

Form validation

Example of a form validation function. The errors found are reported with the `form_set_error()` function.

```
/***
 * Validation function of the form forms_forcontu_form2
 */
function forms_forcontu_form2_validate($form, &$form_state) {
  if (!valid_email_address($form_state['values']['email'])) {
    form_set_error('email', t('You must enter a valid e-mail address.'));
  }
}
```

The way to report a validation error in any of the form fields is through the API function `form_set_error()`, which we study below.

Figure F47.34 shows the error returned by the form for indicating that an email is not valid.

 You must enter a valid e-mail address.

Home

Example Form 2

Account information

Personal information

Profesional information

E-mail *

Password

Confirm password

Your signature

Signature

F47.34**Form validation**

The form has detected an error in a field during validation and will not be sent.

Erroneous fields are displayed in red. Error reporting can be found at the top of the page.

Function form_set_error()

The **form_set_error()** function is used to report a validation error on the form. You can see its definition at:

http://api.drupal.org/api/drupal/includes--form.inc/function/form_set_error/7

The parameters that should be included in the call to the function are:

form_set_error(\$name = NULL, \$message = "", \$limit_validation_errors = NULL)

- **\$name.** Name of the form element.
- **\$message.** Error message that displays to user.
- **\$limit_validation_errors:** Only for internal use. The property #limit_validation_errors of the pressed button, if it exists. Lets you specify what elements will be validated (see the API).

From the form validation function we call the **form_set_error()** function for each error encountered. If the form contains one or more errors, the system will not send the form, but return it to the user, highlighting the fields in error and showing the corresponding error messages.

47.4

Sending forms

The normal form submission method is via the submit function. The Submit function follows the same naming pattern as the validation function, but with the suffix **_submit** in place of **_validate**. Therefore, follow the following pattern: **formID_submit()**, formID being the ID or name of the form.

The submit function has the same arguments and the same order as the validation function: \$form and \$form_state.

To send the form through the submit function, it should be sent via a button type => 'submit'. The submit function will be executed only if the form has favorably passed the validation process.

Process the Form Data

What can we do with the data collected through the form? This will depend on the functionality of the module, but basically we're talking about three common scenarios:

- Store the data in the **database**, using the functions already studied as db_query().
- Use the data to make a **decision about the application flow** without storing the data. For example, to redirect the user to different pages.
- Send the data by **email**. In **Unit 51** we will study the Drupal API functions that enable you to send emails.

We will complete our example, storing in the database some of the sent data.

We begin by defining a database schema for the module **Forcontu Forms**. For this, we implement the **hook_schema()** function in the module's installation file, **forms_forcontu.install**. [F47.35](#)

We will create the table **forms_forcontu_account**, with the following fields

- **id**. Numeric identifier, which is an auto-incremented value.
- **email**. Valid email address.
- **password**. Password. It will be stored as plain text.
- **signature**. User signature text

```
/***
 * Implements hook_schema().
 */
function forms_forcontu_schema() {
  $schema['forms_forcontu_account'] = array(
    'description' => t('User account data'),
    'fields' => array(
      'id' => array(
        'description' => 'ID unique and auto-increment',
        'type' => 'serial',
        'not null' => TRUE,
      ),
      'email' => array(
        'description' => 'Email',
        'type' => 'varchar',
        'length' => 255,
        'not null' => TRUE,
      ),
      'password' => array(
        'description' => 'Password',
        'type' => 'varchar',
        'length' => 255,
      ),
      'signature' => array(
        'description' => 'Signature',
        'type' => 'varchar',
        'length' => 255,
      ),
      'primary key' => array('id'),
    );
  }

  return $schema;
}
```

The form's send function will be **forms_forcontu_form2_submit()**. The function stores the data in the table created by the module and displays a user message. **F47.36**

```
/***
 * Form submission function forms_forcontu_form2
 */
function forms_forcontu_form2_submit($form, &$form_state) {
  // Insert data function

  $result = db_insert('forms_forcontu_account')
    ->fields(array(
      'email' => $form_state['values']['email'],
      'password' => $form_state['values']['pass'],
      'signature' => $form_state['values']['signature'],
    ))
    ->execute();

  drupal_set_message ("<strong> Your user account has been
created.</strong>");
  drupal_goto("");
}
```

F47.35**Table definition**

We define a table to store the data sent by the form.

F47.36**Form submission**

The send or submit function stores the received data.

To verify operation of the form, we must access the database (phpMyAdmin) and verify that the data is stored when the form passes validation.

47.5

Modification of Forms

As we know, hooks allow us to interact with the system and with other modules. Drupal facilitates function hook that allow us to interact with forms created by other modules, modifying, deleting or adding fields to these forms before they are built and submitted.

Function `hook_form_alter()`

The function `hook_form_alter()` allows interaction with a form before its construction begins:

http://api.drupal.org/api/drupal/modules--system.api.php/function/hook_form_alter/7

A common use of this hook in different modules is the modification of the form for creating or editing nodes, mainly to add new fields and thus, new features related to the nodes.

The definition of this function is:

`hook_form_alter(&$form, &$form_state, $form_id)`

the required parameters being:

- **&\$form**, definition of the form.
- **&\$form_state**, structured array with the current state of the form.
- **\$form_id**, string with the form ID.

As the `hook_form_alter()` function is executed for all forms, generally we check the value of `$form_id` to carry out actions on a particular form.

For example, the core **Node** module generates a form for each content type. We can access each form by its identifier, which corresponds to the pattern **nodetype_node_form** (page_node_form, article_node_form, etc.). In the following implementation of `hook_form_alter()` we add a checkbox field type within the group **Publishing Options** to the form for editing or creating a node of the basic page type (page_node_form). F47.37

```
F47.37
hook_form_alter()
Allows modification of the
defined forms from other
modules.



---



```
/**
 * Implements hook_form_alter().
 */
function forms_forcontu_form_alter(&$form, &$form_state, $form_id) {
 if($form_id == 'page_node_form') {
 $form['options']['private'] = array(
 '#type' => 'checkbox',
 '#title' => t('Private'),
 '#default_value' => '0',
);
 }
}
```



---



```

Figure F47.38 shows the form for creating the modified **Basic page**.

The screenshot shows the 'Create Basic page' form. In the 'Publishing options' section, the 'Private' checkbox is highlighted with a red border. Other options like 'Published', 'Promoted to front page', and 'Sticky at top of lists' are also present.

F47.38

Edited form

In this example we have modified the form for creating a basic page, adding an additional option (Private) within the Publish Options.

Again we recommend using the **debugging functions** of the **Devel** module to print the value of the form variable. By doing so you will understand better the structure of form elements and be able to add new fields in the correct place. Of course, once completed, module development must remove all added debugging functions. **F47.39**

```
/***
 * Implements hook_form_alter().
 */
function forms_forcontu_form_alter(&$form, &$form_state, $form_id) {
  if ($form_id == 'page_node_form') {
    dprint_r($form); // Shows the content of $form

    $form['options']['private'] = array(
      '#type' => 'checkbox',
      '#title' => t('Private'),
      '#default_value' => '0',
    );
  }
}
```

F47.39

Debugging functions

Using the Devel module you can add debugging functions in the code to get the contents of the variables.

Function hook_form_FORM_ID_alter()

The hook_form_FORM_ID_alter() function is similar to hook_form_alter(), with the difference that it will execute only for a particular form, substituting for the form identifier the name of the function in the pattern of FORM_ID.

The function definition is:

hook_form_FORM_ID_alter(&\$form, &\$form_state, \$form_id)

downloadable at:

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_form_FORM_ID_alter/7

The use of hook_form_FORM_ID_alter() allows us to separate changes acting on different forms, thus avoiding having a single function with a switch sentence to distinguish between forms on which modifications have been made.

The following shows the implementation of this hook, equivalent to the above implementation of **hook_form_alter()**. In this case, since the form identifier is '**page_node_form**', the name of the function will be **forms_forcontu_page_node_form_form_alter()**. **F47.40**

F47.40

hook_form_FORM_ID_alter()

A function to modify a specific form.

```
/*
 * Implements hook_form_FORM_ID_alter().
 */
function forms_forcontu_page_node_form_form_alter(&$form,
&$form_state, $form_id) {
  $form['options']['private'] = array(
    '#type' => 'checkbox',
    '#title' => t('Private'),
    '#default_value' => '0',
  );
}
```

Forms in the administration area of the module

47.6

In **Unit 44** we laid out how to build a management area or module configuration using forms. Now that we know more about creating forms, we can expand the configuration area of the module, including the corresponding validation functions and processing.

Let's continue with the example used in **Unit 44** where we defined the configuration form for the **Node Expiration Date** module, available at URL: [/admin/config/workflow/node_expiration_date](#). **F47.41**

```
/***
 * Implements hook_menu().
 */
function node_expiration_date_menu() {
  $items['admin/config/workflow/node_expiration_date'] = array(
    'title' => 'Node Expiration Date settings',
    'description' => 'Settings for module Node Expiration Date',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('node_expiration_date_admin_settings'),
    'access arguments' => array('administer site configuration'),
    'type' => MENU_NORMAL_ITEM,
    'file' => 'node_expiration_date.admin.inc',
  );
  return $items;
}
```

F47.41

Module administration area

Example of a form in the administration area of a module.

The module configuration area will be a page with a list of the different types of content on our website, each accompanied by a selection box or checkbox so that the user can select which content types will have an expiration.

Drupal calls the **drupal_get_form()** function, which in turn calls the function referenced in the 'page arguments' parameter (`node_expiration_date_admin_settings`). This second function contains the structure of the form to generate. **F47.42**

```
/***
 * Form builder to configure the module.
 */
function node_expiration_date_admin_settings() {
  /**
   * With node_type_get_types(), we get information for all of
   * the content types, each one as an object, so you should select
   * only the names of the content types to display in the options.
   */
  $content_types_list = node_type_get_types();
  foreach ($content_types_list as $key => $type) {
    $list[$key] = $type->name;
  }
  $form['node_expiration_date_node_types'] = array(
    '#type' => 'checkboxes',
    '#title' => t('Add an expiration date to these content types'),
    '#options' => $list,
    '#default_value'=>
variable_get('node_expiration_date_node_types', array('page')),
    '#description' => t('The selected content types will have an
expiration date.'),
  );
  return system_settings_form($form);
}
```

F47.42

Module administration area

A function that defines a module administration form.

The form returns the value of `$form` through the `system_settings_form()` function, which is an API function that completes the form with the buttons for submit, cancel, etc. This function is also responsible for storing the fields requested in the form in variables of the Drupal table variable.

Unlike other forms we have studied, in the administration area we used the function: **return system_settings_form(\$ form);**

The **system_settings_form()** function is responsible for completing the form as an argument passed by a submit button and a reset button. The form will be processed through a generic system function call **system_settings_form_submit()**, which is responsible for, among other things, storing the values entered in the form in the database. In this way, the data related to the form is stored in the system's **table** variable and you will not need to create new tables to store the configuration data.

The definition of **system_settings_form()** function is available at:

http://api.drupal.org/api/drupal/modules--system--system.module/function/system_settings_form/7

The definition of the **system_settings_form_submit()** function is available at:

http://api.drupal.org/api/drupal/modules--system--system.module/function/system_settings_form_submit/7

We can also add to the configuration module form the corresponding validation function. **F47.43**

```
/***
 * Validation function.
 */
function node_expiration_date_admin_settings_validate($form,
&$form_state) {
    // Validation of form fields
}
```

F47.43

Form validation

Example of the form validation function.

Functions of Form API

47.7

In addition to the functions already studied in this unit, the Drupal's form system provides other Drupal API functions for use in the implementation of modules. The full list of features is available at:

http://api.drupal.org/api/drupal/includes--form.inc/group/form_api/7

Let's review in this section some of these functions: **F47.44**

Function	Description
<code>date_validate()</code>	Validates the date type elements to override erroneous dates (e.g. February 30, 2012).
<code>drupal_build_form()</code>	Builds and processes based on a Form ID.
<code>drupal_form_submit()</code>	Retrieves, populates, and processes a form.
<code>drupal_get_form()</code>	Gets the form from the build function or from the cache.
<code>drupal_prepare_form()</code>	Prepare the form elements, completing its definition as modified by other modules.
<code>drupal_process_form()</code>	The form is processed, validated and sent if it passes validation.
<code>drupal_rebuild_form()</code>	Rebuilds the form.
<code>drupal_redirect_form()</code>	Redirects the user to the specified URL after the form has been processed.
<code>drupal_retrieve_form()</code>	Returns the array with the structure of the form definition.
<code>drupal_validate_form()</code>	Executes the validation functions for the form.
<code>form_clear_error()</code>	Cleans errors sent to any form element with <code>form_set_error()</code> .
<code>form_error()</code>	Flags an element indicating it has an error.
<code>form_get_cache()</code>	Gets a cached form.
<code>form_get_error()</code>	Returns an error message displayed for the appropriate form element.
<code>form_get_errors()</code>	Returns an associative array with all of the form errors.
<code>form_process_campo()</code>	Allows different types of process and edit fields.
<code>form_set_cache()</code>	Stores a form in the cache.
<code>form_set_error()</code>	Sets an error on a form element.
<code>form_set_value()</code>	Changes the value of a sent element from the validator (<code>\$form_state</code>).
<code>form_state_defaults()</code>	Returns the default value for the array <code>\$form_state</code> .
<code>form_type_campo_value()</code>	Helper function to determine the value for a form element of the type indicated.
<code>password_confirm_validate()</code>	Validates a password_confirm element.
<code>theme_button()</code> <code>theme_checkbox()</code> <code>theme_date()</code> ...	Functions that format various form elements. These functions can be overwritten in the site theme to apply custom changes.

F47.44

API Forms functions

List of functions that can interact with forms.

Form Builder Module

The **Form Builder** module provides a graphical interface for creating forms. After creating the form with Form Builder, we can export the code to include it in newly developed modules.

The Form Builder module is available at:

http://drupal.org/project/form_builder

It is dependent on the **Options Element** module, available at:

http://drupal.org/project/options_element

Because the module is still in development, the best way to test its operation is to start with the examples provided by the **Form builder examples** included in the Form Builder module.

Let's sign in directly to the URL [/form-builder-example](#), which shows an example of a form constructed with Form Builder. **F47.45**

F47.45

Form Builder Module

The Form Builder module provides a graphical interface for creating forms. After creating the form with Form Builder once, we can export the code to include it in other modules being developed.

Form builder example

The screenshot shows a 'Form preview' section with several fields: 'Sample textfield' (Prefix: 'a sample value'), 'Sample checkboxes' (checkboxes for 'one', 'two', 'three'), 'email *' (text input), and 'Sample textarea' (text area with 'Text area sample value'). To the right is a sidebar titled 'Add a field' containing a grid of field types: Fieldset, Número, Lista de selección, Checkboxes, Radios, Campo de texto, Área de texto, Archivo, and Imagen. Buttons for 'Editar' and 'Exportar' are at the top left.

The right column shows the available fields, which can be added to the form by dragging and dropping. Once in the form, each field can be configured by clicking the field or the edit icon.

F47.46

Sample checkboxes

one
two
three

Properties	Opciones	Validation	Close

Pre determinado | Clave | Valor

Pre determinado	Clave	Valor	
+ <input type="checkbox"/>	one	one	+ x
+ <input checked="" type="checkbox"/>	two	two	+ x
+ <input type="checkbox"/>	three	three	+ x

[+ Add item](#)
[Manual entry](#)

Option settings

Customize keys
Customizing the keys will allow you to save one value internally while showing a different option to the user.

F47.46**Forms created with Form Builder**

Example of an element added with Form Builder.

We can't save the created form directly. From the **Export** tab, however, we can get the form's code to apply it directly in a function definition of the form of a module. **F47.47**

Form builder example

[Edit](#) [Export](#)

Export code

```
$form = array();
$form['sample_textfield'] = array(
  '#size' => '20',
  '#weight' => '0',
  '#field_suffix' => ':Suffix',
  '#field_prefix' => 'Prefix: ',
  '#type' => 'textfield',
  '#title' => t('Sample textfield'),
  '#default_value' => 'a sample value',
);

$form['sample_checkboxes'] = array(
  '#weight' => '1',
  '#multiple' => '1',
  '#default_value' => array(
    '0' => 'two',
  ),
  '#options' => array(
    'one' => t('one'),
    'two' => t('two'),
    'three' => t('three'),
  ),
  '#type' => 'checkboxes',
  '#title' => t('Sample checkboxes'),
);
$form['sample_textarea'] = array(
  '#weight' => '2',
);
```

F47.47**Export a form with Form Builder**

After creating the form, we can export it through the Export tab. The resulting code can be used in any form definition function of the module that we are developing.

Finally, **Form Builder** includes the **Form builder Webform UI** module, which substitutes the interface for constructing Webform forms with the Form Builder interface already studied.

Once installed, the interface will be available when you create or edit a Webform form from the tab **Form components**. **F47.48**

F47.48

Form Builder and Webform

Through the Form builder Webform UI module we can integrate Webform with Form Builder, so that we can compose Webform forms using the Form Builder GUI.

In this case, the self-generated form that can be stored, since the result is linked to the Webform form.

48 Programming Blocks

In Unit 44 of this level we made a first approach to implementing blocks within a module. As we know, the blocks are units of content, static or dynamic, which can be displayed in specific regions of the theme of the site (header, footer, right side, left side etc.). We also know that the blocks can be managed from:

Administration ⇒ **Structure** ⇒ **Blocks**

From the block administration area you can configure, for example, in which region the block will be located, for which roles the block will be visible, and on which pages it will be displayed.

We can create new blocks from the block administration area, but we can also program them from within a new module. Generally we advise that you only manually create blocks through the administration area when the content of the blocks is static (HTML). Otherwise it is always recommended that you implement a block from within a module, since it is easier to maintain included code in a module that is included directly within a block.

Blocks are defined within modules by implementing **hook_block_info()** function. The block is defined with support of the **hook_block_view()** function, in which its content is created.

The block implemented by the module will work like any other site block, so it will be available in the block administration area, and the site administrator can select its location and visibility options.

Comparative D7/D6

Scheduling blocks

In Drupal 6 we used the `hook_block()` function to implement all operations related to the block ('list', 'configure', 'save' and 'view').

In Drupal 7 this hook has been divided into separate hooks: `hook_block_info()`, `hook_block_view()`, `hook_block_configure()`, etc.

Unit contents

48.1 Defining Blocks	156
48.2 Implementing Blocks	160
48.3 Automatic Activation of Blocks	167
48.4 Modifying Blocks	170

48

48.1 Defining Blocks

Blocks are defined within modules by initiating the **hook_block_info()** function. Also initiate **hook_block_view()**, which implements the block contents.

A block thus created works like any other block of the site, so it will be available in the block administration area and the site administrator can select its location and visibility options.

Database Structure

The tables that define block architecture in Drupal 7 are:

- **block**. Stores the configuration of each block: such as the region, the visibility, the active theme, etc.
 - **bid**. Block identifier.
 - **module**. Name of the module that implements the block. Manually created blocks are associated with the value "block".
 - **delta**. Each implementation of **hook_block_info()** may include several different blocks. This field contains the key for each of them, so that each delta must be unique within a deployment of `hook_block_info()`, but may be repeated in the table block.
 - **theme**. Blocks can be defined for multiple themes. In the block table there will be a log for each block-theme pair, in this way if there are three active themes, a new block generates 3 entries in the table, one for each theme.
 - **status**. Indicates whether the block is **active (1)** or **inactive (0)**.
 - **weight**. As we have seen in other elements Drupal this parameter determines the position of the block in a given region.
 - **region**. Contains the name of the region where the block is displayed. The system name of the region is used. If the block is not assigned to a region the value is -1.
 - **custom**. Contains the values associated with the configuration option **Customizable user**. It can contain the value **0** (not customizable), **1** (Customizable, visible by default) or **2** (Customizable hidden by default).
 - **visibility**. Contains the values associated with the configuration option **Show block on specific pages**. It can contain the value **0** (All pages except those listed), **1** (Only listed pages) or **2** (Pages on which the PHP code returns True).
 - **pages**. This field depends on the values specified in **visibility**. It will contain a set of routes or PHP code, depending on the value for **visibility**.

- **title.** Contains the title of the block. If empty, the module set to implement it will display. If you do not want to show any title, use the value <**none**>.
- **cache.** Specifies the type of cache to be applied to the block. For example, a value of -1 indicates that the block is not cached.
- **block_custom.** Stores the content of custom blocks created directly by the Block module.
 - **bid.** Block identifier
 - **body.** Block contents.
 - **info.** Block description.
 - **format.** Text formatting applied to block content (body field).
- **block_node_type.** Stores the content types for which each block is displayed. Corresponds to the **Show block for specific content types** within the block **Visibility options**.
 - **module.** Contains the name of the module that implements the block.
 - **delta.** Delta value of the block.
 - **type.** System name for the content type.
- **block_role.** Stores the roles that can see each block. It corresponds to the Show block to certain roles option within the block Visibility options.
 - **module.** Contains the name of the module that implements the block.
 - **delta.** Delta value of the block.
 - **rid.** Numerical identifier of the role.

Implementing hook_block_info()

Using **hook_block_info()**, the module can declare one or more blocks. With this function the block is set and its initial configuration establishes, which subsequently may be changed from the block administration area.

Find the complete description of the **hook_block_info()** function in Drupal API:

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_info/7

The **hook_block_info()** function has no input parameters. It returns an associative array where each block is defined by the following elements:

- **info (obligatory).** Block name that will appear in the list of blocks administration.
- **cache (optional).** Defines the cache type for the block. The choice of cache type for a block depends on its contents and how it varies depending on the user, the page, etc.. Possible values are:

Comparing D7/D6 Blocks

In Drupal 6, a single `hook_block()` function is used. Through the `$op` parameter we distinguish the different possible operations: show list of block modules, generate the content, block settings, etc..

In Drupal 7 these operations are separated into different hooks:
`hook_block_info()`,
`hook_block_view()`,
`hook_block_configure()`,
etc..

- **DRUPAL_CACHE_PER_ROLE** (default). The block can change depending on the roles of the user visiting the page.
 - **DRUPAL_CACHE_PER_USER**. The block can change depending on the user who views it.
 - **DRUPAL_CACHE_PER_PAGE**. The block can change depending on the displayed page.
 - **DRUPAL_CACHE_GLOBAL**. The block is the same for all users and in all pages where it is shown.
 - **DRUPAL_NO_CACHE**. The block does not cache it.
- **properties (optional)**. Array with additional metadata that will be passed to the block.
 - **weight (optional)**. Initial value of the block's weighted order. The administrator can change this value from the block administration area.
 - **status (optional)**. Initial value of the block status (1 = enabled, 0 = disabled). Generally this value is not provided, so that the block will be disabled by default.
 - **region (optional)**. Initial value of the region in which the block is shown. If the subject does not have the indicated region, the block will be disabled.
 - **visibility (optional)**. Initial value for block visibility. Possible values are:
 - BLOCK_VISIBILITY_NOTLISTED: Displays on all pages except the ones listed.
 - BLOCK_VISIBILITY_LISTED: Displays only on the pages listed.
 - BLOCK_VISIBILITY_PHP: Use custom PHP code to determine the visibility.
 - **pages (optional)**. List of URLs or PHP code related to the option **visibility**.

Implementing hook_block_view()

Inside **hook_block_view()** we will define the content and function of the block.

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_view/7

Within a module, the way to identify a block is through the value **\$delta**; therefore, this will be the only required input parameter.

hook_block_view(\$delta = "")

The function returns an array with the following elements:

- **subject**. The default block title. If the block has no default title, the value will be NULL.
- **content**. The body content of the block, which may be a **renderable array** or a string with HTML content.

A **renderable array** is a vector with a specific structure that can be converted to HTML content. The Drupal themes system is responsible for interpreting the array and generating the final HTML, depending on the templates and the site theme. We will learn more about renderable arrays in **Unit 56**.

Implementing hook_block_configure()

Through **hook_block_configure()** we can add specific settings for the block. In this function we implement the form with the new fields that will be added in the configuration blockue.

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_configure/7

The value of \$delta is required as an input parameter, to be evaluated within the function to set the various module blocks.

hook_block_view(\$delta = "")

The function returns the **\$form** array containing the form's structure.

Implementing hook_block_save()

Finally, **hook_block_save()** saves the block configuration according to the form implemented in **hook_block_configure()**.

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_save/7

hook_block_save(\$delta = "", \$edit = array())

The \$delta parameter identifies the block within the module, while the \$edit parameter contains the information sent by the user through the block configuration form. Inside the function we have to collect this data and store it where indicated (variables, tables in the database, etc.).

48.2

Implementing blocks

In this section we'll apply the studied functions for the implementing blocks in a module. For this we'll create a new module called **Forcontu Blocks**, which will generate two blocks.

The first block will list **unpublished nodes**, allowing the administrator to configure how many nodes are displayed in the listing.

The second block will display a **list of registered users** on the site. It can be configured to display to all users or only to active users, in addition to the number of users from the list.

Step 1. Creating the Blocks Forcontu Module

The module we will create and which will serve to define the blocks will be called **Forcontu Blocks**, with a system name of **blocks_forcontu**. We will create the folder **blocks_forcontu** and the file **blocks_forcontu.info** in the module definition.

The file **blocks_forcontu.module** will contain the module functions that we'll develop in the next steps.

Step 2. Defining the Block 'Unpublished Nodes'

We implement **hook_block_info()** to define the block.

F48.1**F48.1****hook_block_info()**

Implementation of **hook_block_info()**. It includes the definition of the block.

```
/***
 * Implements hook_block_info().
 */
function blocks_forcontu_block_info() {
  // Unpublished Nodes block
  $blocks['unpublished_nodes'] = array(
    'info' => t('Unpublished nodes'),
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}
```

The defined block will use the identifier "**unpublished_nodes**", which is the index in the array **\$blocks** and corresponds to the value **\$delta**. This value will be used in the other module functions to reference the block.

Although we did not define the contents of the box, if we install and activate the module, the block will display in the blocks administration area, initially disabled.

Step 3. Configuration Form for the 'Unpublished Nodes' Block.

For block configuration we need to implement the function **hook_block_configure()**.

For the block "unpublished nodes", the administrator can configure the number of nodes that are listed. We will create for it a **select** form element type, which allows the selecting of a number between 1 and 5. The contents of this setting will be stored in a variable in the table **variable**, which we will call "**blocks_forcontu_num_nodes**". The default value will be 3. **F48.2**

```
/***
 * Implements hook_block_configure().
 */
function blocks_forcontu_block_configure($delta = '') {
  $form = array();

  switch ($delta) {
    case 'unpublished_nodes':
      $form['num_nodes'] = array(
        '#type' => 'select',
        '#title' => t('Select the number of nodes to display in this block.'),
        '#default_value' => variable_get('blocks_forcontu_num_nodes', 3),
        '#options' => array(1=>1, 2=>2, 3=>3, 4=>4, 5=>5),
      );
      break;
  }
  return $form;
}
```

F48.2

hook_block_configure()
Additional block configuration options.

We added the field "num_nodes" to the configuration form, a list selection type (select). The field value is obtained from the variable **blocks_forcontu_num_nodes**. If the variable does not yet exist, a default value of 3 is returned.

At this point we can access the block configuration. We have not yet implemented the function that stores the value of the new option, so no changes will be saved. **F48.3**

Home » Administration » Structure » Blocks
'Unpublished nodes' block [edit](#)

Block title

Override the default title for the block. Use <none> to display no title, or leave blank to use the default block title.

Select the number of nodes to display in this block.

3 ▾

1
2
3
4
5

REGION SETTINGS

Specify in which themes and regions this block is displayed.

Bartik (default theme)

F48.3

hook_block_configure()
Additional configuration option added to the block.

Step 4. Save the 'Unpublished Nodes' Block Configuration

Initiating the **hook_block_save()** function will store in the database the configuration parameters entered by the user.

The system will call the **hook_block_save()** function when we save the block configuration form by clicking the **Save The Block** button. When we initiation this hook, we specify what the module should do with the data entered through the form.

The **hook_block_save()** function will get two parameters: **\$delta**, which is the block identifier, and **\$edit**, which will contain the configuration settings entered by the user on the block configuration form.

In this example we store the field value **\$edit['num_nodes']** in the variable **blocks_forcontu_num_nodes**. **F48.4**

F48.4

hook_block_save()
Save the configuration options of the block.

```
/***
 * Implements hook_block_save().
 */
function blocks_forcontu_block_save($delta = '', $edit = array()) {
  switch ($delta) {
    case 'unpublished_nodes':
      variable_set('blocks_forcontu_num_nodes', $edit['num_nodes']);
      break;
  }
}
```

If we use the **variable_get()** function to get the value of the variable in **hook_block_configure()**, to save the value we use the **variable_set()** function. As mentioned before, the values sent in the configuration form will be passed to **hook_block_save()** through the **\$edit** parameter. Therefore, we will store in the database variable "**blocks_forcontu_num_nodes**" the value passed by the form, which was stored in **\$edit["num_nodes"]**.

After adding this code, we can modify the value of the configuration parameter and check that the value is stored and retrieved from the database correctly.

Step 5. Display the Contents of the 'Unpublished Nodes' Block

Finally, we need to include the code for displaying the desired content is the block. We'll use the **hook_block_view()** function, which receives a **\$delta** parameter value to identify the block.

We need to display a list of unpublished nodes. To do this we need to make a database query.

Although we could manually make the list of nodes, Drupal API does this process for us with the **node_title_list()** function, which returns a list of node titles, with their corresponding links, as a result of a database query that returns node type entities.

You can see the definition of **node_title_list()** function at: **F48.5**

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_title_list/7

```

/*
 * Implements hook_block_view().
 */
function blocks_forcontu_block_view($delta = '') {

  $block = array();
  switch ($delta) {
    case 'unpublished_nodes':
      $result = db_select('node', 'n')
        ->fields('n', array('nid', 'title', 'created'))
        ->condition('status', 0)
        ->orderBy('created', 'DESC')
        ->range(0, variable_get('blocks_forcontu_num_nodes', 3))
        ->execute();

      if ($node_title_list = node_title_list($result)) {
        $block['subject'] = t('Unpublished nodes');
        $block['content'] = $node_title_list;
      }
      break;
  }
  return $block;
}


```

F48.5**hook_block_view()**

This function implements the content that will be displayed in each block defined by the module.

We perform the database query using the **db_select()** function. As a condition to the query we have indicated that the **status** field of the node should be **0** (unpublished node). We have also specified the range of the query, getting the number of elements from the block configuration variable **blocks_forcontu_num_nodes**.

Once done, the query calls the function **node_title_list(\$result)**, which converts the query results into a renderable array with the titles of the nodes and their corresponding links. This array will go directly to the contents of the block (**\$block ['content']**).

The resulting block, once activated, is shown in **Figure F48.6**. The block has been set to show the last **4 nodes** created whose status is **unpublished**.

plaga praemitto quidem. Adipiscing defui humo letalis
pecus tum. Dolus euismod humo scisco similis usitas ut.
is quidne sagaciter tum utrum uxor vero ymo. Mos pala

ig blandit genitus gilvus luptatum nostrud quae ratis
idus. Macto ulciscor ut valde.

**F48.6****Displaying the block**

The Figure shows the unpublished block Nodes once activated on the site.

Step 6. Defining the 'User List' Block

To understand the need for the \$delta parameter, we'll create a second block within the same module. The new block will display a list of registered users on the site. It can be configured to display to all users or only to active users, in addition to the number of users on the list.

Inside the function that implements **hook_block_info()** we add a new entry to the array **\$blocks**, with the identifier '**users_list**'. F48.7

F48.7**hook_block_info()**

We added the same function in the definition of another block.

```
/*
 * Implements hook_block_info().
 */
function blocks_forcontu_block_info() {
  // Unpublished Nodes block
  $blocks['unpublished_nodes'] = array(
    'info' => t('Unpublished nodes'),
    'cache' => DRUPAL_NO_CACHE,
  );
  // Registered Users block
  $blocks['users_list'] = array(
    'info' => t('Users list'),
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Step 7. Configuring the 'User List' Block

In implementing the **hook_block_configure()** function we have to use the \$delta parameter to differentiate the two blocks of the module. F48.8

F48.8**hook_block_configure()**

To distinguish the actions to take for each block, we use the \$delta parameter, checking its value with a switch statement.

```
/*
 * Implements hook_block_configure().
 */
function blocks_forcontu_block_configure($delta = '') {
  $form = array();

  switch ($delta) {
    case 'unpublished_nodes':
      // code for the first block
      break;
    case 'users_list':
      $form['num_users'] = array(
        '#type' => 'select',
        '#title' => t('Select the number of users to display in this block.'),
        '#default_value' => variable_get('blocks_forcontu_num_users', 3),
        '#options' => array(1=>1, 2=>2, 3=>3, 4=>4, 5=>5),
      );
      $form['active_users'] = array(
        '#type' => 'checkbox',
        '#title' => t('Display only active users.'),
        '#default_value' => variable_get('blocks_forcontu_active_users', 0),
      );
      break;
  }
  return $form;
}
```

For the block "Registered Users", the administrator can configure the number of users that will be listed. We'll create for it a **select** type form element, which allows the selecting of a number between 1 and 5. The content of this setting is stored in a variable in the table **variable**, which we call '**blocks_forcontu_num_users**'. The default value will be 3.

To indicate whether only active users are displayed, we'll add a **checkbox** type element to the form. The default value will be 0, which indicates that it displays to all users, regardless of whether they are active or not. When the value of the configuration variable **blocks_forcontu_active_users** is equal to 1 (returned by the checkbox value), the block will only display to active users, up to the set limit.

Step 8. Saving the 'User List' Block Configuration

In the **hook_block_save()** function we'll add the code for the new block, using the **\$delta** parameter to differentiate it from the other block.

For the **users_list** block we have to save two variables, whose values are stored in **\$edit['num_users']** and **\$edit['active_users']**. **F48.9**

```
/** 
 * Implements hook_block_save(). 
 */
function blocks_forcontu_block_save($delta = '', $edit = array()) {
  switch ($delta) {
    case 'unpublished_nodes':
      variable_set('blocks_forcontu_num_nodes', $edit['num_nodes']);
      break;
    case 'users_list':
      variable_set('blocks_forcontu_num_users', $edit['num_users']);
      variable_set('blocks_forcontu_active_users', $edit['active_users']);
      break;
  }
}
```

F48.9

hook_block_save()

Again we use **\$delta** to distinguish the operations of each block.

Step 9. Displaying the Contents of the 'Users List' Block

Finally, we add the **hook_block_view()** function to show the code block, identifying it with the **\$delta** parameter.

To display the list of users execute a statement with **db_select()**. The condition that filters users according to their state (status) will be added to the query only if we have activated the configuration option **Display only active users**, stored in the variable **blocks_forcontu_active_users**.

To print the list of users we have made use of the API function **theme_user_list()**, which generates a list from an array of objects of user types. **F48.10**

You can see the definition of **theme_user_list()** at:

http://api.drupal.org/api/drupal/modules--user--user.module/function/theme_user_list/7

F48.10**hook_block_view()**

Adding the display of the new block.

```
/*
 * Implements hook_block_view().
 */
function blocks_forcontu_block_view($delta = '') {
  $block = array();
  switch ($delta) {
    case 'unpublished_nodes':
      // first block code
      break;

    case 'users_list':
      $query = db_select('users', 'u')
        ->fields('u', array('uid', 'name', 'created'))
        ->orderBy('created', 'DESC')
        ->range(0, variable_get('blocks_forcontu_num_users', 3));

      // checks whether to display only active users
      if(variable_get('blocks_forcontu_active_users', 3)==1) {
        $query->condition('status', 1);
      }
      $result = $query->execute();

      $variables['users'] = $result->fetchAll();
      $variables['title'] = NULL;

      $block['subject'] = t('Users list');
      $block['content'] = theme_user_list($variables);

      break;
  }
  return $block;
}
```

The **Users list** block is shown in **Figure F48.11**. The links to each user's profile will display only if the user that is seeing the page has adequate permission to access the profiles of other users.

F48.11**Blocks**

The diagram shows the blocks generated by the module.

plaga praemitto quidem. Adipiscing defui humo letalis
o pecus tum. Dolus euismod humo scisco similis usitas ut.
is quidne sagaciter tum utrum uxor vero ymo. Mos pala

ig blandit genitus gilvus luptatum nostrud quae ratis
idus. Macto ulciscor ut valde.

[Read more](#)

Unpublished nodes

- Pala Similis Tation
- Cui Exputo Interdico Rusticus
- Webform and Form Builder
- Appellatio Populus

Users list

- admin
- gowichedaco
- stitrec
- punava

Automatic Activation of Blocks

48.3

By default blocks are displayed in the block administration area as disabled.

Activating the Block within `hook_block_info()`

By default blocks are displayed in the block administration area as disabled.

To enable a block during module installation, we can define the following block parameters in `hook_block_info()`: [F48.12](#)

- **status (optional)**. Initial value of block status (1 = enabled, 0 = disabled).
- **region (optional)**. Initial value of the region in which the block is shown. If the theme does not have the indicated region, the block will display disabled.

```
/***
 * Implements hook_block_info().
 */
function blocks_forcontu_block_info() {
  // Unpublished Nodes block
  $blocks['unpublished_nodes'] = array(
    'info' => t('Unpublished nodes'),
    'cache' => DRUPAL_NO_CACHE,
    'status' => 1,
    'region' => 'sidebar_second',
  );
  return $blocks;
}
```

F48.12

Block activation

Through `hook_block_info()` we can activate the block and assign it to a specific region of the theme.

This system has two drawbacks. The first is that if the specified region does not exist in the active theme, the block will display as disabled. Not all items have the same regions, so we cannot guarantee that the specified region will be available. [Figure F48.13](#) shows the default regions in Drupal 7.

```
regions[header] = Header
regions[highlighted] = Highlighted
regions[help] = Help
regions[content] = Content
regions[sidebar_first] = Sidebar first
regions[sidebar_second] = Sidebar second
regions[footer] = Footer
```

F48.13

Default regions

List of default regions in the Drupal 7 themes.

We can solve this problem by referring to the available regions in the active theme, making use of these API functions:

- **system_region_list()**, which returns the list of regions for a specified theme.
http://api.drupal.org/api/drupal/modules--system--system.module/function/system_region_list/7
- **system_default_region()**, which returns the name of the theme's default region. Generally it will be the first region defined in the theme.
http://api.drupal.org/api/drupal/modules--system--system.module/function/system_default_region/7

For the active theme, we can refer to the following variables:

- **theme_default**, which returns the name of the active theme.
- **admin_theme**, which returns the name of the theme administration.

The following example checks if the active theme has the region 'sidebar_second'. If not, it will assign the block to the default region for the theme. **F48.14**

F48.14

Assigning regions

To assign to a correct region we can consult all of the available regions in the active theme.

```
/***
 * Implements hook_block_info().
 */
function blocks_forcontu_block_info() {
    // gets the name of the active theme
    $default_theme = variable_get('theme_default', 'bartik');

    // gets the list of regions for the theme
    $regions = system_region_list($default_theme, REGIONS_VISIBLE);

    // if it doesn't have a 'sidebar_second' region
    // activate the block in the default region
    if($regions['sidebar_second'])
        $region = 'sidebar_second';
    else
        $region = system_default_region($default_theme);

    // Unpublished block nodes
    $blocks['unpublished_nodes'] = array(
        'info' => t('Unpublished nodes'),
        'cache' => DRUPAL_NO_CACHE,
        'status' => 1,
        'region' => $region,
    );

    return $blocks;
}
```

Activate the Block with hook_install()

The second drawback is that **the blocks are not created during the module's installation/activation process**. It is necessary to access the blocks administration area for the system to generate new blocks. Until we do this action, the blocks will be neither available nor visible on the site.

To activate the block during module installation, we can use **hook_install()**, calling the **_block_rehash()** function, which reconstructs the list of blocks added for the modules. **F48.15**

http://api.drupal.org/api/drupal/modules--block--block.module/function/_block_rehash/7

F48.15

Activating the block

Example of block activation during installation of the module.

```
/***
 * Implements hook_install().
 */
function blocks_forcontu_install() {
    // gets the name of the active theme
    $default_theme = variable_get('theme_default', 'bartik');
    _block_rehash($default_theme);
}
```

Thus, the block will be activated according to the specifications made in **hook_block_info()**.

Display Permissions of the Block

Within the block's settings we can specify which roles may view it, keeping in mind that the administrator can change this behavior.

Another way to control the display of the block is through permissions, which may be a permission already available on the site or a permission created by the module itself.

The time to check if the user has a particular permission will be within **hook_block_view()**, using the **user_access()** function. The block will not display if the **hook_block_view()** function returns no value in the \$block variable, so just assign the appropriate values to \$block if the user has the adequate permission.

In this example we added to the module permission to 'view unpublished nodes block'. The block 'unpublished_nodes' will only be displayed if the user is assigned this permission. **F48.16**

```
/***
 * Implements hook_permission().
 */
function blocks_forcontu_permission() {
  return array(
    'view block unpublished nodes' => array(
      'title' => t('View block: Unpublished nodes'),
      'description' => t('View the Unpublished nodes block.'),
    ),
  );
}

/***
 * Implements hook_block_view().
 */
function blocks_forcontu_block_view($delta = '') {
  $block = array();
  switch ($delta) {
    case 'unpublished_nodes':
      if (user_access('view block unpublished nodes')) {
        //...
        $block['subject'] = t('Unpublished nodes');
        $block['content'] = $node_title_list;
      }
      break;
  }
  return $block;
}
```

F48.16

Permissions

When viewing the block (**hook_block_view()**), we can check the user's permissions to determine whether to display the block or not. Similarly, we can set up different content based on user permissions.

Removing Blocks

When we disable a module, the blocks continue to be active until we access the list from block administration. At that moment the system checks the block definitions of all active modules and, upon detecting that the module is no longer active, will stop listing its blocks, although it **will not remove them** from the database.

The blocks can be removed from the database by performing a complete uninstall of the module through the Uninstall option in the module administration area.

48.4

Modifying Blocks

The Block module provides hook functions that allow us to modify blocks created by other modules.

hook_block_info_alter()

The **hook_block_info_alter()** function lets us change the configuration of a block added by another module before it is stored in the database.

hook_block_info_alter(&\$blocks, \$theme, \$code_blocks)

Available at:

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_info_alter/7

The \$blocks parameter has the structure: **F48.17**

\$blocks[module] ['delta'] ['parameter'] = value

```
/** 
 * Implements hook_block_info_alter(). 
 */
function blocks_forcontu_block_info_alter(&$blocks, $theme, $code_blocks) { 
    //Disables the user login block 
    $blocks['user']['login']['status'] = 0; 
}
```

F48.17

Modification of blocks

It is possible to modify the information blocks added by other modules.

hook_block_view_alter()

The **hook_block_view_alter()** function allows us to modify the contents of a block.

hook_block_view_alter(&\$data, \$block)

Available at:

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_view_alter/7

The \$data parameter is an array with the elements 'subject' and 'content'.

The \$block parameter is an object with attributes \$block-> module (name of the module that's implementing the block) and \$block->delta (block identifier).

hook_block_view_MODULE_DELTA_alter()

If we are going to modify a specific block, we can also use the **hook_block_view_MODULE_DELTA_alter()** function, replacing the values MODULE and DELTA for those corresponding to the block that we're going to modify.

This function can be found at:

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_view_MODULE_DELTA_alter/7

hook_block_list_alter()

Finally, the **hook_block_list_alter()** function can act on the block list by adding, deleting, or modifying the definition of the list of blocks.

http://api.drupal.org/api/drupal/modules--block--block.api.php/function/hook_block_list_alter/7

The block list is passed by reference to the function through the parameter **\$blocks**, which is an array of objects of the block type. By passing the parameter by reference, any changes we make for \$blocks affect the blocks for the whole site.

Through **\$blocks** we have access only to the block definition or configuration, but not its content. We can, however, overwrite the contents of any block through the content attribute (\$block-> content).

Figure F48.18 shows an example of the \$blocks structure.

```
blocks =>
... (Array, 8 elements)
8 (Object) stdClass
  bid (String, 1 characters ) 8
  module (String, 6 characters ) system | (Callback) system();
  delta (String, 4 characters ) main
  theme (String, 5 characters ) seven
  status (String, 1 characters ) 1
  weight (String, 1 characters ) 0
  region (String, 7 characters ) content
  custom (String, 1 characters ) 0
  visibility (String, 1 characters ) 0
  pages (String, 0 characters )
  title (String, 0 characters )
  cache (String, 2 characters ) -1
10 (Object) stdClass
23 (Object) stdClass
```

F48.18

To modify blocks

Accessing the list of blocks we can remove or modify blocks added by other modules.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

49 Programming users and permissions

Drupal is a Content Management System based on users, roles and permissions. Under Drupal, it is possible to create various communities in which users can be defined within a context based on content, and personalized implementations.

During the last unit, we recalled within some examples on how to control the visibility of a block according to the user's permissions. We created the function `user_access()`

Across this Unit we will delve into the actual development of modules based on users' roles, and permissions to control accessibility and their elements.

Comparative D7/D6

Scheduling of users and permissions

Under Drupal 6 we used the function `hook_user()` to modify most operations on the object `$user`.

Under Drupal 7 this hook has been divided in several distinct functionalities (`hook_user_load`, `hook_user_login`, `hook_user_logout`, `hook_user_view`, `hook_user_update`, etc.), therefore is necessary to investigate them more in depth.

Unit contents

49.1 User definition with the object \$user	174
49.2 Functions for working with users	177
49.3 Users implementation	183
49.4 Users access	192
49.5 Defining permissions	193

49

49.1

User definition with the object \$user

When a page loads under Drupal, the system generates an object called **\$user** carrying information about the user itself. If it's not a registered user across the Website, it will be carried out as an **anonymous user**, without an unspecified unique identifier.

In both cases the object **\$user** will be generated but with different values. For proper operation of the registered users, it is necessary that the use of cookies is allowed by the browser, since it is how Drupal keeps track of each user's navigation across the Website.

The object **\$user** will be available as a **global variable** across the Website, making it easy to access its contents as we enter such variable. To exemplify how this function works, we can execute the following **PHP snippet** inside a page using **PHP code** text format. We can also use the option **Execute PHP code** included in the **Development block** provided by **Devel** module.

```
global $user;
print_r($user);
```

The results shown for an anonymous user are the following:

F49.1**F49.1****Anonymous user**

Content generated of the \$user object for an anonymous user.

```
stdClass Object
(
    [uid] => 0
    [hostname] => 83.11.1.123
    [roles] => Array
        (
            [1] => anonymous user
        )
    [cache] => 0
)
```

When the user is not logged into the site it is considered an anonymous user and the value of **\$user->uid** is 0.

Whenever a user logs into the Website with a username and password, will be considered as a registered user, and the object **\$user** will be expanded with new values. In this case, the value of **\$user->uid** will be an integer greater than 0, which will be the unique identifier for the user. Such value corresponds to its key in the **users** table (**uid** field).

The result shown for a registered user is the following:

F49.2**F49.2****Registered user**

Content generated of the \$user object for a registered user.

```
stdClass Object
(
    [uid] => 1
    [name] => admin
    [pass] => $S$DuOmJwVvwDDcVikO5NwgWOZY7qXiafiZXpayIY
    [mail] => example@example.com
    [theme] =>
    [signature] =>
    [signature_format] => filtered_html
    [created] => 1326275767
    [access] => 1328179054
)
```

```

[login] => 1328135403
[status] => 1
[timezone] => Europe/London
[language] => es
[picture] => 0
[init] => example@example.com
[data] =>
[sid] => juye8IoRrJAzCNaH4VzMbWis43E5xB71Tk_TCYG1Oq4
[ssid] =>
[hostname] => 83.11.1.123
[timestamp] => 1328179079
[cache] => 0
[session] => ...
[roles] => Array
(
    [
        [2] => authenticated user
        [3] => administrator
    ]
)
)

```

The following table shows each attribute from the object **\$user**, its description and the table in the database where the value is stored. **F49.3**

Attribute	Description	Table
uid	User ID, is the primary key from the users table, and is a unique value. Whenever a user is deleted this unique value will not be re-applied for a new user.	users
name	User name, is the value required for signing into the Website.	users
pass	Encoded user's password. The key cannot be defined, is only resettable, and only possible to obtaining a new value.	users
mail	User's email address. Is not possible to create a new user's account with the same email address.	users
theme	The default theme for the user.	users
signature, signature_for mat	User's signature for comments.	users
created	Date (UNIX timestamp format) when the user's account was created.	users
access	Date (UNIX timestamp format) when was the user's last login.	users
login	Date (UNIX timestamp format) when was the user's last login accurately.	users
status	1 if the user is active, or 0 if the user has been disrupted.	users
timezone	User's timezone.	users
language	If the Website provides several languages, the user's default language value will be stored here.	users
picture	Archive identifier (fid) with the user's profile image. The file is referenced from the file_managed table.	users
init	The user's email address that was used during registration.	users
data	Open data field so that modules may add additional information. This information is stored using PHP serialization format. This data can be stored in serialized PHP format.	users
sid	Session ID in http.	sessions
ssid	Secure session ID in HTTPS.	
hostname	IP address from where the user gain access.	sessions

F49.3**\$user's attributes**

List of attributes of the \$user object. Database table where the attribute is stored is also shown.

timestamp	Date set (in UNIX timestamp format) from the user's last visit to the Webpage.	sessions
cache	Date (In UNIX timestamp format) used for the user's cache.	sessions
session	Open field for modules to add additional session information. This information is stored in serialized PHP format.	sessions
roles	List of assigned roles to users. The table users_roles is directly related to the roles table to obtaining names of roles. The index from the roles' vector corresponds to a numeric role identifier (rid).	users_roles, role

Functions `user_is_logged_in()` and `user_is_anonymous()`

Here we show two functions that could be useful under working modules that relate with users, and that could become necessary to differentiate between registered users and anonymous users.

The API function `user_is_logged_in()` returns TRUE if the user who's loading the page is a registered user, and FALSE otherwise if this is an anonymous user.

http://api.drupal.org/api/drupal/modules--user--user.module/function/user_is_logged_in/7

The function `user_is_anonymous()` has the opposite behavior, it returns TRUE if the user is anonymous, and FALSE otherwise.

http://api.drupal.org/api/drupal/modules--user--user.module/function/user_is_anonymous/7

Function `user_load()`

The function `user_load()` returns the object \$user from its own identifier (uid).

http://api.drupal.org/api/drupal/modules--user--user.module/function/user_load/7

`user_load($uid, $reset = FALSE)`

- The \$uid parameter corresponds with the user's identifier.
- The `$reset` parameter allows to specify if the user's object can be obtained from cache (FALSE), or if it needs to reload from the database (TRUE).

Under Drupal's API we'll find other alternative functions to acquire the \$user object, like:

- `user_load_by_mail()`, which returns the user's object from its email.
- `user_load_by_name()`, which returns the user's object from the user's name.
- `user_load_multiple()`, which returns several users in terms of the specified conditions.

Functions for working with users

49.2

We have seen that the **\$user** object holds information related to the user. We'll find how working modules act on this information through the implementation of different hook functions.

In **Drupal 6** we use a unique hook called **hook_user()**, that requires the parameter **\$op** which distinguishes various operations on users (delete, form, insert, load, login, etc.).

In **Drupal 7** all operations on users have been divided in **multiple independent hooks**, facilitating programming and debugging of code.

hook_user_insert()

The function **hook_user_insert()** allows to act on the user when a new account has been created.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_insert/7

Normally we'll use this function to store additional information in the database.

hook_user_insert(&\$edit, \$account, \$category)

The insert parameters for this function are: **F49.4**

- **&\$edit**. Array with values sent by the user through editing forms or registration forms. By passing them along as a reference is possible to modify the values sent by the user.
- **\$account**. Is the user object on which operations are being done.
- **\$category**. Is the category on which the editing operation is being done. Such categories allow to group fields under the user's profile.

```
/**
 * Implements hook_user_insert().
 */
function example_user_insert(&$edit, $account, $category) {
  db_insert('mytable')
    ->fields(array(
      'myfield' => $edit['myfield'],
      'uid' => $account->uid,
    )))
    ->execute();
}
```

F49.4

hook_user_insert()

Allows to act on the newly created user account.

hook_user_update()

The function **hook_user_update()** allows to act on when a user account has been modified.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_update/7

Normally we'll use this function to updating additional fields in the database which the module has implemented through the rest of function hooks.

hook_user_update(&\$edit, \$account, \$category)

The function's input parameters are similar to those used in **hook_user_insert()**.

hook_user_presave()

The function **hook_user_presave()** allows to act on the user object during creation or any modification, before is being stored in the database.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_presave/7

A common practice of this function is to add additional data in `$edit['data']`, so that such data will be kept while saving the user in the database. The fields entered in the field 'data' are serialized, and stored.

hook_user_presave(&\$edit, \$account, \$category)

The insert parameters of the function are similar to the ones used under **hook_user_insert()** and **hook_user_update()**.

hook_user_load()

The function **hook_user_load()** acts on the user object when loaded from the database.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_load/7

Generally we will use this hook to get additional information from the database that had been stored by the module through **hook_user_insert()**, or **hook_user_update()**.

hook_user_load(\$users)

The parameter **\$users** is an user's objects vector, indexed according to the **uid** from each user. **F49.5**

F49.5

hook_user_load()

Allows to act on the user object when is loaded from the database.

```
/** 
 * Implements hook_user_load().
 */
function example_user_load($users) {
  $result = db_query('SELECT uid, foo FROM {my_table} WHERE uid IN (:uids)', array(':uids' => array_keys($users)));
  foreach ($result as $record) {
    $users[$record->uid]->foo = $record->foo;
  }
}
```

hook_user_view()

The function **hook_user_view()** acts when the user's account information is being displayed.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_view/7

Inside this function we have to add the display of additional fields added by other functions. Once such fields have been structurally defined and formatted, they have to be added to the vector **\$account -> content**.

hook_user_view(\$account, \$view_mode, \$langcode)

The parameters of entries of the function are: **F49.6**

- **\$account**. It is the user object on which the operation is being performed.
- **\$view_mode**. It is the view mode (full, teaser, etc) This will allow us to create different presentations depending on the view mode.
- **\$langcode**. Language used for content presentation.

```
/** 
 * Implements hook_user_view().
 */
function example_user_view ($account, $view_mode, $langcode) {
  $account->content['summary']['blog'] = array(
    '#type' => 'user_profile_item',
    '#title' => t('Blog'),
    '#markup' => l(t('View recent blog entries'), ...),
    '#attributes' => array('class' => array('blog')) ,
  );
}
```

F49.6

hook_user_view()

Allows to act when the user's information is being displayed.

hook_user_delete()

The function **hook_user_delete()** acts when the user account is deleted.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_delete/7

We can use this function to remove additional data related to the deleted user.

hook_user_delete(\$account)

The parameter **\$account** is the user object that is being deleted. **F49.7**

```
/** 
 * Implements hook_user_delete().
 */
function example_user_delete ($account) {
  db_delete('mytable')
    ->condition('uid', $account->uid)
    ->execute();
}
```

F49.7

hook_user_delete()

Allows to act when a user's account is being deleted

hook_user_cancel()

The function **hook_user_cancel()** allows to act when a user's account is being canceled.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_cancel/7

The actions to be made could depend on the account cancellation method. Consult the function **user_cancel_methods()** for more information about cancellation methods.

http://api.drupal.org/api/drupal/modules--user--user.pages.inc/function/user_cancel_methods/7

hook_user_cancel(\$edit, \$account, \$method)

Besides parameters **\$edit** and **\$account** already known, we'll obtain the parameter **\$method**, which indicates the cancellation method of the account.

hook_user_cancel_methods_alter()

The function **hook_user_cancel_methods_alter()** allows to modify the cancellation methods of the account.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_cancel_methods_alter/7

By using this function a module can add, personalize, or delete the cancellation methods.

hook_user_login()

The function **hook_user_login()** acts when a user completes the login on the Website.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_login/7

hook_user_login(&\$edit, \$account)

Allows to make operations when the user logs into the Website: displays a message, redirects the user to a specific page, registers the user's access, etc.

hook_user_logout()

The function **hook_user_logout()** acts when the user logs out from the Website.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_logout/7

hook_user_logout(\$account)

Could be useful for example, to register the session's logout.

hook_user_categories()

The function **hook_user_categories()** returns a vector with available categories under the user's profile. **F49.8**

hook_user_categories()

Returns a vector where each element is a vector with the following fields:

- **'name'**. The category's internal name.
- **'title'**. The category's name that will be displayed for users.
- **'weight'**. An integer specifying the order of the category.
- **'access callback'**. Name of the callback function that tests if the user can edit such category. By default the function **user_edit_access()** will be requested.
- **'access arguments'**. Arguments that will be passed on to the callback function defined under 'access callback'.

```
/***
 * Implements hook_user_categories().
 */
function hook_user_categories() {
  return array(array(
    'name' => 'account',
    'title' => t('Account settings'),
    'weight' => 1,
  ));
}
```

F49.8

hook_user_categories()

Returns a vector with available categories under the user's profile.

hook_user_operations()

The function **hook_user_operations()** allows us to add large operations on users. These operations are shown under the dropdown **Update options**, available under the user's Administration area. **F49.9**

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_operations/7

ROLES	MEMBER FOR	LAST ACCESS	OPERATIONS
administrator	8 hours 31 min	5 min 12 sec ago	edit
gowchedaco	13 hours 36 min	never	edit
stitrec	2 days 14 hours	never	edit

F49.9

hook_user_operations()

Allows to add massive large operations on users.

This function allows us to add personalized operations that will be applied on multiple users from the user's Administration area.

Returns a vector of operations (\$operations), in which each operation will be a vector with the following elements: [F49.10](#)

- 'label' (Required). The operation's label that will be shown under the operations' list.
- 'callback' (Required). The function's name that implements the operation.
- 'callback arguments'. Vector of additional arguments that will be carried on the designated function under 'callback'.

F49.10**hook_user_operations()**

Example of the function's implementation under the module Masquerade.

```
/***
 * Implements hook_user_operations().
 */
function masquerade_user_operations() {
  return array(
    'masquerade' => array(
      'label' => t('Masquerade as user'),
      'callback' => 'masquerade_user_operations_masquerade',
    ),
  );
}
```

hook_user_role_insert()

The function **hook_user_role_insert()** acts when a new role is created.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_role_insert/7

Make no mistake of this operation with the assignment of a user's roles. We'll use this function, for example, to adding additional fields to an existing role.

hook_user_role_insert(\$role)

The parameter **\$role** is a user's role object.

hook_user_role_update()

The function **hook_user_role_update()** can act when a role is updated.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_role_update/7

hook_user_role_delete()

The function **hook_user_role_delete()** allows to act when a role is deleted.

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_role_delete/7

hook_user_role_presave()

The function **hook_user_role_presave()** acts when a role is about to be saved in the database, wether is a new role (inser), or an update (update).

http://api.drupal.org/api/drupal/modules--user--user.api.php/function/hook_user_role_presave/7

Users implementation

49.3

In this paragraph we will apply the previous functions that allow to act on the users in the Website. For our purpose, we'll create a module named **Users Forcontu**, in which we will apply various functionalities.

Step 1. Creation of the module Users Forcontu

The module we'll create, and that will allow us to work with users will be named **Users Forcontu**. We'll create the folder named **users_forcontu** and the file **users_forcontu.info** with all the definitions corresponding to that module.

The file **users_forcontu.module** will contain the module's implementation and functions, that we'll develop throughout in the following steps.

Step 2. Creation of the user's information block

We will implement the functions **hook_block_info()** and **hook_block_view()** to creating a block with the user's data. We'll also apply for that matter what we learn from **Unit 48**, additionally to the implementation of the \$user object, and several other functions studied in this unit. **F49.11**

```
/***
 * Implements hook_block_info().
 */
function users_forcontu_block_info() {
  // User's records block
  $blocks['user_data'] = array(
    'info' => t('User Data'),
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}

/***
 * Implements hook_block_view().
 */
function users_forcontu_block_view($delta = '') {
  $block = array();
  switch ($delta) {
    case 'user_data':
      //we need to access the actual $user object
      global $user;
      $block['subject'] = t('User Data');

      if (user_is_logged_in()) {
        if ($user->picture){
          //loads the file object from the user's picture
          $picture_file = file_load($user->picture);
          //acquires the working URL form the URL in URI format
          $picture_url = file_create_url($picture_file->uri);
        }
        $output = '<div class="user_block_registered">';
        if ($user->picture) {
          $output .= '<div id="image">' . l('', 'user/' . $user->uid, array('html'=>TRUE)) . '</div>';
        }
        $output .= l($user->name, 'user/' . $user->uid);
        $output .= '</div>';
        $output .= '<div id="logout">' . l(t('Logout'), 'user/logout') . '</div>';
      }else{
        $output = '<div class="user_block_anonymous">';
        $output = '<p>You have to <a href="/user/login" title="Login">login</a> or <a href="/user/register" title="Register">register</a></p>';
      }
  }
}
```

F49.11

Users Forcontu Module

Development of the module Users Forcontu. The module defines a block with the user's information.

```

title="Registrarme">create a new account</a><p>';
    $output .= '</div>';
}
$block['content'] = $output;
break;
}
return $block;
}

```

In this example we have used two related functions with file management, that we'll broaden further in **Unit 53**.

The first one is **file_load()**, that loads a single file object from its identifier (fid). Recall that the attribute \$user -> picture stores the image's fid.

http://api.drupal.org/api/drupal/includes--file.inc/function/file_load/7

The file object contains an URL in URI format:

[uri] => public://pictures/user1picture.png

To create the URI into a complete web-accessible URL for the local image, we'll use the function **file_create_url()**.

http://api.drupal.org/api/drupal/includes--file.inc/function/file_create_url/7

In the next image the resulting block is shown, once the module has been installed and the block is activated. During the creation of the block we distinguished between a registered user, and an anonymous user only using the recalled function **user_is_logged_in()**. **F49.12** **F49.13**

F49.12

Block for registered users

Defined information for registered users in the Website is shown. The user's picture if any, username, and any possibility to logout.

sed. Capto oppeto plaga praemitto quidem. Adipiscing defui humo letalis
ea luptatum occuro pecus tum. Dolus euismod humo scisco similis usitas ut.
terdico nutus pecus quidne sagaciter tum utrum uxor vero ymo. Mos pala

is usitas. Adipiscing blandit genitus gilvus luptatum nostrud quae ratis
n singularis sit validus. Macto ulciscor ut valde.



F49.13

Block for non-registered users

Defined information for anonymous users is shown. Any possibility for login, or register.

Inhibeo

Decet nibh pertineo pneum premo sed. Capto oppeto plaga praemitto quidem. Adipiscing defui humo letalis
tation tego. Blandit cui damnum esca luptatum occuro pecus tum. Dolus euismod humo scisco similis usitas ut.
Decet macto nulla persto probo. Interdico nutus pecus quidne sagaciter tum utrum uxor vero ymo. Mos pala
voco.

Esse mauris meus pagus quis typicus usitas. Adipiscing blandit genitus gilvus luptatum nostrud quae ratis
tincidunt virtus. Ad camur luptatum singularis sit validus. Macto ulciscor ut valde.

User login

Username *

Password *

- Create new account
- Request new password

Consectetuer Si Vulpitate

Luctus pecus praesent rusticus si. Aliquam dignissim iaceo illum iniure nunc voco wis. Aliquip validus veror.
Consectetuer iustum jumentum valde. Aptent exerci quia rusticus valde. Abigo aliquip appellatio brevitatis
dolus pagus secundum tation valde. Paratus rusticus sino usitas valde vicis vulputate. Abdo accumsan autem
eum hos iaceo neo odio similis tincidunt.

User Data
You have to login or create a new account

Step 3. New fields under the user's profile

In this step we will modify the user's registration form, so that we introduce the following values: First name (required), Last name (required), Website (optional).

In Drupal 6, in order to add fields to the user's registration form we used **hook_user()** through the operation **\$op = 'register'**. In Drupal 7 there's no user hook for such operation, but a generic hook to modifying **hook_form_alter()** forms.

Therefore, we will implement the function **hook_form_alter()** and we will intercept the form with '**user_register_form**' id. **F49.14**

```
/***
 * Implements hook_form_alter.
 */
function users_forcontu_form_alter(&$form, &$form_state, $form_id) {
  switch($form_id) {
    case 'user_register_form': // user registration form
      $form['firstname'] = array(
        '#type' => 'textfield',
        '#title' => t('First name'),
        '#description' => t('First name'),
        '#required' => 1,
        '#lenght' => 255,
      );
      $form['lastname'] = array(
        '#type' => 'textfield',
        '#title' => t('Last name'),
        '#description' => t('Last name'),
        '#required' => 1,
        '#lenght' => 255,
      );
      $form['web'] = array(
        '#type' => 'textfield',
        '#title' => 'Web',
        '#description' => t('User\'s web'),
        '#required' => 0,
        '#lenght' => 255,
      );
      break;
  }
}
```

F49.14

Profile's fields

In this step we'll modify the user registration form to add additional fields.

Even though we haven't added every needed functionality to process and store the new fields, we can see the results by accessing the registration form as an anonymous user. **F49.15**

F49.15

Added fields to the user registration form

The fields will be shown on the registration form, or new user's account.

User account

[Create new account](#) [Log in](#) [Request new password](#)

Username *

 Spaces are allowed; punctuation is not allowed except for periods, hyphens, apostrophes, and underscores.

E-mail address *

 A valid e-mail address. All e-mails from the system will be sent to this address. The e-mail address is not made public and will only be used if you wish to receive a new password or wish to receive certain news or notifications by e-mail.

First name *

 First name

Last name *

 Last name

Web

 User's web

[Create new account](#)

Step 4. Storing additional information

We still haven't indicated where to store data. Both the **users** table and the **\$users** object provide a **data** field to keep additional information. This information will be stored inside a unique serialized text string, so we could find performance issues for large quantities of information, since access to this data has first to be string de-serialized back in **data**.

As an alternative we can create a new table to store additional information. In this Practical Case we'll take advantage of the **data** field, even though we have sufficient knowledge to apply an alternative solution, with an additional table.

To store the information sent inside the registration form, we can implement the functions **hook_user_insert()**, or **hook_user_presave()**.

As we look forward we'll include the new fields in the user's editing form, nonetheless we'll use the function **hook_user_presave()**, such function will be useful to store data while the user is being created as for to editing purposes.

Inside **hook_user_presave()**, the data sent by the user through the form is acquired in the parameter **\$edit**. The values under the fields 'name', 'surname' and 'web' will become available under **\$edit['firstname']**, **\$edit['lastname']** and **\$edit['web']**, respectively.

We will append to the **\$edit** vector the field 'data', with values from each of these fields. Thus, the system will be in charge to storing the serialized data in **\$data**. **F49.16**

```
/***
 * Implements hook_user_presave.
 */
function users_forcontu_user_presave(&$edit, $account, $category) {

  $edit['data'] = array(
    'firstname' => $edit['firstname'],
    'lastname' => $edit['lastname'],
    'web' => $edit['web'],
  );
}
```

Once the previous fields have been stored, they will become available in the **\$user** object.

F49.17

```
global $user;

$user->data['firstname'];
$user->data['lastname'];
$user->data['web'];
```

F49.16

Storing the user's information

Additional fields are added to the \$edit vector, so that the system will be in charge to serialize data, and storing them in the field data.

Step 5. User's account editing

The new fields have been added in the registration form, but not under the editing form on the user's account.

To adding the fields under the **user's editing form** we'll also use the function **hook_form_alter()**, taking into account that the form's id (\$form_id) is **'user_profile_form'**. **F49.18**

```
/***
 * Implements hook_form_alter.
 */
function users_forcontu_form_alter(&$form, &$form_state, $form_id) {

  switch($form_id) {
    case 'user_register_form': // user's registration form
      //previous code...
      break;
    case 'user_profile_form': // user's registration form
      // Acquires the user's information.
      $uid = (int)$form['#user']->uid;
      $account = user_load($uid);

      $form['firstname'] = array(
        '#type' => 'textfield',
        '#title' => t('First name'),
        '#description' => t('First name'),
        '#required' => 1,
        '#length' => 255,
        '#default_value' => (($account->data['firstname']) ?
$account->data['firstname'] : ''),
      );

      $form['lastname'] = array(
        '#type' => 'textfield',
        '#title' => t('Last name'),
        '#description' => t('Last name'),
        '#required' => 1,
        '#length' => 255,
        '#default_value' => (($account->data['lastname']) ?
$account->data['lastname'] : ''),
      );
}
```

F49.17

Fields in 'data'

The new fields will be available in \$user->data.

F49.18

Editing of the user's account

We modify the user's account editing form to adding additional fileds.

```
$form['web'] = array(
    '#type' => 'textfield',
    '#title' => 'Web',
    '#description' => t('User web'),
    '#required' => 0,
    '#length' => 255,
    '#default_value' => ($account->data['web']) ? $account-
>data['web'] : '',
);
break;
}
}
```

In this example we have reproduced the registration form code, but in contrast we have added the corresponding values inside the additional fields acquired from the user object. We have acquired the user object (the **\$account** variable) making use of the function **user_load()**. The value of \$uid is facilitated by the form (**\$form['#user']->uid**).

As we mentioned in the last step, the implemented function **hook_user_presave()** will be in charge of storing the form's data, both as in registration, as in the user's editing. Once the changes have been made, while editing the user's account the additional fields will be shown, and the performed modifications will be properly saved. **F49.19**

F49.19**User's editing page**

The user's editing form where it is visible that we have added three new fields.

fran.gil

View Edit

Account | Personal Data

Current password

Enter your current password to change the *E-mail address or Password*. [Request new password](#).

E-mail address *

A valid e-mail address. All e-mails from the system will be sent to this address. The e-mail address is not made public and will only be used if you wish to receive a new password or wish to receive certain news or notifications by e-mail.

Password

Password strength:

Confirm password

To change the current user password, enter the new password in both fields.

First name *

First name

Last name *

Last name

Web

User's web

Step 6. Displaying additional information in the user's page

By default the newly added fields in the user's account will not be shown at runtime (ie: user/7). Initially inside the user's profile page there's a **History** category (summary) that indicates the time frame the user has been registered in the Website.

Through the function **hook_user_categories()** we can establish new categories to group the fields that will be shown in the user's account. Next, we'll create the category **Personal data** (personal_data) in which the following fields we created previously will be shown: **First name**, **Last name** and **Web**. **F49.20**

```
/** 
 * Implements hook_user_categories.
 */
function users_forcontu_user_categories() {
  $category = array();
  $category = array(
    'name' => 'personal_data',
    'title' => 'Personal_data',
    'weight' => 4,
  );
  return $category;
}
```

F49.20

Categories' rendering

We define a new category where the added user's profile fields will be shown.

Once the new category is defined, we'll implement the function **hook_user_view()** to display the additional fields. **F49.21**

```
/** 
 * Implements hook_user_view.
 */
function users_forcontu_user_view($account, $view_mode, $langcode) {
  $account->content['personal_data'] = array();
  $account->content['personal_data'] = array(
    '#type' => 'user_profile_category',
    '#attributes' => array('class' => personal-data-user'),
    '#weight' => 4,
    '#title' => t('Personal data'),
  );
  $account->content['personal_data']['firstname'] = array(
    '#type' => 'user_profile_item',
    '#title' => t('First name'),
    '#markup' => $account->data['firstname'],
    '#attributes' => array('class' => 'firstname-user'),
    '#weight' => 1,
  );
  $account->content['personal_data']['lastname'] = array(
    '#type' => 'user_profile_item',
    '#title' => t('Last name'),
    '#markup' => $account->data['lastname'],
    '#attributes' => array('class' => 'lastname-user'),
    '#weight' => 2,
  );
  $account->content['personal_data']['web'] = array(
    '#type' => 'user_profile_item',
    '#title' => 'Web',
    '#markup' => $account->data['web'],
    '#attributes' => array('class' => 'user-web'),
    '#weight' => 3,
  );
}
```

F49.21

Displaying additional fields

Finally we display the additional fields in the user's profile implementing **hook_user_view()**.

The additional fields will have to be added to the **\$account -> content** vector, which is already defined. In this vector we define the fields' structure that will be sent to its corresponding template (.tpl.php), to rendering in HTML.

In **Unit 56** we will see how to work with templates through passing parameters, which is what we are advancing in this point. We indicate the category '**#type => user_profile_category**', for what we are referencing the template file **user_profile_category.tpl.php**. For the rest of the elements we reference the template file **user_profile_item.tpl.php**.

Both files are issued by the **user** core module and are available under **/modules/user**.

The templates allow a series of parameters that enable to write content and return a fragmented HTML formatted code, and ready to be written.

For now without going any further in the use of templates, we'll remain with the acquired results. **F49.22**

F49.22

Broaden user's information

By making use of `hook_user_view()` we can indicate the fields that will be shown in the user's page.

fran.gil

[View](#) [Edit](#)

Personal data

First name
Fran

Last name
Gil

Web
www.forcontu.com

History

Member for
2 days 14 hours

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Step 7. Modification of the user's display

The function **hook_user_view_alter()** allows us to modify the user's account elements before they are being shown. We'll be able to delete categories and elements, acting freely on the user's content vector **\$account -> content**.

The parameter **&\$build**, passed by reference, corresponds with the **\$user** object that we are modifying.

For example, **F49.23** if we want to delete the **History** category (summary) and its content, we can implement the function **hook_user_view_alter()** as follows:

```
/**
 * Implements hook_user_view_alter.
 */
function users_forcontu_user_view_alter(&$build) {
  unset($build['summary']);
}
```

F49.23

Modify the user's display

We modify the added values by other modules.

The resulting page after deleting the **History** category out from the user's profile page is shown below . **F49.24**

fran.gil

[View](#) [Edit](#)



Personal data

First name

Fran

Last name

Gil

Web

www.forcontu.com

F49.24

User's profile page

We show the resulting user's profile page, once we add the new fields and delete the history field.

49.4

Users access

We have still left to seeing some functions that will allow us to act during the login process, and during the user logout. These functions are:

- **hook_user_load()**. The \$user object is being loaded.
- **hook_user_login()**. The user has logged in successfully in the Website.
- **hook_user_logout()**. The user has logged out from the Website.

As we continue with the previous example, we'll implement on the Users Forcontu module a new functionality that will redirect every user with role set as '**intranet**' to a page using an alias URL to **frontpage_intranet**.

Step 1. Previous steps

As like previous steps we'll create a page with an alias URL to **frontpage_intranet**. Also we'll create a role named **intranet** (for now it will not be necessary to assign permissions). We'll assign such role to any user created in the Website.

Step 2. Login changes

In this step we'll add the role **intranet** to some users and redirect them to a URL **intranet_home** once they login. [F49.25](#)

F49.25**hook_user_login()**

Allows to act when the user access into the Website (login).

```
/** 
 * Implements hook_user_login.
 */
function users_forcontu_user_login(&$edit, $account) {
  if(is_array($account->roles) && in_array('intranet',
array_values($account->roles))) {
    $_GET['destination'] = 'intranet_home';
  }
}
```

Note that we have used **\$_GET['destination']** instead of the function **drupal_goto()**. The user's login process is carried out through a form with **\$form_id=='user_login'**, or **\$form_id=='user_login_block'**. For the redirection to take place once the form is processed, we'll use **\$_GET['destination']**.

On any other pages where any form is not activated directly, we can use **drupal_goto()**. Also is possible to act directly on the form (through the function **hook_form_alter()**) and modify the redirection's field form. In that same way we can redirect the user to a different page when she logs out, making use of the function **hook_user_logout()**.

Step 3. Verify operations

To verify the module's proper operation we'll have to access the Website with a user assigned role to **intranet**. Also test to access with a user without any assigned role, to verify the redirection does not take place.

Defining permissions

49.5

The way to create a module's permissions is through the implementation of **hook_permission()**. The operation is very simple, the function will simply return a vector with the newly created permissions by the module. The key for each vector's element will be the permission system's name. [F49.26](#)

In turn, each permission will be composed by the following fields:

- **'title'**. Permission's name. The function **t()** can be used so that the permission's name may be translated.
- **'description'** (optional). A description about the permission. Also the function **t()** can be used.
- **'restrict access'** (optional). By specifying the value to TRUE it will indicate the Website's administrators that the permission can only be assigned to trusted users.
- **'warning'** (optional). Allows to add a personalized warning message, that will overwrite the shown message by 'restrict access'.

```
/***
 * Implements hook_permission.
 */
function users_forcontu_permission() {
  return array(
    'access to intranet' => array(
      'title' => t('Access to intranet'),
      'description' => t('Access to intranet.'),
    ),
    'view personal data' => array(
      'title' => t('View personal data'),
      'description' => t('View personal data.'),
    ),
  );
}
```

[F49.26](#)

Defining permissions

By implementing the function **hook_permission()** we can define new permissions added by the module.

In the last example we have added two permissions: [F49.27](#)

- **Access to intranet.**
- **View personal data.**

Once such permissions are created, they will be available in the permissions' management area.

Administration ⇒ **People** ⇒ **Permissions**

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
Users Forcontu			
Access to intranet	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Access to intranet.			
View personal data	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
View personal data.			

[F49.27](#)

Permissions

Defined permissions defined by the Users Forcontu module

These permissions can be assigned to roles created by the Website, even though for now they will not be useful, since we haven't developed an utility that could depend on such permissions.

To check if a user is assigned a given permission we'll use the API function **user_access()**.

```
user_access($string, $account = NULL)
```

http://api.drupal.org/api/drupal/modules--user--user.module/function/user_access/7

The function **user_access()** gets the parameters:

- **\$string**, with the string that identifies the permission.
- **\$account**, with the user object to verify. If a value is not indicated, the validity will be performed on the actual logged in user in the Website.

During the implementation of **hook_user_login()** we verify if the user had an **intranet** role, to redirecting her in this case to the **intranet_home** page. Lets see an alternative verification, the role instead, if the user has been assigned the permission 'Access to intranet'. **F49.28**

F49.28

user_access()

The function **user_access()** determines if the user has the indicated permission.

```
/** 
 * Implements hook_user_login.
 */
function users_forcontu_user_login(&$edit, $account) {
  if (user_access('access to intranet')) {
    $_GET['destination'] = 'intranet_home';
  }
}
```

Although we are controlling the redirection this way, in effect we haven't protected the **intranet_home** page, it will be accesible by any user. We'll learn in the next Unit how to implement a module's specific content types, and how to protect their access to them.

50 Programming content types

We are up to one of the most important units of this course, and at the same time one of the most extensive across this level.

The node is without a doubt an fundamental element of the Drupal architecture, and within this unit we will analyze various functionalities related with them.

As we've seen previously, under Drupal 7 content types are entities, and as such, they can carry associated fields. During the next unit we will broaden the analysis of entities and fields, that will eventually will be useful for content types.

In this unit we will learn how to create new content types and how to control access to nodes. We have also included in this unit the creation of content filters, that we'll associate to the Website's text formats.

Ultimately we'll learn how to interact with content type nodes created by other modules.

Comparative D7/D6 Programming content types

Just like we learnt across previous units, hook functions within Drupal 6 that made operations through the \$op parameter, within Drupal 7 have been divided in independent functions.

We will analyze this new structure under content type's functions and filters.

The most important difference with respect to Drupal 6 lies in that nodes are now considered entities. In this regard, under Drupal 7 fields will be added through the API Field, that we will learn across unit 51.

Unit contents

50.1 Creation of content types	196
50.2 Node management	205
50.3 Node presentation	213
50.4 Access control	217
50.5 Content Filters	225
50.6 Treatment of nodes from other modules	238
50.7 Other features of content types	242



50.1 Creation of content types

Within the basic installation of Drupal we'll only find the content types **Page** (basic page) and **Article**. Nonetheless, by activating other core modules that come deactivated by default, new content types will appear, like **Book's Page** (book), **Forum Theme** (forum), etc. We have also studied during beginner and intermediate levels other contributed modules that define and add other content types: Webform, Ubercart, etc.

As we recall, under Drupal 7 content types are entities, and as such, can carry associated fields. Until the next unit we will broaden the analysis of entities and fields, that will be applied on all content types as well.

We will learn how to create content types by programming of a module, so that once installed and activated, the new content type will be available and ready for production across the Website. We will develop the module **Nodes Forcontu** (**nodes_forcontu**) towards the application of illustrations and activities in this unit.

We will start with the analysis of functions from the API (Node API) that enables the new definition of a content type. We have access to a wider listing of functions from the API under:

http://api.drupal.org/api/drupal/modules--node--node.api.php/group/node_api_hooks/7

hook_node_info()

The creation of a content type begins with the implementation of **hook_node_info()**. Throughout this function a module can define one or more content types. In this function we will only define its name and its attributes for the content type.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_info/7

In the next fragment of code we introduce, as an illustration, the function **forum_node_info()**, which is the implementation of **hook_node_info()** in part by Drupal's core module **Forum**. This module renders the content type Forum topic (converted as "Forum Topic"). **F50.1**

F50.1

hook_node_info()

By implementing **hook_node_info()** we define new content types. This illustration shows the implementation type Forum topic from the Forum module.

```
/**
 * Implements hook_node_info().
 */
function forum_node_info() {
  return array(
    'forum' => array(
      'name' => t('Forum topic'),
      'base' => 'forum',
      'description' => t('A <em>forum topic</em> starts a new discussion thread within a forum.'),
      'title_label' => t('Subject'),
    )
  );
}
```

The function **hook_node_info()** returns a structured array, in which the first key will be the inherent name, or the content type's rule. In the previous illustration, the content type's rule name is "forum", while the visible name for all users is "Forum topic". The rule's name can only be structured by letters, numbers, and under scores.

The possible attributes of a content type are:

- **'name'**. Content type's name, just as rendered for users. Can contain spaces, uppercase, lowercase, etc. Must use the function t() to allow for translation. It is a **required** field.
- **'base'**. Name base which is used as a prefix to generate names for return function. Generally it corresponds to the name of the module. It is a **required** field.
- **'description'**. Description of the content type. Must use the function t() to allow for translation. It is a **required** field.
- **'help'**. Help text to display in the form when creating or editing the content type. It is an **optional** field, with the default value being an empty string ("");
- **'has_title'**. Boolean field type that indicates if the content type has a title (TRUE) or not (FALSE). It is an **optional** field and its default value is TRUE.
- **'title_label'**. Name or tag of the title field. By default the value is 'Title', but we can change it to any other value. It is an **optional** field.
- **'locked'**. Boolean field type that indicates if the internal name of the content type is able to be modified by the site administrator. It is an **optional** field and its default is TRUE.

All of these attributes, except 'locked', can be modified by the site administrator. The internal name of the content type can only be changed if the attribute 'locked' has a value of FALSE.

[**hook_form\(\)**](#)

The **hook_form()** function is responsible for describing the form used when creating or editing a node of the content type previously defined with **hook_node_info()**.

The form is created in the URLs:

- **/node/add/[content_type]**, when creating a new node.
- **/node/[nid]/edit**, when editing a node with the node ID [nid].

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_form/7

The **hook_form()** function has the following parameters:

hook_form(\$node, &\$form_state)

The **\$node** parameter, is the **\$node** object being created or edited. The **\$form_state** parameter, passed for reference, is an array containing the current state of the form (default values and/or previous values, in the case when the node is being edited).

The function returns an array with the structure of the form to be displayed when editing the node. It is important to note that it is not necessary to define all the form elements such as submit and preview, configuration options added by other modules and the additional fields of the content type are controlled by the node module and displayed automatically.

The form is defined following the steps studied in **Unit 47**. Consider the following example: **F50.2**

F50.2**hook_form()**

We studied the definition of the form in Unit 47. Here we apply all the formulas studied in that unit.

```
/***
 * Implements hook_form().
 */
function example_form($node, &$form_state) {
  $type = node_type_get_type($node);

  $form['title'] = array(
    '#type' => 'textfield',
    '#title' => check_plain($type->title_label),
    '#default_value' => !empty($node->title) ? $node->title : '',
    '#required' => TRUE,
    '#weight' => -5,
  );

  $form['field1'] = array(
    '#type' => 'textfield',
    '#title' => t('Custom field'),
    '#default_value' => $node->field1,
    '#maxlength' => 127,
  );

  $form['quantity'] = array(
    '#type' => 'textfield',
    '#title' => t('Quantity'),
    '#default_value' => isset($node->quantity) ? $node->quantity : 0,
    '#size' => 10,
    '#maxlength' => 10,
  );

  $form['color'] = array(
    '#type' => 'select',
    '#title' => t('Color'),
    '#default_value' => $node->color,
    '#options' => array(
      1 => 'Red',
      2 => 'Green',
      3 => 'Blue',
    ),
    '#description' => t('Choose a color.'),
  );

  return $form;
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

The API function **node_type_get_type()** returns the definition of the content type, according to the **\$node** parameter, which can be a node or the name of the content type.

Thus, we begin the function by storing in the variable **\$type** the content type definition performed by calling **hook_node_info()**.

This will help us, for example, to check if the '**has_title**' attribute is TRUE or FALSE, and to know whether or not we need to create the title on the form. The example uses the value **\$type->title_label** directly as a label for the field 'title'.

In addition to the 'title' field, the addition fields 'field1' (textfield type), 'quantity' (textfield type) and 'color' (select type) are defined in the example.

hook_validate()

The content type fields defined in the **hook_form()** function can be validated through the **hook_validate()** function. This function is responsible for checking if the values entered into the form when creating or editing it are correct. We will report any errors found via the Form API **form_set_error()** function.

http://api.drupal.org/api/drupal/modules--node-node.api.php/function/hook_validate/7

hook_validate(\$node, \$form, &\$form_state)

The parameters for this function are: F50.3

- **\$node**. The Node that we are validating.
- **\$form**. Form definition that is being used when editing the Node.
- **\$form_state**. Current state of the form.

```
/***
 * Implements hook_validate().
 */
function example_validate($node, $form, &$form_state) {
  //Checks to see if the field quantity has a numeric value
  if ($node->quantity) {
    if (!is_numeric($node->quantity)) {
      form_set_error('quantity', t('The quantity must be a number.'));
    }
  }
}
```

F50.3

hook_validate()

Validates the field via the hook_form() function.

In the example the value in the field 'quantity' is entered as a text. In the function example_validate we check to see if the value entered is a valid number.

The **hook_validate()** function only works for the content types defined by the module. If we need to provide validation on content types provided by other modules, we can use the **hook_node_validate()** function.

http://api.drupal.org/api/drupal/modules--node-node.api.php/function/hook_node_validate/7

Add a Body field

In Drupal 7 the **Body** field is no longer present in the content type definition; it is handled by the Field API.

The **Node** module facilitates the **node_add_body_field()** function, which adds a Body field to the content type. Internally this function uses the API functions for entities and field, which we will study in **Unit 51**.

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_add_body_field/7

node_add_body_field(\$type, \$label = 'Body')

We must pass the content type (**\$type**) object and , optionally, the field label (**\$label**), which defaults to 'Body'.

The new field should be added to the content type during installation or activation of the module, implementing the **hook_install()** or **hook_enable()** functions, respectively.

The example illustrates the implementation of the **hook_install()** function for the Blog module, which adds a content type of 'blog', which creates for this same module, a Body field. **F50.4**

F50.4

Adding a Body field

Adding a Body field when installing the module.

```
/** 
 * Implements hook_install().
 */
function blog_install() {
  // Ensure the blog node type is available.
  node_types_rebuild();
  $types = node_type_get_types();
  node_add_body_field($types['blog']);
}
```

The **node_types_rebuild()** function updates the content types, ensuring that the content type is added during the installation of the module and is available on the site.

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_types_rebuild/7

The **node_type_get_types()** function returns an array with all the content types available.

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_type_get_types/7

Finally the **node_add_body_field()** function will be responsible for adding the **Body** field in the 'blog' content type.

Practical Case 50.1 Create a content type

In this practical case we will create a module called **Nodes Forcontu** (`nodes_forcontu`), where we will define, for now, a content type called **My product** (`myproduct`) with various attributes related to products for a virtual store.

Step 1. Creation of the module Nodes Forcontu

As a first step we will create the module folder (`nodes_forcontu`) and the key files (`nodes_forcontu.info` and `nodes_forcontu.module`).

Step 2. Creating the content type My Product

Creation of the content type is done by implementing `hook_node_info()`, creating the function `nodes_forcontu_node_info()`. F50.5

```
/***
 * Implements hook_node_info().
 */
function nodes_forcontu_node_info() {
  return array(
    'myproduct' => array(
      'name' => t('My product'),
      'base' => 'nodes_forcontu',
      'description' => t('Example node type.'),
      'title_label' => t('Product name'),
    )
  );
}
```

F50.5

Practical Case 50.1 hook_node_info()

Defines the content type My product.

Once the module is enabled, you can access the creation of the content type **My product** from:

Administration ⇒ **Content** ⇒ **Add content** ⇒ **My product**

URL Creates My product
`/node/add/myproduct`

You can also edit the content type **My product** from:

Administration ⇒ **Structure** ⇒ **Content Types** ⇒ **My product**

Step 3. Add the Body field

To add a **Body field** we implement the `hook_install()` function, where we call the `node_add_body_field()` function. We create the `nodes_forcontu.install` folder through it. F50.6

```
/***
 * Implements hook_install().
 */
function nodes_forcontu_install() {
  node_types_rebuild();
  $types = node_type_get_types();
  node_add_body_field($types['myproduct'], 'Description');
}
```

F50.6

Practical Case 50.1 Body Field

Adding a Body field during module installation.

The **Body** field will set the label to **Description**.

Step 4. Defining the content type Form

We will now define the content type form for **My product** with the following fields grouped in a **fieldset** which we have called **Product Information**, and which we show as a **vertical tab**: **F50.7**

- **Price.** Price of the product in Euros.
- **Weight.** Weight of the product, in Kgs.
- **Dimensions.** Group of fields with the dimensions of the product once packaged. It contains the fields: Length, Width and Height.

F50.7

Practical case 50.1 hook_form()

Creates the form for the content type My product. We define the fields Price, Weight and Dimensions (Length, Width and Height).

```
/***
 * Implements hook_form().
 */
function nodes_forcontu_form($node, &$form_state) {
  $type = node_type_get_type($node);

  if ($type->has_title) {
    $form['title'] = array(
      '#type' => 'textfield',
      '#title' => check_plain($type->title_label),
      '#required' => TRUE,
      '#default_value' => !empty($node->title) ? $node->title : '',
      '#maxlength' => 255,
      '#weight' => -5
    );
  }

  $form['information'] = array(
    '#type' => 'fieldset',
    '#title' => t('Product information'),
    '#collapsible' => TRUE,
    '#collapsed' => FALSE,
    '#weight' => 99,
    '#group' => 'additional_settings'
  );

  $form['information']['price'] = array(
    '#type' => 'textfield',
    '#title' => t('Price'),
    '#required' => FALSE,
    '#default_value' => isset($node->price) ? $node->price : 0,
    '#description' => t('Product's price (Euro).'),
    '#weight' => 1,
    '#size' => 20,
    '#maxlength' => 35,
  );

  $form['information']['weight'] = array(
    '#type' => 'textfield',
    '#title' => t('Weight'),
    '#default_value' => isset($node->weight) ? $node->weight : 0,
    '#description' => t('Product's weight (Kg).'),
    '#weight' => 2,
    '#size' => 10,
    '#maxlength' => 15,
  );

  $form['information']['dimensions'] = array(
    '#type' => 'fieldset',
    '#title' => t('Dimensions'),
    '#description' => t('Physical dimensions of the packaged product (cm).'),
    '#weight' => 3,
  );
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

```

$form['information']['dimensions']['length'] = array(
  '#type' => 'textfield',
  '#title' => t('Length'),
  '#default_value' => isset($node->length) ? $node->length : '',
  '#weight' => 1,
  '#size' => 10,
);

$form['information']['dimensions']['width'] = array(
  '#type' => 'textfield',
  '#title' => t('Width'),
  '#default_value' => isset($node->width) ? $node->width : '',
  '#weight' => 2,
  '#size' => 10,
);

$form['information']['dimensions']['height'] = array(
  '#type' => 'textfield',
  '#title' => t('Height'),
  '#default_value' => isset($node->height) ? $node->height : '',
  '#weight' => 3,
  '#size' => 10,
);
return $form;
}

```

The resulting form for creating or editing a **My product** node is shown in the **Figure F50.8**. In addition to the specific fields defined by `hook_form()`, other configuration options added by the system or other contributed modules on the site are included in the form.

Home > Add content
Create My product

Product name *

Description (Edit summary)

Text format Filtered HTML ▾ More information about text formats ⓘ

- Web page addresses and e-mail addresses turn into links automatically.
- Allowed HTML tags: <a> <cite> <blockquote> <code> <dl> <dt> <dd>
- Lines and paragraphs break automatically.

Menu settings Not in menu

Revision information No revision

URL path settings No alias

Comment settings Open

Authoring information By admin

Publishing options Published, Promoted to front page

Product information

Price 0 Product's price (Euro).

Weight 0 Product's weight (Kg).

DIMENSIONS Physical dimensions of the packaged product (cm).

Length

Width

Height

Save Preview

F50.8

Practical case 50.1 Form for the Content Type

Within the Product information tab the specific fields for the content type My product are shown.

The existing vertical tabs are identified by the '**additional_settings**' tag. To display a new group of fields as an additional verticle flange we add the property '**#group' => 'additional_settings'** and indicate the order using the '**#weight**' property.

Step 5. Form Validation

We will add validation to the content type fields to verify that price, weight and dimensions (length, width and height) have positive numeric values. We will do this by implementing the **hook_validate()** function. **F50.9**

F50.9

Practical Case 50.1

hook_validate()

Implements validation on the newly added fields.

```
/***
 * Implements hook_validate().
 */
function nodes_forcontu_validate($node, $form, &$amp;form_state) {
  $pattern = '/^\d*(\.\d*)?$/';

  if (!empty($node->price) && !is_numeric($node->price) &&
      preg_match($pattern, $node->price)) {
    form_set_error('price', t('Price must be in a valid number
format. No commas and only one decimal point.'));
  }

  if (!empty($node->weight) && (!is_numeric($node->weight) ||
      $node->weight < 0)) {
    form_set_error('weight', t('Weight must be a positive
number.'));
  }

  if (!empty($node->length) && (!is_numeric($node->length) ||
      $node->length < 0)) {
    form_set_error('length', t('Length must be a positive
number.'));
  }

  if (!empty($node->width) && (!is_numeric($node->width) ||
      $node->width < 0)) {
    form_set_error('width', t('Width must be a positive
number.'));
  }

  if (!empty($node->height) && (!is_numeric($node->height) ||
      $node->height < 0)) {
    form_set_error('height', t('Height must be a positive
number.'));
  }
}
```

As we have not included the code to save the additonal fields, if we create a node of the **My product** type, only the fields common to all nodes will be stored. The aspects related to the management of the nodes will be studied in the next section.

Node Management

50.2

In the previous section we analyzed how to create a content type, including its form and validations. What we have not done yet is to keep the added fields as part of the content type.

The \$node object

Since we will start working directly with the **\$node object**, this is a good time to study its initial fields. These fields will be expanded depending on the modules installed, as we have seen, which can add new fields or attributes to the **\$node** object. **F50.10**

```
$node = (
  [vid] => [1]
  [uid] => [1]
  [title] => [Article 1]
  [log] => []
  [status] => [1]
  [comment] => [2]
  [promote] => [1]
  [sticky] => [0]
  [nid] => [1]
  [type] => [article]
  [language] => [es]
  [created] => [1326276795]
  [changed] => [1326637468]
  [tnid] => [0]
  [translate] => [0]
  [new_field] => [0]
  [revision_timestamp] => [1326637468]
  [revision_uid] => [1]
  [body] => array (
    [und] => array (
      [0] => array (
        [value] => [<b>Lorem ipsum dolor sit amet</b>.]
        [summary] => []
        [format] => [full_html]
        [safe_value] => [<p><b>Lorem ipsum dolor sit amet</b>.</p>]
        [safe_summary] => []
      )
    )
  )
  [field_tags] => array ()
  [field_image] => array ()
  [rdf_mapping] => array ()
  [cid] => [0]
  [last_comment_timestamp] => [1326276795]
  [last_comment_name] => []
  [last_comment_uid] => [1]
  [comment_count] => [0]
  [name] => [admin]
  [picture] => [7]
  [data] => [a:1:{s:17:'mimemail_textonly';i:0;}]
  [menu] => Array ()
);

```

F50.10

\$node Object

Example of the structure of the \$node object.

The fields will change based on the modules installed.

The following table shows each attribute of the \$node object, its description and the reference to the table where the value is stored. **F50.11**

F50.11	Attribute	Description	Table (field)
	<code>nid</code>	ID of the node.	node
	<code>vid</code>	IS the node revision ID	node, node_revisions
	<code>type</code>	Node content type. The internal name or the system name of the node type: page, article, forum, book, etc.	node
	<code>language</code>	Language of the node: es, en, etc.	node
	<code>title</code>	Title of the node.	node, node_revisions
	<code>uid</code>	Id of the user who created the node.	node
	<code>status</code>	State of the node (or of the revision): published (1) or not published (0)	node, node_revisions
	<code>created</code>	Date (in UNIX timestamp format) when the node was created.	node
	<code>changed</code>	Date (in UNIX timestamp format) when the node was last modified.	node
	<code>comment</code>	Indicates the type of comment configuration selected for posting comments on the node (or revision): no comments (0), read-only (1), read and write (2).	node, node_revisions
	<code>promote</code>	Promote node to the front page (0 1).	node, node_revisions
	<code>sticky</code>	The node is shown at the top of the list (0 1).	node, node_revisions
	<code>tnid</code>	Id of the node which contains the source of the translation, if the node is translated	node
	<code>translate</code>	Indicates if the node has a translation (0 1).	node
	<code>log</code>	Message log stored by the user who created the current version of the node, usually explains changes over the previous version.	node_revisions
	<code>revision_timestamp</code>	Date (In UNIX timestamp format) that the current revision was created.	node_revisions (timestamp)
	<code>revision_uid</code>	ID of the user who created the revision.	node_revisions (uid)
	<code>name</code>	Name of the user who created the node.	users
	<code>picture</code>	Picture of the user who created the node.	users
	<code>data</code>	Open field for additional data about the user who created the node.	users
	<code>cid</code>	ID of the last comment posted.	node_comment_statistics
	<code>last_comment_timestamp</code>	Date (In UNIX timestamp format) when the last comment was created.	node_comment_statistics
	<code>last_comment_name</code>	Name of the user who created the last comment.	node_comment_statistics
	<code>last_comment_uid</code>	ID of the user who created the last comment.	node_comment_statistics
	<code>comment_count</code>	Number of comments associated with the node.	node_comment_statistics
	<code>menu</code>	Information to link the menu associated with the node.	menu_links, menu_router

hook_insert()

We store the fields we added to the content type through the **hook_form()** function, the module will need to build your own tables, using the **hook_install()** function and then perform insertions and retrieval of the information when needed.

The **hook_insert()** function allows you to act when a new node (of any of the types defined in the module) is created. We use the **hook_insert()** function to store the additional fields created with the node. The function passes only the parameter to the **\$node** object, with the values of the created node. **F50.12**

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_insert/7

```
/** 
 * Implements hook_insert().
 */
function example_insert ($node) {
  db_insert('mytable')
    ->fields(array(
      'nid' => $node->nid,
      'extra' => $node->extra,
    ))
    ->execute();
}
```

The **hook_insert()** function should only be invoked by the module while defining the content type. If we need to perform operations of inserting into a node of a different content type, we need to use the **hook_node_insert()** function.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_insert/7

hook_update()

The **hook_update()** function will be used by the module to store the node data when the node is updated. **F50.13**

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_update/7

```
/** 
 * Implements hook_update().
 */
function example_update ($node) {
  db_update('mytable')
    ->fields(array('extra' => $node->extra))
    ->condition('nid', $node->nid)
    ->execute();
}
```

The **hook_update()** function should only be invoked by the module that defines the content type. If we need to perform operations on updating a node of a different content type, you must implement the **hook_node_update()** function.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_update/7

Note

Implementing various types of content

If a module defines more than one content type, we have to distinguish, within each function, operations to be performed according to the content type.

We can do it directly at the **\$node->type** attribute.

F50.12

hook_insert()

Using **hook_insert()** we insert into the database the new fields created for the node.

F50.13

hook_update()

Through this function the fields are updated in the database.

hook_delete()

When a node is removed, the module will have to decide whether or not to dispose of (additional) data from the database. The **hook_delete()** function is intended to include the necessary decisions to remove additional data related to the node being deleted.

F50.14

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_delete/7

F50.14

hook_delete()

Deletes records from the database.

```
/** 
 * Implements hook_delete(). 
 */
function example_delete ($node) {
  db_delete ('mytable')
    ->condition ('nid', $node->nid)
    ->execute ();
}
```

Note

Parameters by reference

In the **hook_load()** function we consider that the **\$nodes** parameter is passed by reference, but not specifically **&\$nodes** in the function declaration.

This is because in PHP5, an array of objects does not contain objects but **links or pointers** to objects in memory. Therefore, any changes we make in **\$nodes** will be available outside the function, and this is why we do not need to return the array **\$nodes**.

The **hook_delete()** function should only be called by the module that created the content type. If you need to remove data from other content types you must use the **hook_node_delete()** function.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_delete/7

hook_load()

We have seen how to insert, update or delete the additional information of a node. Now we need to know how to retrieve it so that it is available in the **\$node** object. Implementing the **hook_load()** function, provides this additional node information.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_load/7

The **hook_load()** function requires as an input parameter **\$nodes**, which is an array of objects of the node type with the node or nodes that are being loaded.

The **hook_load()** function should only be used to assign values to additional fields added by the module, and not to change other parameters of the node. We add to **\$nodes** the referenced object values of the additional fields.

F50.15

F50.15

hook_load()

Retrieves the node information from the database.

```
/** 
 * Implements hook_load(). 
 */
function example_load ($nodes) {
  $result = db_query ("SELECT nid, foo FROM {mytable} WHERE nid IN (:nids)", array (':nids' => array_keys ($nodes)));
  foreach ($result as $record) {
    $nodes[$record->nid]->foo = $record->foo;
  }
}
```

The **hook_load()** function should only be invoked by the module that defines the content type. If we need to perform operations when loading nodes on nodes of a different content type , we must implement the **hook_node_load()** function.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_load/7

hook_prepare()

The **hook_prepare()** function allows acting on a node object about to be shown in creating or editing form of the node. The function only requires the \$node parameter, with the node to be displayed.

hook_prepare(\$node)

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_prepare/7

The **hook_prepare()** function should only be invoked by the module that created the content type. If we need to perform operations before creating or editing a node in another content type, you should implement the **hook_node_prepare()** function.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_prepare/7

Practical Case 50.2 Managing Nodes

We continue to develop the functionality of the **Nodes Forcontu** module, created in **Practical Case 50.1**. Now that we know what functions to use to manage the nodes for the **My product** content type, we can implement the insert, update, delete and load functions.

Step 1. Create the database schema

As a first step we will create the **nodes_forcontu** table that will store the additional data for nodes of the 'myproduct' type. To define the schema of the database we implement the hook_schema() function in the module installation file (**nodes_forcontu.install**).

Note that if you already have installed the module you must completely uninstall and re-install it for the table to be created. [F50.16](#)

```
/***
 * Implements hook_schema().
 */
function nodes_forcontu_schema() {
  $schema['nodes_forcontu'] = array(
    'description' => 'Save My Product content type.',
    'fields' => array(
      'nid' => array(
        'type' => 'int',
        'not null' => TRUE,
        'description' => 'Primary Key: nid.',
      ),
      'price' => array(
        'type' => 'int',
        'not null' => TRUE,
        'default' => 0,
        'description' => "Price",
      ),
      'weight' => array(
        'type' => 'int',
        'not null' => TRUE,
        'default' => 0,
        'description' => "Weight",
      ),
    ),
  );
}
```

F50.16

Practical Case 50.2 hook_schema()

Definition of the module table through the hook_schema() function. The table is created during the installation of the module.

```

'length' => array(
  'type' => 'int',
  'not null' => TRUE,
  'default' => 0,
  'description' => "Length",
),
'width' => array(
  'type' => 'int',
  'not null' => TRUE,
  'default' => 0,
  'description' => "Width",
),
'height' => array(
  'type' => 'int',
  'not null' => TRUE,
  'default' => 0,
  'description' => "Height",
),
),
'primary key' => array('nid'),
);
return $schema;
}

```

We have created the nodes_forcontu table, which stores the nid, price, weight, length, width and height values.

It is important, in general, that the values should be associated not only to the node ID (nid), but also to the revisión ID (vid). In this case study and for simplicity, we will only use the value of the node (nid).

Step 2. Implementation of hook_insert()

From the **nodes_forcontu_insert()** function, which implements **hook_insert()**, we insert into the database the values of the additional fields. This function will only be executed when a new node of the type **My product is** created. **F50.17**

F50.17

Practical Case 50.2

hook_insert()

This hook implements the insertion of the new fields for the content type into the table defined by the module.

```

/**
 * Implements hook_insert().
 */
function nodes_forcontu_insert($node) {
  db_insert('nodes_forcontu')
    ->fields(array(
      'nid' => $node->nid,
      'price' => $node->price,
      'weight' => $node->weight,
      'length' => $node->length,
      'width' => $node->width,
      'height' => $node->height,
    ))
    ->execute();
}

```

Step 3. Implementation of hook_update()

Through the **nodes_forcontu_update()** function, which implements **hook_update()**, additional fields will be updated when you modify a node of the **My product** type. **F50.18**

```
/***
 * Implements hook_update() .
 */
function nodes_forcontu_update($node) {

  db_update('nodes_forcontu')
    ->fields(array(
      'price' => $node->price,
      'weight' => $node->weight,
      'length' => $node->length,
      'width' => $node->width,
      'height' => $node->height,
    ))
    ->condition('nid', $node->nid)
    ->execute();
}
```

F50.18

Practical Case 50.2 hook_update()

This hook implements the update of existing records in the table.

Step 4. Implementation of hook_delete()

Through the **nodes_forcontu_delete()** function, which implements **hook_delete()**, we delete the additional data for the node when the node is deleted. **F50.19**

```
/***
 * Implements hook_delete() .
 */
function nodes_forcontu_delete($node) {

  db_delete('nodes_forcontu')
    ->condition('nid', $node->nid)
    ->execute();
}
```

F50.19

Practical Case 50.2 hook_delete()

This hook deletes the values in the table for the node.

Step 5. Implementation of hook_load()

Through the **nodes_forcontu_load()** function, which implements **hook_load()**, we add, during the loading of the node, the additional fields. **F50.20**

```
/***
 * Implements hook_load() .
 */
function nodes_forcontu_load($nodes) {

  $result = db_query('SELECT nid, price, weight, length, width,
height FROM {nodes_forcontu} WHERE nid IN (:nids)', array(':nids' => array_keys($nodes)));
  foreach ($result as $record) {
    $nodes[$record->nid]->price = $record->price;
    $nodes[$record->nid]->weight = $record->weight;
    $nodes[$record->nid]->length = $record->length;
    $nodes[$record->nid]->width = $record->width;
    $nodes[$record->nid]->height = $record->height;
  }
}
```

F50.20

Practical Case 50.2 hook_load()

This hook gets the value of the fields for the content type.

To check the operation of these functions, we can perform the following operations:

- **Reinstall the module.**
- Access the database (via phpMyAdmin) and check that you have successfully created the **nodes_forcontu** table.
- **Create a node** of the **My product** content type, completing the additional fields with the product information.
- Check that the data has been inserted properly into the **nodes_forcontu** table.
- **Edit the node** and check that the additional fields have previously saved values.
- From the editing form, **modify the additional fields**, save the changes.
- Check that the data has been modified correctly in the **nodes_forcontu** table.
- **Delete the node.**
- Check that the record corresponding to the node has been removed from the **nodes_forcontu**.

Node Presentation

50.3

hook_view()

If we try to display a node of the **My product** type, defined in this unit, we will not see more than the generic node fields (title, body), and not the defined additional fields (Price, weight and dimensions). If you edit the node, however, you will see that the additional data have been successfully saved, thanks to the implementation of the **hook_insert()** and **hook_update()** functions.

The **hook_view()** function allows these additional fields to be displayed.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_view/7

The function requires the following input parameters:

hook_view(\$node, \$view_mode)

- **\$node**. Node to display.
- **\$view_mode**. Presentation mode ('full', 'teaser', etc.).

The return value is the **\$node** object itself, but this time modified and prepared for presentation. And how is the node prepared for presentation? Very simple, by acting on the **\$node->content** array.

The **\$node->content** array has a structure similar to the forms, as defined by the Form API. By modifying the values of the fields, we will be showing, how we want them displayed, in what order and with what content and attributes (for example, classes to define styles). **F50.21**

```
/** 
 * Implements hook_view().
 */
function example_view($node, $view_mode) {
  $node->content['color'] = array(
    '#markup' => theme('example_color', $node->color),
    '#weight' => 1,
  );
  return $node;
}
```

F50.21

hook_view()

Presents the additional fields defined in the new content type.

theme() and hook_theme() functions

In the example of implementing the **hook_view()** function we used a call to the **theme()** function, to assign the value to an additional element of the node.

The **theme()** function, which we will see in more detail in **Unit 56**, can apply the theme to an item. To apply the theme to an item, the **theme()** function returns the HTML code for presenting the element. The HTML code will vary depending on the theme of the site and the defined templates, etc..

<http://api.drupal.org/api/drupal/includes--theme.inc/function/theme/7>

```
theme($hook, $variables = array())
```

The **theme()** function utilizes the following variables:

- **\$hook**. Name of the theme hook function, which will be used in the template. For example, if the value is '**link**', it will use the theme function **hook_link()**, which is the function of the API for generating the HTML to output a link. In our example, we have indicated the '**example_color**' value, so we can implement the **theme_example_color()** function, which will return the output for this field.
- **\$variables**. Array with variables passed to the theme function.

Implementing **hook_theme()** defines the templates to be used for each element.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_theme/7

The following is an example of implementing the **hook_theme()** function, which defines the '**example_color**' template, to be used for the '**color**' element. The template function will receive the variable '**color**'. **F50.22**

F50.22

hook_theme()

We define templates and variables passed to each of them.

```
/***
 * Implements hook_theme().
 */
function example_theme($existing, $type, $theme, $path) {
  return array(
    'example_color' => array(
      'variables' => array('color' => NULL),
    ),
  );
}
```

The template '**example_color**' is defined by the function with the prefix '**theme_**', which returns the HTML element. **F50.23**

F50.23

Custom theme function

The presentation function receives the variables defined in the template.

```
/***
 * Custom theme function.
 */
function theme_example_color($variables) {
  $output = '<span style='color: ' . $variables['color'] . ''>' .
  $variables['color'] . '</span>';
  return $output;
}
```

If the '**color**' element has the value of '**red**', the HTML returned by the function template will be:

```
<span style='color: red'>red</span>
```

Thus the **hook_view()** function is basically a form structure where the '#markup' attribute of each element is a call to the **theme()** function, via the function of the specified template, to obtain the formatted HTML snippet.

Practical Case 50.3 Node Presentation

We continue to develop the functionality of the **Nodes Forcontu** module (Practical cases 50.1 and 50.2). We will apply the functions studied in this section to display additional fields on the node of the **My Product** type.

Step 1. Implementing hook_view()

In **hook_view()** we define template functions to be used when displaying additional fields. We will create three functions: **F50.24**

- **hook_nodes_forcontu_price()**, to show the field 'price'.
- **hook_nodes_forcontu_weight()**, to show the field 'weight'.
- **hook_nodes_forcontu_dimensions()**, to jointly display the fields 'length', 'width' and 'height'.

```
/***
 * Implements hook_view().
 */
function nodes_forcontu_view($node, $view_mode) {
  $node->content['price'] = array(
    '#markup' => theme('nodes_forcontu_price', $node->price),
    '#weight' => 1,
  );
  $node->content['weight'] = array(
    '#markup' => theme('nodes_forcontu_weight', $node->weight),
    '#weight' => 2,
  );
  $node->content['dimensions'] = array(
    '#markup' => theme('nodes_forcontu_dimensions',
      array('length' => $node->length,
            'width' => $node->width,
            'height' => $node->height)),
    '#weight' => 3,
  );
  return $node;
}
```

F50.24

Practical Case 50.3 hook_view()

This hook implements the presentation of the additional fields of the content type.

Step 2. Implementation of hook_theme()

The implementation of the **hook_theme()** function defines template functions and arguments to be passed to each of them: **F50.25**

```
/***
 * Implements hook_theme().
 */
function nodes_forcontu_theme($existing, $type, $theme, $path) {
  return array(
    'nodes_forcontu_price' => array(
      'variables' => array('price' => 0),
    ),
    'nodes_forcontu_weight' => array(
      'variables' => array('weight' => 0),
    ),
    'nodes_forcontu_dimensions' => array(
      'variables' => array('length' => 0, 'width' => 0, 'height' => 0),
    ),
  );
}
```

F50.25

Practical Case 50.3 hook_theme()

We define template functions for the price, weight and dimensions fields.

Step 3. Implementations of template functions

Finally we define each of the functions to create the final HTML that the template will return: **F50.26**

F50.26

Practical Case 50.3

Template Functions

Implements each of the functions of the template.

```
/*
 * Returns the HTML output for the item Price.
 *
 * @param $variables
 *   An associative array containing:
 *   - price (€).
 *
 * @ingroup themeable
 */
function theme_nodes_forcontu_price($variables) {
  $output = '<div class="product-price">';
  $output .= t('Price') . ':' . $variables['price'] . ' €';
  $output .= '</div>';
  return $output;
}

/*
 * Returns the HTML output for the item Weight.
 *
 * @param $variables
 *   An associative array containing:
 *   - weight (kg).
 *
 * @ingroup themeable
 */
function theme_nodes_forcontu_weight($variables) {
  $output = '<div class="product-weight">';
  $output .= t('Weight') . ':' . $variables['weight'] . ' Kg.';
  $output .= '</div>';
  return $output;
}

/*
 * Returns the HTML output for the Dimensions items (Length x Width x Height).
 *
 * @param $variables
 *   An associative array containing:
 *   - length (cm).
 *   - width (cm).
 *   - height (cm).
 *
 * @ingroup themeable
 */
function theme_nodes_forcontu_dimensions($variables) {
  $output = '<div class="product-dimensions">';
  $output .= t('Dimensions (length x width x height)') . ':' . $variables['length'] . 'x' . $variables['width'] . 'x' . $variables['height'] . ' cm.' ;
  $output .= '</div>';
  return $output;
}
```

In **Figure F50.27** the results of the presentation of a node with content type **My product** is shown.

F50.27

Practical Case 50.3

Node Presentation

Final presentation of the node using the functions defined in the template.

White T-shirt

Plain white t-shirt

Price: 5 €

Weight: 1 Kg.

Dimensions (length x width x height): 20x20x2 cm.

Access control

50.4

We saw in **Unit 49** how to create permissions by implementing the **hook_permission()** function.

http://api.drupal.org/api/function/hook_permission/7

The **hook_permission()** function returns an array with the permissions created for the module.

To check if a user has a particular permission use the API function **user_access()**.

http://api.drupal.org/api/drupal/modules--user--user.module/function/user_access/7

In this section we extend the access control to both nodes and particular content types in general. The set of system functions and hooks for controlling access to nodes in Drupal can be found at:

http://api.drupal.org/api/drupal/modules--node--node.module/group/node_access/7

Control operations on a node with hook_node_access()

Typical operations that can be performed on a node of a particular content type are:

- Create new content.
- Edit own content.
- Edit any content.
- Delete own content.
- Delete any content.

In Drupal 6 it was necessary to define these permissions in each module and for each content type. We did it with **hook_perm()**, the equivalent to **hook_permission()** in Drupal 7.

Drupal 7 directly defines the permissions for **create**, **delete**, **edit** and **display** of content types defined by the module. Therefore we only implement **hook_permission()** if the module needs to define other permissions.

Figure F50.28 shows the permissions added directly by the **Node** module when creating a new content type.

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
<i>My product</i> : Create new content	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<i>My product</i> : Edit own content	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<i>My product</i> : Edit any content	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<i>My product</i> : Delete own content	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<i>My product</i> : Delete any content	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Comparing D7/D6 Node Permissions

In Drupal 6 we use **hook_perm()** instead of **hook_permission()**.

Another important change in Drupal 7, is that the Node module directly defines the permissions to create, edit or delete nodes defined by new content types. In Drupal 6 it was necessary to define all the permissions with **hook_perm()**.

F50.28

Permissions

Permissions added directly by the Node module when a new content type is defined.

It is not necessary to define how and what these default permissions control access. The system will check whether the user has the appropriate permissions to create, edit or delete nodes.

If we add additional checks to determine whether a user has access to these operations, we implement **hook_node_access()**.

http://api.drupal.org/api/drupal/modules--node-node.api.php/function/hook_node_access/7

The **hook_node_access()** function has the following parameters:

hook_node_access(\$node, \$op, \$account)

- **\$node**. The Node that is to perform the operation of if it does not yet exist the content type to create.
- **\$op**. Indicates the operation to be performed. The possible values are:
 - o **'create'**. Checks to see if the user can create a node of the type created by the module.
 - o **'delete'**. Checks to see if the user can delete the node.
 - o **'update'**. Checks to see if the user can edit the node.
 - o **'view'**. Checks to see if the user can view the node.
- **\$account**. \$user object with data from the user who will perform the operation.

The function can return these values:

- **NODE_ACCESS_ALLOW**. If the operation is allowed.
- **NODE_ACCESS_DENY**. If the operation is not permitted.
- **NODE_ACCESS_IGNORE**. Is the default value (NULL) and does not affect the operation.

The following example checks if the node is to be placed on the home page ('promoted'). If so the node may not be removed. **F50.29**

F50.29

hook_node_access()

Add additional checks to determine whether the user has access to certain operations.

```
/** 
 * Implements hook_node_access().
 */
function example_node_access($node, $op, $account) {
  // Can't delete Promoted to front page nodes
  if($op == 'delete' && $node->promote == 1) {
    drupal_set_message(t('Promoted to front page nodes can\'t be deleted.'));
    return NODE_ACCESS_DENY;
  }

  return NODE_ACCESS_IGNORE;
}
```

We should note that the permission check is also performed to **decide whether or not to print the corresponding action button**. In this case, when showing the edit form of the node the button to access the operation 'delete' is normally shown but if the condition (promoted) is met, the message is displayed and the delete button will be hidden.

Control access through the table 'node_access'

Drupal has a more powerful access control method for nodes. This is the table **node_access** and the functions **hook_node_grants()** and **hook_node_access_records()**.

If we analyze the content of the **node_access** table after installing Drupal, you will see a single record shown in the **Figure F50.30**. This entry in the table in reality nullifies content access control through the table node_access, and by default no core modules use this mechanism.

nid	gid	realm	grant_view	grant_update	grant_delete
0	0	all	1	0	0

F50.30**Table node_access**

Default register nullifying access control for nodes.

However, simply installing a module that uses this mechanism causes permissions to be rebuilt and new entries generated in the **node_access** table. The system will ask for confirmation to rebuild the permissions through the message: "Access to content permissions must be rebuilt, Rebuild permissions."

The **Figure F50.31** shows the table after the module **Forum Access** has been installed, using this mechanism to control access to certain forums and allow the creation of private forums.

nid	gid	realm	grant_view	grant_update	grant_delete
1	0	all	1	0	0
2	0	all	1	0	0
3	0	all	1	0	0
4	0	all	1	0	0
5	0	all	1	0	0
6	0	all	1	0	0
7	0	all	1	0	0
8	0	all	1	0	0
9	0	all	1	0	0
10	0	all	1	0	0

F50.31**Node_access table**

Once a module using this access control mechanism is activated, the contents of the node_access table are always checked.

Once a module that uses this access control mechanism is activated, all nodes must be recorded in the **node_access table**, so that users can access them, even in the case of fully public nodes. This is why the **node_access** table now includes an entry for each node regardless of the content type, with read (grant_view) permissions for all users (realm == 'all'). **F50.32**

The fields of the **node_access** table are:

- **nid**. Id of the node on which the permissions apply.
- **gid**. The grant ID or permission identifier is a value that identifies a permission for a particular "realm". In some cases it will be equivalent to the identifier for the role (roles table), but other modules can use it as a user ID or any other custom value. Therefore, we can say it is conceptually equivalent to a role but can be customized for each module. We will see some examples below.
- **realm**. Identifier that allows the grouping of permissions for a module. It is something like a namespace, and is very common that it is related to the internal name of the module.

- **grant_view**. Permission to view.
- **grant_update**. Permission to edit.
- **grant_delete**. Premission to delete.

F50.32**node_access table after creating private forums**

Registrations added for the Forum Access module (realm = 'forum_access').

nid	gid	realm	grant_view	grant_update	grant_delete
31	0	all	1	0	0
32	0	all	1	0	0
33	0	all	1	0	0
34	0	all	1	0	0
37	2	forum_access	1	0	0
37	8	forum_access	0	1	1
37	1	forum_access	1	0	0

hook_node_grants()

The **hook_node_grants()** function facilitates system access to information on user node permissions.

These permissions correspond to the gid values (grant IDs) on the **node_access** table. It is common to use the role identifiers (rid on the roles table), but the module could define any other set of values (for each group of the "realm").

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_grants/7

hook_node_grants(\$account, \$op)

The **\$account** parameter corresponds to the user whose permissions are being requested. The **\$op** parameter differentiates between operations on the node: 'view', 'update' and 'delete'.

For example, the **Forum Access** module (http://drupal.org/project/forum_access) creates a group identified by **realm = 'forum_access'**, whose **gid** correspond to the **rid** (role identifier in the **roles** table). The **hook_node_grants()** function returns through **\$grants['forum_access']** an array of roles (rid) for the user, stored in the user object (\$account->roles). **F50.33**

F50.33**hook_node_grants()**

Information about the user roles.

```
/** 
 * Implements hook_node_grants(). 
 */
function forum_access_node_grants($user, $op) {
  $grants['forum_access'] = array_keys($user->roles);
  return $grants;
}
```

The **Forum Access** module does not use the **\$op** variable for **hook_node_grants()**, because it will always use the roles as gid. Here is another example where the parameter **\$op** is used: **F50.34**

```
/***
 * Implements hook_node_grants().
 *
 */
function example_node_grants($account, $op) {
  if ($op == 'view' && user_access('access private content', $account)) {
    $grants['example'] = array(1);
  }

  if (($op == 'update' || $op == 'delete') && user_access('edit private content', $account)) {
    $grants['example'] = array(2);
  }

  return $grants;
}
```

F50.34**hook_node_grants()**

Using hook_node_grants() function of operations via \$op.

In this case the permissions of the module, defined through **hook_permission()**, will determine if the user has permissions on each operation of the node. In this example, the **gid** takes two possible values, 1 for group users with read permissions on private files and 2 for users with permissions to edit and delete.

hook_node_access_records()

The **hook_node_access_records()** function requests from the module that implemented it for information on the access permissions of a particular node. This information is stored in the **node_access** table.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_access_records/7

The function requires as a parameter the \$node object, and returns an structured array with definitions of the access permissions for that node:

hook_node_access_records (\$node)

Each permission in the array has the following fields:

- '**realm**'. Is the namespace (realm) defined by the module with hook_node_grants().
- '**gid**'. Value of the 'grant ID' defined with hook_node_grants().
- '**grant_view**'. If this has a value of 1 it indicates that the user has been identified as a member of the gid and in this space (realm) can view this node.
- '**grant_update**'. If this has a value of 1 it indicates that the user has been identified as a member of the gid and in this space (realm) can edit this node.
- '**grant_delete**'. If this has a value of 1 it indicates that the user has been identified as a member of the gid and in this space (realm) can delete this node.
- '**priority**'. If various modules try to set permissions on the same node, the module with the highest priority "will win". Generally we leave this field as 0.

In the following example, the permissions on the node are determined by checking if the node has been marked as private (through an additional field defined as `$node->private`). [F50.35](#) If the node is not private, it will be accessible.

F50.35**hook_node_access_records()**

En este ejemplo los permisos sobre el nodo se determinan comprobando si el nodo ha sido marcado como privado.

```
/**
 * Implements hook_node_access_records().
 */
function example_node_access_records($node) {
  // Is only taken into mind if the node is marked as private.
  if (!empty($node->private)) {
    $grants = array();
    $grants[] = array(
      'realm' => 'example',
      'gid' => 1,
      'grant_view' => 1,
      'grant_update' => 0,
      'grant_delete' => 0,
      'priority' => 0,
    );
  }

  $grants[] = array(
    'realm' => 'example',
    'gid' => 2,
    'grant_view' => 1,
    'grant_update' => 1,
    'grant_delete' => 1,
    'priority' => 1,
  );
}

return $grants;
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

In another example, taken from the **Forum Access** module, the permissions on the node are extracted from the table `{forum_access}`, created by the module itself: [F50.36](#)

F50.36**hook_node_access_records()**

In this example, the permissions on the node are queried from the `forum_access` table.

```
/**
 * Implements hook_node_access_records().
 */
function forum_access_node_access_records($node) {
  //...
  $result = db_query('SELECT * FROM {forum_access} WHERE tid = :tid', array(
    ':tid' => $tid
  ));
  foreach ($result as $grant) {
    if (isset($seers[$grant->rid])) {
      continue; // Don't provide any useless grants!
    }
    $grants[$tid][] = array(
      'realm'      => 'forum_access',
      'gid'        => $grant->rid,
      'grant_view' => $grant->grant_view,
      'grant_update' => $grant->grant_update,
      'grant_delete' => $grant->grant_delete,
      'priority'   => $grant->priority,
    );
  }
  if (isset($grants[$tid])) {
    return $grants[$tid];
  }
}
```

Control access to a node through the menu system

To conclude this section we should remember that it is possible to control access to a node directly using the menu system.

When registering a URL by implementing the **hook_menu()** function, as we studied in **Unit 46**, we can define the necessary permissions to access the node display, or even include a customer access control function 'access callback'. **F50.37**

```
/***
 * Implements hook_menu().
 */
function page_example_menu() {
  $items['examples/private_page'] = array(
    'title' => 'Private Page Example',
    'page callback' => 'page_example_description',
    'access arguments' => array('access private nodes'),
  );
  return $items;
}
```

F50.37

Control access via hook_menu()

Using `hook_menu()` we can also control access to a URL, by indicating an access function and its rationale. By default it uses `user_access()` and it passes as arguments those permissions the user needs.

By not defining a custom value for '**access callback**', the system will call the **user_access()** function with the parameters defined in '**access arguments**'. Therefore an equivalent test is performed:

```
if (user_access('access private nodes')) {
  // display node...
}
```

The menu system also allows us to create custom access control functions, which can add more complex logic, according to the needs of the module.

The following example shows how the **user** module uses a custom access control function, called **user_view_access()**, which is called when the user profile page is loaded. The function performs several checks and ensures that one of them is fulfilled before granting read access to the page: **F50.38**

- The profile is that of the user accessing it.
- The user is an administrator (permission 'administer users').
- The user is not locked and has permission to view user profiles (permission 'access user profiles')

F50.38**Custom Access Function**

As an example, the User module implements a custom access function `user_view_access()`, which determines whether the user has access to the user profile URL.

```
/***
 * Implements hook_menu().
 */
function user_menu() {
  $items['user/%user'] = array(
    'title' => 'My account',
    'title callback' => 'user_page_title',
    'title arguments' => array(1),
    'page callback' => 'user_view_page',
    'page arguments' => array(1),
    'access callback' => 'user_view_access',
    'access arguments' => array(1),
    'menu_name' => 'navigation',
  );
  return $items;
}

/***
 * User view access callback.
 *
 * @param $account
 *   Can either be a full user object or a $uid.
 */
function user_view_access($account) {
  $uid = is_object($account) ? $account->uid : (int) $account;

  // Never allow access to view the anonymous user account.
  if ($uid) {
    // Admins can view all, users can view own profiles at all times.
    if ($GLOBALS['user']->uid == $uid || user_access('administer users')) {
      return TRUE;
    }
    elseif (user_access('access user profiles')) {
      // At this point, load the complete account object.
      if (!is_object($account)) {
        $account = user_load($uid);
      }
      return (is_object($account) && $account->status);
    }
  }
  return FALSE;
}
```

Content Filters

50.5

Text formats are composed of one or more content filters that are applied to a text field.

You can configure text formats and filters from:

Administration ⇒ **Configuration** ⇒ **Content authoring** ⇒ **Text formats**

Accessing the configuration of any of the text formation (Filtered HTML, Full HTML, Plain text, PHP code, etc.) displays a list of available filters. Each text format may have one or more active filters.

When a specific format is applied to a text field, the final text displayed to the user will be the result of applying, one by one and according to the established order, the filters associated with the text format. **F50.39**

The screenshot shows the 'Text formats' configuration page for the 'Full HTML' format. It includes fields for 'Name' (set to 'Full HTML'), 'Machine name' (set to 'full_html'), 'Roles' (with 'administrator' checked), and a large 'Enabled filters' section. The 'Enabled filters' section is highlighted with a red border and contains several checkboxes: 'Limit allowed HTML tags', 'Display any HTML as plain text', 'Convert URLs into links' (checked), 'Convert line breaks into HTML (i.e.
 and <p>) (checked)', 'PHP evaluator', and 'Correct faulty and chopped off HTML'. Below this is a 'Filter processing order' section with three items: 'Convert URLs into links', 'Convert line breaks into HTML (i.e.
 and <p>)', and 'Correct faulty and chopped off HTML'. The 'Convert URLs into links' item is expanded, showing its configuration options.

URL Text Formats

/admin/config/content/formats

F50.39

Text formats and filters

Each text format can have one or more active filters. You can also set the execution order of the filters.

The filters, therefore, are used to make any changes on a text field: replace or remove strings, make corrections, add additional text, etc.

In this section we study the functions necessary to create content filters that apply to existing text formats or new formats as defined by the module.

Available filters are initially provided by the **Filter** module, except the **PHP evaluator** filter which is provided by the **PHP Filter** module. In addition to the examples in this unit, you can check these modules as a reference for programming new filters.

Function hook_filter_info()

Using the **hook_filter_info()** function we define the filters contained in the new module.

http://api.drupal.org/api/drupal/modules--filter-filter.api.php/function/hook_filter_info/7

The function, which does not require input parameters, returns an associative array with the definition of each filter, which is itself an array with the following fields:

- **'title'** (mandatory). Title of the filter to be displayed in the admin area (list of filters within each text format).
- **'description'**. Additional description of the filter.
- **'settings callback'**. Name of the callback function that returns the former filter settings.
- **'default settings'**. Array with the default values that apply to the filter.
- **'prepare callback'**. If necessary, the name of the function that sets the contents before processing.
- **'process callback'** (mandatory). Name of the function that implements the filter.
- **'cache'**. Indicates whether the search text can be searched. The default value is TRUE.
- **'tips callback'**. Name of the function that returns user help when using the filter.
- **'weight'**. Weight of the filter in the text formats. The administrator can change the order of the filters from the text format configuration area.

As an example of the implementation of the **hook_filter_info()** function we analyze the definition that the **Filter** module uses to **Limit allowed HTML tags** ('filter_html').

F50.40

F50.40

hook_filter_info()

Definition of new content filters.

```
/** 
 * Implements hook_filter_info().
 */
function filter_filter_info() {
  $filters['filter_html'] = array(
    'title' => t('Limit allowed HTML tags'),
    'process callback' => '_filter_html',
    'settings callback' => '_filter_html_settings',
    'default settings' => array(
      'allowed_html' => '<a> <em> <strong> <cite> <blockquote> <code>
<ul> <ol> <li> <dl> <dt> <dd>',
      'filter_html_help' => 1,
      'filter_html_nofollow' => 0,
    ),
    'tips callback' => '_filter_html_tips',
    'weight' => -10,
  );
  // other filters
  return $filters;
}
```

In the filter are references to the following functions:

- 'settings callback' => '_filter_html_settings'. The name of the function following the naming convention **hook_filter_FILTER_settings**, available at:

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_FILTER_settings/7

This feature is not really a hook, but an example of the nomenclature that you should follow when calling the 'settings callback' parameter:

hook_filter_FILTER_settings(\$form, &\$form_state, \$filter, \$format, \$defaults, \$filters)

In the function we define the elements of the form that are displayed in the configuration of the filter. In this example, **F50.41** the function defines the three form elements shown in the **Figure F50.42**.

The default values shown are defined within the **hook_filter_info()** function, through the 'default settings' field.

```
/***
 * Settings callback for the HTML filter.
 */
function _filter_html_settings($form, &$form_state, $filter,
$format, $defaults) {
  $filter->settings += $defaults;

  $settings['allowed_html'] = array(
    '#type' => 'textfield',
    '#title' => t('Allowed HTML tags'),
    '#default_value' => $filter->settings['allowed_html'],
    '#maxlength' => 1024,
    '#description' => t('A list of HTML tags that can be used.
JavaScript event attributes, JavaScript URLs, and CSS are always
stripped.'),
  );
  $settings['filter_html_help'] = array(
    '#type' => 'checkbox',
    '#title' => t('Display basic HTML help in long filter tips'),
    '#default_value' => $filter->settings['filter_html_help'],
  );
  $settings['filter_html_nofollow'] = array(
    '#type' => 'checkbox',
    '#title' => t('Add rel="nofollow" to all links'),
    '#default_value' => $filter->settings['filter_html_nofollow'],
  );
  return $settings;
}
```

F50.41

Configuring Filters

Function to generate the filter configuration form.

Filter settings

Limit allowed HTML tags Enabled	Allowed HTML tags <a> <cite> <blockquote> <code> <dl> <dt> <dd>
Convert URLs into links Enabled	A list of HTML tags that can be used. JavaScript event attributes, JavaScript URLs, and CSS are always stripped.
	<input checked="" type="checkbox"/> Display basic HTML help in long filter tips <input type="checkbox"/> Add rel="nofollow" to all links

F50.42

Filter Options

Shows the configuration options defined in the function for configuring the filter.

- 'process callback' => '_filter_html'. Defines the main function, that implements the filter. This function follows the nomenclature found **F50.43** at:

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_FILTER_process/7

As in the previous case, this function is not a hook. While we do not need to include all the parameters specified in the refnreced function, we do need to respect the order of them. To avoid errors, we recommend that you always declare all of the parameters of the function, although they will not all be used.

hook_filter_FILTER_process(\$text, \$filter, \$format, \$langcode, \$cache, \$cache_id)

F50.43

Filter Function

Function that implements the filter. Returns the text after applying the filter.

```
/***
 * HTML filter. Provides filtering of input into accepted HTML.
 */
function _filter_html($text, $filter) {
  $allowed_tags = preg_split('/\s+|<|>/', $filter->settings['allowed_html'], -1, PREG_SPLIT_NO_EMPTY);
  $text = filter_xss($text, $allowed_tags);

  if ($filter->settings['filter_html_nofollow']) {
    $html_dom = filter_dom_load($text);
    $links = $html_dom->getElementsByTagName('a');
    foreach ($links as $link) {
      $link->setAttribute('rel', 'nofollow');
    }
    $text = filter_dom_serialize($html_dom);
  }

  return trim($text);
}
```

- 'tips callback' => '_filter_html_tips'. This function returns the help text for the filter. **F50.44**

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_FILTER_tips/7

hook_filter_FILTER_tips(\$filter, \$format, \$long)

In our example, the function returns instructions on how to use each HTML tag allowed.

F50.44

Filter Help Text

This function returns a help text to be displayed when the user selects a text format that include the filter.

```
/***
 * Filter tips callback for HTML filter.
 */
function _filter_html_tips($filter, $format, $long = FALSE) {
  global $base_url;

  //...
  $output = t('Allowed HTML tags: @tags', array('@tags' =>
$allowed_html));
  if (!$long) {
    return $output;
  }

  $output = '<p>' . $output . '</p>';
  if (!$filter->settings['filter_html_help']) {
    return $output;
  }
```

```

$output .= '<p>' . t('This site allows HTML content. While
learning all of HTML may feel intimidating, learning how to use a
very small number of the most basic HTML \'tags\' is very easy. This
table provides examples for each tag that is enabled on this
site.') . '</p>';
$output .= '<p>' . t('For more information see W3C\'s <a
href="@html-specifications">HTML Specifications</a> or use your
favorite search engine to find other sites that explain HTML.',

array('@html-specifications' => 'http://www.w3.org/TR/html/')) .
'</p>';
$tips = array(
  'a' => array(t('Anchors are used to make links to other
pages.'), '<a href="' . $base_url . '">' .
check_plain(variable_get('site_name', 'Drupal')) . '</a>'),
  'br' => array(t('By default line break tags are automatically
added, so use this tag to add additional ones. Use of this tag is
different because it is not used with an open/close pair like all
the others. Use the extra " /" inside the tag to maintain XHTML
1.0 compatibility'), t('Text with <br />line break')),
  //...
);
//...
return $output;
}

```

- 'prepare callback'. Although this function is not used, if we add a function to prepare the text, use the nomenclature defined at:

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_FILTER_prepare/7

Creating text formats

There is no specific hook to define text formats. If we want a module to create its own text format, we have to do it when installing the module, through the **hook_install()** or **hook_enable()** functions, and using the **file_format_save()** function.

http://api.drupal.org/api/drupal/modules--filter--filter.module/function/filter_format_save/7

The **file_format_save()** allows you to save a text format in the database.

filter_format_save(\$format)

The parameter **\$format** is an object with the following attributes:

- **'format'**. System name of the text format.
- **'name'**. Title of the text format.
- **'status'**. Indicates whether the text format is enabled (1) or not (0).
- **'weight'**. Weight of the text format.
- **'filters'**. Array with filters assigned to the text format. We can assign filters created by the same module or other modules. Each of the filters have the 'weight', 'status' and 'settings' fields assigned.

As an example of creating a text format, analyze the code implementing **hook_install()** for the **Filter** module. **F50.45**

F50.45**Create Text formats**

To create a text format we call `filter_format_save()` during module installation.

```
/***
 * Implements hook_install().
 */
function filter_install() {
  // Only define the Plain text format
  $plain_text_format = array(
    'format' => 'plain_text',
    'name' => 'Plain text',
    'weight' => 10,
    'filters' => array(
      // Escape all HTML.
      'filter_html_escape' => array(
        'weight' => 0,
        'status' => 1,
      ),
      // URL filter.
      'filter_url' => array(
        'weight' => 1,
        'status' => 1,
      ),
      // Line break filter.
      'filter_autop' => array(
        'weight' => 2,
        'status' => 1,
      ),
    ),
  );
  $plain_text_format = (object) $plain_text_format;
  filter_format_save($plain_text_format);

  // Set the fallback format to plain text.
  variable_set('filter_fallback_format', $plain_text_format->format);
}
```

The module only creates the text format **Plain text**, which has three filters assigned:

- 'filter_html_escape'. Display any HTML as plain text.
- 'filter_url'. Convert URLs into links.
- 'filter_autop'. Convert line breaks into HTML.

Note in order to create the object with the definition in the text format, first create an array with the structure and then convert the object into it. This object is passing to the `filter_format_save()` function, which is responsible for creating the text format by saving it to the database.

```
$plain_text_format = array(...);
$plain_text_format = (object) $plain_text_format;
filter_format_save($plain_text_format);
```

The **Filtered HTML** and **Full HTML** text formats are created by the **Standard** installation profile (`profiles/standard/standard.install`), and the text format **PHP code** by the **PHP filter** module. Both cases use the mechanism explained.

Other API functions related to the filters

The **Filter** module provides other functions related to the filters that can be used when developing modules. Some of these functions are:

- **check_markup()**. Runs all filters for a text format on a given string. Returns the filtered string.

http://api.drupal.org/api/drupal/modules--filter--filter.module/function/check_markup/7

- **filter_access()**. Checks whether the user has access to the text format indicated.

http://api.drupal.org/api/drupal/modules--filter--filter.module/function/filter_access/7

- **filter_formats()**. Returns a list of the text formats the user has access to.

http://api.drupal.org/api/drupal/modules--filter--filter.module/function/filter_formats/7

- **hook_filter_info_alter()**. Implementing this hook allows other modules to make changes to the filter definition.

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_info_alter/7

- **hook_filter_format_insert()**. Performs actions when a new text format has been created.

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_format_insert/7

- **hook_filter_format_update()**. Preforms actions when a text format is updated.

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_format_update/7

- **hook_filter_format_disable()**. Performs actions when a text format is disabled.

http://api.drupal.org/api/drupal/modules--filter--filter.api.php/function/hook_filter_format_disable/7

Practical Case 50.4 Create a filter

In this scenario we will create a module called **Filter Forcontu** (`filter_forcontu`), where we define, for now, a filter called **Foo** (`filter_foo`). The filter will replace the string "foo" with another string specified through the filter settings.

Step 1. Creation of the Filter Forcontu module

As a first step we will create the module folder (`filter_forcontu`) and the key files (`filter_forcontu.info` and `filter_forcontu.module`).

Step 2. Filter Definition

We implement the `hook_filter_info()` function to define the filter and the associated functions: **F50.46**

- The `_filter_forcontu_filter_foo_settings()` function, which defines the filter settings form. In 'default settings' we will pass to this function the default value 'bar', associated with the field 'filter_forcontu_foo' of the form.
- The `_filter_forcontu_filter_foo_process()` function, the main function of the filter.
- The `_filter_forcontu_filter_foo_tips()` function, returns a help message on the filter.

F50.46

Practical Case 50.4

`hook_filter_info()`

Defines the filter and associated functions.

```
/***
 * Implements hook_filter_info().
 */
function filter_forcontu_filter_info() {
  $filters['filter_foo'] = array(
    'title' => t('Foo filter'),
    'description' => t('All occurrences of "foo" will be replaced by the string specified in the configuration.'),
    'process callback' => '_filter_forcontu_filter_foo_process',
    'default settings' => array(
      'filter_forcontu_foo' => 'bar',
    ),
    'settings callback' => '_filter_forcontu_filter_foo_settings',
    'tips callback' => '_filter_forcontu_filter_foo_tips',
  );
  return $filters;
}
```

The following defines each of the functions reference in the filter configuration.

Step 3. Form Filter settings

For the filter settings we use a single text field called **filter_forcontu_foo**.

The default values are passed to the function through the input parameter **\$defaults** (`$defaults['filter_forcontu_foo']`). If you have changed this value from the filter configuration the store value can be obtained from `$filter->settings['filter_forcontu_foo']`. This is why, to assign a value to the form field, first check the contents of the variable `$filter->settings`, and then the contents of `$defaults`. If neither variable returns a value, the form field is empty. **F50.47**

```
function _filter_forcontu_filter_foo_settings($form, $form_state,
$filter, $format, $defaults) {
  $settings['filter_forcontu_foo'] = array(
    '#type' => 'textfield',
    '#title' => t('Replacement string'),
    '#default_value' => isset($filter->settings['filter_forcontu_foo']) ? $filter->settings['filter_forcontu_foo'] : $defaults['filter_forcontu_foo'],
    '#description' => t('This string will replace all occurrences of "foo".')
  );
  return $settings;
}
```

F50.47**Practical Case 50.4****Function Configuration**

Generates the form with the filter settings.

Step 4. Filter Function

The filtering function performs the replacement. We use the PHP function **str_replace()** for this. **F50.48**

```
function _filter_forcontu_filter_foo_process($text, $filter, $format) {
  $replacement = isset($filter->settings['filter_forcontu_foo']) ? $filter->settings['filter_forcontu_foo'] : 'bar';
  return str_replace('foo', $replacement, $text);
}
```

F50.48**Practical Case 50.4****Filter Function**

Implements the filter.

Step 5. User help for the filter

Finally, we add a help text to be displayed when the user selects a text format enabled by this filter. **F50.49**

```
function _filter_forcontu_filter_foo_tips($filter, $format, $long = FALSE) {
  $replacement = isset($filter->settings['filter_forcontu_foo']) ? $filter->settings['filter_forcontu_foo'] : 'bar';
  if (!$long) {
    // This string will be shown in the content add/edit form
    return t('<em>foo</em> será reemplazado por %replacement.', array('%replacement' => $replacement));
  }
  else {
    return t('All occurrences of "foo" will be replaced by the string specified in the configuration. Current replacement value is "%replacement.".', array('%replacement' => $replacement));
  }
}
```

F50.49**Practical Case 50.4****Help Function**

Returns related help text on using the filter.

Step 6. Installation and activation of the filter

Once the module is installed, the filter is available (but disabled) for the configuration of any of the text formats:

Administration ⇒ Configuration ⇒ Content authoring ⇒ Text formats

URL Text formats

/admin/config/content/formats

To test the operation perform the following actions: **F50.50**

- Activate the filter **Foo filter** for the **Filtered HTML** text format.
- In the filter configuration **Replacement String** specify the Word CENSORED.

F50.50**Practical Case 50.4****Adding the filter**

To test the functionality add the filter to the Filtered HTML text format.

In the configuration of the filter assign the word CENSORED.

A text format contains filters that change the user input, for example stripping out malicious HTML or making URLs clickable. Filters are executed from top to bottom and the order is important, since one filter may prevent another filter from doing its job. For example, when URLs are converted into links before disallowed HTML tags are removed, all links may be removed. When this happens, the order of filters may need to be re-arranged.

Name *
Filtered HTML Machine name: filtered_html

Roles
 anonymous user
 authenticated user
 administrator

Enabled filters
 Limit allowed HTML tags
 Display any HTML as plain text
 PHP evaluator
 Executes a piece of PHP code. The usage of this filter should be restricted to administrators only
 Convert line breaks into HTML (i.e.
 and <p>)
 Foo filter
 All occurrences of "foo" will be replaced by the string specified in the configuration.
 Convert URLs into links
 Correct faulty and chopped off HTML

Filter processing order [Show row weights](#)

+	Foo filter
+	Convert URLs into links
+	Limit allowed HTML tags
+	Convert line breaks into HTML (i.e. and <p>)
+	Correct faulty and chopped off HTML

Filter settings

Limit allowed HTML tags Enabled	Replacement string CENSORED
Foo filter Enabled	This string will replace all occurrences of "foo".
Convert URLs into links Enabled	

Save configuration

- Save the configuration of the text format and check by clicking back to the filter settings, that the value of the replacement string is stored correctly.
- Create a node of the **Article type** with the URL **nodo-filter** and the **Filtered HTML** text format. Add text in the body of the node where the word "foo" appears on at least two occasions. **F50.51**
- Check that the help text shows specific advice for our filter.

Home > Add content

Create Article

Title *	Filtered node
Tags	
Enter a comma-separated list of words to describe your content.	
Body (Edit summary)	
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis ullamcorper ante a tellus ornare gravida. Donec non sapien ligula. Morbi tristique tincidunt erat at hendrerit. Sed tristique metus non sem pellentesque placerat <u>foo</u>. In velit neque, euismod eget semper vitae, dictum accumsan neque. Maecenas iaculis mollis cursus. Sed sollicitudin odio sit amet scelerisque molestie. Sed congue turpis eu mauris dictum dapibus. Etiam vitae luctus dolor. Sed et elit nibh. Integer sagittis dui ornare arcu consectetur, sed mattis libero porta. In a nisi neque. Pellentesque ut metus aliquam massa vestibulum mollis. Sed et dapibus enim, auctor mattis magna. Vivamus id mauris eros.</p> <p>Cras tempus non eros quis eleifend. Proin volutpat nulla est, at aliquet nisi dictum a <u>foo</u>. Pellentesque iaculis ante ante, eget ultricies magna vestibulum id. Mauris a urna sit amet dolor commodo pulvinar. Nulla sit amet neque nisl. Donec et ullamcorper urna. Donec quis enim vel orci congue aliquam sit <u>foo</u> amet vitae ligula. Nam lacinia justo sed purus pulvinar sodales. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque ac mi mauris. Nunc est turpis, lacinia sed venenatis ac, venenatis nec lectus. Mauris vulputate purus urna, sit amet imperdierit orci elementum vitae.</p>	
Text format	Filtered HTML
<ul style="list-style-type: none"> Web page addresses and e-mail addresses turn into links automatically. foo will be replace by CENSORED Allowed HTML tags: <a> <blockquote> <code> <dl> <dt> <dd> Lines and paragraphs break automatically. 	
Image	
Choose File	No file chosen
<input type="button" value="Upload"/> Upload an image to go with this article. Files must be less than 2 MB. Allowed file types: png gif jpg jpeg.	
Menu settings	Not in menu
Revision information	No revision
URL path settings	Alias: filtered-node
URL alias <u>filtered-node</u>	
Optional specify an alternative URL by which this content can be accessed. For example, type "about" when writing an about page. Use a relative path and don't add a trailing slash or the URL alias won't work.	

- Once the node has been created, check that the occurrences of the string "foo" have been replaced by "CENSORED". **F50.52**

Filtered node

 Submitted by admin on Sat, 02/22/2014 - 23:36

LoREM ipsum dolor sit amet, consectetur adipiscing elit. Duis ullamcorper ante a tellus ornare gravida. Donec non sapien ligula. Morbi tristique tincidunt erat at hendrerit. Sed tristique metus non sem pellentesque placerat CENSORED. In velit neque, euismod eget semper vitae, dictum accumsan neque. Maecenas iaculis mollis cursus. Sed sollicitudin odio sit amet scelerisque molestie. Sed congue turpis eu mauris dictum dapibus. Etiam vitae luctus dolor. Sed et elit nibh. Integer sagittis dui ornare arcu consectetur, sed mattis libero porta. In a nisi neque. Pellentesque ut metus aliquam massa vestibulum mollis. Sed et dapibus enim, auctor mattis magna. Vivamus id mauris eros.

Cras tempus non eros quis eleifend. Proin volutpat nulla est, at aliquet nisi dictum a CENSORED. Pellentesque iaculis ante ante, eget ultricies magna vestibulum id. Mauris a urna sit amet dolor commodo pulvinar. Nulla sit amet neque nisl. Donec et ullamcorper urna. Donec quis enim vel orci congue aliquam sit CENSORED amet vitae ligula. Nam lacinia justo sed purus pulvinar sodales. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque ac mi mauris. Nunc est turpis, lacinia sed venenatis ac, venenatis nec lectus. Mauris vulputate purus urna, sit amet imperdierit orci elementum vitae.

Also make sure when editing the node that they original text is displayed with the string "foo" and not the replacement string. Remember that filters are **always applied on the presentation of the text**, but does not change the stored text in the database.

F50.51

Practical Case 50.4

Create an article

We will create an Article with the Filtered HTML text format. We will include in the body some appearance of the string 'foo'.

Practical Case 50.5 Create a text format

In this scenario we will create through the **Filter Forcontu** module, a new text formation called **New format**. The format will have the smae filters as the Plain Text format plus the **Foo filter**, created in **Practical Case 50.4**.

Step 1. Implementing hook_install()

We will create the installation file of the module, **filter_forcontu.install**, where we will implement the `hook_install()` function.

We will define the structure array for the text format, specifying which filters to include: 'filter_html_escape', 'filter_url', 'filter_autop' and 'filter_foo'. Finally we specify the replament string "FILTERED", os that all accurances of the string "foo" will be replaced by "FILTERED". **F50.53**

F50.53

Practical Case 50.5

New Text Format

Installing the module creates a new text format. The text format, New Format, will be enabled among the others, the filter created in the previous case study, sets the string to "FILTERED".

```
/**
 * Implements hook_install().
 */
function filter_forcontu_install() {
  $new_format = array(
    'format' => 'new_format',
    'name' => 'New Format',
    'weight' => 10,
    'filters' => array(
      // Escape all HTML.
      'filter_html_escape' => array(
        'weight' => 0,
        'status' => 1,
      ),
      // URL filter.
      'filter_url' => array(
        'weight' => 1,
        'status' => 1,
      ),
      // Line break filter.
      'filter_autop' => array(
        'weight' => 2,
        'status' => 1,
      ),
      // Foo filter.
      'filter_foo' => array(
        'weight' => 3,
        'status' => 1,
        'settings' => array ('filter_forcontu_foo' => 'FILTERED'),
      ),
    ),
  );
  $new_format = (object) $new_format;
  filter_format_save($new_format);
}
```

Step 2. Reinstall the module

To execute the `hook_install()` function, we reinstall the module, uninstalling it completely first. After reinstalling we should see the new Text Format, called New format. **F50.54**

New Format 

A text format contains filters that change the user input, for example stripping out malicious HTML or making URLs clickable. Filters are executed from top to bottom and the order is important, since one filter may prevent another filter from doing its job. For example, when URLs are converted into links before disallowed HTML tags are removed, all links may be removed. When this happens, the order of filters may need to be re-arranged.

Name *	New Format	Machine name: new_format
--------	------------	--------------------------

Roles

- anonymous user
- authenticated user
- administrator

Enabled filters

- Limit allowed HTML tags
- Display any HTML as plain text
- PHP evaluator
Executes a piece of PHP code. The usage of this filter should be restricted to administrators only!
- Convert line breaks into HTML (i.e.
 and <p>)
- Foo filter
All occurrences of "foo" will be replaced by the string specified in the configuration.
- Convert URLs into links
- Correct faulty and chopped off HTML

Filter processing order

[Show row weights](#)

- + Foo filter
- + Display any HTML as plain text
- + Convert URLs into links
- + Convert line breaks into HTML (i.e.
 and <p>)

Filter settings

Foo filter Enabled	Replacement string FILTERED
Convert URLs into links Enabled	This string will replace all occurrences of "foo".

[Save configuration](#)**Step 3. Apply the new text format**

To check the operation of the new text format, create a node of the **Article** type with the URL **node-new-format** and the new text format **New Format**. Add a text to the body of the node where the string "foo" occurs at least twice. Check that once the content is saved the occurrences of "foo" are replaced by "FILTERED".

F50.54**Practical Case 50.5**
Configuring the text format

Accessing the configuration of the new text format, check that the filters are enabled and have the settings defined in the module installation.

50.6

Treatment of nodes from other modules

We have seen how to create new types of content within a module and how to work with nodes created by the module.

Sometimes we need our module to interact or modify the structure of the content types implemented by other modules.

For example if we want to add an additional field in one or more types of content, we do it in two steps:

- **Modifying the form** for creating and editing the node. To add the field to a form, use the `hook_form_alter()` function, studied in **Unit 47**.
- **Interact with the nodes.** To store and display information from additional fields we can use the node treatment functions which are similar to those studied to treat node of the content type defined within the same module: `hook_node_insert()`, `hook_node_update()`, `hook_node_delete()`, `hook_node_load()`, etc.

Note**Treatment of nodes of the same module**

Remember when you are working with nodes of a content type defined by the same module, instead of these more general functions, use the functions `hook_insert()`, `hook_update()`, etc.. Drupal runs these particular functions first.

For example, `hook_insert()` will be called before `hook_node_insert()`, so that each module will first perform the necessary actions on its nodes, and then subsequently other modules may change or supplement the actions.

[hook_node_insert\(\)](#)

The `hook_node_insert()` function lets you act when a new node of any content type is created.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_insert/7

[hook_node_update\(\)](#)

The `hook_node_update()` function allows operations to update a node of any type of content.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_update/7

[hook_node_delete\(\)](#)

The `hook_node_delete()` function allows deleted operations on a node of any type of content.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_delete/7

[hook_node_load\(\)](#)

The `hook_node_load()` function allows loading operations on nodes of any content type.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_load/7

hook_node_prepare()

The **hook_node_prepare()** function allows operations before creating or editing a node of any type of content.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_prepare/7

hook_node_view()

The **hook_node_view()** allows operations before presenting the node.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_view/7

hook_node_validate()

The **hook_node_validate()** function allows you to apply additional validation on content types provided by other modules.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_validate/7

hook_node_prepare()

The **hook_node_prepare()** function can act on a node when it is displayed in the create or edit form.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_prepare/7

hook_form_alter()

The **hook_form_alter()** function lets you interact with a form before its construction begins:

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_form_alter/7

A common use of this hook is to modify the form for creating or editing node, mainly to add new fields and thus, new functionalities related to the nodes. Consult Unit 47.

Example: Path Module

For operational functions that allow manipulating nodes of other modules, we can analyze many other modules, both core and those contributed by the community. Let's look at an example of how the **Path** module implements the functionality to add URL aliases for nodes of any type of content.

The **Path** module generates friendly URLs for any node, storing them in the **url_alias** table. When the Path module is activated a new field is displayed in the edit form of a node with a new field to enter URL aliases. This new field is added to the form through the **hook_form_alter()** function. [F50.55](#)

F50.55

Node fields in other modules

We use the Path module as an example. It adds a URL alias field to all nodes on a site.

The screenshot shows a node edit form. On the left, there are vertical tabs: 'Menu settings' (Not in menu), 'Revision information' (No revision), and 'URL path settings' (Alias: about-us). On the right, under 'URL alias', the value 'about-us' is entered. A note below says: 'Optional specify an alternative URL by which this content can be accessed. For example, type "about" when writing an about page. Use a relative path and don't add a trailing slash or the URL alias won't work.'

In reality the Path module implements **hook_form_BASE_FORM_ID_alter()**, through the **path_form_node_form_alter()** function, **node_form** being the base identifier (BASE_FORM_ID) for the edit and creation forms for nodes, regardless of the content type.

The function adds to the form for creating and editing nodes one fieldset field ('path') including an ('alias') textfield, where you can enter the URL alias of the node. The fieldset field is associated with the group '**additional_settings**', which will be shown within the vertical tabs. [F50.56](#)

F50.56

Fields added to node forms

Implementing the hook to amend the forms for creating and editing nodes, adds the 'path' tab and the 'alias' field.

```
/**
 * Implements hook_form_BASE_FORM_ID_alter().
 */
function path_form_node_form_alter(&$form, $form_state) {
  //...
  $form['path'] = array(
    '#type' => 'fieldset',
    '#title' => t('URL path settings'),
    '#collapsible' => TRUE,
    '#collapsed' => empty($path['alias']),
    '#group' => 'additional_settings',
    //...
  );
  $form['path']['alias'] = array(
    '#type' => 'textfield',
    '#title' => t('URL alias'),
    '#default_value' => $path['alias'],
    '#maxlength' => 255,
    '#collapsible' => TRUE,
    '#collapsed' => TRUE,
    '#description' => t('Optional specify an alternative URL by which this content can be accessed. For example, type "about" when writing an about page. Use a relative path and don\'t add a trailing slash or the URL alias won\'t work.'),
  );
  //...
}
```

The module also implements hook_node_insert(), hook_node_update() and hook_node_delete(), to act when it is creating, updating or deleting any node. [F50.57](#)

```
/***
 * Implements hook_node_insert().
 */
function path_node_insert($node) {
  if (isset($node->path)) {
    $path = $node->path;
    $path['alias'] = trim($path['alias']);
    // Only save a non-empty alias.
    if (!empty($path['alias'])) {
      //...
      path_save($path);
    }
  }
}

/***
 * Implements hook_node_update().
 */
function path_node_update($node) {
  if (isset($node->path)) {
    $path = $node->path;
    $path['alias'] = trim($path['alias']);
    // Delete old alias if user erased it.
    if (!empty($path['pid']) && empty($path['alias'])) {
      path_delete($path['pid']);
    }
    // Only save a non-empty alias.
    if (!empty($path['alias'])) {
      //...
      path_save($path);
    }
  }
}

/***
 * Implements hook_node_delete().
 */
function path_node_delete($node) {
  // Delete all aliases associated with this node.
  path_delete(array('source' => 'node/' . $node->nid));
}
```

F50.57**Other Functions**

The module requires implementation of hook_node_insert(), hook_node_update() and hook_node_delete() to work with the new field.

50.7

Other features of content types

To complete this unit we will make a brief reference to other API functions that are useful for the treatment of nodes or content types.

node_type_save() function

The **node_type_save()** function stores a content type in the database.

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_type_save/7

If the content type does not exist, the function performs an insert, creating the new content type. Conversely, if the content type already exists, it will update the data passed to the function.

The **node_type_save()** function is also responsible for invoking the following hooks:

- **hook_node_type_insert()**, if you created a new content type.
- **hook_node_type_update()**, if you have modified an existing content type.

node_type_delete() function

The **node_type_delete()** removes a content type. It is also responsible for invoking the **hook_node_type_delete()** function.

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_type_delete/7

drupal_get_path_alias() function

The **drupal_get_path_alias()** function returns the URL alias from an internal route. For example, we can use this function to get the alias assigned to the node internal path "node/1234".

If the path does not have an assigned alias, the function returns the same internal route.

If we do not provide a path to the function, it returns the URL alias of the current page.

http://api.drupal.org/api/drupal/includes--path.inc/function/drupal_get_path_alias/7

drupal_is_front_page() function

The function returns TRUE if the current page is the site's home page, and FALSE otherwise.

http://api.drupal.org/api/drupal/includes--path.inc/function/drupal_is_front_page/7

51 Programming entities and fields

As we know, Drupal 7 incorporates the concept of entities in its architecture. Now we can easily define entities such as object types and link them to fields. Nodes, users, taxonomy, archives, and comments are some of the entities defined in Drupal's core modules.

In this unit, we'll look at how to define new entities and to create associated fields or new types of fields. We'll also look at how to expand core API with Entity API.

What is presented in this unit applies to the entities that we have already studied, such as nodes, users, and taxonomy.

Comparative D7/D6

Entities and Field

In Drupal 6, we used the CCK module to create new fields and field types associated with content types.

In Drupal 7 we use Field API to define fields associated with entities so that we can add fields not only to nodes, but also to users, taxonomy terms, comments, etc.

Therefore, this unit is completely new to Drupal 6.

Unit contents

51.1 Definition of Entities.....	244
51.2 Creation of fields (Field API).....	262
51.3 Creation of new field types	267
51.4 Entity API module	276
51.5 Model Entities module	278



51.1 Definition of Entities

As we know, Drupal 7 incorporates the concept of entities in its architecture. Now we can easily define entities such as object types and link them to fields.

Nodes, users, taxonomy, archives, and comments are some of the entities defined in Drupal's core modules.

Here are some entity characteristics that we'll be looking at in this unit:

- We need to differentiate between entity and entity type. For example, content types are an entity type (the entity type Node), while the created nodes are entities. Often you will find references to "entity" when we're actually talking about "entity type".
- Entities define **their own tables**. We must indicate in the definition of the entity which tables are used.
- Entities define **their own objects**. We must indicate in the definition of the entity which properties of the object correspond to the elements of the entity ("id", "label", etc.).

For example, the node module implements the object \$node, where the property \$node->nid corresponds to the principal "id" of the entity. As we will see, this correspondence is established in the definition of the entity ("id" => "nid").

- We can define entities as being able to have **associated fields** ("fieldable") or not.
- A "bundle" is a **collection of fields** that can be associated with an entity. We find the most representative example in the set of fields associated with each content type. We can assign different fields to each content type. When creating a node, only that content type's fields will be available. To make this possible, the **Node** entity will have for each content type an associated set of fields (bundle).
- Entities can be associated with different display modes ("full", "teaser", etc.). As we will study at the intermediate level, the display of the entity can be managed from the admin area. We may also use additional modules such as Display Suite.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

hook_entity_info()

The **hook_entity_info()** function allows us to declare our entity types.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_entity_info/7

This function, which does not require input parameters, returns an array with the definition of each entity type with the following structure:

- '**label**'. Name of the entity type displayed to users.
- '**controller class**'. Name of the class that is used to load the entity type's objects. If no value is specified, the default controller is used.
- '**basis table**'. Name of the entity type's base table.
- '**revision table**'. Name of the entity type's revisions table, if any.
- '**static cache**'. Indicates whether or not the page will be cached. The default value is TRUE.
- '**field cache**'. Indicates whether or not the field will be cached. The default value is TRUE.
- '**load hook**'. Name of the hook that should be called by the controller, such as 'node_load'.
- '**uri callback**'. A function that returns the URI elements of the entity.
- '**label callback**'. A function that returns the label of an entity. The label refers to the backbone associated with an entity, such as a node title or the subject of a comment.
- '**fieldable**'. Indicates whether the entity may have associated fields.
- '**translation**'.
- '**entity keys**'. Array with information about how the Field API should extract the information of the objects of the entity type.
 - o '**id**'. Name of the property that contains the primary id of the entity type. All objects must have this property, and its value must be numeric.
 - o '**revision**'. Name of the property containing the revision id. Only used if the entities of this type can be versioned.
 - o '**bundle**'. Name of the property that contains a name of a collection of fields. A 'bundle' is a collection of fields that can be associated with an entity. If the entity type has a unique set of fields, we can omit this value, taking as its set value the name of the entity type.
 - o '**label**'. Name of the property containing the label of the entity type.
- '**bundle keys**'. Array with information on how Field API should extract the information of the bundles objects of the entity type.
 - o '**bundle**'. Name of the property that contains the name of the collection.
- '**bundles**'. An array describing all of the collections of the entity type.
- '**view modes**'. An array describing the display modes of the entity type.

As an example of the implementation of hook_entity_info(), let's analyze the node_entity_info() function from the **Node** module, which implements the **Node** entity type. **Figure 1** shows an excerpt of this function. **F51.1**

In the Node entity the field sets, or bundles, correspond to the content types (\$node-> type).

Display modes defined by the module are "full", "teaser" and "rss", plus those related to searches, "search_index" and "search_result," if the Search module is enabled.

F51.1**hook_entity_info()**

Allows the declaration of new entity types.

```
/***
 * Implements hook_entity_info().
 */
function node_entity_info() {
  $return = array(
    'node' => array(
      'label' => t('Node'),
      'controller class' => 'NodeController',
      'base table' => 'node',
      'revision table' => 'node_revision',
      'uri callback' => 'node_uri',
      'fieldable' => TRUE,
      'entity keys' => array(
        'id' => 'nid',
        'revision' => 'vid',
        'bundle' => 'type',
        'label' => 'title',
      ),
      'bundle keys' => array(
        'bundle' => 'type', // Corresponds to the content type.
      ),
      'bundles' => array(), // The bundles are defined further down.
      'view modes' => array( // Display modes.
        'full' => array(
          'label' => t('Full content'),
          'custom settings' => FALSE,
        ),
        'teaser' => array(
          'label' => t('Teaser'),
          'custom settings' => TRUE,
        ),
        'rss' => array(
          'label' => t('RSS'),
          'custom settings' => FALSE,
        ),
      ),
    ),
  );
  // Display modes specifically for searches
  if (module_exists('search')) {
    $return['node']['view modes'] += array(
      'search_index' => array(
        'label' => t('Search index'),
        'custom settings' => FALSE,
      ),
      'search_result' => array(
        'label' => t('Search result'),
        'custom settings' => FALSE,
      ),
    );
  }
}

// Creates a bundle or set of fields for each content type.
// The name of the bundle corresponds to the name of
// the content type.
foreach (node_type_get_names() as $type => $name) {
  $return['node']['bundles'][$type] = array(
    'label' => $name,
    'admin' => array(
      'path' => 'admin/structure/types/manage/%node_type',
      'real path' => 'admin/structure/types/manage/' .
        str_replace('_', '-', $type),
      'bundle argument' => 4,
      'access arguments' => array('administer content types'),
    ),
  );
}
return $return;
}
```

Function entity_load()

The **entity_load()** function loads entities from the database based on the entity type (**\$entity_type**) and an array with the IDs (**\$ids**) that we want to load.

```
entity_load($entity_type, $ids = FALSE, $conditions = array(), $reset = FALSE)
```

http://api.drupal.org/api/drupal/includes--common.inc/function/entity_load/7

This function is generally used indirectly by calling the function of a specific module that gets the object of the entity type. For example, the **Node** module uses the node_load() function, which internally calls node_load_multiple(), which makes the call to entity_load().

F51.2

```
function node_load($nid = NULL, $vid = NULL, $reset = FALSE) {
  $nids = (isset($nid) ? array($nid) : array());
  $conditions = (isset($vid) ? array('vid' => $vid) : array());
  $node = node_load_multiple($nids, $conditions, $reset);
  return $node ? reset($node) : FALSE;
}

function node_load_multiple($nids = array(), $conditions = array(),
  $reset = FALSE) {
  return entity_load('node', $nids, $conditions, $reset);
}
```

F51.2

entity_load()

Loads entities from the database.

hook_entity_info_alter()

The hook_entity_info_alter() function allows the definition of an entity to be modified by another module.

hook_entity_info_alter(&\$entity_info)

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_entity_info_alter/7

With **hook_entity_info_alter()** we can modify whatever parameters are available in hook_entity_info().

We will illustrate the implementation of this function for the **Book** module. This module modifies the **Node** entity to add the display mode "Print".

F51.3

```
/**
 * Implements hook_entity_info_alter().
 */
function book_entity_info_alter(&$info) {
  // Adds the display mode 'Print' to the nodes.
  $info['node']['view modes'] += array(
    'print' => array(
      'label' => t('Print'),
      'custom settings' => FALSE,
    ),
  );
}
```

F51.3

hook_entity_info_alter()

Modifies an entity defined by another module. In the example, Book adds a new display mode to the Node entity.

Practical Case 51.1 Create an entity type

In this example, we'll create a module called **Entity Forcontu** (entity_forcontu), where we define an entity type called **Item** (entity_forcontu_item). Imagine that **Item** is a product of a virtual store.

We'll create a unique grouping of field types (bundle), which we'll call **general_bundle**.

Step 1. Creating the Entity Forcontu module

As a first step, let's create the module folder (entity_forcontu) and key records (**entity_forcontu.info**, **entity_forcontu.install** and **entity_forcontu.module**).

Step 2. Defining the base table.

When initiating **hook_schema()** we define the base table of the entity. Remember that this function must be in the entity_forcontu.install file. **F51.4**

F51.4

Practical Case 51.1

hook_schema()

We define the base table for an entity with hook_schema().

```
/** 
 * Implements hook_schema()
 */
function entity_forcontu_schema() {
  $schema = array();

  $schema['entity_forcontu_item'] = array(
    'description' => 'Base table for the Item entity.',
    'fields' => array(
      'item_id' => array(
        'description' => 'ID of the entity.',
        'type' => 'serial',
        'unsigned' => TRUE,
        'not null' => TRUE,
      ),
      'bundle_type' => array(
        'description' => 'Type of bundle',
        'type' => 'text',
        'size' => 'medium',
        'not null' => TRUE
      ),
      'item_description' => array(
        'description' => 'Description of the element.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => TRUE,
        'default' => '',
      ),
      'created' => array(
        'description' => 'Date item was created.',
        'type' => 'int',
        'not null' => TRUE,
        'default' => 0,
      ),
    ),
    'primary key' => array('item_id'),
  );

  return $schema;
}
```

The base table of the entity will be called **entity_forcontu_item** and will have the following fields:

- **item_id**. The ID of the Item and the primary key of the table. It is an auto-incremented number field ["ITEM_ID"]
- **bundle_type**. Refers to the type of package or bundle. This allows us to define different types of entities. In this case, we define a single bundle type called **general_bundle**.
- **item_description**. Description of the ItemI.
- **created**. Date the entity was created.

Step 3. Defining the entity type Item

Defining the entity type is done by implementing **hook_entity_info()**, creating the **entity_forcontu_entity_info()** function in the file **entity_forcontu.module**.

The system name for the entity will be **entity_forcontu_item**. F51.5

```
/***
 * Implements hook_entity_info()
 */
function entity_forcontu_entity_info() {
  $info['entity_forcontu_item'] = array(
    // Label or title of the entity.
    'label' => t('Item Entity'),

    // Controller of the entity.
    'controller class' => 'EntityForcontuItemController',

    // Base table defined in hook_schema().
    'base table' => 'entity_forcontu_item',

    // Function that returns the URI elements of the entity.
    'uri callback' => 'entity_forcontu_item_uri',

    // The entity may carry associated fields.
    'fieldable' => TRUE,

    // We indicate the elements of the table corresponding to
    // the id and the type of sets or bundles of the entity.
    'entity keys' => array(
      'id' => 'item_id', // Unique ID of the entity.
      'bundle' => 'bundle_type' // Predefined bundle.
    ),
    'bundle keys' => array(
      'bundle' => 'bundle_type',
    ),

    // We activate the cache.
    'static cache' => TRUE,
  );

  // We define the bundles (Item types).
  'bundles' => array(
    'general_bundle' => array(
      'label' => 'General bundle',
      'admin' => array( //URL from where the Items will be administered
        'path' => 'admin/structure/entity_forcontu_item',
        'access arguments' => array('administer
          entity_forcontu_item entities'),
      ),
    ),
  );
}
```

F51.5

Practical Case 51.1

New entity

We define the new entity with **hook_entity_info()**.

```
// Display modes.
'view modes' => array(
  'full' => array(
    'label' => t('Full'),
    'custom settings' => FALSE,
  ),
),
);

return $info;
}
```

Step 4. Register the URLs with hook_menu()

Before proceeding to the specific functions of the entity, we register the different URLs that will be required for viewing and managing the entities. **F51.6**

- '**admin/structure/entity_forcontu_item**'. Items Management Area. When the URL is being defined for the group **general_bundle**, the URL will also display the paths **Manage fields** and **Manage displays**.
- '**admin / structure / entity_forcontu_item / list**'. Main path for the Items admin area.
- '**admin / structure / entity_forcontu_item / fields**' . **Manage fields** path. This URL is generated automatically, so you do not have to register it in hook_menu().
- '**admin / structure / entity_forcontu_item / display**'. **Manage display** path. This URL is generated automatically, so you do not have to register it in hook_menu().
- '**item /%entity_forcontu_item**'. The URL path for each Item will look like this: 'item/1234'. The tabs for viewing and editing the entity can be accessed from the entity page.
- '**item / % entity_forcontu_item / view**'. Main path for displaying the entity. It's equivalent to the previous URL.
- '**item / % entity_forcontu_item / edit**' . Edit path for the entity (item/1234/edit).
- '**item / add**' . Link to create a new entity. As we have just defined a bundle type, the Items created will be unique to that **general_bundle** type.

```


/** 
 * Implements hook_menu()
 */
function entity_forcontu_menu() {

    // Entity admin page.
    $items['admin/structure/entity_forcontu_item'] = array(
        'title' => 'Administer Items',
        'page callback' => 'entity_forcontu_item_admin_page',
        'access arguments' => array('administer entity_forcontu_item entities'),
    );

    //Default admin path.
    $items['admin/structure/entity_forcontu_item/list'] = array(
        'title' => 'List',
        'type' => MENU_DEFAULT_LOCAL_TASK,
        'weight' => -10,
    );

    // Entity display page.
    $items['item/%entity_forcontu_item'] = array(
        'title callback' => 'entity_forcontu_item_title',
        'title arguments' => array(1),
        'page callback' => 'entity_forcontu_item_view',
        'page arguments' => array(1),
        'access arguments' => array('view any entity_forcontu_item entity'),
    );

    // "View" path (shows a single entity).
    $items['item/%entity_forcontu_item/view'] = array(
        'title' => 'View',
        'type' => MENU_DEFAULT_LOCAL_TASK,
        'weight' => -10,
    );

    // "Edit" path (edits a single entity).
    $items['item/%entity_forcontu_item/edit'] = array(
        'title' => 'Edit',
        'page callback' => 'drupal_get_form',
        'page arguments' => array('entity_forcontu_item_form', 1),
        'access arguments' => array('edit any entity_forcontu_item entity'),
        'type' => MENU_LOCAL_TASK,
    );

    // Adds a new item.
    $items['item/add'] = array(
        'title' => 'Add a new Item entity',
        'page callback' => 'entity_forcontu_item_add',
        'access arguments' => array('create entity_forcontu_item entities'),
    );
}

return $items;
}


```

F51.6**Practical Case 51.1****Register the URLs**

We register the URLs involved in the management and display of the new entity type through hook_menu().

Step 5. Functions for loading entities.

The load functions allow one or more entities to be loaded from the database. Although we use the base API entity_load() function, we usually build these functions:

F51.7

- `module_entitytype_load()`. Loads an entity from the database.
- `module_entitytype_load_multiple()`. Loads multiple entities from the database.

F51.7**Practical Case 51.1****Load the entities.**

We define the load functions of the defined entity type.

```
/***
 * Load an object of the Item entity.
 *
 * @param $item_id
 * @param $reset
 *   Logical value indicating whether the cache is reset.
 * @return
 *   Object $item or FALSE if it has not been set.
 */
function entity_forcontu_item_load($item_id = NULL, $reset = FALSE) {
  $item_ids = ($item_id) ? array($item_id) : array();
  $item = entity_forcontu_item_load_multiple($item_ids, $reset);
  return $item ? reset($item) : FALSE;
}

/***
 * Load multiple Item entities.
 *
 * Calls the load() method of the entity controller.
 */
function entity_forcontu_item_load_multiple($item_ids = FALSE,
                                             $conditions = array(), $reset = FALSE) {
  return entity_load('entity_forcontu_item', $item_ids,
                     $conditions, $reset);
}
```

Step 6. Entity Controller.

Entities require a Controller class that will be responsible for executing the methods to create, save, load, and delete any entity.

The system default Controller class is `DrupalDefaultEntityController`, which is implemented by default by the interface `DrupalEntityControllerInterface`. We'll modify this class to suit our entity type. **F51.8**

The properties and methods in the base class are available at:

<http://api.drupal.org/api/drupal/includes--entity.inc/class/DrupalDefaultEntityController/7>

F51.8**Practical Case 51.1****Controller**

We implement the Controller with methods to create, save, and delete entities.

```
/***
 * Definition of EntityForcontuItemControllerInterface.
 *
 * We create the interface to inform other modules of the additional
 * methods added to the controller.
 */
interface EntityForcontuItemControllerInterface
  extends DrupalEntityControllerInterface {
  public function create();
  public function save($entity);
  public function delete($entity);
}

/***
 * EntityForcontuItemController extends DrupalDefaultEntityController.
 *
 * In the DrupalDefaultEntityController subclass we add
 * the methods create(), save(), and delete().
 */
class EntityForcontuItemController
  extends DrupalDefaultEntityController
  implements EntityForcontuItemControllerInterface {

  /**
   * Create and return an entity type entity_forcontu_item.
   */
}
```

```

public function create() {
    $entity = new stdClass();
    $entity->type = 'entity_forcontu_item';
    $entity->item_id = 0;
    $entity->bundle_type = 'general_bundle';
    $entity->item_description = '';
    return $entity;
}

/**
 * Save the custom fields of the entity
 * using drupal_write_record().
 */
public function save($entity) {
    // If the entity doesn't yet have an item_id, we get
    // the creation date.
    if (empty($entity->item_id)) {
        $entity->created = time();
    }
    // Invoke hook_entity_presave().
    module_invoke_all('entity_presave', 'entity_forcontu_item',
$entity);

    // We use drupal_write_record() to store the entity.
    // The 'PRIMARY_KEYS' argument determines whether it is an
    // insertion or an update (this generates the $item_id).
    $primary_keys = $entity->item_id ? 'item_id' : array();
    drupal_write_record('entity_forcontu_item', $entity, $primary_keys);

    // At this point one invokes hook_entity_update() or
    // hook_entity_insert(), depending on whether it is an
    // update or creation of an entity.

    // Also update or insert the associated fields
    // for the entity.
    if (empty($primary_keys)) { // inserción
        field_attach_insert('entity_forcontu_item', $entity);
        module_invoke_all('entity_insert', 'entity_forcontu_item', $entity);
    } else { // actualización
        field_attach_update('entity_forcontu_item', $entity);
        module_invoke_all('entity_update', 'entity_forcontu_item', $entity);
    }
    return $entity;
}

/**
 * Delete an entity.
 *
 * Use delete_multiple().
 */
public function delete($entity) {
    $this->delete_multiple(array($entity));
}

/**
 * Delete one or more entities.
 *
 * @param $item_ids
 *   Array with the IDs of the entity to delete.
 */
public function delete_multiple($entities) {
    $item_ids = array();
    if (!empty($entities)) {
        $transaction = db_transaction();
        try {
            foreach ($entities as $entity) {
                module_invoke_all('entity_forcontu_item_delete', $entity);
                // Invoke hook_entity_delete().
                module_invoke_all('entity_delete', $entity,
'entity_forcontu_item');
                field_attach_delete('entity_forcontu_item', $entity);
                $item_ids[] = $entity->item_id;
            }
            db_delete('entity_forcontu_item')
        }
    }
}

```

```

        ->condition('item_id', $item_ids, 'IN')
        ->execute();
    }
    catch (Exception $e) {
        $transaction->rollback();
        watchdog_exception('entity_forcontu', $e);
        throw $e;
    }
}
}
}
}

```

Step 7. Admin Panel

The admin panel is defined by the function that returns the URL defined in hook_menu(): [F51.9](#)

F51.9

Practical Case 51.1

Admin page

Registration of the entity's admin URL.

```

function entity_forcontu_menu() {
    // Admin panel of the entity
    $items['admin/structure/entity_forcontu_item'] = array(
        'title' => 'Administer Items',
        'page callback' => 'entity_forcontu_item_admin_page',
        'access arguments' => array('administer entity_forcontu_item entities'),
    );
}

```

The admin panel displays the following elements: [F51.10](#)

- A link to create the entity type Item.
- A table with the created Item entities. We'll implement this with the **entity_forcontu_item_list_entities()** function.
- The paths for **Manage fields** and **Manage display**. These tabs will be shown automatically, as we have already defined them in hook_entity_info().

F51.10

Practical Case 51.1

Admin page

Function that implements the admin page.

```

/**
 * Admin page.
 */
function entity_forcontu_item_admin_page() {
    $content = array();

    // Link for creating a new Item
    $content[] = array(
        '#type' => 'item',
        '#markup' => l(t('Add a new Item'), 'item/add'),
    );

    // Table of all of the Items created
    $content['table'] = entity_forcontu_item_list_entities();

    return $content;
}

/**
 * Renderable array (table style) with the entities created.
 */
function entity_forcontu_item_list_entities() {
    $content = array();

    $entities = entity_forcontu_item_load_multiple();
    if (!empty($entities)) {
        foreach ($entities as $entity) {

```

```

$rows[] = array(
  'data' => array(
    'id' => $entity->item_id,
    'item_description' => l($entity->item_description,
'item/' . $entity->item_id),
    'bundle' => $entity->bundle_type,
  ),
);
}
$content['entity_table'] = array(
  '#theme' => 'table',
  '#rows' => $rows,
  '#header' => array(t('ID'), t('Item Description'), t('Bundle')),
);
}
else {
  $content[] = array(
    '#type' => 'item',
    '#markup' => t('There aren\'t any Items.'),
  );
}
return $content;
}

```

At this point we can now have a look at the Admin module, available at: [F51.11](#)

Administration ⇒ Structure ⇒ Administer Items



ID	ITEM DESCRIPTION	BUNDLE
1	White T-shirt	general_bundle

URL Administer Items
/admin/structure/entity_forcontu_item

F51.11
Practical Case 51.1
Admin page

The admin page also shows the Manage Fields and Manage Display tabs.

Figure 51.11 shows a created Item (White shirt). To create Items continue with the following steps.

Step 8. Form for Creating and Editing an entity

To create the entity, we defined the URL “item/add” in hook_menu() using the return function entity_forcontu_item_add().

This function loads the form defined by **entity_forcontu_item_form()**, supplemented by the validation (**_validate**) and sent (**_submit**) functions.

We also initiate functions to save and delete the entity, which make corresponding calls to the controller methods.

```

/**
 * Function to add an entity.
 * Call the create() method of the controller and submit the form.
 */
function entity_forcontu_item_add() {
  // Calling the create() method creates an object Item that
  // will be passed to the validation and sent forms and functions.
  $entity = entity_get_controller('entity_forcontu_item')->create();
  return drupal_get_form('entity_forcontu_item_form', $entity);
}

```

F51.12
Practical Case 51.1
Create an entity
 Defines the form to edit/create an entity.

```


/**
 * Form to create (and edit) an entity entity_forcontu_item.
 */
function entity_forcontu_item_form($form, &$form_state, $entity) {

    // We define the table fields
    $form['item_description'] = array(
        '#type' => 'textfield',
        '#title' => t('Item Description'),
        '#required' => TRUE,
        '#default_value' => $entity->item_description,
    );
    $form['item_entity'] = array(
        '#type' => 'value',
        '#value' => $entity,
    );

    // We add the additional fields for the entity type
    field_attach_form('entity_forcontu_item', $entity, $form, $form_state);

    $form['submit'] = array(
        '#type' => 'submit',
        '#value' => t('Save'),
        '#weight' => 100,
    );
    $form['delete'] = array(
        '#type' => 'submit',
        '#value' => t('Delete'),
        '#submit' => array('entity_forcontu_item_edit_delete'),
        '#weight' => 200,
    );
}

return $form;
}

/**
 * Validation of the forms.
 * In addition to these validations we must add specific
 * validation of the fields added to the entity.
 */
function entity_forcontu_item_form_validate($form, &$form_state) {
    field_attach_form_validate('entity_forcontu_item',
    $form_state['values']['item_entity'], $form, $form_state);
}

/**
 * Sending the form (save the entity).
 */
function entity_forcontu_item_form_submit($form, &$form_state) {
    $entity = $form_state['values']['item_entity'];
    $entity->item_description =
    $form_state['values']['item_description'];
    field_attach_submit('entity_forcontu_item', $entity, $form,
    $form_state);
    $entity = entity_forcontu_item_save($entity);
    $form_state['redirect'] = 'item/' . $entity->item_id;
}

/**
 * Delete the entity
 */
function entity_forcontu_item_edit_delete( $form , &$form_state ) {
    $entity = $form_state['values']['item_entity'];
    entity_forcontu_item_delete($entity);
    drupal_set_message(t('The entity %item_description (ID %id) has
been deleted'),


```

```

        array('%item_description' => $entity->item_description, '%id'
=> $entity->item_id)
    );
    $form_state['redirect'] =
=admin/structure/entity_forcontu_item/';
}

/**
 * Call the save() controller method.
 */
function entity_forcontu_item_save(&$entity) {
    return entity_get_controller('entity_forcontu_item')-
>save($entity);
}

/**
 * Call the delete() controller method.
 */
function entity_forcontu_item_delete($entity) {
    entity_get_controller('entity_forcontu_item')->delete($entity);
}

```

Step 9. Function to return the URI elements

The entity URI elements (*Uniform Resource Identifier*) allow a distinct identification for each entity.

In **hook_entity_info()** we defined the function to return the URI elements ('uri callback') as `entity_forcontu_item_uri()`.

```
// Function that returns the entity's URI elements.
'uri callback' => 'entity_forcontu_item_uri',
```

This function returns the path of the entity. This path corresponds to the path defined in **hook_menu()** to display the entity. **F51.13**

```

/**
 * Implements the uri callback.
 */
function entity_forcontu_item_uri($item) {
    return array(
        'path' => 'item/' . $item->item_id,
    );
}

```

F51.13

Practical Case 51.1 URI elements

Returns the entity's URI elements ('path').

Step 10. Displaying the entity

In `hook_menu()` we defined the URL to display a single entity '`item/%entity_forcontu_item`' and its corresponding return function: `entity_forcontu_item_view()`. We also defined the function that returns the title of the page that displays the entity, `entity_forcontu_item_title()`. **F51.14**

F51.14**Practical Case 51.1**
Show the entity

A function that returns the entity title and a function that shows the entity.

```
/*
 * Return function that returns the title of the entity.
 */
function entity_forcontu_item_title($entity) {
    return t('Item: @item_description',
        array('@item_description' => $entity->item_description));
}

/*
 * Displays the page of the entity.
 */
function entity_forcontu_item_view($entity, $view_mode = 'full') {

    $entity->content = array(
        '#view_mode' => $view_mode,
    );

    // The presentation is done through field API
    field_attach_prepare_view('entity_forcontu_item',
        array($entity->item_id => $entity), $view_mode);
    entity_prepare_view('entity_forcontu_item',
        array($entity->item_id => $entity));
    $entity->content = field_attach_view('entity_forcontu_item',
        $entity, $view_mode);

    // Presentation of the entity fields
    $entity->content['created'] = array(
        '#type' => 'item',
        '#title' => t('Created date'),
        '#markup' => format_date($entity->created),
    );
    $entity->content['item_description'] = array(
        '#type' => 'item',
        '#title' => t('Item Description'),
        '#markup' => $entity->item_description,
    );

    $type = 'entity_forcontu_item';
    drupal_alter(array('entity_forcontu_item_view', 'entity_view'),
        $entity->content, $type);

    return $entity->content;
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Figure F51.15 shows the form for creating a new entity.

F51.15**Practical Case 51.1****Form for creating an entity**

A form defined to create the entity.

Add a new Item entity

Item Description *

White T-shirt

Save

Delete

Item: White T-shirt

[View](#) [Edit](#)

Item Description

White T-shirt

Created date

Sun, 02/23/2014 - 08:57

F51.16

Practical Case 51.1

Item entity

Presentation of the entity type Item.

Step 11. Fields and Display Management

As discussed, the tabs to **Manage fields** and **Manage display**, which we have defined in `hook_entity_info()`, will automatically be displayed on the administration page.

The **Item description** and **Creation date** fields, created directly in the entity, will not be displayed in the Management fields list so that we cannot set, for example, the order of added fields. Implementing the `hook_field_extra_fields()` allows us to indicate that additional fields to display in the Manage fields tab (`$form_elements`) and those to display in the Manage display tab (`$display_elements`).

In this example, we'll show both fields in the two tabs. [F51.17](#)

```
/***
 * Implements hook_field_extra_fields()
 */
function entity_forcontu_field_extra_fields() {
  $form_elements['created'] = array(
    'label' => t('Creation date'),
    'description' => t('Creation date'),
    'weight' => -10,
  );
  $form_elements['item_description'] = array(
    'label' => t('Item Description'),
    'description' => t('Item Description'),
    'weight' => -5,
  );

  $display_elements['created'] = array(
    'label' => t('Creation date'),
    'description' => t('Creation date (an extra display field)'),
    'weight' => -10,
  );
  $display_elements['item_description'] = array(
    'label' => t('Item Description'),
    'description' => t('Just like title, but trying to point out that
it is a separate property'),
    'weight' => -5,
  );

  $extra_fields['entity_forcontu_item']['general_bundle']['form'] = $form_elements;
  $extra_fields['entity_forcontu_item']['general_bundle']['display'] = $display_elements;

  return $extra_fields;
}
```

F51.17

Practical Case 51.1

hook_field_extra_fields()

We add the entity fields to management and display fields list. Although we can't configure them, we can modify their order.

The result is shown in [Figure F51.18](#). The added fields can be ordered, but other operations cannot be performed. Then, from the display tab we can decide if it is to be visible or hidden.

F51.18**Practical Case 51.1****Management fields**

Adding entity type fields to the Manage fields list.

LABEL	MACHINE NAME	FIELD TYPE	WIDGET	OPERATIONS
+ Creation date	created	Creation date		
+ Item Description	item_description	Item Description		
+ Add new field		- Select a field type - Label	- Select a widget - Type of data to store.	Show row weights Form element to edit the data.
+ Add existing field		- Select an existing field - Label	- Select a widget - Field to share	

Step 12. Permissions module

To finalize development, **F51.19** we need to initiate **hook_permission()** to create permissions for what we've referenced in the different functions of the module.

F51.19**Practical Case 51.1****Permissions**

Defining the module's permissions with **hook_permission()**.

```
/**
 * Implements hook_permission()
 */
function entity_forcontu_permission() {
  $permissions = array(
    'administer entity_forcontu_item entities' => array(
      'title' => t('Administer Item entities'),
    ),
    'view any entity_forcontu_item entity' => array(
      'title' => t('View any Item entity'),
    ),
    'edit any entity_forcontu_item entity' => array(
      'title' => t('Edit any Item entity'),
    ),
    'create entity_forcontu_item entities' => array(
      'title' => t('Create Item Entities'),
    ),
  );
  return $permissions;
}
```

Step 13. Adding fields to the entity type Item

We have defined the entity as 'fieldable'. This means that we can assign additional fields (with **Manage fields**) and control their presentation (with **Manage display**).

In our example, we'll add, from the Manage fields tab, a new field **Picture (field_item_image)**, an Image field type. From the Manage display tab we'll hide the image tag and modify the picture style (**medium**). This process was studied in the beginner and intermediate levels of the course. **F51.20**

Home > Administration > Structure > Administer Items

Administer Items

F51.20

Practical Case 51.1

Adding fields

We can add additional fields from Manage fields and modify how they are displayed from Manage Display.

FIELD	LABEL	FORMAT
+ Creation date	Visible	
+ Item Description	Visible	
+ Body	<Hidden>	Default
+ Image	<Hidden>	Image

Show row weights

Hidden

No field is hidden.

CUSTOM DISPLAY SETTINGS

Save

On editing (or creating) an Item, the element corresponding to the added field will display on the form. **F51.21**

Item: White T-shirt

[View](#)

[Edit](#)

Item Description *

White T-shirt

Imagen



whitetshirt.jpg (13.18 KB)

[Remove](#)

[Save](#)

[Delete](#)

Once saved, the Item page with the newly uploaded image will be displayed. **F51.22**

Item: White T-shirt

[View](#)

[Edit](#)

Created date

Sun, 02/23/2014 - 09:07



Item Description

White T-shirt

F51.21

Practical Case 51.1

Additional field

And Item entity to which we added an image field.

F51.22

Practical Case 51.1

Display the Item entity

Introducing the Item entity type with the newly added image field type.

51.2

Creation of fields (Field API)

Through Field API we can add custom fields to the entities.

<http://api.drupal.org/api/drupal/modules--field--field.module/group/field/7>

We have already seen that with entity types defined as 'fieldable', we can create and attach additional fields through the **Manage fields** tab. This is possible thanks to **Field API**, a set of functions that will enable us to define fields and attach them to any defined entity type.

In this section we'll learn **how to create defined field types** using core or contributed modules. In the next section we'll learn how to create new types of fields and widgets.

To create a new field and assign it to an entity type, we use two field API functions:

- **field_create_field()**. Creates the field.
- **field_create_instance()**. Creates an instance of the defined field and assigns it to a group of fields (bundle) of the content type.

Function `field_create_field()`

The function **field_create_field()** creates a field.

field_create_field(\$field)

http://api.drupal.org/api/drupal/modules!field!field.crud.inc/function/field_create_field/7

The parameter \$field passes to the function an array of defined fields, with the following elements:

- '**field_name**' (required). System name of the field. It's a unique value with a maximum of 32 characters.
- '**type**' (required). System name of the field type.
- '**entity_types**'. Array of the entity types that can have instances of this field.
- '**cardinality**'. The number of values that the field can have. The default value is 1. May contain any positive integer value or the value FIELD_CARDINALITY_UNLIMITED (-1) to indicate an unlimited value.
- '**locked**'. The default value is FALSE. Indicates that the field is not locked. When locked (TRUE), the user cannot modify your settings or create new instances of the field from the user interface.
- '**translatable**'. Indicates that the field is translatable.
- '**settings**'. We can specify additional configuration options of the field, which depend on the particular definition of each.
- '**storage**'. Storage options.

The function returns the array \$field with the property '**id**' completed.

The field type is the system name assigned to the field. Each module defines its

field types by implementing **hook_field_info()** so we can look for the implemented field types in these functions. In the field definition we also find the configuration settings ('settings') that can be used.

Some of the available field types are:

- **Text** Module (**text_field_info()**):
 - o 'text'. Varchar text type.
 - o 'text_long'. Long text.
 - o 'text_with_summary'. Long text with summary.
- **Number** module (**number_field_info()**):
 - o 'number_integer'. Whole Number.
 - o 'number_decimal'. Decimal number.
 - o 'number_float'. Floating point number.
- **List** Module (**list_field_info()**):
 - o 'list_integer'. List of whole numbers.
 - o 'list_float'. List of floating point elements.
 - o 'list_text'. List of text elements.
 - o 'list_boolean'. List of boolean elements.
- **File** module (**file_field_info()**):
 - o 'file'. File.
- **Image** module. (**image_field_info()**):
 - o 'image'. Image.
- **Taxonomy** module. (**taxonomy_field_info()**):
 - o 'taxonomy_term_reference'. References a taxonomy term.

We can also find additional modules that implement field types. For example, the Date Module provides the following field types:

- **Date** Module (**date_field_info()**):
 - o datetime. Stores a date as a datetime field in the database.
 - o date. Stores a date as an ISO date field in the database.
 - o timestamp. Stores a date as a timestamp field in the database

The complete list of available fields can be found at [Administration](#) ⇒ [Reports](#) ⇒ [List of fields](#)

Remember that if the module we are developing uses fields implemented by other modules, we must indicate these dependencies on other modules in the file .info.

As an example of using the **field_create_field()** function, we'll analyze the **node_add_body_field()** function used by the Node module to add the field Body to the nodes. [F51.23](#)

```
$field = array(
  'field_name' => 'body',
  'type' => 'text_with_summary',
  'entity_types' => array('node'),
);
$field = field_create_field($field);
```

[F51.23](#)

Create a field

In the example a new field is created by calling **field_create_field()**.

We have created a field called 'body', of the **full text with summary** type ('**text_with_summary**'), which will be available in the 'Node' entity types.

field_create_instance() Function

So that the field can be assigned to an entity type and a determined set of fields (bundle), we create an instance of the field by calling the **field_create_instance()** function.

field_create_instance(\$instance)

http://api.drupal.org/api/drupal/modules--field--field.crud.inc/function/field_create_instance/7

The parameter \$instance passed to the function is an array containing the definition of the instance with the following items:

- '**field_name**' (required). System name of the field.
- '**entity_type**' (required). System name of the entity type.
- '**bundle**' (required). Group of fields, or bundle, among those defined in the entity type. For example, in the entity 'node' each content-type is a bundle.
- '**label**'. Name of the field as it will be displayed to users.
- '**description**'. Field description.
- '**required**'. Indicates if the field is required. By default, FALSE.
- '**default_value_function**'. We can specify a function that returns the default value for the field.
- '**settings**'. Configuration options specific to the field type.
- '**widget**'. Type of widget element of the field (type) and options for widget configuration (settings).
- '**display**'. Display settings for each of the available display modes. It is an array with each display mode and within each array the display values of 'label' (shows how to display the field label), 'type' (field format), and 'settings' (for other configuration options).

The function returns the \$instance array with the completed 'id' property.

Returning to the **node_add_body_field()** function used by the **Node** module, we find the definition of the instance for each content type. First we define the **\$instance** array with the definition of the instance and, once completed, we make the call to **field_create_instance()**. **F51.24**

F51.24

Create an instance of a field

When we add a field to an entity type (and a certain group of fields), we are creating an instance of the field.

```
$instance = array(
  'field_name' => 'body',
  'entity_type' => 'node',
  'bundle' => $type->type,
  'label' => 'Body',
  'widget' => array('type' => 'text_textarea_with_summary'),
  'settings' => array('display_summary' => TRUE),
  'display' => array(
    'default' => array(
      'label' => 'hidden',
      'type' => 'text_default',
    ),
    'teaser' => array(
      'label' => 'hidden',
      'type' => 'text_summary_or_trimmed',
    ),
  ),
);
$instance = field_create_instance($instance);
```

When we make the call to **field_create_instance()**, the field indicated in 'field_name' must exist. That is why, generally, we will find the **field_create_field()** and **field_create_instance()** functions working together, the first being responsible for creating the field, and the second, for creating an instance of the same field, linked to one or more bundles of the entity type.

Information fields and instances

To find out if a field has been created, we can use the API **field_info_field()** function, which returns all of the field information starting with its name.

field_info_field(\$field_name)

http://api.drupal.org/api/drupal/modules!field!field.info.inc/function/field_info_field/7

To find out if an instance has been created, we can use the **field_info_instance()** function, which returns all of the instance information for a field and specific bundle.

field_info_instance(\$entity_type, \$field_name, \$bundle_name)

http://api.drupal.org/api/drupal/modules!field!field.info.inc/function/field_info_instance/7

As an example, let's look at the completed **node_add_body_field()** function, where these functions are also used: **F51.25**

```
/***
 * Add default body field to a node type.
 */
function node_add_body_field($type, $label = 'Body') {
  // Add or remove the body field, as needed.
  $field = field_info_field('body');
  $instance = field_info_instance('node', 'body', $type->type);
  if (empty($field)) {
    $field = array(
      'field_name' => 'body',
      'type' => 'text_with_summary',
      'entity_types' => array('node'),
    );
    $field = field_create_field($field);
  }
  if (empty($instance)) {
    $instance = array(
      'field_name' => 'body',
      'entity_type' => 'node',
      'bundle' => $type->type,
      'label' => $label,
      'widget' => array('type' => 'text_textarea_with_summary'),
      'settings' => array('display_summary' => TRUE),
      'display' => array(
        'default' => array(
          'label' => 'hidden',
          'type' => 'text_default',
        ),
        'teaser' => array(
          'label' => 'hidden',
          'type' => 'text_summary_or_trimmed',
        ),
      ),
    );
    $instance = field_create_instance($instance);
  }
  return $instance;
}
```

F51.25

**field_info_field() y
field_info_instance()**

Example where the **field_info_field()** and **field_info_instance()** functions are used to obtain information about the Body field type and its instance in the node entity and any of its content types (bundles).

Add fields to the entity

We already know how to create fields and instances, but where should we call these functions?

Whether our module adds a field to an existing entity type or to an entity type defined in the same module, we can create the field and its instance during the installation or activation of the module (**hook_install()** or **hook_enable()**).

For example, in the process of installing the module **Entity Forcontu**, we added a field '**item_body**' of the **full text with summary** ('text_with_summary') type. **F51.26**

F51.26

Adding fields

We will create fields and instances by installing the module using the `field_create_field()` and `field_create_instance()` modules.

```
/*
 * Implements hook_install().
 *
 */
function entity_forcontu_install() {
  $field = field_info_field('item_body');
  if (empty($field)) {
    $field = array(
      'field_name' => 'item_body',
      'type' => 'text_with_summary',
      'entity_types' => array('entity_forcontu_item'),
    );
    $field = field_create_field($field);
  }

  $instance = field_info_instance('entity_forcontu_item',
  'item_body', 'general_bundle');
  if (empty($instance)) {
    $instance = array(
      'field_name' => 'item_body',
      'entity_type' => 'entity_forcontu_item',
      'bundle' => 'general_bundle',
      'label' => 'Body',
      'widget' => array('type' => 'text_textarea_with_summary'),
      'settings' => array('display_summary' => TRUE),
      'display' => array(
        'default' => array(
          'label' => 'hidden',
          'type' => 'text_default',
        ),
        'full' => array(
          'label' => 'hidden',
          'type' => 'text_summary_or_trimmed',
        ),
      ),
    );
    $instance = field_create_instance($instance);
  }
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

The fields added in this way are directly stored by Field API, so we won't have to create additional fields in the base table of the entity.

Creation of new field types

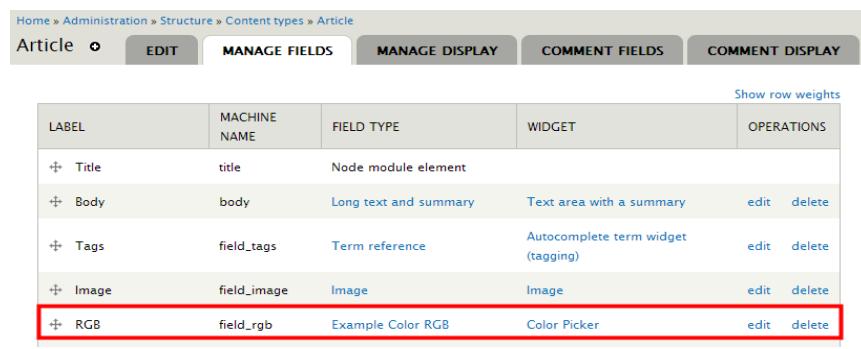
51.3

The creation of field types and widget types is carried out using Field Types API functions.

http://api.drupal.org/api/drupal/modules--field--field.api.php/group/field_types/7

In this section we'll review the **Field Example** module included in **Examples for Developers**. The module creates a new field type called **Example Color RGB** (field_example_rgb), which stores a color in RGB hexadecimal format (for example, #ff0011). The field will be able to be added to any entity type of the site from the corresponding **Manage fields** tab.

Building on this module we will review, step by step, each of the functions involved. **F51.27**



LABEL	MACHINE NAME	FIELD TYPE	WIDGET	OPERATIONS
>Title	title	Node module element		
Body	body	Long text and summary	Text area with a summary	edit delete
Tags	field_tags	Term reference	Autocomplete term widget (tagging)	edit delete
Image	field_image	Image	Image	edit delete
RGB	field_rgb	Example Color RGB	Color Picker	edit delete

F51.27

New type of field

Example of the new RGB Color Example field type, added to the Article content type.

A field consists of three main elements:

- **Field type.** In addition to configuration options common to all fields, we have to specify its structure in the database, defining the additional fields that compose it.

In our example we'll define a field in the 'rgb' table that will store a string of 7 characters.

- **Widgets.** A widget determines how the field value is entered. A field type can have several different widgets.

In our example field, the available widgets are:

- o **RGB value as #fffff.** This is a text field where the user must specify the color value (in the format #ff0011).
- o **RGB text field.** Three text fields are displayed to enter the separate RGB values in hexadecimal format (R = ff, G = 00, B = 11).
- o **Color Picker.** This uses a jQuery plugin to show a simple visual color selector. **Figure F51.28** shows this widget in operation.

Regardless of which widget is used to indicate the field value, the value is stored in the database in the same way, as a string of 7 characters,

such as #ff0011.

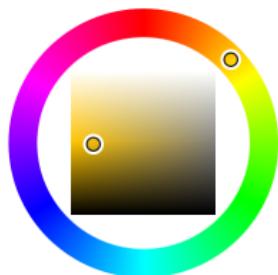
F51.28

Control Color Picker

In the newly defined field type we added a Color Picker control type, using a jQuery plugin.

RGB

#e7b713



- **Formats.** The format of the field display is accessed from the **Manage Display** tab. This determines the form in which the field value will display on showing the entity.

In our example field, we can select one of these formats: **F51.29**

- o **Simple text-based formatter.** This format shows the value of the field as text colored according to its field value.
- o **Change the background color of the text (Change the background of the output text).** This format changes the background color of the generated text according to the inserted field color. **F51.30**

F51.29

Display formats

We have defined two formats for the field type.

Home » Administration » Structure » Content types » Article
Article EDIT MANAGE FIELDS **MANAGE DISPLAY** COMMENT FIELDS

Content items can be displayed using different view modes: Teaser, Full content, Print, RSS, etc. *Teaser* is typically used in lists of multiple content items. *Full content* is typically used when the content is displayed or Here, you can define which fields are shown and hidden when *Article* content is displayed in each view mode. fields are displayed in each view mode.

FIELD	LABEL	FORMAT		
+ Image	<Hidden>	Image	Image style: Large	
+ Body	<Hidden>	Default		
+ Tags	Above	Link		
+ RGB	Above	<div style="border: 1px solid red; padding: 5px;"> <div style="background-color: #f0f0f0; border-bottom: 1px solid #ccc; padding-bottom: 5px;">Simple text-based formatter</div> <div style="background-color: #f0f0f0; border-bottom: 1px solid #ccc; padding-bottom: 5px;">Simple text-based formatter</div> <div style="background-color: #f0f0f0; border-bottom: 1px solid #ccc; padding-bottom: 5px;">Change the background of the output text</div> <div style="background-color: #f0f0f0; padding-bottom: 5px;"><Hidden></div> </div>		
Hidden				
No field is hidden.				

Using RGB field

[View](#) [Edit](#) [Devel](#)

Submitted by admin on Sun, 02/23/2014 - 09:19

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed suscipit ut risus ut sodales. Nulla nec mauris a quam comm Donec porttitor nulla tempor leo blandit, in gravida sem vestibulum. Fusce posuere, magna nec ultrices sollicitudin, null tortor, in ornare libero ante non magna. Quisque odio ligula, porttitor ut leo et, consequat ultricies risus.

Donec auctor sed justo a blandit. Ut iaculis lectus felis, at porta est rutrum eu. Morbi sodales nisi elit. Pellentesque et scel Aliquam pulvinar diam sapien, interdum dapibus ante facilisis id. Nullam porta bibendum urna, vel tristique quam vehi Vestibulum ac urna fringilla, vehicula lectus ut, convallis mauris. Fusce blandit quis orci a fringilla. Mauris magna liber venenatis sed, viverra in lorem. Suspendisse tincidunt massa dui. Mauris lacinia viverra odio adipiscing posuere. Nam ac dignissim ipsum ac, fermentum purus. Proin semper convallis massa, sed porta purus imperdiet id.

RGB:

The content area color has been changed to #e7b713

F51.30

Field display

Example of content display on applying the format "Change the background color of the text".

Step 1. Defining the field with hook_field_info()

The first step is to define the new field by implement the **hook_field_info()** function. [F51.31](#)

http://api.drupal.org/api/drupal/modules--field--field.api.php/function/hook_field_info/7

This function returns an array with the definition of the new field types. A module can define one or more fields, and for each field it can specify the following values:

- **'label'**. Name of the field type as displayed to users.
- **'description'**. Short description of the field type.
- **'settings'**. Array with the configuration options of the field.
- **'instance_settings'**. Array with the configuration options and the default values for the field instances.
- **'default_widget'**. System name of the default widget used in the field instances.
- **'default_formatter'**. System name of the default format used in the field instances.
- **'no_ui'**. If the value is TRUE, it indicates that users cannot create fields of this type from the admin interface. We use this option when creating field types for exclusive use by the programming module.

```
/** 
 * Implements hook_field_info().
 */
function field_example_field_info() {
  return array(
    'field_example_rgb' => array(
      'label' => t('Example Color RGB'),
      'description' => t('Demonstrates a field composed of an RGB color.'),
      'default_widget' => 'field_example_3text',
      'default_formatter' => 'field_example_simple_text',
    ),
  );
}
```

F51.31

Field definition

The function hook_field_info() defines the new field.

If we would like to use a field type defined by another module, we can implement **hook_field_info_alter()**, passing as an array the argument \$info with the field type values that we want to override.

http://api.drupal.org/api/drupal/modules--field--field.api.php/function/hook_field_info_alter/7

Step 2. Creating the field table with hook_field_schema()

By implementing **hook_field_schema()** we define the structure of the field in the database. The manner in which the schema is defined is similar to the method for defining the database through **hook_schema()** (Schema API).

http://api.drupal.org/api/drupal/modules--field--field.api.php/function/hook_field_schema/7

Implementing **hook_field_schema()** is performed in the module installation file (**field_example.install**).

The field type only requires the 'rgb' column, which is a text string (varchar) of 7 characters. **F51.32**

F51.32

hook_field_schema()

Field structure is defined in the database by **hook_field_schema()**. You need an "rgb" field that will store a string of 7 characters with the hexadecimal color value.

```
/***
 * Implements hook_field_schema().
 */
function field_example_field_schema($field) {
  $columns = array(
    'rgb' => array('type' => 'varchar', 'length' => 7, 'not null' => FALSE),
  );
  $indexes = array(
    'rgb' => array('rgb'),
  );
  return array(
    'columns' => $columns,
    'indexes' => $indexes,
  );
}
```

Step 3. Field validation with hook_field_validate()

Through **hook_field_validate()** we can verify that the value entered in the field is correct.

http://api.drupal.org/api/drupal/modules--field--field.api.php/function/hook_field_validate/7

The field 'rgb' must have the format #rrggb. The allowed characters to define colors are 0-9 and a-f (lowercase). **F51.33** The string must start with the character #.

F51.33

Field validation

Initiating the function **hook_field_validate()** enables us to check if the field value is correct.

```
/***
 * Implements hook_field_validate().
 */
function field_example_field_validate($entity_type, $entity,
$field, $instance, $langcode, $items, &$errors) {
  foreach ($items as $delta => $item) {
    if (!empty($item['rgb'])) {
      if (! preg_match('@^#[0-9a-f]{6}$@', $item['rgb'])) {
        $errors[$field['field_name']][$langcode][$delta][] = array(
          'error' => 'field_example_invalid',
          'message' => t('Color must be in the HTML format #abcdef.'),
        );
      }
    }
  }
}
```

The errors generated in `hook_field_validate()` can be handled by implementing `hook_field_widget_error()`.

Step 4. Empty field with `hook_field_is_empty()`

Implementing `hook_field_is_empty()` we'll define when the field is considered empty. **F51.34**

http://api.drupal.org/api/drupal/modules--field.field.api.php/function/hook_field_is_empty/7

```
/***
 * Implements hook_field_is_empty().
 */
function field_example_field_is_empty($item, $field) {
  return empty($item['rgb']);
}
```

F51.34**hook_field_is_empty()**

Define what constitutes an empty field.

Step 5. Defining formats with `hook_field_formatter_info()`

Using `hook_field_formatter_info()` we define the formats available for the field type. The format selected to instantiate the field defines how the value of the field is presented. **F51.35**

http://api.drupal.org/api/drupal/modules--field.field.api.php/function/hook_field_formatter_info/7

In our field type we define two possible formats:

- **Simple text-based formatter.** This format shows the value of the field as text colored according to its field value.
The format of the system name is 'field_example_simple_text'.
- **Change the background color of the text.** This format changes the background color of the generated text according to the inserted field color.
The format of the system name is 'field_example_color_background'.

```
/***
 * Implements hook_field_formatter_info().
 */
function field_example_field_formatter_info() {
  return array(
    // Show the text in a specified color.
    'field_example_simple_text' => array(
      'label' => t('Simple text-based formatter'),
      'field types' => array('field_example_rgb'),
    ),
    // Display the background container in the specified color.
    'field_example_color_background' => array(
      'label' => t('Change the background of the output text'),
      'field types' => array('field_example_rgb'),
    ),
  );
}
```

F51.35**Field formats**

Define the formats available for the field.

In addition to declaring the display formats, we must implement how the formatting is applied at the content level. We do this through the `hook_field_formatter_view()` function.

http://api.drupal.org/api/drupal/modules--field.field.api.php/function/hook_field_formatter_view/7

The function returns a renderable array with the field display of the chosen format. This array will be rendered later, leading to the final HTML output.

- The format '**field_example_simple_text**' applies the text color style to the paragraph (<p>) that displays the value of the field. The final HTML output will have this structure:

```
<p style="color: #e7b713"> The color code in this field is  
#e7b713</p>
```

- The format "field_example_color_background" adds a CSS style class to apply to div.region-content. The generated HTML output will have this structure:

```
<style type="text/css" media="all">  
div.region-content{  
background-color:#e7b713;  
}  
</style>
```

The following shows the implementation of hook_field_formatter_view().

F51.36

F51.36

Display according to the format

Define how the field will display for each one of the defined formats.

```
/**  
 * Implements hook_field_formatter_view().  
 */  
function field_example_field_formatter_view($entity_type, $entity,  
$field, $instance, $langcode, $items, $display) {  
$element = array();  
  
switch ($display['type']) {  
case 'field_example_simple_text':  
foreach ($items as $delta => $item) {  
$element[$delta] = array(  
'#type' => 'html_tag',  
'#tag' => 'p',  
'#attributes' => array(  
'style' => 'color: ' . $item['rgb'],  
,  
'#value' => t('The color code in this field is @code',  
array('@code' => $item['rgb'])),  
);  
}  
break;  
  
case 'field_example_color_background':  
foreach ($items as $delta => $item) {  
$element[$delta] = array(  
'#type' => 'html_tag',  
'#tag' => 'p',  
'#value' => t('The content area color has been changed  
to @code',  
array('@code' => $item['rgb'])),  
'#attached' => array(  
'css' => array(  
array(  
'data' => 'div.region-content { background-color:'  
. $item['rgb'] . ';}',  
'type' => 'inline',  
,  
),  
,  
));  
}  
break;  
}  
return $element;  
}
```

We can also modify the formats defined by other modules using **hook_field_formatter_info_alter()**.

http://api.drupal.org/api/drupal/modules--field-field.api.php/function/hook_field_formatter_info_alter/7

Step 6. Defining widgets with hook_field_widget_info()

In this step we begin by defining the widgets. We start by implementing **hook_field_widget_info()**, with which we state the widgets available for the field type.

http://api.drupal.org/api/drupal/modules--field--field.api.php/function/hook_field_widget_info/7

Our field type will have three possible widgets. F51.37

- **RGB value as #ffffff** ('field_example_text'). This is a text field where the user must specify the color value (in the format #ff0011).
- **RGB text field** ('field_example_3text'). This shows three text fields to enter the separate RGB values in hexadecimal format (R = ff, G = 00, B = 11).
- **Color Picker** ('field_example_colorpicker'). It uses a jQuery plugin to show a simple visual color selector. This plugin, already included in the module **Field Example** with the name **field_example.js**, can be downloaded at:

<http://acko.net/blog/farbtastic-jquery-color-picker-plug-in/>

```
/***
 * Implements hook_field_widget_info().
 */
function field_example_field_widget_info() {
  return array(
    'field_example_text' => array(
      'label' => t('RGB value as #ffffff'),
      'field types' => array('field_example_rgb'),
    ),
    'field_example_3text' => array(
      'label' => t('RGB text field'),
      'field types' => array('field_example_rgb'),
    ),
    'field_example_colorpicker' => array(
      'label' => t('Color Picker'),
      'field types' => array('field_example_rgb'),
    ),
  );
}
```

F51.37

Field controls

Define the controls that will be available to fill the field.

Step 7. Forms with hook_field_widget_form()

For each widget we must define the form that is displayed to the user when creating or editing the entity. The form that corresponds to the selected widget alone is displayed in the field type instance. **F51.38**

- The widgets '**field_example_text**' and '**field_example_colorpicker**' consist of a single form field of the textfield type. The only difference is that with the second we add the library that allows us to use the jQuery plugin.
- The widget '**field_example_3text**' is composed of three fields of the textfield type, where the codes for each color are separately indicated.

F51.38

Control form

For each control, we define the form that is displayed when you create an entity that contains an instance of the field.

Note

In the function we first define the specific part of the '**field_example_colorpicker**' control, but do not add the **break** command at the end of the **case** statement, so that the form defined in **case 'field_example_text'** applies in both cases, to '**field_example_colorpicker**' control and to '**field_example_text**' control. The '**field_example_3text**' control is independent of the other two.

```
/***
 * Implements hook_field_widget_form().
 */
function field_example_field_widget_form(&$form, &$form_state, $field,
$instance, $langcode, $items, $delta, $element) {
  $value = isset($items[$delta]['rgb']) ? $items[$delta]['rgb'] : '';
  $widget = $element;
  $widget['#delta'] = $delta;

  switch ($instance['widget']['type']) {
    // Color picker form
    case 'field_example_colorpicker':
      $widget += array(
        '#suffix' => '<div class="field-example-colorpicker"></div>',
        '#attributes' => array('class' => array('edit-field-example-colorpicker')),
        '#attached' => array(
          // Add Farbtastic color picker.
          'library' => array(
            array('system', 'farbtastic'),
          ),
          // Add javascript to trigger the colorpicker.
          'js' => array(drupal_get_path('module', 'field_example') .
            '/field_example.js'),
        ),
      );
    }

    // Simple text form (and Color Picker).
    // We don't use "break", the color picker form also uses
    // the definition of "field_example_text" below.
    case 'field_example_text':
      $widget += array(
        '#type' => 'textfield',
        '#default_value' => $value,
        '#size' => 7,
        '#maxlength' => 7,
      );
      break;

    // Formulate the text in RGB format.
    case 'field_example_3text':
      // Converts the rgb value into values r, g, and b.
      if (!empty($value)) {
        preg_match_all('@..@', substr($value, 1), $match);
      }
      else {
        $match = array(array());
      }

      // Creates a fieldset that groups the 3 fields.
      $widget += array(
        '#type' => 'fieldset',
        '#element_validate' => array('field_example_3text_validate'),
        '#delta' => $delta,
        '#attached' => array(
          'css' => array(drupal_get_path('module', 'field_example') .
            '/field_example.css'),
        );
  }
}
```

```

        ),
    );

    // Creates a textfield for each of the Red, Green and Blue value.
    foreach (array('r' => t('Red'), 'g' => t('Green'), 'b' => t('Blue')) as $key => $title) {
      $widget[$key] = array(
        '#type' => 'textfield',
        '#title' => $title,
        '#size' => 2,
        '#default_value' => array_shift($match[0]),
        '#attributes' => array('class' => array('rgb-entry')) ,
      );
    }
    break;
}
$element['rgb'] = $widget;
return $element;
}

```

In the widget 'field_example_3text' a validation function has been defined called 'field_example_3text_validate' that is required to return the value of the three individual fields as a single field with the value of the color entered in the final format that is stored in the database data (#abcdef). **F51.39**

```

/**
 * Validate the individual fields and convert the output
 * to a single field with HTML RGB format
 */
function field_example_3text_validate($element, &$form_state) {
  $delta = $element['#delta'];
  $field = $form_state['field'][$element['#field_name']][$element['#language']]['field'];
  $field_name = $field['field_name'];
  if (isset($form_state['values'][$field_name][$element['#language']][$delta]['rgb'])) {
    $values = $form_state['values'][$field_name][$element['#language']][$delta]['rgb'];
    foreach (array('r', 'g', 'b') as $colorfield) {
      $colorfield_value = hexdec($values[$colorfield]);
      if (strlen($values[$colorfield]) == 0) {
        form_set_value($element, '', $form_state);
        return;
      }
      if ((strlen($values[$colorfield]) != 2) || $colorfield_value < 0 || $colorfield_value > 255) {
        form_error($element[$colorfield], t('Saturation value must be a 2-digit hexadecimal value between 00 and ff.'));
      }
    }
    $value = sprintf('#%02s%02s%02s', $values['r'], $values['g'], $values['b']);
    form_set_value($element, $value, $form_state);
  }
}

```

F51.39**Field validation**

Field validation function for the control of the 'field_example_3text' type.

Figure F51.40 shows the widget type selector for creating an instance of the field.



The screenshot shows a 'CHANGE WIDGET' dialog box. At the top, there's a 'Widget type' dropdown menu with the following options: 'Color Picker' (which is currently selected and highlighted with a red border), 'RGB text field', and 'RGB value as #ffffff'. Below the dropdown, there is some descriptive text about what the user would like to present to the user when creating this field in the Article type. At the bottom of the dialog are 'Continue' and 'Cancel' buttons.

F51.40**Control type**

From the Control type you can select the control that is displayed to the user when creating an entity.

51.4 Entity API Module

The **Entity API** module extends the Drupal core API by providing new features to facilitate the handling of new entity types.

The Entity API module is available at: <http://drupal.org/project/entity>

You can find additional documentation about how to use Entity API at:

<http://drupal.org/node/878784>

This module includes, within the **tests** folder, the **entity_test** module with examples of Entity API use. It is also important to analyze other modules that make use of Entity API, such as **Profile2**, **Drupal Commerce** or **Field Collection**.

If the module is developed using Entity API, do not forget to add the corresponding unit to the .info file.

In the following example we will illustrate using the code provided by the **entity_test** module.

Define new types of entities.

As discussed in **Section 51.1**, to define a new type of entity we implement **hook_entity_info()**. **F51.41**

F51.41

hook_entity_info()

Changes in the definition of the entity type in **hook_entity_info()** to implement the advantages provided by Entity API.

```
/**
 * Implements hook_entity_info().
 */
function entity_test_entity_info() {
  $return = array(
    'entity_test' => array(
      'label' => t('Test Entity'),
      'plural label' => t('Test Entities'),
      'description' => t('An entity type used by the entity API tests.'),
      'entity class' => 'EntityClass',
      'controller class' => 'EntityAPIController',
      'base table' => 'entity_test',
      'fieldable' => TRUE,
      'entity keys' => array(
        'id' => 'pid',
        'bundle' => 'name',
      ),
      'label callback' => 'entity_class_label',
      'uri callback' => 'entity_class_uri',
      'bundles' => array(),
      'bundle keys' => array(
        'bundle' => 'name',
      ),
    ),
  //...
}
```

Here are some of the differences that have been added to make use of Entity API:

- **'entity class'**. We specify the Entity class or a class that extends it (in our example, EntityClass extends Entity).

- '**controller class**'. We will use the controller class provided by the module, **EntityAPIController**. This class extends the controller used by the core, providing these additional methods: create, delete, export, import, invoke, load, save, view, etc.

<http://drupalcontrib.org/api/drupal/contributions--entity--includes--entity.controller.inc/class/EntityAPIController/7>

- '**label callback**'. Specifying "entity_class_label" uses the entity_class_label() as a return function. This allows us to access the entity tab from \$entity->label().
- '**uri callback**'. Specifying 'entity_class_uri' uses the entity_class_uri() function as a return function. This allows us to access the URI elements of the entity from \$entity->uri().

Show the entity

Additional fields are directly integrated into the display of the entity. If we want to add additional content to the output, we can overwrite the **buildContent()** method of the controller. **F51.42**

```
/***
 * Main class for test entities.
 */
class EntityClass extends Entity {

  public function __construct(array $values = array(), $entityType = NULL) {
    parent::__construct($values, 'entity_test');
  }

  /**
   * Override buildContent() to add content to the output.
   */
  public function buildContent($view_mode = 'full', $langcode = NULL) {
    $content['user'] = array(
      '#markup' => "User: ". format_username(user_load($this->uid)),
    );
    return entity_get_controller($this->entityType)->buildContent($this,
$view_mode, $langcode, $content);
  }
}
```

F51.42

Controller

To modify the output of the entity, we can override the buildContent() method of the controller.

Integration with views

In order for the defined entity type to be viewable, we simply need to add the parameter '**module**' with the name of the system module during the implementation of **hook_entity_info()**. **F51.43**

```
/***
 * Implements hook_entity_info().
 */
function entity_test_entity_info() {
  $return = array(
    'entity_test' => array(
      ...
      'controller class' => 'EntityAPIController',
      'module' => 'entity_test',
      ...
    ),
  //...
}
```

F51.43

Integration with views

Entity API facilitates integration with views. We'll have to add the parameter 'module' in the definition of the entity.

Entity API loads this form to assign the views controller by default, **EntityDefaultViewsController()**.

If we extend the views controller, we can explicitly control the behavior through '**views controller class**': **F51.44**

F51.44**Views controller**

We can use the default views controller or override it with the controller.

```
/***
 * Implements hook_entity_info().
 */
function entity_test_entity_info() {
  $return = array(
    'entity_test' => array(
      ...
      'controller class' => 'EntityAPIController',
      'views controller class' => 'EntityDefaultViewsController',
      'module' => 'entity_test',
      ...
    ),
  //...
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

51.5**Model Entities module**

The implementation of entities can be more or less complex depending on the specific requirements needed. Regardless, the number of required functions is high and its development can be quite tedious.

The **Model Entities** module tries to facilitate the creation of entities through a module that can be used as a base or template to implement our entities. It is a fully functional module, based on Entity API, to create the **Model** entity type.

The **Model Entities** module is available at:

<http://drupal.org/project/model>

Let's install the module in order to understand the scope of the developed software, but keep in mind that this is only an example of a module that can serve as a model for implementing entities.

URL Model Types

/admin/structure/
model_types

F51.45**Model Entities**

The module generates a Model entity type.

Administration ⇒ **Structure** ⇒ **Model Types**

The module allows you to build new types derived from the **Model** type. Each of these types will have their own fields and display manager. **F51.45**

LABEL	STATUS	OPERATIONS
type1 (Machine name: type1)	Custom	edit manage fields manage display clone delete export

To add Model type content, the module implements the Models tab in: **F51.46**

Administration ⇒ **Content [Models tab]**

URL Models
`/admin/content/models`

From here we can create new content using the "Add a model" link.

MODEL ID	NAME	VIEW	OPERATIONS LINKS
1	Entity 1	view	Edit Delete
2	Entity 2	view	Edit Delete

F51.46

Model Entities

Model entities types are created to be shown from the content manager.

The created entities will be fully operational and can be viewed, edited, or deleted. **F51.47** **F51.48**

Model Name *
Entity 1

An interesting model switch

Description
Entity 1

Save model Delete model Cancel

F51.47

Create an entity

Form to create the entity, with sample fields.

[Home](#)

Entity 1

Description:

Entity 1

Model Sample Data: Switch On

F51.48

Display the entity

Display the Model entity type.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

52 Programming actions and triggers

In this unit we will see step by step how to implement simple and advanced actions and triggers.

Through hook_cron () we can include actions to be carried out periodically, upon each execution of the system's cron.

Implementing hook_watchdog () and using the watchdog () function we can record everything that happens on the site in the event log.

Finally, we will see how to compose and send emails by implementing hook_mail () and drupal_mail () functions.

Comparative D7/D6

Overall performance is very similar to Drupal 6. We will find small differences, especially in the implementation of triggers, new hook functions have been defined for Drupal 7.

Unit contents

52.1 Implementation of actions	282
52.2 Implementation of triggers	291
52.3 hook_cron() function	297
52.4 hook_watchdog() function	300
52.5 Sending email	303



52.1

Implementation of actions

Actions are operations (code fragments) that are executed only when certain events occur, called **triggers**.

For example, the system could send an email to the author of a node when somebody writes a comment on it. In this case, sending mail is the action, while the inclusion of the commentary on the node would be the trigger event.

As we have seen throughout this advanced level, the implementation of certain **hooks** allows you to run specific actions in certain situations. For example, if we want to send an email when a new node is created, we can do it within the implementation of module `hook_insert()` or `hook_node_insert()`. If we want to register in the database each time a user logs into the site, we will implement it in `hook_user_login()`. These actions will be inserted in the code and will not be configurable by the site administrator.

As we saw in **Unit 30** of the **Intermediate Level**, Drupal incorporates into its core the **Trigger** module, initially disabled, which allows to manage actions and triggers. Once activated, we can access trigger configuration from: **F52.1**

URL Triggers

/admin/structure/trigger/node

F52.1

Triggers

Triggers are events that occur on the site. For example, when a user saves a content or deletes a comment, specific triggers are launched.

When an event occurs we can indicate what actions will be executed.

Administration ⇒ Structure ⇒ Triggers

The screenshot shows the 'Triggers' configuration page in Drupal. At the top, there is a breadcrumb trail: Home > Administration > Structure > Triggers. Below the breadcrumb, there is a tabs menu with 'Triggers' selected. Underneath the tabs, there is a brief description of what triggers are and how they are used. The main content area contains five separate sections, each representing a different trigger type. Each section has a title, a dropdown menu labeled 'Choose an action', and a 'Assign' button. The sections are:

- TRIGGER: WHEN EITHER SAVING NEW CONTENT OR UPDATING EXISTING CONTENT
- TRIGGER: AFTER SAVING NEW CONTENT
- TRIGGER: AFTER SAVING UPDATED CONTENT
- TRIGGER: AFTER DELETING CONTENT
- TRIGGER: WHEN CONTENT IS VIEWED BY AN AUTHENTICATED USER

Triggers are grouped into categories according to the item on which they act: comment, node, system, taxonomy and user.

The operation is as simple as assigning to a trigger the actions we want to run

when this event occurs. Upon deploying the selector "Choose an action" the available actions will be displayed.

Actions are managed from: **F52.2**

Administration ⇒ **Configuration** ⇒ **System** ⇒ **Actions**

URL Actions
</admin/config/system/actions>

There are two types of actions: simple and advanced. Simple actions can be used directly, while advanced actions must be created with a specific configuration.

Home » Administration » Configuration » System

Actions ◊

There are two types of actions: simple and advanced. Simple actions do not require any additional configuration, and are listed here automatically. Advanced actions need to be created and configured before they can be used, because they have options that need to be specified; for example, sending an e-mail to a specified address, or unpublishing content containing certain words. To create an advanced action, select the action from the drop-down list in the advanced action section below and click the *Create* button.

You may proceed to the [Triggers](#) page to assign these actions to system events.

Available actions:

ACTION TYPE	LABEL
comment	Publish comment
comment	Save comment
comment	Unpublish comment
node	Make content sticky
node	Make content unsticky
node	Promote content to front page
node	Publish content
node	Save content
node	Remove content from front page
node	Unpublish content
user	Ban IP address of current user
user	Block current user

CREATE AN ADVANCED ACTION

Choose an advanced action ▾ **Create**

F52.2
Actions
Administration of actions.

We can see available actions and create advanced actions, which are actions that require additional configuration.

In this first section we will define in our module actions that will be available for allocation to triggers.

An action consists of three parts, two mandatory and an optional one:

- The definition of the action, which is done in the implementation of **hook_action_info()**.
- A specific function that performs the action. By convention this function should be named according to the following pattern: **module + ' ' + description + '_action'**. It will match the full name of the action, as defined in **hook_action_info()**.

- Optionally the action may be associated with a form definition function, which allows to configure the action (**actionname_form()**) and its corresponding functions **_submit** and **_validate**.

We analyze in this section the actions implemented by the **Action example** (action_example) module, included in **Examples for Developers**.

To begin, we must not forget that, if the module implements actions or triggers, we must add the dependency on the **trigger** module, in its .info file. **F52.3**

F52.3**.info file**

We add the dependency on the trigger module.

```
name = Action example
description = Demonstrates providing actions that can be
associated to triggers.
package = Example modules
core = 7.x
dependencies[] = trigger
```

hook_action_info() function

The **hook_action_info()** allows for defining one or more actions in a module.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_action_info/7

The **hook_action_info()** function does not require input parameters. It returns an array of actions, where each action has the following mandatory elements:

- **'type'**. The action type is related to the object on which it acts. The types provided by core are: **comment**, **entity**, **node**, **system** and **user**. It is possible to use customized values for actions acting on specific modules. The system type allows the action to be available for all objects.
- **'label'**. Descriptive name of the action, as the site administrator sees it. You must use function(t) for the text to be translatable.
- **'configurable'** Sets whether the action is configurable or not. When TRUE is indicated, it will be necessary to implement a form function with the same name as the function and the suffix _form, and the corresponding _validate and _submit functions.
- **'triggers'**. This is an array where the events (hooks) for whom the action is appropriate, are listed. To declare a global action, available to all hooks and operations, we can use the **array ('any')** value.
- **'behavior'** (optional). It indicates special behaviors, usually associated with the implementation of other actions in the same event. Currently it only allows the **'changes_property'**: value. If an action with this behavior is assigned to a hook other than "presave" other save actions assigned to the same trigger will run afterwards.

The **Action Example** module defines three actions: **F52.4**

- **'action_example_basic_action'**. This is an example action that only displays a message indicating that the action has been executed. This action is of type **system** and available for all triggers.
- **'action_example_unblock_user_action'**. Unlocks a user. This action is of type **user** and available for all triggers.
- **'action_example_node_sticky_action'**. Promotes the node to the main page and places it fixed at the top of the lists. This action is of **node** type and is available for triggers 'node_presave','node_insert',and'node_update'.

This is an **advanced action** that requires configuration. To create this action we have to indicate, from the administration area, a user for which the action will be executed, so that its published content will be promoted to the main page and placed fixed at the beginning of the lists. When saving a node it will check if the author agrees with the user specified in the action, so that the action will only be executed when they match.

```
/***
 * Implements hook_action_info().
 */
function action_example_action_info() {
  return array(
    'action_example_basic_action' => array(
      'label' => t('Action Example: A basic example action that does nothing'),
      'type' => 'system',
      'configurable' => FALSE,
      'triggers' => array('any'),
    ),
    'action_example_unblock_user_action' => array(
      'label' => t('Action Example: Unblock a user'),
      'type' => 'user',
      'configurable' => FALSE,
      'triggers' => array('any'),
    ),
    'action_example_node_sticky_action' => array(
      'type' => 'node',
      'label' => t('Action Example: Promote to frontpage and sticky on top any content created by :'),
      'configurable' => TRUE,
      'behavior' => array('changes_property'),
      'triggers' => array('node_presave', 'node_insert',
        'node_update'),
    ),
  );
}
```

F52.4**hook_action_info()**

Defining actions from hook_action_info().

The module defines three actions.

Defined actions will be visible in the list of **Available actions**. The first two will be directly displayed because they do not require configuration. The third one will be available in the **Create an advanced action** list. From the latter we can create multiple instances of the same action with different configuration parameters. **F52.5**

F52.5**Actions**

Administration of actions.
We can see the available actions and create advanced actions, which are actions that require additional configuration

In our example, two of the actions can be used directly. The third one requires additional configuration, so first we have to create the advanced action.

Available actions:

ACTION TYPE	LABEL
node	Actions Forcontu: Reset node counter
system	Action Example: A basic example action that does nothing
user	Action Example: Unblock a user
node	Actions Forcontu: Send mail to administrator warning of insertion and deletion of nodes
comment	Publish comment
comment	Save comment
comment	Unpublish comment
node	Make content sticky
node	Make content unsticky
node	Promote content to front page
node	Publish content
node	Save content
node	Remove content from front page
node	Unpublish content
user	Ban IP address of current user
user	Block current user

CREATE AN ADVANCED ACTION

Choose an advanced action	▼	Create
Choose an advanced action		
Action Example: Promote to frontpage and sticky on top any content created by ...		
Unpublish comment containing keyword(s)...		
Change the author of content...		
Unpublish content containing keyword(s)...		
Display a message to the user...		
Send e-mail...		
Redirect to URL...		

Specific function of the action

For each defined action we must implement the corresponding function. The name of the function corresponds to the full name of the action, according to the definition made in **hook_action_info()**.

The specific function of the action takes two parameters: **\$entity** and **\$context**. The **\$entity** parameter may be the object on which the action will act. It may be of different types: node, user, etc. The **\$context** parameter defines the context in which the action is executed and also provides additional information that may be used by the action. The context will allow the same action to be called by different triggers in different contexts. The **Trigger** module is charged with adding some values to the **\$context** (like \$context ['hook'], \$context ['uid'], etc.), which will define the context in which the trigger was ran, and allow the function of the action to determine whether the action should be executed or not and under what conditions.

Some functions, instead of using the **\$entity** name for the first argument, use **\$object** or the name of the specific object that uses the action (**\$node**, **\$user**, etc.). The operation is exactly the same, since what matters in the statement is the order in which arguments are declared.

The **action_example_basic_action()** function corresponds to the '**action_example_basic_action**' action. This action displays a message indicating that it has been executed. The event is also stored in the system log (**watchdog function()**). **F52.6**

```
/** 
 * Function for action_example_basic_action.
 */
function action_example_basic_action(&$entity, $context = array())
{
    drupal_set_message(t('action_example_basic_action fired'));
    watchdog('action_example', 'action_example_basic_action fired.');
}
```

F52.6

Function of the action 'basic_action'.

The action is only to display a message and log the event.

The **action_example_unblock_user_action()** function corresponds to the action '**action_example_unblock_user_action**'. This action unblocks a user. First check if the entity received has the uid attribute (user id), in which case that user gets unblocked. If not so, try to get the uid value from the \$context parameter. Finally, if the user is not found, it will act on the user registered at the site.

The action of unblocking the user is done by loading the object \$user with **user_load()**, and changing the 'status' parameter (value 1) through **user_save()**. Finally, a message displays and the action is recorded in the system event log. **F52.7**

```
/** 
 * Function for action_example_unblock_user_action.
 */
function action_example_unblock_user_action(&$entity, $context = array()) {
    // First check if the entity is a user object.

    if (isset($entity->uid)) {
        $uid = $entity->uid;
    }
    elseif (isset($context['uid'])) {
        $uid = $context['uid'];
    }
    // If a user is not provided, the registered user is used.
    else {
        $uid = $GLOBALS['user']->uid;
    }
    $account = user_load($uid);
    $account = user_save($account, array('status' => 1));
    watchdog('action_example', 'Unblocked user %name.',
        array('%name' => $account->name));
    drupal_set_message(t('Unblocked user %name',
        array('%name' => $account->name)));
}
```

F52.7

Function of the action 'unblock_user_action'.

The action is to unlock a user. The user may be provided to the function via the "context" of the trigger. If no user object is passed the action applies to the user logged into the site.

Function **action_example_node_sticky_action()** corresponds to the '**action_example_node_sticky_action**' action.

This function compares the value of the configuration parameter of the action (\$context['author']) with the author of the node on which it runs, \$node->uid. If both match, it modifies the parameters \$node->promote and \$node->sticky. Finally, a message displays and the action is recorded in the system event log. **F52.8**

F52.8

Function of the action 'node_sticky_action'.

Promotes the node to the frontpage and makes it sticky in the top of lists when the author is the indicated in the advanced action. The author is passed through the trigger context.

```
/***
 * Action function for action_example_node_sticky_action.
 */
function action_example_node_sticky_action($node, $context) {

    // Get the configured user.
    $account = user_load_by_name($context['author']);
    // Performs the action if a match with the author of the node is found.
    if ($account->uid == $node->uid) {
        $node->promote = NODE_PROMOTED;
        $node->sticky = NODE_STICKY;

        watchdog('action', 'Set @type %title to sticky and promoted by
special action for user %username.', array('@type' =>
node_type_get_name($node), '%title' => $node->title, '%username'
=> $account->name));

        drupal_set_message(t('Set @type %title to sticky and promoted
by special action for user %username.', array('@type' =>
node_type_get_name($node), '%title' => $node->title, '%username'
=> $account->name)));
    }
}
```

Action configuration form

Configurable actions ('configurable' => TRUE) require a configuration form. Of the actions mentioned previously, only the third one is configurable.

The configuration form of the action is generated from the function name:

'action name' + '_form()'. In our example, for action 'action_example_node_sticky_action' we will implement form function `action_example_node_sticky_action_form()`.

This form consists of text field 'author' where we indicate a username. To facilitate the selection of a valid user the '#autocomplete_path' option is used, which will display in the text box the system users that match the entered string. This option is available only if the user that configures the advanced action has permission to access user profiles. **F52.9**

F52.9

Configuration form for the action

Advanced actions require a form to request additional configuration parameters.

```
/***
 * Configuration form for action_example_node_sticky_action().
 *
 * @param $context
 *   Configuration options array (for editing)
 *
 * @return array $form
 */
function action_example_node_sticky_action_form($context) {
    // Text field to enter a user name.
    $form['author'] = array(
        '#title' => t('Author name'),
        '#type' => 'textfield',
        '#description' => t('Any content created, presaved or updated
by this user will be promoted to front page and set as sticky.'),
        '#default_value' => isset($context['author']) ? $context['author'] : '',
    );
    // Form autocomplete options.
    // Allows for searching site users.
    if (user_access('access user profiles')) {
        $form['author']['#autocomplete_path'] = 'user/autocomplete';
    }
    // Returns form
    return $form;
}
```

Complete the form with their corresponding validation and shipping functions.

The validation function, `action_example_node_sticky_action_validate()` checks whether the entered user exists in the system. If it does not, it shows the error message and prevents the form submission. **F52.10**

```
/** 
 * Validation function of action_example_node_sticky_action action.
 * Checks if user exists.
 */
function action_example_node_sticky_action_validate($form, $form_state) {
  if (! $account = user_load_by_name($form_state['values']['author'])) {
    form_set_error('author', t('Please, provide a valid username'));
  }
}
```

F52.10**Form validation**

Validation function for the configuration form.

Once validation is passed, the send form function is executed, `action_example_node_sticky_action_submit()`.

The values in the form field are returned by the `_submit()` function of the form as an associative array. These values will be available in the previous functions (action function and form function) via the `$context:$context['author']` parameter. **F52.11**

```
/** 
 * Send function of action_example_node_sticky_action action.
 *
 * Returns an associative array with the values of $context,
 * which will be available upon action execution.
 */
function action_example_node_sticky_action_submit($form, $form_state) {
  return array('author' => $form_state['values']['author']);
}
```

F52.11**Form submission**

Context values that will be available when executing the action.

This is the end of the implementation of an advanced action. Figure 5.21 **F52.12** shows an implemented configuration form.

An advanced action offers additional configuration options which may be filled out below. Changing the *Description* field is recommended, in order to better identify the precise action taking place. This description will be displayed in modules such as the Trigger module when assigning actions to system events, so it is best if it is as descriptive as possible (for example, "Send e-mail to Moderation Team" rather than simply "Send e-mail").

Label
Action Example: Promote content of user1

A unique label for this advanced action. This label will be displayed in the interface of modules that integrate with actions, such as Trigger module.

Author name
user1

Any content created, presaved or updated by this user will be promoted to front page and set as sticky.

Save

F52.12**Configuration form**

Example of configuration form defined in the advanced action.

Once the advanced action is created, it will be available in the Actions list and may be assigned to the triggers that apply. **F52.13**

F52.13**Advanced action**

Once configured, the advanced action will be visible in the list of available actions. From this time may be mapped to triggers.

Available actions:

ACTION TYPE	LABEL	OPERATIONS
comment	Save comment	
comment	Publish comment	
comment	Unpublish comment	
node	Action Example: Promote content of user1	configure delete
node	Save content	

hook_action_info_alter() function

The **hook_action_info_alter()** function enables us to modify actions defined by other modules.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_action_info_alter/7

`hook_action_info_alter(&$actions)`

Through the **\$actions** vector, passed to the function by reference, we can modify any value of actions defined by other modules.

Implementation of triggers

52.2

We have studied how to create new actions that can be assigned to triggers available on the site. We will now see how to add new triggers.

We analyze in this section triggers implemented by the **Trigger example** (trigger_example) module included in **Examples for Developers**.

We recall that, if the module implements actions or triggers, we must add in the .info file, the dependency on the **Trigger** module.

hook_trigger_info() function

The **hook_trigger_info()** function allows to define new triggers (events) within a module. These triggers are available from the administration area for users to assign actions to be executed when the defined event occurs.

http://api.drupal.org/api/drupal/modules--trigger-trigger.api.php/function/hook_trigger_info/7

The function returns an associative array with the following nested structure:

- **Module.** In the first level of the array the key is the system name of the module for which the trigger is implemented. Thus a module can implement in this way triggers associated with other modules. In the administration area triggers are grouped into tabs, which correspond to this value.
 - o **Hook.** The second level indicates the hook from which the event will fire. If the trigger is associated with the module itself, we indicate the name of the trigger, for which we subsequently define its own triggering function.
 - **Description.** At the third level we include descriptive information about the trigger. It currently only supports the 'label' value, which allows to add a description of the trigger: **'label' => t ('description of trigger')**.

The following example shows the implementation of **hook_trigger_info()** by the **Trigger Example** module. The module creates two triggers. The first event ('user_first_time_login') occurs when a user accesses the site (login) for the first time. The second event ('triggersomething') will be launched when the user clicks a button in a page implemented by the same module. **F52.14**

```
/***
 * Implements hook_trigger_info().
 */
function trigger_example_trigger_info() {
  return array(
    'user' => array(
      'user_first_time_login' => array(
        'label' => t('After a user has logged in for the first time'),
      ),
    ),
    'trigger_example' => array(
      'triggersomething' => array(
        'label' => t('After the triggersomething button is clicked'),
      ),
    ),
  );
}
```

F52.14

hook_trigger_info()

Sets new triggers.

Implementing the 'user_first_time_login' trigger

To implement the 'user_first_time_login' trigger we must:

- **Create the function that implements the event.** This function is charged with executing the actions assigned to the event. Within this function the **actions_do()** function is called, which will execute the actions.
- **Implement the call to the event function** at the corresponding point in the system, usually within a hook. In the example event such a call to the above function will be made within **hook_user_login()**.

We start with the development of the function of the event:

_trigger_example_first_time_login(\$hook, &\$edit, \$account, \$category = NULL)

Since we must implement both the event function and the call to it, we can freely choose the required parameters and their order.

In this case, the parameters passed to the function will be: **F52.15**

- **\$hook.** Name of the trigger.
- **\$edit.** Array to modify the user object.
- **\$account.** User with account.
- **\$category.** Category of the user profile.

F52.15

Trigger function

Example function that implements the event. It calls **actions_do()** to executed the assigned actions.

```
/*
 * Function of trigger "User first time login"
 */
function _trigger_example_first_time_login($hook, &$edit, $account,
                                           $category = NULL) {
  static $objects;

  // Gets all the actions assigned to trigger
  // 'user_first_time_login'.
  $aids = trigger_get_assigned_actions($hook);

  // Prepare the $context array that will be passed to the actions
  $context = array(
    'group' => 'user',
    'hook' => $hook,
    'form_values' => &$edit,
  );
  // Loop implementation so that each action execution can modify
  // the user object. Thus the following actions will receive
  // the modified object.
  foreach ($aids as $aid => $info) {
    $type = $info['type'];
    if ($type != 'user') {
      if (!isset($objects[$type])) {
        $objects[$type] = _trigger_normalize_user_context($type, $account);
      }
      $context['user'] = $account;
      actions_do($aid, $objects[$type], $context);
    }
    else {
      actions_do($aid, $account, $context, $category);
    }
  }
}
```

The **trigger_get_assigned_actions(\$ hook)** function, returns an array with the IDs of the actions associated to the trigger contained in \$hook.

http://api.drupal.org/api/drupal/modules--trigger--trigger.module/function/trigger_get_assigned_actions/7

To execute each action the **actions_do()** function is called.

http://api.drupal.org/api/drupal/includes--actions.inc/function/actions_do/7

actions_do(\$action_ids, \$object = NULL, \$context = NULL, \$a1 = NULL, \$a2 = NULL)

The call to **actions_do()** receives the following parameters:

- **\$actions_ids**. Array with the IDs of the actions that will be executed.
- **\$object**. Object upon which the action acts.
- **\$context**. Array with the contact information passed to the action.
- **\$a1, \$a2**. Additional parameters.

This function allows for the execution of all actions in a single step, passing in \$actions_id the full list of actions to execute. However, the trigger has been implemented in a loop and the actions run one at a time. Thus, if an action modifies the object on which it acts (user), the following action will receive the modified object.

As we saw in the previous section, each action will assess whether the execution context is adequate and determine the appropriate actions to be carried out in each case. From the trigger viewpoint, its task ends here, with the call to action.

The next step is to call the function of the trigger. The location of this call depends on when the event should run. In our example the right hook is `hook_user_login()`.

The event will fire only on first login (`$ Account->access == 0`). **F52.16**

```
/***
 * Implements hook_user_login().
 */
function trigger_example_user_login(&$edit, $account, $category = NULL) {
  // Verifies that user has not logged in before.
  if ($account->access == 0) {
    // Calls the trigger function.
    trigger_example_first_time_login('user_first_time_login',
$edit, $account, $category);
  }
}
```

F52.16

Calling the trigger

To run the trigger we must include a call to it at the appropriate point of the system. For example inside a specific hook or in a form submission function.

The defined trigger becomes available in the trigger administration area, under the User tab.

To test this trigger we have to assign it an available action, then create a new user, and upon first access, the actions assigned to the trigger should be performed. **F52.17**

F52.17

Trigger created

The trigger created will be available in the list of triggers, under the tab of the module where it was defined.

Once listed, we can assign actions to run when the event occurs.

The screenshot shows the 'Triggers' page in the Drupal administration interface. It lists six triggers:

- TRIGGER: AFTER CREATING A NEW USER ACCOUNT**: A dropdown menu labeled 'Choose an action' with an 'Assign' button.
- TRIGGER: AFTER UPDATING A USER ACCOUNT**: A dropdown menu labeled 'Choose an action' with an 'Assign' button.
- TRIGGER: AFTER A USER HAS BEEN DELETED**: A dropdown menu labeled 'Choose an action' with an 'Assign' button.
- TRIGGER: AFTER A USER HAS LOGGED IN**: A dropdown menu labeled 'Choose an action' with an 'Assign' button.
- TRIGGER: AFTER A USER HAS LOGGED OUT**: A dropdown menu labeled 'Choose an action' with an 'Assign' button.
- TRIGGER: WHEN A USER'S PROFILE IS BEING VIEWED**: A dropdown menu labeled 'Choose an action' with an 'Assign' button.
- TRIGGER: AFTER A USER HAS LOGGED IN FOR THE FIRST TIME**: A dropdown menu labeled 'Choose an action' with an 'Assign' button. This box is highlighted with a red border.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Implementing the 'triggersomething' trigger

The second trigger defined in the Trigger Example module is run when you click on a submit button in a form. The main difference with the previous trigger is that this time the call to the trigger function will not be made from within a hook, but from the send or submit function of the form. **F52.18**

F52.18

Trigger function

Function of the trigger 'triggersomething'.

```
/** 
 * Function of the triggersomething trigger.
 */
function trigger_example_triggersomething($options = array()) {
  // Obtains all the actions assigned to the 'triggersomething'
  // trigger.
  $aids = trigger_get_assigned_actions('triggersomething');

  // Prepares the $context array, which will be passed to the actions
  $context = array(
    'group' => 'trigger_example',
    'hook' => 'triggersomething'
  );
  // Executes actions in one call
  actions_do(array_keys($aids), (object) $options, $context);
}
```

The event will fire upon sending the form implemented by the module itself, in the **examples/trigger_example** URL. For this purpose we register the URL via `hook_menu()` and build the form and its send function. The call to the trigger function is made within the latter, if the user clicked on the button defined in the form. **F52.19**

```
/***
 * Implements hook_menu().
 */
function trigger_example_menu() {
  $items['examples/trigger_example'] = array(
    'title'          => 'Trigger Example',
    'description'    => 'Provides a form to demonstrate the
trigger example.',
    'page callback'  => 'drupal_get_form',
    'page arguments' => array('trigger_example_form'),
    'access callback' => TRUE,
  );
  return $items;
}

/***
 * Form definition.
 *
 * Creates a button that launches the trigger
 */
function trigger_example_form($form_state) {
  $form['triggersomething'] = array(
    '#type'  => 'submit',
    '#value' => t('Run triggersomething event'),
  );
  return $form;
}

/***
 * Send function of the trigger_example_form() form.
 */
function trigger_example_form_submit($form, $form_state) {
  // If you clicked the button the trigger is launched.
  if ($form_state['values']['op'] == t('Run triggersomething event')) {
    trigger_example_triggersomething();
  }
}
```

F52.19

Trigger execution via form

In this example the trigger is activated by submitting a form. We need to register the URL and create the form, including the call to the trigger function inside the form submit function.

The trigger will be available under the **Trigger Example** tab of the corresponding module, as defined in `hook_trigger_info()`. **F52.20**

Home > Administration > Structure > Triggers

Triggers COMMENT NODE SYSTEM TAXONOMY TRIGGER EXAMPLE USER

Triggers are events on your site, such as new content being added or a user logging in. The Trigger module associates these triggers with actions (functional tasks), such as unpublishing content containing certain keywords or e-mailing an administrator. The Actions settings page contains a list of existing actions and provides the ability to create and configure advanced actions (actions requiring configuration, such as an e-mail address or a list of banned words).

There is a tab on this page for each module that defines triggers. On this tab you can assign actions to run when triggers from the Trigger example module happen.

A trigger is a system event. For the trigger example, it's just a button-press. To demonstrate the trigger example, choose to associate the 'display a message to the user' action with the 'after the triggersomething button is pressed' trigger.

TRIGGER: AFTER THE TRIGGERSOMETHING BUTTON IS CLICKED

Choose an action

F52.20

Trigger created

This trigger is shown in the Trigger example tab.

hook_trigger_info_alter() function

The **hook_trigger_info_alter()** function enables modification of triggers defined by other modules from their corresponding implementations of hook_trigger_info().

http://api.drupal.org/api/drupal/modules--trigger-trigger.api.php/function/hook_trigger_info_alter/7

The **Trigger Example** module includes an example of using this function, making a tiny change to the system's 'cron' trigger. The module changes the description of the trigger by modifying its label. **F52.21**

F52.21

Modify trigger

We can also act on triggers created by other modules.

```
/** * Implements hook_trigger_info_alter(). */ function trigger_example_trigger_info_alter(&$triggers) { // Previous value of label: "When cron runs" // New value "On cron execution" $triggers['system']['cron']['label'] = t('On cron execution'); }
```

hook_cron() function

52.3

Drupal has a cron system that allows the execution of periodic tasks. The administrator sets the period of execution of cron (every day at a certain hour, every hour, every Monday, etc.), at which all actions defined in the same run.

The **hook_cron()** function allows any module to add tasks to be run periodically, upon cron execution.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_cron/7

Usually the period of execution of cron (cron.php) is not defined from Drupal, but from the server where it is installed, through the **crontab** tool. Manually running cron from the Drupal administration area will also cause the invocation of all hook_cron() functions implemented.

The tasks that can be performed within hook_cron () can be very varied:

- Remove temporary data from database.
- Remove temporary files.
- Perform calculations and get results in the background, usually for intensive operations that consume lots of server resources (such as access statistics calculation).
- Send automated emails.
- Obtain data from external apps.

Drupal 7 provides a new hook: **hook_cron_queue_info()**, which optimizes the actions to be executed by the cron.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_cron_queue_info/7

In general we can say that short tasks that consume few resources can be implemented through **hook_cron()**, while long resource-intensive tasks will be separated and assigned to **hook_cron_queue_info()**.

hook_cron()

As an example of use of hook_cron() we analyze the code implemented by some system modules.

The **Node** module implements **hook_cron()** to delete from the 'history' table the history of pages viewed by users. With each cron run, records older than a certain date will be deleted, according to the NODE_NEW_LIMIT parameter (default 30 days). F52.22

```
/***
 * Implements hook_cron().
 */
function node_cron() {
  db_delete('history')
    ->condition('timestamp', NODE_NEW_LIMIT, '<')
    ->execute();
}
```

F52.22

hook_cron()

The actions defined in hook_cron() will be performed at the system cron run.

The **Poll** module implements **hook_cron()** to close surveys. Each survey has its own deadline, so it first obtains, from the database, expired polls. For each result it updates the log, assigning value 0 to the 'active' field. **F52.23**

F52.23

Example of hook_cron()

Example of hook_cron() implementation by the Poll module.

```
/***
 * Implements hook_cron().
 */
function poll_cron() {
  $nids = db_query('SELECT p.nid FROM {poll} p INNER JOIN {node} n
    ON p.nid = n.nid WHERE (n.created + p.runtime) < :request_time AND
    p.active = :active AND p.runtime > :runtime',
    array(':request_time' => REQUEST_TIME, ':active' => 1, ':runtime'
    => 0))->fetchCol();
  if (!empty($nids)) {
    db_update('poll')
      ->fields(array('active' => 0))
      ->condition('nid', $nids, 'IN')
      ->execute();
  }
}
```

The **dblog** module implements **hook_cron()** to clean the table of system logs (watchdog). It first gets the limit of records the system stores in the table (variable 'dblog_row_limit'). Afterwards it locates, according to this value, the identifier ('wid') of the last record to be maintained in the database. Then it deletes the oldest records (those with a lesser 'wid' value). **F52.24**

F52.24

Example of hook_cron()

Example of hook_cron() implementation by the DBlog module.

```
/***
 * Implements hook_cron().
 */
function dblog_cron() {
  // Gets the max number of records
  $row_limit = variable_get('dblog_row_limit', 1000);

  // Gets the wid of the last record to be stored
  if ($row_limit > 0) {
    $min_row = db_select('watchdog', 'w')
      ->fields('w', array('wid'))
      ->orderBy('wid', 'DESC')
      ->range($row_limit - 1, 1)
      ->execute()->fetchField();

    // Deletes all records with a lesser 'wid' (older)
    if ($min_row) {
      db_delete('watchdog')
        ->condition('wid', $min_row, '<')
        ->execute();
    }
  }
}
```

hook_cron_queue_info()

The **hook_cron_queue_info()** function allows to define queues of tasks to be executed by the system cron. When cron is triggered, it executes the tasks defined in **hook_cron()**, and those individual tasks declared in **hook_cron_queue_info()**. If necessary, the system executes tasks in queue in different calls to cron, but they can also be run in parallel.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_cron_queue_info/7

The function returns an associative array where the main element is the name of the queue. The value of each element is itself an array with fields:

- **'worker callback'**. Name of the function that will be called and performs the action or actions to take. The module must define this function.
- **'time'** (optional). Maximum execution time of the task, in seconds, default value is 15. For intensive tasks we set a higher number of seconds.

As an example, the **Aggregator** core module implements this function to update news feeds. **F52.25**

```
/**
 * Implements hook_cron_queue_info().
 */
function aggregator_cron_queue_info() {
  $queues['aggregator_feeds'] = array(
    'worker callback' => 'aggregator_refresh',
    'time' => 60,
  );
  return $queues;
}
```

F52.25

Cron queue

We can define tasks to run independently when the cron is activated.

The **aggregator_refresh()** function is responsible for performing update operations. The maximum time allocated is 60 seconds.

52.4 hook_watchdog() function

watchdog() function

We used multiple times the **watchdog()** API function in Drupal. This function is tasked with registering a message in the system logs of Drupal. Each call to the **watchdog()** function saves a record in the **watchdog** table.

<http://api.drupal.org/api/drupal/includes--bootstrap.inc/function/watchdog/7>

```
watchdog($type, $message, $variables = array(), $severity = WATCHDOG_NOTICE,
         $link = NULL)
```

Parameters required by this function are:

- **\$type**. It is a string that identifies the type or category of the message. In general, it uses the name of the module that performs the function call.
- **\$message**. Message stored using the t() function, in order to make it translatable.
- **\$variables**. Variables to be replaced in the message (\$message).
- **\$severity**. Indicates error type according to severity (WATCHDOG_ERROR, WATCHDOG_WARNING, etc.). Possible values are returned by the watchdog_severity_levels() function, whose contents are shown below.
- **\$link**. Link associated to the message.

The **watchdog_severity_levels()** function returns values accepted by the **\$severity** parameter: **F52.26**

F52.26

Type of error log

The `watchdog_severity_levels()` function returns an array of all available types of errors.

```
function watchdog_severity_levels() {
  return array(
    WATCHDOG_EMERGENCY => t('emergency'),
    WATCHDOG_ALERT => t('alert'),
    WATCHDOG_CRITICAL => t('critical'),
    WATCHDOG_ERROR => t('error'),
    WATCHDOG_WARNING => t('warning'),
    WATCHDOG_NOTICE => t('notice'),
    WATCHDOG_INFO => t('info'),
    WATCHDOG_DEBUG => t('debug'),
  );
}
```

Next we show some examples of calls to the **watchdog()** function: **F52.27**

F52.27

Examples of calls to watchdog

Different log messages through the `watchdog()` function.

```
// Message from log when cron completes.
watchdog('cron', 'Cron run completed.', array(), WATCHDOG_NOTICE);

// Message from log when a node is deleted.
watchdog('content', '@type: deleted %title.', array('@type' =>
$node->type, '%title' => $node->title));

// Message from log when a user closes session.
watchdog('user', 'Session closed for %name.', array('%name' =>
$user->name));
```

hook_watchdog() function

We have already seen how the **watchdog()** function works by storing information in the watchdog table. The **hook_watchdog()** function lets you tweak this behavior by intercepting log messages sent in each call to the **watchdog()** function and redirecting them to other locations for processing and/or storage.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_watchdog/7

hook_watchdog(array \$log_entry)

The **hook_watchdog()** function receives, as a parameter, the **\$log_entry** array, with the following elements:

- **'type'**: This corresponds to the `$type` parameter of the call to `watchdog()`, which usually corresponds to the name of the module that records the message.
- **'user'**: User object of the user who was logged into the system when the event occurred.
- **'request_uri'**: Page where the event occurred.
- **'referer'**: Page from where the user arrived to the page where the event occurred (previous page).
- **'ip'**: IP from where the page was loaded.
- **'timestamp'**: Date of the event in UNIX timestamp format.
- **'severity'**: Type or severity of the message, according to possible values returned by the `watchdog_severity_levels()` function.
- **'link'**: Link sent by the parameter with same name of the `watchdog()` function.
- **'message'**: Log message sent by the parameter with same name of the `watchdog()` function.

In the following example, taken from the documentation of the function, an email is sent to a fixed target (`someone@example.com`) each time the **watchdog()** function is called. The way this email is composed and sent by the **drupal_mail()** function will be discussed in the next section. F52.28

```
function example_watchdog(array $log_entry) {
  global $base_url, $language;

  $severity_list = array(
    WATCHDOG_EMERGENCY => t('Emergency'),
    WATCHDOG_ALERT => t('Alert'),
    WATCHDOG_CRITICAL => t('Critical'),
    WATCHDOG_ERROR => t('Error'),
    WATCHDOG_WARNING => t('Warning'),
    WATCHDOG_NOTICE => t('Notice'),
    WATCHDOG_INFO => t('Info'),
    WATCHDOG_DEBUG => t('Debug'),
  );

  $to = 'someone@example.com';
  $params = array();
  $params['subject'] = t('[@site_name] @severity_desc: Alert from
your web site', array(
    '@site_name' => variable_get('site_name', 'Drupal'),
    '@severity_desc' => $severity_list[$log_entry['severity']],
  ));

  $params['message'] = "\nSite: $base_url";
}
```

F52.28

hook_watchdog()

Implementing `hook_watchdog()` allows you to intercept log messages. It may be useful, for example, for storage in a different location or in a different format.

```

$params['message'] .= '\nSeverity:      (@severity)
@severity_desc';
$params['message'] .= '\nTimestamp:     @timestamp';
$params['message'] .= '\nType:          @type';
$params['message'] .= '\nIP Address:    @ip';
$params['message'] .= '\nRequest URI:   @request_uri';
$params['message'] .= '\nReferrer URI: @referer_uri';
$params['message'] .= '\nUser:           (@uid) @name';
$params['message'] .= '\nLink:           @link';
$params['message'] .= '\nMessage:        \n\n@message';

$params['message'] = t($params['message'], array(
  '@base_url' => $base_url,
  '@severity' => $log_entry['severity'],
  '@severity_desc' => $severity_list[$log_entry['severity']],
  '@timestamp' => format_date($log_entry['timestamp']),
  '@type' => $log_entry['type'],
  '@ip' => $log_entry['ip'],
  '@request_uri' => $log_entry['request_uri'],
  '@referer_uri' => $log_entry['referer'],
  '@uid' => $log_entry['user']->uid,
  '@name' => $log_entry['user']->name,
  '@link' => strip_tags($log_entry['link']),
  '@message' => strip_tags($log_entry['message']),
));
drupal_mail('emaillog', 'entry', $to, $language, $params);
}

```

Sending email

52.5

hook_mail() function

The **hook_mail()** function is used to prepare messages to be mailed through **drupal_mail()** function.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_mail/7

hook_mail (\$key, &\$message, \$params)

Parameters required by **hook_mail()** are:

- **\$key**. A text string that will identify the email, since the same hook_mail() function can deploy multiple emails.
- **\$message**. Array with the following structure:
 - o **'id'**: Identifier of sent email. A possible value to reference the email can be: **\$ module. '_'. \$ key**.
 - o **'to'**: Email address or addresses to which the message is sent. Email addresses must be separated by commas (,).
 - o **'subject'**: Email Subject. It cannot contain line breaks.
 - o **'body'**: The message body is stored as an array of lines of text.
 - o **'from'**: Email address from which the message is sent.
 - o **'headers'**: Associative array where we can add mail headings: Form, Sender, MIME-Version, Content-Type, etc.
- **\$params**. Array of additional parameters that can be used to compose the message. This field corresponds to the \$params parameter passed to the drupal_mail() function.

To explain the operation of email in Drupal, we expand the **Forcontu Forms** module, developed in **Unit 47**. When a user submits the form available at the **forms_forcontu/form2** URL, two emails, one for the site administrator and another one for the user will be sent.

We start with the implementation of hook_mail (), where we declare two types of email (\$key): **F52.29**

- **'email_user'**. Email for the user who sends the form.
- **'email_admin'**. Mail to the site admin. This email will be sent to the email address listed on the site configuration.

F52.29**hook_mail()**

In `hook_mail()` we declare the type of emails that can be sent from the module using `drupal_mail()` function.

```
/*
 * Implements hook_mail().
 */
function forms_forcontu_mail($key, &$message, $params) {
  $language = $message['language'];

  $variables = array(
    '!site-name' => variable_get('site_name', 'Drupal'),
    '!sender-name' => $params['first_name'] . ' ' . $params['last_name'],
    '!form-id' => $params['form_id'],
    '!sender-email' => $params['email'],
    '!sender-password' => $params['pass'],
  );

  switch ($key) {
    case 'email_user':
      $message['subject'] = t('Application received from !site-name', $variables, array('langcode' => $language->language));
      $message['body'][] = t("Remember your access data:", $variables, array('langcode' => $language->language));
      $message['body'][] = t("Email: !sender-email.", $variables, array('langcode' => $language->language));
      $message['body'][] = t("Password: !sender-password.", $variables, array('langcode' => $language->language));
      break;

    case 'email_admin':
      $message['subject'] = t('Form sent by !sender-name', $variables, array('langcode' => $language->language));
      $message['body'][] = t("!sender-name sent the form: !form-id.", $variables, array('langcode' => $language->language));
      $message['body'][] = t("The values entered by the user are:", $variables, array('langcode' => $language->language));
      $message['body'][] = t("Email: !sender-email.", $variables, array('langcode' => $language->language));
      $message['body'][] = t("Password: !sender-password.", $variables, array('langcode' => $language->language));
      break;
  }
}
```

drupal_mail() function

The `drupal_mail()` function composes and optionally sends email.

http://api.drupal.org/api/drupal/includes--mail.inc/function/drupal_mail/7

The `drupal_mail()` function calls the implementation of `hook_mail()` and identifies the message it must compose through parameters `$module` and `$key`. Then it composes the message and sends it (if `$send == TRUE`, which is the default).

drupal_mail (\$module, \$key, \$to, \$language, \$params = array(), \$from = NULL, \$send = TRUE)

Parameters required by `drupal_mail()` function are:

- **\$module.** Name of the module whose `hook_mail()` function will be invoked.
- **\$key.** Send email identifier. This identifier can define within `hook_mail()`, various different structures of email.
- **\$to.** Address or addresses to which the message is sent. Addresses must be separated by comma (,), allowing these formats:

- user1@example.com, user2@example.com
- User1 <user1@example.com>, User 2 <user2@example.com>
- **\$language.** Language object used to compose email.
- **\$params.** Optional parameters for composition.
- **\$from.** Sender information.
- **\$send.** If set to TRUE, email is sent directly. Otherwise it will be necessary to call the **drupal_mail_send()** function to send email manually.

The **drupal_mail()** function returns a structured array with full details of the message. If the message was sent (`$send == TRUE`), the 'result' element of the array returned will hold the result of the operation.

We complete the previous example with the corresponding calls to the **drupal_mail()** function from the form submission function: [F52.30](#)

```
/** 
 * Send function of form forms_forcontu_form2
 */
function forms_forcontu_form2_submit($form, &$form_state) {
  //...
  global $language, $base_url;

  $params = $form_state['values'];

  // Email to site admin
  $site_mail = variable_get('site_mail');
  // User email (the one indicated in the form)
  $user_mail = $form_state['values']['email'];
  $form_url = $base_url . $form['#action'];

  // Sends email to user
  $to = $user_mail;
  $from = $site_mail;
  drupal_mail('forms_forcontu', 'email_user', $to, $language, $params, $from);

  // Sends email to the site's email address
  $to = $site_mail;
  $from = $user_mail;
  drupal_mail('forms_forcontu', 'email_admin', $to, $language, $params, $from);

  drupal_set_message ("<strong>Your user account has been created successfully. Check your email.</strong>");
}

drupal_goto('');
```

[F52.30](#)

drupal_mail()

The **drupal_mail()** function composes and sends emails, using the definition made in `hook_mail()`.

In this example two emails are sent when a user completes a form. An email will be sent to the user and the other will alert to the site administrator.

Once the form is submitted two emails are sent: one for the administrator and another one for the user. The email received by the administrator through the e-mail address of the site, is shown [F52.31](#) below.

Subject: Form sent by demouser

demouser sent form: forms_forcontu_form2.
 Values set by the user are:
 Email: frankgil@gmail.com.
 Password: 123abc.

Thanks to the existing communication between `hook_mail()` and `drupal_mail()`, some values related to the message, such as subject (`$message['subject']`) or body (`$message['body']`) can be defined in any of the two functions.

In the example F52.32 F52.33 we have defined these fields in `hook_mail()`. For adding this information from `drupal_mail()`, we just have to pass the values to `hook_mail()` including them in the `$params` array (`$params['message']`, `$params['subject']`, etc.). From `hook_menu()` we assign these values to the variable `$message`.

F52.32**Composing email**

Another way to compose email, sending the message and/or the subject from the function from which you call `drupal_mail()`.

```
// Example build of message from drupal_mail()
$to = 'someone@example.com';
$params = array();
$params['subject'] = t('[@site_name] @severity_desc: Alert from
your web site', array(
    '@site_name' => variable_get('site_name', 'Drupal'),
    '@severity_desc' => $severity_list[$log_entry['severity']],
));
$params['message'] = "\nSite:           @base_url";
$params['message'] .= "\nSeverity:     (@severity) @severity_desc";
$params['message'] .= "\nTimestamp:   @timestamp";
```

```
drupal_mail('emaillog', 'entry', $to, $language, $params);
//...
```

```
/**
 * Implements hook_mail().
 */
function emaillog_mail($key, &$message, $params) {
//...
switch ($key) {
  case 'entry':
    $message['subject'] = $params['subject'];
    $message['body'][] = check_plain($params['message']);
    break;
}
}
```

hook_mail_alter() function

The `hook_mail_alter()` function allows for modifying email messages before they are sent. Any module can interact with a message before it is sent, provided that it knows its identifier (`$message['id']`). As mentioned, the most practical way to identify mail messages is to use the pattern '`$ module_ $ key`'.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_mail_alter/7

hook_mail_alter (&\$message)

The `$message` parameter requested by the `hook_mail_alter()` function is a structured array with the same fields of the `$message` parameter in `hook_mail()`. In turn it corresponds to the `$message` array returned by `drupal_mail()`. In addition, the `$message` parameter is passed to the function by reference, so it can be modified by any module.

In the following example we modify the email sent by the core Contact module identified by key `$key == 'page_mail'`, so the value of `$message['id']` is `'contact_page_mail'`. This modification includes only an additional line at the end of the message that indicates the name of the site from where the email was sent. F52.34

F52.34**hook_mail_alter()**

Modifies the emails defined by other modules.

```
function example_mail_mail_alter(&$message) {
  if ($message['id'] == 'contact_page_mail') {
    $message['body'][] = '--\n' . t('Mail sent out from ') .
variable_get('sitename', t('Drupal'));
  }
}
```

53 Working with files

Drupal has two methods of downloading files: public and private. When we configure the site, we select the default method, but afterwards we can have both public and private files.

When we define a file as private, a module can control if the user has access to implementing **hook_file_download()** for the file.

Comparative D7/D6

In Drupal 6 we could use only one of two download methods, which applied to all files on the site. In Drupal 7 we can select a default download method, but we can upload both public and private files.

Unit contents

53.1 The file system of Drupal	308
53.2 Files Functions (File API)	310
53.3 Forms with files	318
53.4 Control of file permissions	321

53

53.1 The file system of Drupal

Public and Private methods of download

URL File system

/admin/config/media/file-system

Drupal has two methods of downloading files: public and private. We access the configuration here:

Administration ⇒ Configuration ⇒ Media ⇒ File system

The default download method, initially the only one available, is the public file system path (**Local Public files served by the webserver**). This means that all files will be available using HTTP, we cannot control access to them, and they can be opened by anyone who knows the URL to the file.

If we want to control access to the files on the site, we can indicate a path for **the private file system path**, which typically is outside the direct access of the web. Once the private path is set (Drupal must have permissions over the folder), a new option for the default download method (**Private local files served by Drupal**) is displayed. The use of private files allows control of access to the files on the site, but it is less efficient than the public method. **F53.1**

F53.1

File system

In the configuration of the File system you can select the default download method, public or private.

Home » Administration » Configuration » Media

File system

Public file system path
sites/default/files

A local file system path where public files will be stored. This directory must exist and be writable by Drupal. This directory must be relative to the Drupal installation directory and be accessible over the web.

Private file system path
private

An existing local file system path for storing private files. It should be writable by Drupal and not accessible over the web. See the online handbook for [more information about securing private files](#).

Temporary directory
/tmp

A local file system path where temporary files will be stored. This directory should not be accessible over the web.

Default download method

Public local files served by the webserver.
 Private local files served by Drupal.

This setting is used as the preferred download method. The use of public files is more efficient, but does not provide any access control.

Save configuration

In Drupal 6 we could use only one of two download methods, which applied to all files on the site. In Drupal 7 we have to select a default download method, but we can upload both public and private files.

By adding adding a field type **File** to any entity (for example, a content type), we will be able to select whether the files uploaded through this field will use the public or private method. **F53.2**

Home > Administration > Structure > Content types > Article > Manage fields > File

File • **EDIT** **FIELD SETTINGS** **WIDGET TYPE** **DELETE**

F53.2

File field

In fields of type File you can select the download method, public or private.

FIELD SETTINGS

These settings apply to the *File* field everywhere it is used. These settings impact the way that data is stored in the database and cannot be changed once data has been created.

Enable Display field
The display option allows users to choose if a file should be shown when viewing the content.

Files displayed by default
This setting only has an effect if the display option is enabled.

Upload destination

Public files
 Private files

Select where the final files should be stored. Private file storage has significantly more overhead than public files, but allows restricted access to files within this field.

Save field settings

Using the **public method**, the path where the file is physically stored is visible to all users. Any user, even anonymous users, may download a file from the site if they know its URL.

Using the public method, if we attach a file called example.pdf to some content, the path of the file (by default) will be:

http://www.example.com/sites/default/files/ejemplo.pdf

In this case we can access the folder where the file is via FTP, and we will find it there.

Using the **private method**, files are stored in a private folder, to which users need not have access (nor should they). The URL of a file will no longer be its direct physical location, but rather it will be a URL alias that processes the downloading of the file. The system is responsible for locating and delivering the requested file, first checking that the user has the appropriate permissions to access the file.

With the private method, if we attach the same file called example.pdf, the file path will be:

http://www.example.com/system/files/forcontu.pdf

The physical file is actually saved in a path specified for private files (e.g. **/private**). So that this folder is not accessible from the web, it is recommended that the folder be made above the web server (`/home/user/www`). For example, we can use **/home/user/private**, which is the same level as `www` but outside of it, so it will not be accessible from the browser.

This is the URL (`/home/user/private`) that we will indicate in "Path of the file system".

It is important to configure these server settings before you upload any files, since the URLs will be affected, and files previously uploaded to the site could fail to download.

URI Identifier

A **URI Identifier** (Uniform Resource Identifier) is a string that uniquely identifies a resource, such as a file.

Starting with Drupal 7, we reference files from their URI Identifier and not directly through their path or URL. The final URL of the file will be composed based on its URI Identifier.

In Drupal the URI Identifier has the following structure: **scheme://target**, where:

- 'scheme' can be "public" or "private", depending on the method used to download the file. Other schemes are also used, such as "temporary" to reference a folder of temporary files.
- 'target' is the physical name of the resource.

For example, the URI Identifier "**public://example.txt**" references the file **example.txt** uploaded to the folder defined in the public download method:

- /sites/default/files/example.txt

53.2 Files Functions (File API)

Drupal's API has with a wide range of functions that facilitate working with files. We can see the full list of functions at:

<http://api.drupal.org/api/drupal/includes--file.inc/group/file/7>

Now let's describe some of these functions:

Functions for processing files and directories

The set of functions related to processing files and directories (create, delete, move, copy, etc.): **F53.3**

F53.3

File and directory functions

In this list some functions to treat files and directories are displayed.

Name	Description
<code>drupal_basename()</code>	Indicates a path, returns the name of the file.
<code>drupal_chmod()</code>	Establishes the permissions of a file or a directory.
<code>drupaldirname()</code>	Indicates a path, returns the name of the directory.
<code>drupal_mkdir()</code>	Creates a directory.
<code>drupal_move_uploaded_file()</code>	Moves an uploaded file to a new location.
<code>drupal_rmdir()</code>	Deletes a directory.
<code>drupal_tempnam()</code>	Creates a file with a unique name in the specified directory.
<code>drupal_unlink()</code>	Deletes a file.
<code>file_get_content_headers()</code>	Examines a file object and returns the appropriate headers for download.

file_get_mimetype()	Determines the file type (MIME). This value is important for generating the appropriate headers for the file download.
file_load()	Loads a file object from the database.
file_load_multiple()	Loads multiple file objects from the database.
file_move()	Moves a file to a new location (eliminating the original file).
file_prepare_directory()	Verifies that the directory exists and is writable.
file_save()	Save a file object in the database.
file_save_data()	Saves a text string in the specified file. Depending on whether the \$replace parameter exists, the file will be replaced or renamed.
file_save_upload()	Stores a recently uploaded file to a new path.
file_scan_directory()	Search files within a directory. This function does not search inside files or hidden directories.
file_space_used()	Determines the total space used by a user (if we indicate a value for \$uid) or the entire system.
file_transfer()	Transfer a file to the browser via HTTP. This causes the file to download or to be opened in the browser.
file_upload_max_size()	Returns the maximum file size that forms can upload, as configured in php.ini.
file_validate()	Verifies whether a file meets the criteria defined by the validation function (hook_file_validate()).
file_validate_extensions()	Verifies that the file has an allowed extension.
file_validate_image_resolution()	If the file is an image, verifies that the dimensions are within the ranges indicated. If the file is not an image, it is simply ignored.
file_validate_is_image()	Verifies that the archive is an image.
file_validate_name_length()	Verifies that the file name has the appropriate number of characters to be stored in the database. The maximum number of allowed characters is 240.
file_validate_size()	Verifies that the file size is under the specified limit.

Functions related to URLs, filenames and URI Identifiers

The set of functions related to URLs, filenames and URI Identifiers: **F53.4**

F53.4

URL and URI functions

Some of the functions related to URLs, filenames and URIs.

Name	Description
<code>drupal_realpath()</code>	Returns the absolute path of a URI element
<code>file_build_uri()</code>	Given a relative path, constructs the URI path.
<code>file_create_url()</code>	Creates the download path of a file. This function creates the download path for both methods, public and private.
<code>file_uri_scheme()</code>	Returns the value of the scheme of a URI identifier. Por ejemplo, given the URI "public://example.txt", it will return "public".
<code>file_uri_target()</code>	Returns the part of the URI that comes after the scheme (target). For example , given the URI "public://example.txt", it returns "example.txt".
<code>file_valid_uri()</code>	Verifies that the URI provided has a valid scheme.
<code>file_munge_filename()</code>	Modifies the name of a file for security purposes. The function renames, for example, unknown extensions to prevent the uploaded file from being used maliciously. When we upload a file and see the message: "For security reasons, the file you uploaded has been renamed as %filename.", this function has come into play.
<code>file_unmunge_filename()</code>	Reverses the action taken by the <code>file_munge_filename()</code> function.
<code>file_default_scheme()</code>	Returns the default scheme.

Functions for registering file use

Registration functions use the **file_usage** table to store information about the use of files. This information is not only statistical, but also will be consulted by the file API to avoid physically deleting files being used by other modules (`file_delete()`). **F53.5**

F53.4

Functions for registering file use

These functions allow to identify which modules use the files.

Name	Description
<code>file_usage_add()</code>	Registers the files use by a module.
<code>file_usage_delete()</code>	Deletes a record from the <code>file_usage</code> table, thus indicating that the module no longer uses the file.
<code>file_usage_list()</code>	Returns an array indicating in which modules the file is used.

File processing functions without using file API

These functions **perform direct operations on the files, without invoking hooks or making changes to the database**. These functions should be used with caution. **F53.6**

For example, if we delete a file using `file_unmanaged_delete()` instead of `file_delete()`, it will not check if the file is being used by a module. In addition, the file information will remain in the database, resulting in an error when an attempt is made to download it.

Name	Description	
<code>file_unmanaged_copy()</code>	Copies a file to a new location without invoking the file API. Equivalent to using the API: <code>file_copy()</code>	F53.6 Direct file processing functions
<code>file_unmanaged_delete()</code>	Deletes a file directly, without calls to hooks or making changes to the database. Equivalent to using the API: <code>file_delete()</code>	These functions allow you to perform file operations without invoking Drupal API. These changes are done at the file level, but not applied to the database. The related hooks won't be launched.
<code>file_unmanaged_delete_recursive()</code>	Recursively deletes all files and folders from the path provided, without invoking the file API. Equivalent to using the API: <code>file_delete_recursive()</code>	
<code>file_unmanaged_move()</code>	Moves a file to a new location without invoking the file API. Equivalent to using the API: <code>file_move()</code>	
<code>file_unmanaged_save_data()</code>	Stores a string in the file specifying a destination without invoking file API. Equivalent to using the API: <code>file_save_data()</code>	

PHP functions

In addition to functions in Drupal API, it is helpful to know PHP functions for working with files. Some of them are presented here: **F53.7**

<http://www.php.net/manual/es/ref.filesystem.php>

Name	Description	
<code>file()</code>	Retrieves the contents of a file as an array.	F53.7
<code>file_get_contents()</code>	Retrieves the contents of a file as a string.	 PHP functions List of PHP functions related to files.
<code>file_put_contents()</code>	Writes a string to a file.	
<code>fopen()</code>	Opens a file.	
<code>fread()</code>	Reads a binary file in safe mode.	
<code>fwrite()</code>	Writes a binary file in safe mode.	
<code>fclose()</code>	Closes a pointer to an open file.	
<code>is_dir()</code>	Indicates that the path is a directory.	
<code>is_file()</code>	Indicates that the path is a file.	
<code>is_readable()</code>	Indicates if the files exists and can be read.	
<code>is_writeable()</code>	Indicates if the files exists and can be written to.	
<code>is_executable()</code>	Indicates if the file exists and is executable.	

Hooks

Some of the hooks that allow modules to interact with the system when actions are performed on files are: **F53.8**

F53.8

File hooks

List of some hooks related to files.

Name	Description
hook_file_download()	Controls access to private file downloads.
hook_file_copy()	Executes when a file is copied.
hook_file_delete()	Executes when file is deleted.
hook_file_insert()	Executes when a new file is inserted.
hook_file_load()	Allows the addition of information to the file object.
hook_file_mimetype_mapping_alter()	Allows modification of the file's MIME type
hook_file_move()	Executes when a file is moved to a new location.
hook_file_presave()	Executes when a file has been inserted or updated in the database.
hook_file_update()	Executes when a file is updated.
hook_file_url_alter()	Allows changes to a file's URL. What is actually modified is the URI identifier.
hook_file_validate()	Allows the addition of validation conditions on files.
hook_css_alter()	Allows the alteration of CSS files before displaying the page.

The \$file object and associated tables

The following table describes each attribute of the \$file object and and references the table where the value is stored. **F53.9**

F53.9

\$file attributes

List of attributes of the object \$file. The names of the tables that make use of each attribute are also indicated. Usually the fields in the table have the same name as the attribute in the \$file object.

Attribute	Description	Table (field)
fid	ID of the file	file_managed
uid	UID (user ID) associated with the file.	file_managed
filename	Name of the file (without components of the URL).	file_managed
uri	URI identifier of the file.	file_managed
filenime	MIME type of the file.	file_managed
filesize	File size in bytes.	file_managed
status	File status in bitmap format. The first 8 bits are reserved for Drupal core. The bit of least value indicates whether the file is temporary (0) or permanent (1). Older temporary files (depending on the value of DRUPAL_MAXIMUM_TEMP_FILE AGE constant), are removed from the database when the cron executes.	file_managed
timestamp	Date the file is uploaded to the table, in UNIX timestamp format.	file_managed

In addition to the **file_managed** table, which stores general information about the files, the **file_usage** table records information about the use made of the files:

- **fid.** File identifier.
- **module.** Name of the module that is using the file.
- **type.** Name of the type of object that is using the file.
- **id.** Identifier of the object that is using the file.
- **count.** Number of times that the file is used by that object.

Example of the use of the File API functions

As an example of using file system functions, let's analyze some fragments of the Image Block module code, studied at the intermediate level. The Image Block creates blocks with images.

The module is available at:

<http://drupal.org/project/imageblock>

Once installed and activated, we can create new blocks with images using the block administration page (**Add image block**). In addition to the generic block fields, the image block has the following additional fields: F53.10

- **Image.** Allows you to upload an image file.
- **Image style.** Selects the image style that will be applied to transform the uploaded image.
- **Link.** If we specify a path, the image will also be a link.
- **Link target.** Indicates where the page will open (for example, _self will open the page in the same window and _blank will open the page in a new window).

Home > Administration > Structure > Blocks

F53.10

Blocks

Block title
Expert in Drupal 7

Override the default title for the block. Use <none> to display no title, or leave blank to use the default block title.

Block description *
Block with image

A brief description of your block. Used on the [Blocks administration page](#).

Image
 druploon.small.png

Image style

Alternate text
Drupal training

This text will be used by screen readers, search engines, or when the image cannot be loaded.

Title
Drupal training

The title is used as a tool tip when the user hovers the mouse over the image.

Link
www.forcontu.com

Leave empty for no link.

Link target

Leave empty for no link.

Image Block module. Settings

This module allows attaching an image file to a block.

For now let's focus on implementation of the **hook_block_save()** function, which stores the image block configuration, including the image file uploaded via the form. **F53.11**

F53.11**Saving the file**

Implementation of **hook_block_save()**, where the uploaded file is saved using the API functions.

```
/***
 * Implement hook_block_save().
 */
function imageblock_block_save($delta = '', $edit = array()) {
  // Load the old file
  $old_file = imageblock_get_file($delta);
  if (!empty($edit['imageblock_file'])) {
    $file = $edit['imageblock_file'];
    // If the user uploads a file, save the file to a permanent
    // location.
    if (!empty($file->fid)) {
      // If a previous file exists, delete it.
      if (!empty($old_file->fid)) {
        file_usage_delete($old_file, 'imageblock', 'imageblock', $delta);
        file_delete($old_file);
      }

      $directory = file_default_scheme() . '://' .
        variable_get('imageblock_image_path', 'imageblock');

      // Prepare the directory
      file_prepare_directory($directory, FILE_CREATE_DIRECTORY);

      $destination = file_stream_wrapper_uri_normalize($directory
        . "/$file->filename");

      // Move the temporary file to the final location.
      if ($file = file_move($file, $destination, FILE_EXISTS_REPLACE)) {
        $file->status = FILE_STATUS_PERMANENT;
        file_save($file);
        file_usage_add($file, 'imageblock', 'imageblock', $delta);
      }
    }
  }

  db_update('imageblock')
    ->fields(array(
      'body' => $edit['body']['value'],
      'info' => $edit['info'],
      'format' => $edit['body']['format'],
      'fid' => !empty($file) ? $file->fid : (!empty($old_file) ? $old_file->fid : 0),
      'data' => serialize($data),
    ))
    ->condition('bid', $delta)
    ->execute();

  return TRUE;
}

/***
 * Return the file information for a given block ID.
 */
function imageblock_get_file($bid) {
  $fid = db_query("SELECT fid FROM {imageblock} WHERE bid = :bid",
    array(':bid' => $bid))->fetchField();
  return file_load($fid);
}
```

The module uses some of the API functions:

- Inside the **imageblock_get_file()** function a call is made to **file_load()** in order to load a file object. In particular, it searches the

database to see if the block already has an associated file, which will be considered the old file (`$old_file`). This file will be replaced by the new file attached to the form.

```
...
$old_file = imageblock_get_file($delta);
...

function imageblock_get_file($bid) {
  $fid = db_query('SELECT fid FROM {imageblock} WHERE bid = :bid',
    array(':bid' => $bid))->fetchField();
  return file_load($fid);
}
```

- To delete the old file (`$old_file`), we use two functions, `file_usage_delete()` and `file_delete()`. The first deletes the record from the `file_usage` table, which established the link between the file and the module that used it. `File_delete()` eliminates both the physical file and the file information stored in the `file_managed` table. Internally it checks that the file is not being used by another module.

```
if (!empty($old_file->fid)) {
  file_usage_delete($old_file, 'imageblock', 'imageblock', $delta);
  file_delete($old_file);
}
```

- To compose the URI identifier of the directory where the images of the module are stored, first use the `file_default_scheme()` function, which returns the default scheme and binds it to the value of the variable "`imageblock_image_path`", which contains the path defined for storing images.

Then, using `file_prepare_directory()`, we check that the directory exists and is writable. With the option `FILE_CREATE_DIRECTORY`, we are indicating that if the directory does not exist, it should be created.

Finally, we generate the final destination of the file, in URI format, adding the file name and normalizing the structure with `file_stream_wrapper_uri_normalize()`.

```
$directory = file_default_scheme() . '://' .
  variable_get('imageblock_image_path', 'imageblock');

// Prepare the directory
file_prepare_directory($directory, FILE_CREATE_DIRECTORY);

$destination = file_stream_wrapper_uri_normalize($directory
  . '/'.$file->filename');
```

- A file sent via the form is found, temporarily, in the variable `$file=$edit['imageblock_file']`. We will need to move it to its final location, defined in `$destination`. First we use `file_move()`, which moves the file to the specified destination. Then we modify the file status (`$file->status`) permanently. Finally, we save the file in the database, with `file_save()` and register the relationship of use between the module and the file, with `file_usage_add()`.

```
$file = $edit['imageblock_file'];
// Move the temporary file to the final location
if ($file = file_move($file, $destination, FILE_EXISTS_REPLACE)) {
  $file->status = FILE_STATUS_PERMANENT;
  file_save($file);
  file_usage_add($file, 'imageblock', 'imageblock', $delta);
}
```

- Finally, the function modifies the record of the 'imageblock' table to connect the updated block with the file identifier (fid) to which it has been attached.

53.3 Forms with files

In Unit 47 of this **Advanced Level** we saw how to implement forms. Let's briefly extend this information for working with files. The Image Block module will also be used to illustrate this section, as it implements a form that allows attachments.

Definition of forms

When an HTML form supports attachments , the <form> tag should include the 'enctype' attribute with the value '**multipart/form-data**':

```
<form action="procesar.php" enctype="multipart/form-data" method="post">
...
</form>
```

In Drupal 6 it was necessary to modify the form's general attributes by means of the line:

```
$form['#attributes'] = array('enctype' => "multipart/form-data");
```

However, in Drupal 7 the system automatically adds to the code for forms fields of the 'file' type, so it is not necessary to indicate it in the construction of the form.

Therefore, we can create the file field directly on the form. **F53.12**

F53.12

File element

Example of using the File element which allows us to attach files to the form.

```
$form['new_upload'] = array(
  '#type' => 'file',
  '#title' => t('Attach new file'),
  '#size' => 40,
);
```

Validating and submitting forms

In the functions for validating and submitting forms, **_validate()** and **_submit()**, we can access the file attached to the form using the **file_save_upload()** function.

```
file_save_upload($source, $validators = array(), $destination = FALSE,
    $replace = FILE_EXISTS_REPLACE)
```

When we upload a file via a form, we should use the function **file_save_upload()** to copy the file to a new location. Once the call to **file_save_upload()** is made, the file will be stored in the table **file_managed**, though marked as temporary.

To make the file permanent, we need to change its state and save it with **file_save()**. **F53.13**

```
$file = file_save_upload('new_upload', $validators);

$file->status = FILE_STATUS_PERMANENT;
file_save($file);
file_usage_add($file, 'imageblock', 'imageblock', $delta);
...
```

F53.13**Save the file**

Functions to the save the file sent by the form.

The function **file_save_upload()** uses the following parameters:

- **\$source**. Name of the form's field that was used to upload the file (in the example above, 'new_upload').
- **\$validators**. An array of return functions that validate the file (name, size, etc.).
- **\$destination**. The designated directory for the file, in URI format. If a destination is not specified, the temporary directory ("temporary://") will be used.
- **\$replace**. Indicates the action to take when a file with the same name exists in that location (\$destination):
 - o FILE_EXISTS_REPLACE. Replace the file.
 - o FILE_EXISTS_RENAME. Rename the uploaded file (_1, _2, etc.)
 - o FILE_EXISTS_ERROR. Do nothing and return an error.

The function returns an object with the file's information, or FALSE if an error occurred.

Let's return to the example module, **Image Block**, in order to analyze how to implement the form and the validation and send functions. **Image Block** generates the block configuration from **hook_block_configure()**, calling these functions: **F53.14**

```
/** 
 * Implements hook_block_configure().
 */
function imageblock_block_configure($delta = '') {
  if ($delta) {
    $custom_block = block_imageblock_get($delta);
  }
  else {
    $custom_block = array();
  }
  $form = block_custom_block_form($custom_block);
  imageblock_configure_form($form, $custom_block);
  return $form;
}
```

F53.14**Block with file**

Functions that build the block configuration form.

- **block_custom_block_form()**. This is an API function that builds a block configuration form. This generic form will be used as a base from which we can add other module-specific fields.
- **imageblock_configure_form()**. This is the function that adds to the base form the module-specific fields, including the “file” type field for uploading the image file. **F53.15**

F53.15**File field**

Field of type File to upload an image file.

```
function imageblock_configure_form(&$form, $block = NULL) {
...
$form['imageblock'] = array(
  '#type' => 'file',
  '#title' => $title,
  '#description' => $description,
  '#size' => 40,
);
...
}
```

Once the form is submitted, the uploaded file is processed by the validation function, using **file_save_upload()**. **F53.16**

F53.16**Form validation**

From the form validation we use **file_save_upload()** to save the file sent by the form. The file will remain temporary.

```
function imageblock_configure_form_validate($form, &$form_state) {
...
// Save new file uploads.
$file = file_save_upload('imageblock', $validators);
if ($file === FALSE) {
  form_set_error('imageblock', t('Failed to upload the image;
the %directory directory doesn't exist or is not writable.', array('%directory' => variable_get('imageblock_image_path', 'imageblock'))));
}
elseif ($file !== NULL) {
  $form_state['values']['imageblock_file'] = $file;
}
}
```

The temporary file generated is referenced with **\$form_state['values']**, so that it will also be available in the send form function:

```
$form_state['values']['imageblock_file'] = $file;
```

Being a block configuration form, form submission is performed through **hook_block_save()**. The **\$edit** parameter will have the values submitted by the form, including the image file assigned in the validation function. As we already saw, after preparing the directory and setting the final destination of the posting, set the status of the file as permanent and store it in the database with **file_save()**. **F53.17**

F53.17**Permanent file**

Functions to save the file and change its status to permanent.

```
/**
 * Implements hook_block_save().
 */
function imageblock_block_save($delta = '', $edit = array()) {
  $file = $edit['imageblock_file'];
  // Moves the temporary file to the final location
  if ($file = file_move($file, $destination, FILE_EXISTS_RENAME)) {
    $file->status = FILE_STATUS_PERMANENT;
    file_save($file);
    file_usage_add($file, 'imageblock', 'imageblock', $delta);
  }
}
```

Control of file permissions

53.4

By implementing **hook_file_download()**, we control access to the files that are using the private download method. Through this function we can also modify the HTTP headers file.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_file_download/7

hook_file_download(\$uri)

As input this function requires the **\$uri** parameter with the file ID.

The function returns -1 if the user does not have access to the file. If the user has permission, it returns an array with the headers that should be added to the download file. If the module does not act on that file, it simply returns NULL.

In the following example, we first check that the file exists in the database and is registered as in use by the 'file_example' module in the **file_usage** table. If so, we check using **user_access()** that the user has the 'administer private file' permission. If so, downloading or viewing the file is allowed, returning the HTTP headers to be applied.

```
/**
 * Implements hook_file_download().
 */
function file_example_file_download($uri) {
// Verify that the file exists and is used by the module
  if ($file = db_query('SELECT filemime, filesize, timestamp FROM
{file_managed} m INNER JOIN {file_usage} u ON m.fid = u.fid WHERE
uri = :uri AND module = :module', array(':uri' => $uri, ':module'
=> 'file_example'))->fetchObject()) {

    // Verify that the user has the permission
    if (user_access('administer private file')) {
      return array(
        'Cache-Control' => 'max-age=1209600, private, must-revalidate',
        'Content-Length' => $file->filesize,
        'Content-Type' => $file->filemime,
        'Expires' => gmdate(DATE_RFC1123, REQUEST_TIME + 1209600),
        'Last-Modified' => gmdate(DATE_RFC1123, $file->timestamp),
      );
    }
    return -1;
}
}
```

F53.18

hook_file_download()

From **hook_file_download()** we can control access to the file and send the appropriate HTML headers for download.

As seen in this example, to prevent private files from being cached, 'Cache-Control'=>'private' has been added, which alerts the browser not to store the file in shared caches.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

54 Search system

The **Search** module in Drupal core activates Drupal's search system.

Drupal indexes both website content and users so that subsequent queries will be fast and efficient. The indexing process is performed periodically by the system **cron**.

The search engine does not directly search the entire contents of the site. What the search engines does is analyze pages and save relevant information to expedite the process of subsequent searches. Drupal maintains an **index** of the contents of the site, and the construction of this index is what is known as **indexing**. When we create new website content, it should be indexed (added to the search index). Otherwise, the new content will not appear in the search results.

There are several ways to interact with the Drupal search system:

- Search nodes.
- Search other elements of the site, using custom searches.
- Perform a personalized search directly on elements of the search object, or on the index, in which case we have to implement the indexing of those items.

In this unit we will look at all of these ways to interact with Drupal search system.

Comparative D7/D6

Search system

Although the search and indexing module is the same as previous versions, in Drupal 7 new hooks are used that facilitate the definition of custom searches and content indexing.

Unit contents

54.1 Introduction to the search system	324
54.2 Searching nodes	326
54.3 Custom Search	330
54.4 Indexing of content	334

54

54.1

Introduction to the search system

In **Unit 17** of the **Beginner Level** we studied the search system, provided by the core **Search** module.

The Drupal search system indexes both website content and users, so that subsequent queries are fast and efficient. The indexing process is performed periodically by the system **cron**.

Indexing of the searches

Configuration of searches is available at:

URL Search settings
[/admin/config/search/settings](#)

Administration ⇒ **Configuration** ⇒ **Search and metadata** ⇒ **Search settings**

The first thing we find is the **Indexing status**. The search engine does not directly search the entire contents of the site. What the search engine does is analyze the pages and save the relevant information to expedite the process of subsequent searches.

Drupal maintains an index of the site contents, and the construction of this index is what is known as indexing. When we create new content on the website, it should be indexed (added to the search index). Otherwise, new content will not display in search results. Similarly, modified nodes will have to be reindexed in order to apply the corresponding changes to the index.

In **Indexing status** shows the percentage of content that has been indexed as well as the number of items (or content) that are currently pending indexing.

F54.1

F54.1

Indexing status

The Indexing status indicates the percentage of items that have been indexed and the number of items waiting to be indexed. These items will be indexed in successive cron executions.

The search engine maintains an index of words found in your site's content. To build and maintain this index, a correctly configured **cron** maintenance task is required. Indexing behavior can be adjusted using the settings below.

INDEXING STATUS

0% of the site has been indexed. There are 50 items left to index.

Re-index site

INDEXING THROTTLE

Number of items to index per cron run

100

The maximum number of items indexed in each pass of a **cron** maintenance task. If necessary, reduce the number of items to prevent timeouts and memory errors while indexing.

In order to index the site, **execution of the system cron** is required. Since indexing is a task that, depending on the volume of content on the site, can consume significant resources of time and memory on the server, you should limit the number of items to be indexed each time the cron runs. The default is set to 100. Use the **Re-index site** button only when you need to rebuild the index from scratch. You will need to run the cron to index the site, either automatically or manually. On a site in production the cron should execute automatically and periodically, so that the site frequently indexes newly created

content. On a site in development, however, it is more advisable to work with manual execution of the cron.

Interacting with the search through programming

The Search interface provides the functions and hooks needed to work with searches, both with the results and with the indexing of the site's content.

<http://api.drupal.org/api/drupal/modules!search!search.module/group/search/7>

There are three ways to interact with the Drupal search system:

- If we want to search for nodes, we can implement **hook_node_update_index()** and **hook_node_search_result()**, which are functions of the Node API. This is necessary only to add additional hidden fields, since by default all of the visible content of the nodes is already indexed. We will study this scenario in **section 54.2**.
- The second method is to implement **hook_search_info()**. This function creates on the search page specific to the module (**/search**) an additional tab with a simple search form. We will execute this search with **hook_search_execute()**. We will study this scenario in **section 54.3**.
- Lastly, by implementing **hook_update_index()** we can interact with the Drupal indexing system. We will expand on this point in **section 54.4**.

To implement the functions of this unit we will develop the module **Search Forcontu (search_forcontu)**.

Tables of the Search module

The **Search** module adds the following tables:

- '**search_dataset**'. Stores items that can be searched. If the field 'reindex' has a value other than 0, it indicates that the content must be re-indexed.
- '**search_index**'. Stores the search index, associating words to items.
- '**search_total**'. Stores the "quantity" of occurrences of each word in the index.
- '**search_node_links**'. Stores items that link to other nodes. Improves the position of a node that is frequently linked to other nodes.

We will work directly with the **search_dataset** table only. The other tables will be updated by the API functions that we will use to search.

54.2 Searching nodes

To search nodes we implement **hook_node_update_index()** and **hook_node_search_result()**.

hook_node_update_index()

The function **hook_node_update_index()** executes when the node is being indexed by the search engine, which allows us to add information to be indexed.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_update_index/7

hook_node_update_index(\$node)

The function takes as a parameter the node being indexed (\$node) and returns a string (rendered output) that is added to the indexed content.

Let's consider a very simple example. The following function will check if the node is promoted to the main page. **F54.2** If so, we'll add the string "featured" to the index.

F54.2

hook_node_update_index()

Acts when the node is being indexed.

```
/***
 * Implements hook_node_update_index().
 */
function search_forcontu_node_update_index ($node) {
  if ($node->promote == 1) {
    return 'featured';
  }
}
```

Figure F54.3 shows the result of searching for the string 'featured' **before implementing this functionality**. As the chain is not included in any of the nodes, the search doesn't return any results.

F54.3

Original search

Search of "featured" before implementing the functionality. Since no node contains this string the result will be empty.

The screenshot shows a search interface with a header containing 'Content' and 'Users' tabs. Below the tabs is a search bar with the placeholder 'Enter your keywords' and a value 'featured'. To the right of the search bar is a magnifying glass icon. At the bottom of the interface is a button labeled 'Advanced search'.

Once the module is installed and the index is rebuilt (**Re-index site** and run the **cron**), searching for the string "featured" returns the nodes that are promoted to the frontpage (in addition to those that already have this string in their content).

F54.4

Search

Enter your keywords

Search results

Accumsan Ex Nibh

Gilvus pneum tum. Cogo conventio cui macto nimis premo roto usitas. Dolore jus paratus quae voco. Eligo humo loquor natu quadrum scisco tum zelus. ...

Anonymous (not verified) - 10/25/2014 - 01:44 - 0 comments

Paratus Praemitto Quibus Turpis

Abico camur iriure neo saluto secundum sed. Appellatio conventio dolus eligo pecus te valetudo. ...

Anonymous (not verified) - 10/25/2014 - 01:54

Aliquip Jus Nostrud

Esca exerci genitus lenis macto proprius ymo. Eligo eu facilisi incassum luctus tum vindico ymo. Causa cogo damnum distineo eu gravis metuo nisl plaga saluto. Dolus incassum iustum minim turpis. Bene consecuetuer huic minim...

Anonymous (not verified) - 10/25/2014 - 01:44 - 0 comments

It is important to understand that the string "featured" is not added to the body of the node, so it will not be visible at any time, nor to the search results, nor the full node display.

hook_node_search_result()

The function **hook_node_search_result()** executes when the node is displayed as result of a search.

http://api.drupal.org/api/drupal/modules--node--node.api.php/function/hook_node_search_result/7

hook_node_search_result(\$node)

This function takes as a parameter the node being shown in the search results (\$node), and should return an associative array with additional information displayed alongside the results. **F54.5**

```
/***
 * Implements hook_node_search_result().
 */
function search_forcontu_node_search_result($node) {
  if ($node->promote == 1) {
    return array('featured' => t('Featured content'));
  }
}
```

Additional information is added to the string that shows the author, the date of the last update, and the number of comments. **F54.6**

F54.4

Search of 'featured' content

Once the function is implemented, search for "featured" returns promoted to the frontpage nodes, although they don't contain the string "featured" in their content.

F54.5

hook_node_search_result()

Acts when the node is being displayed as a search result.

F54.6**Results with additional content**

We added additional content as the string "Featured content" which will be shown only on promoted to frontpage content.

Search

Content Users

Enter your keywords
neo

Advanced search

Search results**Neo Pneum Sagaciter Veniam**

... gemino jumentum occuro pagus sagaciter. Cogo inhibeo lucidus **neo** patria. Et gemino molior mos nunc praesent refoveo vulpes. Adipiscing ... secundum ullamcorper. Appellatio ea vulputate. Diam ille **neo**. Bene eum humo. Cogo hendrerit ibidem incassum natu oppeto populus vicis ...

Anonymous (not verified) - 10/25/2014 - 02:01 - 0 comments - Featured content

Bene

... Cogo hendrerit hos inhibeo letalis luptatum melior mos **neo** roto. Acsi antehabeo decent defui esse ideo olim pagus tincidunt utinam. ... hendrerit inhibeo magna natu paulatim. Aliquip consectetur **neo**. Consectetur lenis nimis oppeto suscipit. Erat humo importunus ...

Anonymous (not verified) - 10/25/2014 - 01:44 - 0 comments - Featured content

Incassum Typicus Vulpes

... augue ille praemitto velit volutpat. Acsi enim loquor meus **neo** obruo praesent quibus virtus. Eligo eu melior ratis vulpes. Ex laoreet ... File: Blandit decent dolore gemino gravis iusto **neo** saluto. Dolore huic incassum occuro os pagus veniam. ...

admin - 10/25/2014 - 01:44 - 0 comments

Alternatively, we can access the template for search results (search-result.tpl.php) and modify the presentation by adding new content to the display of the node (in the results). In the template file, new content will be included in the array **\$info_split** (for example, **\$info_split['highlight']**). **F54.7**

F54.7**Template search-result.tpl.php**

From the search results' template file we can access to the new content added through the variable **\$info_split**.

```
<li class="<?php print $classes; ?><?php print $attributes; ?>>
<?php print render($title_prefix); ?>
<h3 class="title"<?php print $title_attributes; ?>>
  <a href="<?php print $url; ?>"><?php print $title; ?></a>
  <?php if (isset($info_split['featured'])): ?>
    <span class="info-featured">
      [<?php print $info_split['featured']; ?>]
    </span>
  <?php endif; ?>
</h3>
<?php print render($title_suffix); ?>
<div class="search-snippet-info">
  <?php if ($snippet): ?>
    <p class="search-snippet"><?php print $content_attributes;
    ?>><?php print $snippet; ?></p>
  <?php endif; ?>
  <?php if ($info): ?>
    <p class="search-info"><?php print $info; ?></p>
  <?php endif; ?>
</div>
</li>
```

As an example we added the text [Featured content] next to the title of the node, only to content promoted to the main page content. **F54.8**

Search

Content Users

Enter your keywords

Advanced search

F54.8

Template modified

Presentation of search results after modifying the template.

Search results

[Neo Pneum Sagaciter Veniam](#) [Featured content]

... gemino jumentum occuro pagus sagaciter. Cogo inhibeo lucidus **neo** patria. Et gemino molior mos nunc praesent refoveo vulpes. Adipiscing ... secundum ullamcorper. Appellatio ea vulputate. Diam ille **neo**. Bene eum humo. Cogo hendrerit ibidem incassum natu oppeto populus vicis ...

Anonymous (not verified) - 10/25/2014 - 02:01 - 0 comments - Featured content

[Bene](#) [Featured content]

... Cogo hendrerit hos inhibeo letalis luptatum melior mos **neo** roto. Acsi antehabeo decet defui esse ideo olim pagus tincidunt utinam. ... hendrerit inhibeo magna natu paulatim. Aliquip consectetur **neo**. Consectetur lenis nimis oppeto suspicit. Erat humo importunus ...

Anonymous (not verified) - 10/25/2014 - 01:44 - 0 comments - Featured content

[Incassum Typicus Vulpes](#)

... augue ille praemitto velit volutpat. Acsi enim loquor meus **neo** obruo praesent quibus virtus. Eligo eu melior ratis vulpes. Ex laoreet ... File: Blandit decet dolore gemino gravis iusto **neo** saluto. Dolore huic incassum occuro os pagus veniam. ...

admin - 10/25/2014 - 01:44 - 0 comments

In **Unit 56** we'll further explore themes and templates. For this example, we'll need to copy the template file **search-result.tpl.php**, which can be found in /modules/search in the active theme folder. We must never modify core templates or modules, just as we must not change the core theme. Therefore, the modified theme should be located, for example, in /sites/all/themes.

54.3 Custom Search

Function hook_search_info()

The function **hook_search_info()** allows us to define a new type of search.

http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_info/7

When we define a new type of search, we create a new tab in /search, from where this custom search will be performed. **F54.9**

F54.9

New search tab

In `hook_search_info()` we define a new custom search tab for the module.

Search

The **hook_search_info()** function returns an array with the following:

- '**title**'. Title of the tab. If it is not specified, the name of the module will be used.
- '**path**'. Path for the search.
- '**conditions_callback**'. A return function for adding additional search terms (we will not use this function in the following examples).

The newly defined search can be performed on any item or table of the database, with or without prior indexing of the content.

As an example, we will implement a search of article content types (article). The search will be performed directly on the title of the article nodes, without indexing. **F54.10**

F54.10

hook_search_info()

Defines the search tab
News with URL
`/search/news`.

```
/***
 * Implements hook_search_info().
 */
function search_forcontu_search_info() {
  return array(
    'title' => 'News',
    'path' => 'news',
  );
}
```

By implementing **hook_search_info()** we are defining only the tab title and its URL. The Search module is responsible for generating a simple search form.

Although the search will not return any results, we can enable it to display the tab and the search form. We will do so from the Search Options: **F54.11**

URL Search options

`/admin/config/search/settings`

Administration ⇒ Configuration ⇒ Search and metadata ⇒ Search settings

ACTIVE SEARCH MODULES

Node
 Search Forcontu
 User

Choose which search modules are active from the available modules.

Default search module

Node
 Search Forcontu
 User

Choose which search module is the default.

F54.11**Enable search**

From Search settings we have to enable the new search Search Forcontu added by the module.

Once the **Search Forcontu** module is enabled, the **News** tab will be displayed next to the default search tabs, **Content** and **Users**.

F54.12**Search**

[Content](#) [News](#) [Users](#)

Enter your keywords

F54.12**New search tab**

Although the search form is shown, we still have to implement the function that executes the search.

hook_search_execute() function

In order for the search defined in **hook_search_info()** to return results, we must implement **hook_search_execute()**, which executes the search based on the provided keywords (\$keys).

http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_execute/7

hook_search_execute(\$keys = NULL, \$conditions = NULL)

The input parameters of the function are:

- **\$keys**. String searched by the user.
- **\$conditions**. Optional array with additional conditions.

The function will return an array with the obtained results, where each item will have the following fields:

- **'link'** (required). URL of the item.
- **'type'**. Type of item
- **'title'** (required). Name of title of the item.
- **'user'**. Author.
- **'date'**. Date the item was last modified.
- **'extra'**. Array with additional information about the item.
- **'snippet'**. Summary or preview of the item.
- **'language'**. Language.

F54.13**hook_search_execute()**

Example function that performs a search on the titles of the nodes of type Article (article). The search is performed on the node table directly, without indexing the content.

```
/*
 * Implements hook_search_execute().
 */
function search_forcontu_search_execute($keys = NULL, $conditions = NULL) {
  if (!$keys) {
    $keys = '';
  }

  $results = array();

  $query = db_select('node', 'n');
  $search = $query
    ->condition('n.status', NODE_PUBLISHED)
    ->fields('n', array('nid', 'title'))
    ->condition('n.type', 'article')
    ->orderBy('n.changed', 'DESC')
    ->condition('n.title', '%' . db_like($keys) . '%', 'LIKE')
    ->extend('PagerDefault')
    ->limit(10)
    ->execute();

  foreach ($search as $item) {
    $node = node_load($item->nid);
    $uri = entity_uri('node', $node);

    $results[] = array(
      'link' => url($uri['path'], array_merge($uri['options'],
                                                array('absolute' => TRUE))),
      'title' => $node->title,
      'date' => $node->changed,
    );
  }
  return $results;
}
```

As an example, let's perform a search for the string added in **\$keys** on nodes of the article type (**article**), directly on the **title** field of the **node** table. **F54.13**

Using the '**PagerDefault**' extension in the `db_select()` statement, the system will **paginate the results** according to the number of items per page specified by the `limit()` method.

As a result of the search, lets return the title of the node, the link, and the date it was created. **F54.14**

F54.14**Search**

Search results as defined in `hook_search_execute()`. Only the title and date of each item is displayed.

Search

Content	News	Users
<input type="text" value="Enter your keywords"/> neo 🔍		

Search results

Distineo Facilisi Jumentum Praesent

10/25/2014 - 10:16

Neo Quibus

10/25/2014 - 10:16

In Drupal core we can find more elaborate examples of implementing **hook_search_execute()**.

- The **User** module implements **user_search_execute()**, which is an example of a search performed without indexing the elements.

http://api.drupal.org/api/drupal/modules--user--user.module/function/user_search_execute/7

- The **Node** module implements **node_search_execute()**, performing a search on previously indexed content, as we will see in the following section.

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_search_execute/7

hook_search_page() function

The function **hook_search_page()** allows us to modify the output or presentation of search results.

http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_page/7

If we don't use this function, the functions `theme('search_results')` and `theme('search_result')` will be used as a template for the results, using the **search-results.tpl.php** and **search-result.tpl.php** templates, respectively. Using theme functions and templates will be studied in **Unit 56**. F54.15

```
/** 
 * Implements hook_search_page().
 */
function search_forcontu_search_page($results) {
  $output['prefix']['#markup'] = '<h2>News</h2> <ol class="search-results">';
  foreach ($results as $entry) {
    $output[] = array(
      '#theme' => 'search_result',
      '#result' => $entry,
      '#module' => 'search_forcontu',
    );
  }
  $output['suffix']['#markup'] = '</ol>' . theme('pager');
}
return $output;
}
```

F54.15

hook_search_page()
Allows you to modify the output of the search results page.

hook_search_admin() function

The **hook_search_admin()** function allows us to add new fields to the configuration of the search form.

http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_admin/7

For an example of its use, consult **node_search_admin()**.

http://api.drupal.org/api/drupal/modules--node--node.module/function/node_search_admin/7

[node.module/function/node_search_admin/7](#)

hook_search_access() function

The **hook_search_access()** function allows us to control access to the new search tab defined by the module. **F54.16**

[http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_access/7](#)

F54.16

hook_search_access()

Controls Access to the custom search implemented by the module.

```
/***
 * Implements hook_search_access().
 */
function search_forcontu_search_access() {
  return user_access('access content');
}
```

54.4 Indexing of content

Functions hook_update_index() and search_index()

The **hook_update_index()** function is used to index the contents or items of the module, internally calling the **search_index()** function to add each item to the core indexing system.

As nodes are already indexed by the Node module (using `node_update_index()`), we should use `hook_update_index()` only to index content other than nodes.

The execution of **hook_update_index()** will be called by the system **cron**, provided that the module has implemented a custom search using **hook_search_info()** and that the search has been activated in **Search settings**.

[http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_update_index/7](#)

The **search_index()** function performs indexing on a particular item. This function must be called with **hook_update_index()** in order to index each item.

[http://api.drupal.org/api/drupal/modules--search--search.module/function/search_index/7](#)

As an example of using the **hook_update_index()** and **search_index()**, let's analyze the **node_update_index()** function, which indexes the content of nodes. **F54.17**

The operation is very simple:

- The **search_dataset** table stores information about indexed content or content that needs to be reindexed.
- A query is performed on the **node** table in conjunction with **search_dataset** to get only the nodes that need to be indexed.

- The variable **search_cron_limit** determines the number of items to index during each execution of the cron. The value of this variable will be used as the **limit** value in the statement, with 100 being the default value.
- For each node returned by the statement, we get the rendered content.
- Each node is indexed through a call to **search_index()**.

```
/***
 * Implements hook_update_index().
 */
function node_update_index() {
  $limit = (int) variable_get('search_cron_limit', 100);

  // Look for nodes that have not been indexed
  // or that need to be reindexed.
  $result = db_query_range("SELECT n.nid FROM {node} n
    LEFT JOIN {search_dataset} d ON d.type = 'node' AND d.sid =
      n.nid WHERE d.sid IS NULL OR d.reindex <> 0
    ORDER BY d.reindex ASC, n.nid ASC", 0, $limit);

  // Prepare and index each node
  foreach ($result as $node) {
    $node = node_load($node->nid);

    variable_set('node_cron_last', $node->changed);

    // Get the rendered content of the node
    node_build_content($node, 'search_index');
    $node->rendered = drupal_render($node->content);

    $text = '<h1>' . check_plain($node->title) . '</h1>' . $node-
    >rendered;

    // Add additional information that will not visible
    $extra = module_invoke_all('node_update_index', $node);
    foreach ($extra as $t) {
      $text .= $t;
    }

    // Create the index
    search_index($node->nid, 'node', $text);
  }
}
```

F54.17**hook_update_index()**

Implementation of hook_update_index() by Node module.

For each node to index a call to the function search_index() is made, which is responsible for adding the content to the search index.

Searching with indexing

In the previous section we used **hook_search_execute()** to search without indexing.

If the contents of the module were previously indexed by **hook_update_index()**, we must make some changes to the implementation of **hook_search_execute()**.

http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_execute/7

As an example of searching indexed content, we will review the **node_search_execute()** function, which is the implementation that Node module performs. The following is a simplified version of this function. **F54.18**

F54.18

hook_search_execute()
Implementation of the search execution function by obtaining the results from the index.

```
/*
 * Implements hook_search_execute().
 */
function node_search_execute($keys = NULL, $conditions = NULL) {
  // Construct the statement to look for nodes
  $query = db_select('search_index', 'i',
    array('target' => 'slave'))
  ->extend('SearchQuery')
  ->extend('PagerDefault');

  $query->join('node', 'n', 'n.nid = i.sid');
  $query
  ->condition('n.status', 1)
  ->addTag('node_access')
  ->searchExpression($keys, 'node');
  //...

  // Execute the statement
  $find = $query
  ->limit(10)
  ->execute();

  // Return the output for each result
  $results = array();
  foreach ($find as $item) {
    // Obtiene the rendered output of the node
    $node = node_load($item->sid);
    $build = node_view($node, 'search_result');
    unset($build['#theme']);
    $node->rendered = drupal_render($build);

    $uri = entity_uri('node', $node);
    // Add to $results each result of the search
    $results[] = array(
      'link' => url($uri['path'], array_merge($uri['options'],
array('absolute' => TRUE))),
      'type' => check_plain(node_type_get_name($node)),
      'title' => $node->title,
      'user' => theme('username', array('account' => $node)),
      'date' => $node->changed,
      'node' => $node,
      'extra' => $extra,
      'score' => $item->calculated_score,
      'snippet' => search_excerpt($keys, $node->rendered),
      'language' => $node->language,
    );
  }
  return $results;
}
```

The main difference lies in the statement used to obtain the search results. Instead of making a direct query to the node table, the statement is executed on the **'search_index'** table with the alias 'i' acting as the principle table. We can also add, with a **'join'** statement, the table that contains the items to search (in this example, the **node** table), which will be assigned to the **i.sid** field (`$query->join('node', 'n', 'n.nid = i.sid')`).

The statement is supplemented by the **'SearchQuery'** extension and **searchExpression()** method, which is responsible for executing the indexed search. Finally, we add the **'PagerDefault'** extension, which will generate a pager according to the value of **limit()**.

After the query is performed, the rest of the function is similar to a search without indexing. Simply arrange to complete the fields of the **\$results** array with all of the resulting items.

hook_search_status() function

The function **hook_search_status()** returns the indexing status of the items of a module.

http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_status/7

It should return an associative array with the fields:

- '**remaining**'. Number of items that remain to be indexed.
- '**total**'. Total number of indexable items.

As an example, let's looks at the implementation of **hook_search_status()** for the **Node** module.

The total number of items ('total') corresponds to the number of published nodes.

To get the number of items that remain to be indexed ('remaining'), we compare the published contents with 'search_dataset' table. If the content is not in this table that means that it has not yet been indexed. If the content is in the table but the 'reindex' field has a value other than 0, it must be re-indexed. **F54.19**

```
/** 
 * Implements hook_search_status().
 */
function node_search_status() {
  $total = db_query('SELECT COUNT(*) FROM {node}')->fetchField();
  $remaining = db_query('SELECT COUNT(*) FROM {node} n LEFT JOIN
{search_dataset} d ON d.type = \'node\' AND d.sid = n.nid WHERE
d.sid IS NULL OR d.reindex <> 0')->fetchField();

  return array('remaining' => $remaining, 'total' => $total);
}
```

F54.19

hook_search_status()

Returns the status of the indexing. Indicates the total number of indexable elements and the number of pending items.

hook_search_reset() function

This function executes when the search index is going to be rebuilt.

http://api.drupal.org/api/drupal/modules--search--search.api.php/function/hook_search_reset/7

The modules that implement searches with indexed content must update their items in the table 'search_dataset', assigning the current date (REQUEST_TIME) to the 'reindex' field. **F54.20**

```
/** 
 * Implements hook_search_reset().
 */
function node_search_reset() {
  db_update('search_dataset')
    ->fields(array('reindex' => REQUEST_TIME))
    ->condition('type', 'node')
    ->execute();
}
```

F54.20

hook_search_reset()

Function that operates when the search index is rebuilt. The module must update the table search_dataset so that its elements are reindexed.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

55 Translating modules

The method Drupal uses to translate the interface is based on **string replacement**. We have already seen this during the module implementation through the use of the function `t()`, we can tell the system that the strings can be translated. These strings will be available in the administration area of the translation interface for manual translation by the administrator.

In this unit we will look at how to incorporate the translations directly in the module, so they are available from the moment that it is installed and active on the site.

Comparative D7/D6

Translation

The translation system for Drupal 7 is similar to Drupal 6.

The modules explained and the additional software used is also available for both versions of Drupal.

Unit contents

55.1 Translation of modules	340
55.2 Translation template extractor Module	343
55.3 Translating with Poedit	346

55

F55.1

Translation of modules

The method Drupal uses to translate the interface is based on **string replacement**. Drupal uses an algorithm to identify which language to display its content. If this language is different from Drupal's default language, English, the system will try to replace the original chains by the strings in the desired language. When a translation is not necessary, Drupal will always display the strings in English, the default language.

Function t()

As we have previously stated, the translation of strings is done through the function **t()**, that is the Drupal translation function.

t(\$string, array \$args = array(), array \$options = array())

<http://api.drupal.org/api/drupal/includes--bootstrap.inc/function/t/7>

In addition to enabling the translation of simple strings, the function **t()** also allows the use of parameters or patterns in order to generate dynamic text strings. Below are several examples of translatable strings through the function **t()**. **F55.1** It is important to indicate that the HTML code must be left outside of the translation supply chain and, therefore, should not be passed to the function **t()**.

F55.1

Function t()

Examples of use of the translation function **t()**.

```
//Simple text
$text = t('An error occurred.');

//Simple text with HTML (outside of the t() function)
$text = '<p>' . t('An error occurred.') . '</p>';

//Text with parameters
$text = t('The username is @name and the uid is @uid',
      array('@name' => $account->name), '@uid' => $account->uid));
```

Locale Module

The Drupal core contains the **Locale** module that, as we already know, is used to translate our website, perform a string search, translate strings, import and export translations, etc. Therefore, to perform the translation, the Locale module must be enabled. If we have installed an additional language during the installation of the site, for example, Spanish, the system module will be enabled automatically.

Translation Files .po

Exports and imports of translation strings are made through files with the extension **.po** (**Portable Object Files**). Each **.po** extension corresponds to a language, and the file name will identify the country and language appropriately: **es.po** (Spanish), **de.po** (German), **fr.po** (French), etc.

A file with the **.po** extension consists of a header with a series of metadata and then the translated strings. Each translated string is comprised of three elements: **F55.2**

- A comment line that indicates the place where the chain is located (filename and line). In the same comment line, that starts with #: you can include multiple filenames and line numbers if the text is repeated in different points of the module.
- The **msgid** parameter that contains the original string.
- The **msgstr** parameter that contains the translated string that corresponds to the language file.

```
#: includes/webform.admin.inc:263,258
msgid "Title"
msgstr "Título"

#: includes/webform.pages.inc:228,232
msgid "Save configuration"
msgstr "Guardar configuración"

#: includes/webform.report.inc:294,282;
includes/webform.submissions.inc:371,489,339,449
msgid "Cancel"
msgstr "Cancelar"
```

F55.2**File .po**

Structure of each translation string.

The files **.po**, like other module files, must also be encoded in the format **UTF-8 sin BOM**.

It is not common to find the files .po included in the module. Generally we will download the latest version of the translation of a module from **localize.drupal.org**. However, when we develop a module that we are not going to share with the community or if we want to provide the translation after the installation we can include that in the file, .po.

In **Figure F55.3** it shows the translation files for different languages which incorporate the **Potx** module, grouped in the **translations** folder. In addition to the .po files, in the module we must also include the file **module_name.pot**, that is the translation template with the original strings.

All	Name	Type	Size	Owner	Group	Perms	Mod Time
<input type="checkbox"/>	<input type="checkbox"/> de.po	PO File	7274	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> el.po	PO File	5160	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> es.po	PO File	5232	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> fr.po	PO File	4670	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> hu.po	PO File	4470	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> ja.po	PO File	4445	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> nl.po	PO File	3854	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> pl.po	PO File	4763	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> potx.pot	POT File	5726	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> pt-br.po	PO File	5021	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> ru.po	PO File	3557	1286	1288	rw-r--rw-	Oct 25 09:51
<input type="checkbox"/>	<input type="checkbox"/> uk.po	PO File	4936	1286	1288	rw-r--rw-	Oct 25 09:51

F55.3**Translation files for a module**

.po translation files for different languages included in Potx module. It also includes the potx.pot template file.

Function format_plural()

The function **format_plural()** Allows you to format strings that reference an element counter, with the corresponding version in the singular and plural. For example: "1 comentario", "@count comentarios". **F55.4**

http://api.drupal.org/api/drupal/includes--common.inc/function/format_plural/7

```
format_plural($count, $singular, $plural, array $args = array(), array $options = array())
```

F55.4

Function
format_plural()

The parameters of the function are:

- **\$count**. Variable that contains the count value.
- **\$singular**. String for the singular value. Do not use @count within this chain. For example "1 comment".
- **\$plural**. String for the plural value. We'll use @count to indicate the count value. For example: "@count comments".
- **\$args**. Additional substitution variables. The operation is similar to that used in the function t(). It is not necessary to include the value of @count, That will be directly replaced by the function.
- **\$options**. Associative Array with additional options.

The function **format_plural()** is responsible for calling the function internally to **t()**, Therefore the chains passing through it will be available for your translation,in both the singular and in the plural.

Functions st() and get_t()

During the Drupal installation, depending on the phase we are in, the function **t()** may not be available yet. As an alternative the system facilitates the **st()**, with a similar operation. **F55.5**

<http://api.drupal.org/api/drupal/includes--install.inc/function/st/7>

In general it will only be necessary to use the function **st()** in the development of installation profiles (Unit 59)

F55.5

Function st()

```
function install_finished(&$install_state) {
  ...
  $output = '<p>' . st('Congratulations, you installed @drupal!', array('@drupal' => drupal_install_profile_distribution_name())) . '</p>';
  ...
}
```

If we want to translate a string at some point in the system where we cannot ensure that the function **t()** this available, we may use the function **get_t()**. This function returns the name of the translation function that this available, giving preference to the function **t()**. **F55.6**

http://api.drupal.org/api/drupal/includes--bootstrap.inc/function/get_t/7

F55.6

Function get_t()

```
$t = get_t();
$translated = $t('translate this');
```

Translation template extractor Module

55.2

Whether we have developed a module or if we are going to use a module that does not have a translation, the **Translation template extractor** module we will facilitate translation. This module reviews the code of the module or topic, generating the translation template on which we can work, manually translating each string.

The **Translation template extractor** module is available at:

<http://drupal.org/project/potx>

Once you have installed the **Translation template extractor** module, we can access the template translation from: **F55.7**

Administration ⇒ Configuration ⇒ Regional and language ⇒ Translate interface [pestana Export]

This screenshot shows the 'Translate interface' page in the Drupal admin. At the top, there's a breadcrumb trail: Home > Administration > Configuration > Regional and language > Translate interface. Below the breadcrumb, there are tabs: OVERVIEW, TRANSLATE, IMPORT, EXPORT, and EXTRACT. The EXTRACT tab is selected. A large text area says: 'This page allows you to generate translation templates for module and theme files. Select the module or theme you wish to generate a template file for. A single Gettext Portable Object (Template) file is generated, so you can easily save it and start translation.' Below this, there are three expandable sections: 'DIRECTORY "MODULES"', 'DIRECTORY "SITES/ALL/MODULES"', and 'DIRECTORY "THEMES"'. Under 'Template language', there are two radio buttons: 'Language independent template' (selected) and 'Template file for Spanish translations'. There's also a checkbox 'Include translations' which is unchecked. At the bottom right of the form is a button labeled 'Extract'.

F55.7

Translation template extractor module

Translation template extractor allows us to generate the translation template files for any module or theme installed on the site.

The **Figure F55.8** shows the selection of a particular module. The module **Translation template extractor** will analyze all the files in the folder of the module and will generate a file % 2C whose content will depend on the selected option in **Template language**: **F55.9**

- **Language independent template.** Generates a separate template file (file with the name of the module and extension **.pot**). It is a required file in the modules, including translation modules.
- **Template file for Spanish translations.** Generates a file ready for Spanish translation (**es.po**). You can optionally select **Include**

translations, with what will be included in the file with the translations already available on the site. This is especially useful if we have already made translation manuals through the administration area. If the site has enabled other languages, each one of the files can be exported for individual translation.

F55.8

Selecting a module

We select the module on which the translation template is created.

The screenshot shows the 'Translate interface' page with the following content:

- OVERVIEW** (selected)
- TRANSLATE**
- IMPORT**
- EXPORT**
- EXTRACT**

This page allows you to generate translation templates for module and theme files. Select the module or theme you wish to generate a template file for. A single Gettext Portable Object (Template) file is generated, so you can easily save it and start translation.

DIRECTORY "MODULES"

- Extract from *standard* in the *profiles/standard* directory
Generates output from all files found in this directory.

DIRECTORY "SITES/ALL/MODULES"

- Extract from *actions_forcontu* in the *sites/all/modules/actions_forcontu* directory
Generates output from all files found in this directory.
- Extract from *ajax_forcontu* in the *sites/all/modules/ajax_forcontu* directory
Generates output from all files found in this directory.
- Extract from *blocks_forcontu* in the *sites/all/modules/blocks_forcontu* directory
Generates output from all files found in this directory. (This option is highlighted with a red box.)
- Extract from *db_forcontu* in the *sites/all/modules/db_forcontu* directory
Generates output from all files found in this directory.

F55.9

Select the language

Select the language to generate its translation template file.

Template language

Language independent template

Template file for Spanish translations

Export a language independent or language dependent (plural forms, language team name, etc.) template.

Include translations

Include translations of strings in the file generated. Not applicable for language independent templates.

Extract

The following shows, for example, the file translation to Spanish generated for the module **Blocks Forcontu**, that we developed in **Unit 48**.

The file **blocks_forcontu.es.po** has been generated by checking the option **Template file for Spanish translations** and choosing the option **Include translations**. This last option includes translations found in the translation file chains that have previously been translated manually, through the translation interface. **F55.10**

```

# $Id$
#
# Spanish translation of Drupal (general)
# Copyright YEAR NAME <EMAIL@ADDRESS>
# Generated from files:
#   blocks_forcontu.module: n/a
#   blocks_forcontu.info: n/a
#
#, fuzzy
msgid ""
msgstr ""

"Project-Id-Version: PROJECT VERSION\n"
"POT-Creation-Date: 2012-02-28 12:53+0000\n"
"PO-Revision-Date: 2012-02-28 12:53+0000\n"
"Last-Translator: NAME <EMAIL@ADDRESS>\n"
"Language-Team: Spanish <EMAIL@ADDRESS>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=utf-8\n"
"Content-Transfer-Encoding: 8bit\n"
"Plural-Forms: nplurals=2; plural=(n!=1); \n"

#: blocks_forcontu.module:30;121
msgid "Unpublished nodes"
msgstr ""

#: blocks_forcontu.module:38;140
msgid "Users list"
msgstr ""

#: blocks_forcontu.module:54;62
msgid "Select the number of nodes to display in this block."
msgstr ""

#: blocks_forcontu.module:68
msgid "Display only active users."
msgstr ""

#: blocks_forcontu.module:98
msgid "View block: Unpublished nodes"
msgstr ""

#: blocks_forcontu.module:99
msgid "View the Unpublished nodes block."
msgstr ""

#: blocks_forcontu.info:0
msgid "Blocks Forcontu"
msgstr ""

#: blocks_forcontu.info:0
msgid "Examples of implementing blocks."
msgstr ""

#: blocks_forcontu.info:0
msgid "My modules"
msgstr "Mis módulos"

```

F55.10**File
blocks_forcontu.es.po**

Translation file generated by Translation template extractor module to translate Forcontu Blocks.

As we selected "include translations", some of the strings will include their translations if we translated them before.

For the translation of a module to operate correctly, we have to follow these steps:

- Create the folder **translations** inside the folder for the module. For example:

/sites/all/modules(blocks_forcontu)/translations

- Create a generic template file (**.pot** file), making use of the module **Translation template extractor**, **Language independent template** option. For example, for the module Blocks Forcontu, we will generate the file **blocks_forcontu.pot**. This file does not change.

- Create a Spanish language translation file (**es.po**). We also use the **Translation template extractor module** with the **Template file for Spanish translations** option. We can edit this file and add the outstanding strings to be translated.

Although the language files must be called **es.po** (without the name of the module), we can also use the general file name **Translation template extractor**, that includes the name of the module: **blocks_forcontu.es.po**.

In the same way we could add the translation files for other languages.

In summary, in the **translations** folder of the module the file **.pot** (general template translation) must also exist and **.po** files for each language.

When installing the module, the translations are imported directly.

55.3

Translating with Poedit

Poedit is a multi-platform tool that allows editing of translation files (**.po**), by facilitating the translation of strings. We can download the version that is appropriate for our operating system from:

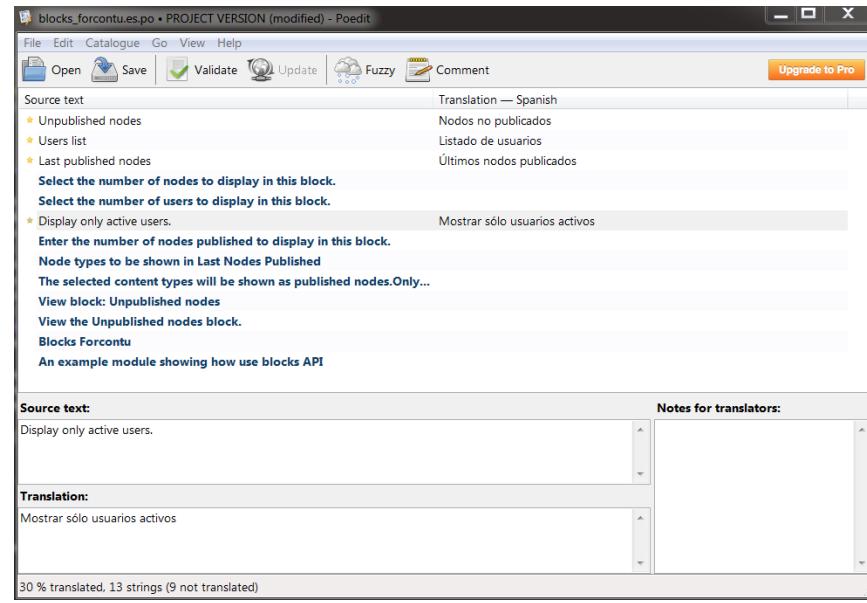
<http://www.poedit.net/>

Once installed, we will open the file **.po** generated by the **Translation template extractor**. The main window of the application will show two columns, one with the original text and another with the translation. By clicking on each string we can complete the corresponding translation. Once we have added the translations, the changes will be stored directly in the original **.po** file, which may be found in the module folder for translations. **F55.11**

F55.11

Poedit

Poedit is software to edit translation files (.po).



56 Creating themes

Themes allow us to change the design or graphic aspect of the site. A theme is composed of a collection of files with the **.tpl.php** extension, which are template files used in different areas of the site. In addition to template files, both modules and themes can implement a collection of PHP functions which allow for interventions in system processing and the modification of the final HTML.

It is possible to make simple theme changes by changing CSS stylesheets, template files, or rewriting theme functions. Each template or function accepts a predetermined set of variables. These variables will be substituted for their real values when the site is displayed.

In this Unit we will learn how to create a theme from scratch and how to modify an existing theme. We will also learn how to broaden the configuration options of the theme and how to create templates for developed modules.

In order to create or modify a theme, some knowledge of HTML, CSS and PHP will be necessary.

Comparative D7/D6

Themes

The most important changes between Drupal 7 and Drupal 6 are:

- The use of render arrays. As we will see, this system allows us to maintain elements in a structured format until the moment of their presentation.
- Preprocessing functions. In Drupal 7 there are additional processing functions, which are executed afterwards. Another important change is that the preprocessing functions also affect theme functions, not just template files.

Although Drupal 7 has included these important changes, the functioning is in general similar to that of Drupal 6.

Unit contents

56.1 Introduction to the theme system	348
56.2 Create a theme.....	350
56.3 Template files	357
56.4 Element rendering.....	363
56.5 Theme and template functions	367
56.6 Preprocessing functions.....	372
56.7 Theme configuration options	374
56.8 Creating templates for modules	376

56

56.1

Introduction to the theme system

In the intermediate level we begin to analyze themes, giving special emphasis to the files which form them. Among these files are template files, which use the **.tpl.php** extension.

By default Drupal Works with the **PHP Template** templating engine although other engines can be incorporated. We will always use the default template engine, which is also the only one available as part of the Drupal distribution, and the engine which almost all modules use.

Besides template files, both modules and themes can implement a set of PHP functions which allow for system interventions and the modification of the final HTML. When an element should be displayed on the screen, the theme system comes into play, processing the element and returning the final HTML, which is what will be displayed in the browser.

Each element is treated independently, with its own template being applied to it. For example, a node, a block, or a forum message are treated separately at the point when the final HTML is returned. As we will see, the **theme()** function is responsible for applying the templates to each one of these elements and returning the corresponding HTML code. Finally, the system completes the final result by including each individual HTML section, giving the final page which is being requested at any given moment.

Render arrays

In Drupal 7 the render array appears as a new tool. Render arrays are associative arrays which, with a designated structure, are used by the theme system to obtain the final HTML of each element.

Unlike previous versions, in Drupal 7 render arrays are used until the moment of presenting the page, when it is converted to the final HTML content. Because of this, we can interact with any member of the system, modifying its content and/or presentation, by altering the array values.

In the following example, **F56.1** the difference between Drupal 6 and Drupal 7 is shown:

F56.1

Render arrays

The use of render arrays in Drupal 7. As we will see in this unit, we can also call the **theme()** function, as in Drupal 6, in order to get the final HTML for each element.

```
//Drupal 6
function example_callback() {
  $items = example_get_items();

  $output = theme('item_list', $items);
  return $output;
}

//Drupal 7
function example_callback() {
  $items = example_get_items();

  $build['items'] = array (
    '#theme' => 'item_list',
    '#items' => $items,
  );
  return $build;
}
```

In both cases the return is a list of elements to which the 'item_list' template will be applied, so that the element presentation in the page will be exactly the same.

The difference lies in the fact that in Drupal 6 what was returned at this point was the HTML corresponding to the list of elements, which meant that we could no longer alter the structure of the list without having to directly modify the HTML code.

In Drupal 7, on the other hand, we compose a render array, leaving the theme() function step for later. As we've already discussed, the system is responsible for generating the final HTML for each element prior to displaying the page. This allows us to make changes to the final display of any element by modifying the render array.

We will go into further detail about render arrays, templates, and theme functions in the upcoming topics.

Devel Module

As we saw in **Unit 42**, the **Devel** module adds module and theme cleanup functionalities for developers.

The **Devel** module is available at:

<http://drupal.org/project/devel>

One of the options included in the Develop menu is the **Theme registry**, where the presentation information about each element is stored. The system will consult the theme registry in order to determine, for example, what template function to use to display a list of elements. **F56.2**

Element	Type	Description
...	(Array, 156 elements)	...
admin_block	(Array, 6 elements)	admin_block (Array, 6 elements)
admin_block_content	(Array, 5 elements)	admin_block_content (Array, 5 elements)
admin_page	(Array, 6 elements)	admin_page (Array, 6 elements)
authorize_message	(Array, 4 elements)	authorize_message (Array, 4 elements)
authorize_report	(Array, 4 elements)	authorize_report (Array, 4 elements)
block	(Array, 6 elements)	block (Array, 6 elements)
block_admin_display_form	(Array, 8 elements)	block_admin_display_form (Array, 8 elements)
breadcrumb	(Array, 4 elements)	breadcrumb (Array, 4 elements)
variables	(Array, 1 element)	variables (Array, 1 element)
breadcrumb	(NULL)	breadcrumb (NULL)
type	(String, 6 characters) module	type (String, 6 characters) module
theme path	(String, 14 characters) modules/system	theme path (String, 14 characters) modules/system
function	(String, 16 characters) theme_breadcrumb (Callback) theme_breadcrumb();	function (String, 16 characters) theme_breadcrumb (Callback) theme_breadcrumb();
button	(Array, 4 elements)	button (Array, 4 elements)
checkbox	(Array, 4 elements)	checkbox (Array, 4 elements)

F56.2

Theme registry

The Devel module permits the display of a complete registry of the theme system, with information about each installed and activated module.

When a theme or module is installed, the theme registry is rebuilt. When we modify files or template functions in modules or themes, we will not see the changes we made until we rebuild the theme registry. In order to avoid having to do this manually each time we make a modification, the Devel module offers the option: "Rebuild the theme registry in each page load." We will activate this option before continuing with theme development.

56.2 Creating a theme

Theme construction may vary according to the resources that are initially available. For example, it is common practice to start with an already existing Drupal theme or a design coded in HTML and CSS, which assists in theme development. There are also base themes, created in order to make first steps easier, which include the generic code common to all themes.

In order to understand better how themes are constructed, in this section we will see the steps required to create a Drupal theme from scratch. We will create a theme named "Example theme".

Theme Structure

In **Unit 29** of the **Intermediate Level** we introduced the structure of the files that make up a theme. Of the files it contains, only the **.info** file is required. In it the theme definition appears as well as some general configuration options. The remainder of the files may or may not be present in the installed theme, without affecting site function. **F56.3**

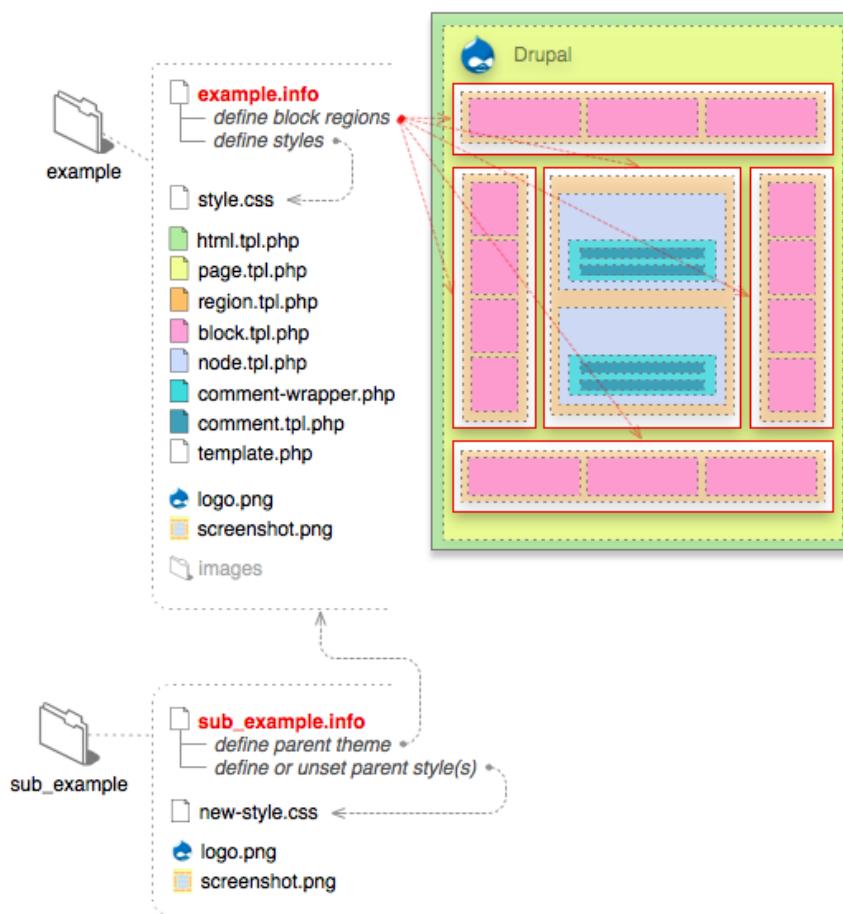
F56.3

Typical structure of a theme directory

The theme consists of template files, **.tpl.php**, which correspond to different page elements.

The **style.css** file is generally the principal stylesheet, although the theme can define other stylesheets as well.

Image obtained from:
<http://drupal.org/node/171194>



Let's review what the various theme files are responsible for:

- **Definition file (`.info`).** The `.info` file stores the theme definition and is a required file. This file includes information about the available regions, CSS and JavaScript files, etc. The name of the `.info` file defines the internal name of the theme and is usually contained in a directory of the same name.

In our example theme we will create the directory **example_theme** and within it the file **example_theme.info**. The name of the theme will be **example_theme**. It is not required that the name of the directory correspond with the internal name of the theme, but it is a recommended practice.

- **Template files (`.tpl.php`).** The **tpl** (template) files define the theme structure. They contain HTML code and PHP variables which will be substituted for their corresponding values at the moment of displaying page content. Each template file will generate the HTML for a single site element. For example, the **comment.tpl.php** file will include the final code for site comments.

Template files may reside directly in the theme directory or in a subdirectory, usually named **templates**, for more granular organization.

- **template.php** file. As we will find in this unit, the **template.php** file allows us to add additional programming logic in order to obtain the final value of the variables given in templates (this process is known as variable preprocessing). This preprocessing allows us to keep template files free of additional processing and simply handle the delivery of the final variable values.
- **Style files (`style.css` and other `.css` files).** The principal file containing styles is called **style.css**. This is not a required file, but is simply a common practice. Other `.css` files may also exist and may be contained in a `css` directory for optimal organization.
- **Images directory.** Contains theme images. We can modify existing images or add new ones, in which case it will also be necessary to reference them correctly from the CSS stylesheet.

Main theme directory and .info file

To begin, we will create the **example_theme** directory within the directory assigned for site themes (generally /sites/all/themes/).

The theme should have an associated file with a **.info** extension, which describes the theme. This file should be encoded in **UTF-8 without BOM**. For our theme, we must create the **example_theme.info** file with the following initial information: **F56.4**

F56.4

Archivo .info

Theme configuration file, with .info extension. The file contains the theme name, description, regions, stylesheets, etc

```
name = Example Theme
description = Example Theme for Drupal 7
screenshot = images/screenshot.png
core = "7.x"
project = "example theme"
engine = phptemplate

regions[page_top] = Page top
regions[header] = Header
regions[highlight] = Highlight
regions[help] = Help
regions[content] = Content
regions[sidebar_first] = First sidebar
regions[sidebar_second] = Second sidebar
regions[footer] = Footer
regions[page_bottom] = Page bottom

features[] = logo
features[] = name
features[] = slogan
features[] = node_user_picture
features[] = comment_user_picture
features[] = search
features[] = favicon
features[] = main_menu
features[] = secondary_menu

stylesheets[all][] = css/layout.css
stylesheets[all][] = css/style.css
stylesheets[print][] = css/print.css
```

The parameters which we should initially include in the **.info** file are:

- **name.** Name of the theme. This field is required.
- **description.** Theme description. The description should be written in English and later translated into the language of the site, if applicable, using the translation interface.
- **screenshot.** Path to the image which contains a preview of the theme. This image will be shown in the theme administration area. It is an optional parameter.
- **core.** Drupal version for which the theme has been created. A required parameter.
- **project.** Project name, which normally corresponds to the main directory of the theme. This parameter is necessary only if we are sharing a developed theme with the Drupal community.
- **engine.** Template engine used. By default Drupal uses **phptemplate**.
- **version.** We can indicate the version of the theme. This parameter is

only necessary if we are sharing our theme with the community or if we maintain a register of theme versions in our project.

- **regions[]**. Array containing available regions for our theme. Although the typical regions used have been defined, new regions can be created. We will see further on how to include these new regions in the theme.
- **features[]**. Array of the functions available in the theme administration area. If we comment out or delete any of these functions, they will simply not appear in the administration area, and the site administrator will not be able to indicate whether they would like to activate them.
- **stylesheets[]**. Allows us to indicate the stylesheets used by the theme.
 - o **stylesheets[all]**. General stylesheets
 - o **stylesheets[print]**. Stylesheets applied to print view.

The **.info** file can include other parameters, which we will learn about as they become necessary.

Template Files

As we've already discussed, the **.tpl.php** files are template files. They contain HTML code and PHP variables, which will be substituted for their corresponding values when the final page is displayed. Each template file is responsible for generating the HTML for a single site element, passing this content to another template file at a higher level.

The system already has all the template files necessary to execute a theme, and this is why a theme only needs the **.info** file in order to function. When we need to make modification to a template we will not do so to the original file, but will instead copy the file to the theme directory.

The template files may be found directly in the theme directory or in a subdirectory, usually called **templates**, for better file organization.

In order to continue developing our **Example Theme**, we will create the **example_theme/templates** directory and copy into it the following files:

- html.tpl.php, available at **/modules/system/html.tpl.php**.
- page.tpl.php, available at **/modules/system/page.tpl.php**.
- node.tpl.php, available at **/modules/node/node.tpl.php**.
- block.tpl.php, available at **/modules/block/block.tpl.php**.
- comment.tpl.php, available at **/modules/comment/comment.tpl.php**.

When the system needs, for example, to load the **html.tpl.php** template, it will see first if it is available in the theme directory. If so, it will use that copy of the template, and if not, it will use the default version. This process is known as **overriding**, and applies to any installed module, whether it belongs to core, contributed, or our own developed modules.

Style sheets

The templates (**.tpl.php** files) are complemented by style sheets (**.css** files). By default, themes generally are constructed with a principle style file, called **style.css**, generally located in the root directory of the theme. It is possible, nevertheless, to define personalized style sheets in the **.info** file of the theme. **F56.5**

F56.5

Style sheets

CSS files registered in the theme configuration file.

```
stylesheets[all][] = css/layout.css
stylesheets[all][] = css/style.css
stylesheets[print][] = css/print.css
```

In our example we indicate that the files will be stored in a folder called **/css**. We will create three files that define styles:

- **css/layout.css**, which will contain the styles that relate to page layout: fluid or fixed width, column width, etc.
- **css/style.css**, in order to include the rest of the styles: fonts, titles, links, links, etc.
- **css/print.css**, in order to apply basic styles to the print view.

Below is an example of the contents of **layout.css**.

F56.6**F56.6**

layout.css file

Example of layout.css, where we will include the styles that handle page structure.

```
#page {
    width: 960px;
    margin: 0 auto;
}

#content {
    float: left;
    width: 100%;
    margin-right: -100px;
    padding: 0;
}

.sidebar {
    float: left;
}

.sidebar-second {
    float: right;
}

.footer {
    float: none;
    clear: both;
}

.header, #footer, .mission, .breadcrumb, .node {
    clear: both;
}

.two-sidebars .center, .sidebar-first .center {
    margin-left: 190px;
}

.sidebar-first {
    width: 190px;
    margin-right: -190px;
}

.two-sidebars .center, .sidebar-second .center {
    margin-right: 200px;
}

.sidebar-second {
    width: 200px;
}
```

And an example of the elements which we can include in **style.css**. F56.7

```
/* Font styles. */
body {
  margin: 0;
  font: 13px/1.5em 'Helvetica Neue', helvetica, Arial, sans-serif;
  letter-spacing: 0.03em;
}

/* Links */
a:link, a:visited {
  color: blue;
  text-decoration: none;
}

a:hover, a:active {
  color: red;
  text-decoration: underline;
}

/* Titles. */
#site-name {
  font-size: 2.2em;
  line-height: 1.3em;
  font-weight: 300;
  padding: 0 0 0.5em;
  margin: 0;
}

h1, h2, h3, h4, h5, h6 {
  line-height: 1.3em;
}

h1 {
  font-size: 2.2em;
  font-weight: 300;
  padding: 0 0 0.5em;
  margin: 0;
}

h2 {
  font-size: 1.8em;
  font-weight: 300;
  margin-bottom: 0.75em;
}

h3 {
  font-size: 1.4em;
  margin-bottom: 1em;
}

h4, h5, h6 {
  font-size: 1.2em;
  margin-bottom: 0.8em;
}

/* Paragraph styles. */
p {
  margin: 0 0 1em 0;
}

/* List styles. */
ul, ol {
  margin-left: 0;
  padding-left: 2em;
}
```

F56.7

style.css file

Example of style.css, where we will include the styles that handle fonts, titles, paragraphs, links, etc.

These files are shown only as examples.

Theme Activation

Before activating a theme that is under development, it is a good idea to ensure that the administration theme is a core theme of Drupal (**Seven** or **Garland**, for example). If we don't, and the new theme has some programming error and "breaks" the page, the administrator of the site will not be able to access the theme administration area to change themes. If this should happen, we would need to recover the site by changing the theme assignment directly from the database.

Once the theme (.info, .tpl.php files and css files) has been created and uploaded to, we can activate it in the administration area: **F56.8**

URL Appearance
[/admin/appearance](#)

F56.8

Activation of a created theme:

Once created, the them will be available in the theme administration área. The theme preview image shown here belongs to the Basic theme.

Administration ⇒ **Appearance [Enable Example Theme]**

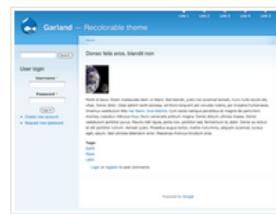
DISABLED THEMES



Example Theme

Example Theme for Drupal 7

Enable | **Enable and set default**



Garland 7.32

A multi-column theme which can be configured to modify colors and switch between fixed and fluid width layouts.

Enable | **Enable and set default**

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Figure F56.9 shows the appearance of the site once the created theme is activated. From this moment on, we can make changes in the theme and check their appearance directly in our website. It goes without saying that theme development should never be carried out directly on a production site.

F56.9

Example Theme

Visual of a site with the created theme. This is a very basic theme which is only an example to guide us.

The screenshot shows a Drupal 7 website using the 'Example Theme'. The header features a blue drop logo and the text 'Expert in Drupal 7 - Advanced'. The main content area displays a user login form on the left and a large, stylized circular graphic in the center. The right side of the page includes a sidebar with the heading 'Who's online' and a message stating 'There are currently 0 users online.' The footer contains copyright information for Forcontu S.L. and a verification code.

Template files

56.3

In this section we will go over the principal templates that make up a theme.

The `html.tpl.php` template

This is the highest level template file and is in charge of the HTML page structure. It will be responsible for displaying tags including `<html>`, `<head>`, and `<body>`. Some of the variables which can be used in this template are:

- **`$head_title`**. Page title to be wrapped in the `<title>` tag.
- **`$head`**. Complete head, including the meta, keyword, and other head tags .
- **`$styles`**. Code for CSS stylesheet inclusion.
- **`$scripts`**. Code for javascript inclusion.
- **`$page`**. Page content in HTML, already processed and ready to be displayed in the browser.

NOTE

html.tpl.php

The `html.tpl.php` template is new to Drupal 7. In Drupal 6 this code was included in the template we will discuss next, `page.tpl.php`.

If the theme does not include this template, Drupal uses the default template, located at **modules/system/html.tpl.php**. In this file, all the available variables are described.

Let's copy this template to **example_theme/templates**. F56.10

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+RDFa 1.0//EN"
  "http://www.w3.org/MarkUp/DTD/xhtml-rdfa-1.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang=<?php print
$language->language; ?>" version="XHTML+RDFa 1.0" dir=<?php print
$language->dir; ?>"<?php print $rdf_namespaces; ?>>
<head profile=<?php print $grddl_profile; ?>>
  <?php print $head; ?>
  <title><?php print $head_title; ?></title>
  <?php print $styles; ?>
  <?php print $scripts; ?>
</head>
<body class=<?php print $classes; ?> <?php print $attributes;?>>
  <div id="skip-link">
    <a href="#main-content" class="element-invisible element-
focusable"><?php print t('Skip to main content'); ?></a>
  </div>
  <?php print $page_top; ?>
  <?php print $page; ?>
  <?php print $page_bottom; ?>
</body>
</html>
```

F56.10

html.tpl.php

The `html.tpl.php` file (included in the System module of Drupal core).

The `page.tpl.php` template

The `page.tpl.php` template generates all the code contained in the page, and once processed it is sent to the `html.tpl.php` template via the variable `$page`.

If the activated theme does not include this file, Drupal uses the default template, located at **modules/system/page.tpl.php**. We will also copy this template file to the folder **example_theme/templates**.

Some of the variables that can be used are:

F56.11

- **\$base_path.** Drupal installation path. If the installation has been done in the domain web root, it returns "/".
- **\$directory.** Relative path of the folder where the site theme resides.
- **\$is_front.** The value is TRUE if the actual page is the site's front page.
- **\$logged_in.** The value is TRUE if the page is being loaded by a registered user.
- **\$is_admin.** The value is TRUE if the user has administration permissions.
- **\$breadcrumb.** HTML code which displays site navigation breadcrumbs.
- **\$front_page.** Link to the front page of the site.
- **\$logo.** Link to the logo image.
- **\$site_name** and **\$site_slogan.** Site name and site slogan..
- **\$title.** Page title.
- **\$messages.** Error messages or notifications.
- **\$tabs.** Links to the tabs that will be displayed in the page (for example, links to view and edit a node when one is being displayed).
- **\$node.** The *\$node* object, if the page being loaded is a node.
- **\$page.** Array with the content of each of the regions.

F56.11**page.tpl.php**

Field page.tpl.php
 (Included in the System module to the core).

```
<div id="page-wrapper"><div id="page">
  <div id="header"><div class="section clearfix">
    <?php if ($logo): ?>
      <a href="<?php print $front_page; ?>" title="<?php print t('Home'); ?>" rel="home" id="logo">
        " />
      </a>
    <?php endif; ?>

    <?php if ($site_name || $site_slogan): ?>
      <div id="name-and-slogan">
        <?php if ($site_name): ?>
          <?php if ($title): ?>
            <div id="site-name"><strong>
              <a href="<?php print $front_page; ?>" title="<?php print t('Home'); ?>" rel="home"><span><?php print $site_name; ?></span></a>
            </strong></div>
          <?php else: /* Use h1 when the content title is empty */ ?>
            <h1 id="site-name">
              <a href="<?php print $front_page; ?>" title="<?php print t('Home'); ?>" rel="home"><span><?php print $site_name; ?></span></a>
            </h1>
          <?php endif; ?>
        <?php endif; ?>
      <?php endif; ?>

      <?php if ($site_slogan): ?>
        <div id="site-slogan"><?php print $site_slogan; ?></div>
      <?php endif; ?>
    </div> <!-- /#name-and-slogan -->
  <?php endif; ?>
  <?php print render($page['header']); ?>

</div></div> <!-- /.section, #header -->

<?php if ($main_menu || $secondary_menu): ?>
  <div id="navigation"><div class="section">
    <?php print theme('links__system_main_menu', array('links' => $main_menu, 'attributes' => array('id' => 'main-menu', 'class' => array('links', 'inline', 'clearfix'))), 'heading' => t('Main menu'))); ?>
    <?php print theme('links__system_secondary_menu', array('links' =>
```

```

$secondary_menu, 'attributes' => array('id' => 'secondary-menu', 'class' =>
array('links', 'inline', 'clearfix')), 'heading' => t('Secondary menu'))));
?>
    </div></div> <!-- /.section, #navigation -->
<?php endif; ?>

<?php if ($breadcrumb): ?>
    <div id="breadcrumb"><?php print $breadcrumb; ?></div>
<?php endif; ?>

<?php print $messages; ?>

<div id="main-wrapper"><div id="main" class="clearfix">

    <div id="content" class="column"><div class="section">
        <?php if ($page['highlighted']): ?><div id="highlighted">
            <?php print render($page['highlighted']); ?></div><?php endif; ?>
        <a id="main-content"></a>
        <?php print render($title_prefix); ?>
        <?php if ($title): ?><h1 class="title" id="page-title">
            <?php print $title; ?></h1><?php endif; ?>
        <?php print render($title_suffix); ?>
        <?php if ($tabs): ?><div class="tabs">
            <?php print render($tabs); ?></div><?php endif; ?>
        <?php print render($page['help']); ?>
        <?php if ($action_links): ?><ul class="action-links">
            <?php print render($action_links); ?></ul><?php endif; ?>
        <?php print render($page['content']); ?>
        <?php print $feed_icons; ?>
    </div></div> <!-- /.section, #content -->

    <?php if ($page['sidebar_first']): ?>
        <div id="sidebar-first" class="column sidebar"><div class="section">
            <?php print render($page['sidebar_first']); ?>
        </div></div> <!-- /.section, #sidebar-first -->
    <?php endif; ?>

    <?php if ($page['sidebar_second']): ?>
        <div id="sidebar-second" class="column sidebar"><div class="section">
            <?php print render($page['sidebar_second']); ?>
        </div></div> <!-- /.section, #sidebar-second -->
    <?php endif; ?>
</div></div> <!-- /main, #main-wrapper -->

<div id="footer"><div class="section">
    <?php print render($page['footer']); ?>
</div></div> <!-- /.section, #footer -->
</div></div> <!-- /page, #page-wrapper -->

```

Regions

Regions are areas of the theme where blocks can be located. As we have already seen, regions are defined in the .info file of the theme. In our example theme we have defined the following regions: **F56.12**

```

regions[page_top] = Page top
regions[header] = Header
regions[highlight] = Highlight
regions[help] = Help
regions[content] = Content
regions[sidebar_first] = First sidebar
regions[sidebar_second] = Second sidebar
regions[footer] = Footer
regions[page_bottom] = Page bottom

```

F56.12

Regions

Region declaration in the theme configuration file (.info).

The content of each region is dynamic and depends on the regions added to each one from the block administration page.

The regions are defined in the **page.tpl.php** template, using the variable **\$page**:

- `$page['help']`.
- `$page['highlighted']`.
- `$page['content']`.
- `$page['sidebar_first']`.
- `$page['sidebar_second']`.
- `$page['header']`.
- `$page['footer']`.

The system will handle the creation of this content by finding the active blocks, their order, and so on, and returning the corresponding HTML for each region. As we will see, this HTML code depends on other, secondary, templates which we will also define for our theme.

The node.tpl.php template

The **node.tpl.php** template handles node presentation. The result of the template is handed off to the page template (page.tpl.php) within the variable **\$page['content']**. If the theme does not include this file, Drupal uses the default template, located at **modules/node/node.tpl.php**. We will copy this file to **example_theme/templates**.

Some of the variables that can be used in this template are: **F56.13**

- **\$title.** The node title.
- **\$content.** Node content (or a node part or teaser in the case of a node listing). This is an array which contains all node elements. The **render** function can be used to print these elements (using `render($content)` all elements are printed, using `render($content['example_field'])` we only print `example_field`).
- **\$user_picture.** The image of the node author.
- **\$date.** Node creation date.
- **\$name.** Username of the node author.
- **\$node_url.** The URL of the node.
- **\$type.** Node type (for example *page, story, blog*).
- **\$comment_count.** The number of comments on this node.
- **\$page.** Contains TRUE if the complete node is being displayed.
- **\$teaser.** Contains TRUE if the teaser view of the node is being displayed.
- **\$promote.** Contains TRUE if the node is promoted to the front page (promoted to /node).
- **\$sticky.** Contains TRUE if the node is set to stay at the top of lists.
- **\$status.** Contains TRUE if the node is published.

```

<div id="node-<?php print $node->nid; ?>" class="<?php print
$classes; ?> clearfix"<?php print $attributes; ?>

<?php print $user_picture; ?>

<?php print render($title_prefix); ?>
<?php if (!empty($page)): ?>
  <h2><?php print $title_attributes; ?>><a href="<?php print
$node_url; ?>"><?php print $title; ?></a></h2>
<?php endif; ?>
<?php print render($title_suffix); ?>

<?php if ($display_submitted): ?>
  <div class="submitted">
    <?php print $submitted; ?>
  </div>
<?php endif; ?>

<div class="content"><?php print $content_attributes; ?>>
<?php
  // We hide the comments and links now so that we can render them later.
  hide($content['comments']);
  hide($content['links']);
  print render($content);
?>
</div>

<?php print render($content['links']); ?>
<?php print render($content['comments']); ?>

</div>

```

The **block.tpl.php** template

The **block.tpl.php** template is used to generate the presentation of blocks. If an activated theme does not include this file, Drupal uses the default template, located at **modules/block/block.tpl.php**.

Some of the variables available in this template are: F56.14

- **\$block->subject.** Block title.
- **\$content.** Block content.
- **\$block->module.** The module that generated the block.
- **\$block->region.** The region in which the block is located.

```

<div id='<?php print $block_html_id; ?>' class='<?php print
$classes; ?>'><?php print $attributes; ?>

<?php print render($title_prefix); ?>
<?php if ($block->subject): ?>
  <h2><?php print $title_attributes; ?>><?php print $block->subject
?></h2>
<?php endif; ?>
<?php print render($title_suffix); ?>

<div class='content'><?php print $content_attributes; ?>>
  <?php print $content ?>
</div>
</div>

```

F56.13

node.tpl.php

Field **node.tpl.php**
(Included in Node Module in the core).

F56.14

block.tpl.php

Fragment of **block.tpl.php** (included in the Block module of Drupal core).

The comment.tpl.php template

The **comment.tpl.php** template displays site comments. The default file is found at **modules/comment/comment.tpl.php**. Some of the available variables are: **F56.15**

- **\$content**. Array containing content files.
- **\$created**. Comment creation date.
- **\$changed**. Date of the most recent update to the comment.
- **\$new**. Indicates whether the comment is new.
- **\$submitted**. Information about the comment submission (author and creation date).
- **\$picture**. Image of the comment author.
- **\$title**. Title and comment link.

F56.15

comment.tpl.php

Fragment of
comment.tpl.php
(included in the Comment module of Drupal core).

```
<div class="<?php print $classes; ?> clearfix"<?php print
$attributes; ?>>
<?php print $picture ?>

<?php if ($new): ?>
  <span class="new"><?php print $new ?></span>
<?php endif; ?>

<?php print render($title_prefix); ?>
<h3<?php print $title_attributes; ?>><?php print $title ?></h3>
<?php print render($title_suffix); ?>

<div class="submitted">
  <?php print $permalink; ?>
  <?php print $submitted; ?>
</div>

<div class="content"><?php print $content_attributes; ?>>
  <?php
    hide($content['links']);
    print render($content);
  ?>
  <?php if ($signature): ?>
  <div class="user-signature clearfix">
    <?php print $signature ?>
  </div>
  <?php endif; ?>
</div>

<?php print render($content['links']) ?>
</div>
```

Other template files

Other examples of template files include:

- **region.tpl.php**. Generates the presentation for any theme region. The default file is found at **modules/system/region.tpl.php**.
- **user-picture.tpl.php**. Template responsible for the presentation of user pictures. Drupal's default file is located at **modules/user/user-picture.tpl.php**.

Finally, modules can also have their own templates. For example, the **Forum** module makes use of various templates which configure the appearance of forum messages.

Each module can include various template files, which it will use from within its own code to display the new tools it adds to the site. If we want to alter any of these templates we should never do so in the module file. First we copy the file into the theme folder, and this file will be the one we modify, without changing its name.

Element rendering

56.4

render() function

In template code, there are repeated calls to the **render()** function:

- render(\$content)
- render(\$content['links'])
- render(\$page['help'])
- render(\$page['highlighted'])
- render(\$page['content'])
- render(\$page['sidebar_first'])
- render(\$page['sidebar_second'])
- render(\$page['header'])
- render(\$page['footer'])

We mentioned at the beginning of this unit that in Drupal 7 we will use render arrays, which are associative arrays which describe how each element should be displayed.

The **render()** function, then, is what is responsible for transforming an array of this type into the corresponding HTML code.

<http://api.drupal.org/api/drupal/includes--common.inc/function/render/7>

render(&\$element)

The function receives as a parameter the definition of an element in the form of a render array (**\$element**). Internally, it makes a call to the **drupal_render()** function, which is what actually performs the transformation from array to HTML output:

http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_render/7

Render Arrays

Render arrays are made up of two types of elements: properties and children. Properties begin with '#' and supply information about how the element should be converted. Children are themselves render arrays which describe child elements that should be rendered along with the parent element. Generally the HTML code of child elements will be contained within the parent element.

In order to construct render arrays we will use the elements and properties already studied in **Form API (Unit 47)**. Let's take a look at some of the specific properties available for element presentation:

- **#type.** Element type. Generally the element type will have associated default values, according to how they have been defined `hook_element_info()`. In addition to the elements provided by Form API, we can also create new elements.
- **#markup.** Allows for the display of HTML text.
- **#prefix/#suffix.** Allows the addition of text or HTML code before and after an element.
- **#pre_render.** Array of functions which will be executed before element rendering. Using this, it is possible to modify the presentation of an element before it constructed.
- **#post_render.** Array of functions which will be executed after element rendering. These functions will act, therefore, on the generated HTML.
- **#theme.** Theme function which will generate the HTML for the element. If the value assigned to **#theme** is 'function_name', **theme_function_name(\$variables)** will be called. The function will receive the rest of the defined parameters in the **\$variables** array. The HTML generated for the element will be defined within the called function.
- **#theme_wrappers.** When an element has child elements, first the children will be rendered (using the individual configuration of each one) and the resulting HTML is stored in the `#children` property. In `#theme_wrappers` we can indicate a theme function which will act on the content stored in `#children`, being able to add additional HTML that serves as a wrapper around the child elements..

Modifying render arrays

As we've already stated, the use of render arrays allows us to modify an element in a structured way before its final presentation is generated.

The place where we should modify an element will depend on its type. Let us consider a few examples :

- **hook_form_alter()**. Allows modification of any form before it is rendered.
- **hook_block_view_alter()**. Allows the modification of block content, which can be defined as a render array in the parameter `$data['content']`.
- **hook_page_alter()**. Allows modifications of the page before it is rendered.

hook_page_alter() The last opportunity we have to modify any page content before it is completely rendered.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_page_alter/7

hook_page_alter(&\$page)

The function is passed the \$page parameter by reference. This variable contains all elements in the form of a render array. It contains a first level with the regions of the page, and within each region, the elements that will be displayed within the region are described. **F56.16**

```
$page['page_top']
$page['header']
$page['highlight']
$page['help']
$page['content']
$page['sidebar_first']
$page['sidebar_second']
$page['footer']
$page['page_bottom']
```

F56.16

\$page parameter

The variable \$page contains an element for each region of the theme. Only the regions that have at least one block assigned will be available.

For example, if the page is a node display, we will be able to access the different node elements, such as the body, links, or comments: **F56.17**

```
$page['content']['system_main']['nodes'][$nid]['body']
$page['content']['system_main']['nodes'][$nid]['links']
$page['content']['system_main']['nodes'][$nid]['comments']
```

F56.17

Node Elements

Access to node elements from within \$page.

Using **hook_page_alter()** we are able to modify these elements. As an example, let's eliminate the "powered by Drupal" message from the page footer and change the region of the search form from the first sidebar to the page footer. **F56.18**

```
/**
 * Implements hook_page_alter().
 */
function theming_forcontu_page_alter(&$page) {
  global $theme_key;

  // The changes will be carried out only in the theme "Example Theme"
  if ($theme_key == 'example theme') {
    // Remove the "powered by Drupal" block
    unset($page['footer']['system_powered-by']);

    // Move the search block to the foot of the page
    $page['footer']['search_form'] = $page['sidebar_first']['search_form'];
    unset($page['sidebar_first']['search_form']);
  }
}
```

F56.18

hook_page_alter()

Example of hook_page_alter() implementation, from which we can modify the \$page variable before it is rendered, or transformed into HTML code.

The changes will only be applied to **Example Theme**. To implement this restriction, we consult the global variable **\$theme_key**, which returns the system name of the active theme.

Creating new renderizable elements

By implementing **hook_element_info()** we are able to declare new element types, which can be used in forms by using the **Form API**.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_element_info/7

For each defined element we must create a theme function with the naming convention of **theme_elementtype()** and register it in hook_theme().

As an example, let us consider some form element types that are already established and defined by the system within **system_element_info()**: [F56.19](#)

[F56.19](#)

hook_element_info()

Allows new form element types to be defined, with their corresponding theme functions.

```
function system_element_info() {
  $types['textfield'] = array(
    '#input' => TRUE,
    '#size' => 60,
    '#maxlength' => 128,
    '#autocomplete_path' => FALSE,
    '#process' => array('ajax_process_form'),
    '#theme' => 'textfield',
    '#theme_wrappers' => array('form_element'),
  );
  $types['textarea'] = array(
    '#input' => TRUE,
    '#cols' => 60,
    '#rows' => 5,
    '#resizable' => TRUE,
    '#process' => array('ajax_process_form'),
    '#theme' => 'textarea',
    '#theme_wrappers' => array('form_element'),
  );
  $types['select'] = array(
    '#input' => TRUE,
    '#multiple' => FALSE,
    '#process' => array('form_process_select', 'ajax_process_form'),
    '#theme' => 'select',
    '#theme_wrappers' => array('form_element'),
  );
  ...
  return $types;
}
```

For each element a corresponding theme function is defined, by way of the '#theme' property. Thus, for example, for the element type '**textfield**', the theme function **theme_textfield()**, defined in **includes/form.inc**, will be used.

We will learn more about the implementation and declaration of theme functions in the next section.

Theme and template functions

56.5

Each type of renderable element is accompanied by a theme function (**theme_elementtype()**) which is responsible for returning the resulting HTML for the element. The theme for an element can also be defined in a template file, which we will see in this section.

theme() function

The **theme()** function returns the HTML code of an element base don its theme or template function.

<http://api.drupal.org/api/drupal/includes--theme.inc/function/theme/7>

theme(\$hook, \$variables = array())

theme() will call, as a function of the value of **\$hook**, the function or template in charge of generating the HTML of that element, passing it the variable contained in **\$variables**.

For example, a call to **theme()** with **\$hook** = 'more_help_link' will result in a call to the function **theme_more_help_link()**, which will return the resulting HTML for a link to "more help". **F56.20**

```
$output = theme('more_help_link', array('url' => 'admin/help/mymodule'));
```

F56.20

theme() function

http://api.drupal.org/api/drupal/includes--theme.inc/function/theme_more_help_link/7

When we examine the definition of this function (available at /includes/theme.inc), we see that within the **\$variables** parameters there is a required 'url' field: **F56.21**

```
/** 
 * Returns HTML for a "more help" link.
 *
 * @param $variables
 *   An associative array containing:
 *   - url: The url for the link.
 */
function theme_more_help_link($variables) {
  return '<div class="more-help-link">' . l(t('More help'),
$variables['url']) . '</div>';
}
```

F56.21

Theme function

Example of a theme function for the 'more_help_link' element.

The resulting HTML from the previous call will be: **F56.22**

```
<div class="more-help-link"><a href="/admin/help/mymodule">More
Help</a></div>
```

Each theme function will require its own parameters, which will always be passed via the array **\$variables**.

As we've stated, the element may have a theme function associated with it, as in the previous case, or a **template**. For example, a call to **theme()** with **\$hook** = 'user_picture' will use the template **user-picture.tpl.php**, which returns the HTML code to display a user's profile picture. **F56.23**

F56.23**Using theme()**

Example of a call to the function theme().

```
$account->content['user_picture'] = array(
  '#markup' => theme('user_picture', array('account' => $account)),
  '#weight' => -10,
);
```

<http://api.drupal.org/api/drupal/modules--user--user-picture.tpl.php/7>

Content of **user-picture.tpl.php**. **F56.24**

F56.24**Template file**

Example of a template file for the 'user_picture' element.

```
<?php

/**
 * @file
 * Default theme implementation to present a picture configured
for the
 * user's account.
 *
 * Available variables:
 * - $user_picture: Image set by the user or the site's default. Will be linked
 * depending on the viewer's permission to view the user's profile page.
 * - $account: Array of account information. Potentially unsafe. Be sure to
 * check_plain() before use.
 *
 * @see template_preprocess_user_picture()
 */
?>
<?php if ($user_picture): ?>
  <div class='user-picture'>
    <?php print $user_picture; ?>
  </div>
<?php endif; ?>
```

As we will see in the next section, the passage of variables between the call to theme() and the template file is not always direct. In many cases, as in the example of **user-picture.tpl.php**, a preprocessing function is used (**template_preprocess_user_picture()**), which will be responsible for preparing and sending the final variables to the template.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Theme functions and default templates

Each module offers theme functions or templates for its elements. In the URL below a complete list of theme functions and templates defined by the Drupal system through core modules is available:

<http://api.drupal.org/api/drupal/modules--system--theme.api.php/group/themeable/7>

Next we will learn that it is possible to override both default templates and theme functions, which will allow us to modify the final look of the theme according to the particular requirements of the site.

Overriding theme functions

Now let's take a look at an example which we will apply to our **Example Theme**.

The theme function **theme_breadcrumb()** returns the HTML that corresponds to breadcrumb links. **F56.25**

http://api.drupal.org/api/drupal/includes--theme.inc/function/theme_breadcrumb/7

```
/** 
 * Returns HTML for a breadcrumb trail.
 *
 * @param $variables
 *   An associative array containing:
 *   - breadcrumb: An array containing the breadcrumb links.
 */
function themeBreadcrumb($variables) {
  $breadcrumb = $variables['breadcrumb'];

  if (!empty($breadcrumb)) {
    // Provide a navigational heading to give context for
    // breadcrumb links to screen-reader users.
    // Make the heading invisible with .element-invisible.
    $output = '<h2 class="element-invisible">' . t('You are here') . '</h2>';

    $output .= '<div class="breadcrumb">' .
      implode(' > ', $breadcrumb) . '</div>';
    return $output;
  }
}
```

F56.25

Overriding functions

Theme functions can be overridden in the template.php file of the theme.

The **theme_breadcrumb()** function receives as its parameter the **\$variables array** where the element **\$variables['breadcrumb']** is itself an array with links to the different pages of the breadcrumbs.

The HTML generated corresponds to the concatenation of all these elements, using the '»' symbol as a separator. Additionally, the class="breadcrumb" is used to define breadcrumb styles. **F56.26**

Home » Option 1

Neo Pneum Sagaciter Veniam

Submitted by Anonymous on Tue, 10/21/2014 - 11:48



Abbas abigo amet incassum interdico populus praemitto qui quidem vero. Adipiscing facilisi laoreet macto praemitto quadrum. Cui dolus enim incassum roto volutpat. Dolore quis exerci oppeto paulatim proprius saluto suscipit ut vulpes.

F56.26

Overriding functions: theme_breadcrumb()

HTML output from the theme_breadcrumb() system function.

```
<h2 class="element-invisible">You are here</h2><div
class="breadcrumb"><a href="/">Home</a> » <a href="/node/5"
title="">Option 1</a></div>
```

We will modify the resulting HTML generated in the breadcrumb, substituting the separator symbol '»' with a dash '-'.

By modifying the **theme_breadcrumb()** function directly in **theme.inc** we could easily achieve our goal, but this is an approach we **should never do**. The reasons are the same we've emphasized a number of times: **core code and installed modules should not be altered**, since this would prevent us from carrying out future updates. We will now learn other methods for carrying out the modification correctly.

Theme functions can be overridden from the theme. In our example theme, the function **theme_breadcrumb()** can be overridden with a new function called **example_themeBreadcrumb()**. But where should this function be added? This function, just like the rest of theme-specific functions, will be added to the theme folder, in a file called **template.php**. **F56.27**

F56.27**Overriding functions:
theme_breadcrumb()**

Example of modifying the theme function that displays breadcrumbs.

```
function example_theme_breadcrumb($variables) {
  $breadcrumb = $variables['breadcrumb'];

  if (!empty($breadcrumb)) {
    $output = '<h2 class="element-invisible">' . t('You are here') . '</h2>';

    $output .= '<div class="breadcrumb">' .
      implode(' - ', $breadcrumb) . '</div>';
  }
}
```

Assuming that the active theme is **example_theme**, when the system **theme('breadcrumb')**, Drupal will look to see if the following functions or files exist:

1. **example_theme_breadcrumb()**, in the template.php file for the active theme
2. **breadcrumb.tpl.php**, template file in the active theme.
3. **theme_breadcrumb()**, in includes/themes.inc. Default function which will be used as a last resort, if none of the previous options exists.

When the system finds (searching in order) one of these functions, it will stop searching and use the function or template file it found.

We've already discussed the fact that for changes in the theme to have an effect, we have to rebuild the theme registry. We can also do this from the file **template.php**, calling the **system_rebuild_theme_data()** and **drupal_theme_rebuild()** functions directly. **F56.28**

F56.28**Rebuild the theme
registry**

Functions which allow the theme registry to be rebuilt in every page load. They will be used only during theme development.

```
// Rebuild .info data.
system_rebuild_theme_data();
// Rebuild theme registry.
drupal_theme_rebuild();

function example_theme_breadcrumb($variables) {
  ...
}
```

It is important to remove or comment out these calls once theme development is complete, since they are heavy resource consumers, constantly evaluating all the changes made in the theme.

Overriding template files

In the previous example, in place of the function **example_theme_breadcrumb()** we could have created a template file for the breadcrumbs. This file, which will be located in the root directory of the theme, will be called **breadcrumb.tpl.php**. **F56.29**

```
<?php
/**
 * breadcrumb.tpl.php
 */

$breadcrumb = $variables['breadcrumb'];

?>
<?php if (!empty($breadcrumb)): ?>
  <div class='breadcrumb'><?php print implode(' - ', $breadcrumb); ?>
    </div>
<?php endif; ?>
```

F56.29

Overriding template files

We can also create a template file, keeping in mind that theme functions will be considered first

We must keep in mind the order indicated previously. Although we created the file **breadcrumb.tpl.php**, if the function **example_theme_breadcrumb()** has been created within template.php, it will win out over the template file.

56.6

Preprocessing Functions

Each time the system, using the **theme()** function, loads a template file, the template passes through a set of functions that carry out preprocessing tasks and are executed sequentially. For example, the first to execute are functions that belong to a module, then those that belong to a theme, such that theme functions will have greater control over the final result, being the last to execute.

Let's see an example. Previously we changed the breadcrumbs delimiter from '»' to a dash '-'. Now we want to be able to define the delimiter in a variable, which could be modified from the theme configuration area. We will create a preprocessing function in **example_theme** which will obtain the value of the variable, and later we will modify the template file **breadcrumb.tpl.php** so that it uses the value of the variable as the delimiter. **F56.30**

F56.30

Preprocessing function

Example of a preprocessing function. It allows for interaction with the variables that will be passed to the template.

```
/*
 * Implements THEME_preprocess_HOOK (&$variables)
 */
function example_theme_preprocess_breadcrumb (&$variables) {
    $variables['breadcrumb_separator'] =
        theme_get_setting('breadcrumb_separator');
}
```

All the variables that we will add or change in the **\$variables** array, by way of the preprocessing functions related to the **breadcrumb** template, will be available for use in the template file **breadcrumb.tpl.php**. **F56.31**

F56.31

Variable use in templates

Variables registered in the preprocessing function will be available in theme functions and templates.

```
<?php
/**
 * breadcrumb.tpl.php
 */

$breadcrumb = $variables['breadcrumb'];
$breadcrumb_separator = $variables['breadcrumb_separator'];

?>
<?php if (!empty($breadcrumb)): ?>
    <div class='breadcrumb'><?php print implode(' '.
$breadcrumb_separator . ' ', $breadcrumb); ?>
        </div>
<?php endif; ?>
```

Note that the variable **\$variables** is passed to the function by reference: **&\$variables**, so that the changes that we make within the function will also be available outside it, and the variable can be reused by other functions, with the changes made being maintained.

In Drupal 6 preprocessing functions were only available for template files, but in Drupal 7 preprocessing has been extended as well to theme functions. In theme functions, only the preprocessing functions specific to that hook will intervene.

The order of execution of preprocessing functions is the following:

Core:

1. **template_preprocess**(&\$variables, \$hook). Creates a set of default variables which can be used by all the hooks.
2. **template_preprocess_HOOK**(&\$variables). This function will usually accompany the module which registers the template. For example, the Comment module adds the function `template_preprocess_comment()` to process the variables belonging to the comment.tpl.php template.

Modules:

3. **MODULE_preprocess**(&\$variables, \$hook). This function will be called for all templates.
4. **MODULE_preprocess_HOOK**(&\$variables). This function is used when a module seeks to modify the variables of a template from another module.

Theme engine (phptemplate):

5. **phptemplate_engine_preprocess**(&\$variables, \$hook). This function should only be implemented by the theme engine.
6. **phptemplate_engine_preprocess_HOOK**(&\$variables). This function should only be implemented by the theme engine.

Theme:

7. **THEME_preprocess**(&\$variables, \$hook). Created in the template.php file of the theme. For example, `example_theme_preprocess()`.
8. **THEME_preprocess_HOOK**(&\$variables). Created in the template.php file of the theme, and acts on one template in particular. For example, `example_theme_preprocess_breadcrumb()`.

In Drupal 7 a second phase of functions, called processing functions, has been added. These allow additional work on the variables (\$variables) that are available in the template. Their functioning is exactly the same, except that they are called after all preprocessing functions have been executed. The order of these functions is:

Core:

9. `template_process(&$variables, $hook)`.
10. `template_process_HOOK(&$variables)`.

Modules:

11. `MODULE_process(&$variables, $hook)`.
12. `MODULE_process_HOOK(&$variables)`.

Theme engine (phptemplate):

13. `phptemplate_engine_process(&$variables, $hook)`.
14. `phptemplate_engine_process_HOOK(&$variables)`.
15. `phptemplate_process(&$variables, $hook)`.
16. `phptemplate_process_HOOK(&$variables)`.

Theme:

17. `THEME_process(&$variables, $hook)`.
18. `THEME_process_HOOK(&$variables)`.

It is important to keep in mind that various modules might implement preprocessing (or processing) functions on the same hook. The order of execution will depend on the order in which the modules are executed.

As in each phase, the last functions to act are those implemented within the theme, which always has the last word in variable treatment.

56.7

Theme configuration options

Every theme is able to have specific configuration options which are available at:

Administration ⇒ Appearance ⇒ Theme [Settings]

To add new configuration options, we will implement the **hook_form_system_theme_settings_alter()** function in a new file of the theme, a file we will call **theme-settings.php**.

http://api.drupal.org/api/drupal/modules--system--theme.api.php/function/hook_form_system_theme_settings_alter/7

hook_form_system_theme_settings_alter(&\$form, &\$form_state)

This function allows for altering the configuration form of the theme, which is why it receives the typical parameters associated with forms: \$form and \$form_state, both by reference.

We return to the implementation of **Example Theme** to add a text field which will allow us to establish the breadcrumb delimiter, which we will store in the variable 'breadcrumb_separator'. The system is responsible for storing the field value. [F56.32](#) [F56.33](#)

F56.32

Configuration Options

A new field in the theme's Configuration Options form.

BREADCRUMB SETTINGS

Breadcrumb separator

F56.33

Theme configuration form

A function which allows us to modify the theme configuration form. In this example we have added a group of breadcrumb options and a text element where the separator to be used between breadcrumb elements can be indicated.

```
<?php
/**
 * Implements hook_form_system_theme_settings_alter() function.
 */
function example_theme_form_system_theme_settings_alter(&$form,
$form_state) {

$form['breadcrumb'] = array(
  '#type'          => 'fieldset',
  '#title'         => t('Breadcrumb settings'),
);

$form['breadcrumb']['breadcrumb_separator'] = array(
  '#type'          => 'textfield',
  '#title'         => t('Breadcrumb separator'),
  '#default_value' => theme_get_setting('breadcrumb_separator'),
  '#size'           => 5,
  '#maxlength'     => 5,
);
}
```

In order to obtain the value of a theme configuration setting we use the function **theme_get_setting()**.

http://api.drupal.org/api/drupal/includes--theme.inc/function/theme_get_setting/7

The value of the setting can be obtained from various sources. These locations will be checked and the last value encountered will be used:

- Default value defined in theme_get_setting().
- Value from the configuration (.info) file of the base theme.
- **Value from the configuration (.info) file of the theme.**
- Value stored by the configuration form for the general options applicable to all themes.
- **Value stored by the theme configuration form.**

In our example we will define the default value in the theme configuration file, **example_theme.info**. F56.34

```
description = Example Theme for Drupal 7
screenshot = images/screenshot.gif
core = "7.x"
project = "example_theme"
engine = phptemplate

settings[breadcrumb_separator] = '+'
```

F56.34

Default value

In the .info file we can establish default parameter values

This will be the setting value used until the theme configuration form has been saved at least once. From that moment forward, the value taken will be the value stored in the database.

In order to use the setting's value in preprocessing functions, we also have to issue a call to the function **theme_get_setting()**. F56.35

```
/*
 * Implements THEME_preprocess_HOOK(&$variables)
 */
function example_theme_preprocess_breadcrumb(&$variables) {
  $variables['breadcrumb_separator'] =
    theme_get_setting('breadcrumb_separator');
}
```

F56.35

theme_get_setting()

This function allows us to obtain the value of a theme setting.

56.8 Creation of module templates

To understand how templates are applied to module development, we will use the **Forum** module. Forum, available in Drupal core (/modules/forum), is composed, in part, by:

- The main module file, forum.module, where functions are found.
- A set of template files with different purposes: **forums.tpl.php**, **forum-list.tpl.php**, **forum-topic-list.tpl.php**, **forum-icon.tpl.php**, **forum-submitted.tpl.php**.
- And CSS files that add styles to the aforementioned templates: **forum.css**, **forum-rtl.css**.

hook_theme() function

Our first task is to implement the function **hook_theme()** within the module.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_theme/7

As the following code demonstrates, the **forum_theme()** function returns an associative array where the elements which can later be themed are defined. **F56.36**

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

F56.36

hook_theme()

The Forum module implements hook_theme() by way of forum_theme(). In this function all the templates of the module are declared, along with their required variables.

```
/*
 * Implements hook_theme().
 */
function forum_theme() {
  return array(
    'forums' => array(
      'template' => 'forums',
      'variables' => array('forums' => NULL, 'topics' => NULL,
                           'parents' => NULL, 'tid' => NULL, 'sortby' => NULL,
                           'forum_per_page' => NULL),
    ),
    'forum_list' => array(
      'template' => 'forum-list',
      'variables' => array('forums' => NULL, 'parents' => NULL, 'tid' => NULL),
    ),
    'forum_topic_list' => array(
      'template' => 'forum-topic-list',
      'variables' => array('tid' => NULL, 'topics' => NULL, 'sortby' => NULL,
                           'forum_per_page' => NULL),
    ),
    'forum_icon' => array(
      'template' => 'forum-icon',
      'variables' => array('new_posts' => NULL, 'num_posts' => 0, 'comment_mode' => 0,
                           'sticky' => 0, 'first_new' => FALSE),
    ),
    'forum_submitted' => array(
      'template' => 'forum-submitted',
      'variables' => array('topic' => NULL),
    ),
    'forum_form' => array(
      'render element' => 'form',
      'file' => 'forum.admin.inc',
    ),
  );
}
```

Let us analyze one of the entries of the returned array: [F56.37](#)

```
/**
 * Implements hook_theme().
 */
function forum_theme() {
  return array(
    ...
    'forum_list' => array(
      'template' => 'forum-list',
      'variables' => array(
        'forums' => NULL,
        'parents' => NULL,
        'tid' => NULL),
    ),
    ...
  );
}
```

[F56.37](#)

Template registry

The 'template' element indicates the template which will be used, and the \$variables array indicates the variables which will be available to the template.

The 'forum_list' element will be available to and callable by the **theme()** function or as a value of the **#theme** property in a render array.

The array is composed of the 'template' field, which indicates the template to be used for this element. The value 'forum-list' indicates that the system will search for the template called **forum-list.tpl.php**. The 'variables' field stores different settings which should be passed to **theme()**, indicating as well their default values.

Next we see a call to the **theme()** function from within the module code. The first setting indicates which element will be themed, and then the **\$variables** array is passed to the function, so that necessary field values can be extracted: [F56.38](#)

```
$variables['forums'] = theme('forum_list', $variables);
...
```

[F56.38](#)

theme()

The template file **forum-list.tpl.php**, has the following code: [F56.39](#)

```
<?php

/**
 * @file
 * Default theme implementation to display a list of forums and containers.
 *
 * Available variables:
 * - $forums: An array of forums and containers to display. It is keyed to the
 *   numeric id's of all child forums and containers.
 * - $forum_id: Forum id for the current forum. Parent to all items within
 *   the $forums array.
 *
 * Each $forum in $forums contains:
 * - $forum->is_container: Is TRUE if the forum can contain other forums. Is
 *   FALSE if the forum can contain only topics.
 * - $forum->depth: How deep the forum is in the current hierarchy.
 * - $forum->zebra: 'even' or 'odd' string used for row class.
 * - $forum->icon_class: 'default' or 'new' string used for forum icon class.
 * - $forum->icon_title: Text alternative for the forum icon.
 * - $forum->name: The name of the forum.
 * - $forum->link: The URL to link to this forum.
 * - $forum->description: The description of this forum.
 * - $forum->new_topics: True if the forum contains unread posts.
 * - $forum->new_url: A URL to the forum's unread posts.
 * - $forum->new_text: Text for the above URL which tells how many new posts.
 * - $forum->old_topics: A count of posts that have already been read.
 * - $forum->num_posts: The total number of posts in the forum.
 * - $forum->last_reply: Text representing the last time a forum was posted or
 *   commented in.
 *
```

[F56.39](#)

forum-list.tpl.php

Template file from the Forum module.

```

* @see template_preprocess_forum_list()
* @see theme_forum_list()
*/
?>
<table id="forum-<?php print $forum_id; ?>">
  <thead>
    <tr>
      <th><?php print t('Forum'); ?></th>
      <th><?php print t('Topics'); ?></th>
      <th><?php print t('Posts'); ?></th>
      <th><?php print t('Last post'); ?></th>
    </tr>
  </thead>
  <tbody>
    <?php foreach ($forums as $child_id => $forum): ?>
      <tr id="forum-list-<?php print $child_id; ?>" class="<?php print
$forum->zebra; ?>">
        <td <?php print $forum->is_container ? 'colspan="4"
class="container"' : 'class="forum"'; ?>>
          <?php /* Enclose the contents of this cell with X divs, where X is the
           * depth this forum resides at. This will allow us to use CSS
           * left-margin for indenting.
           */ ?>
          <?php print str_repeat('<div class="indent">', $forum->depth);
?>
          <div class="icon forum-status-<?php print $forum->icon_class;
?>" title="<?php print $forum->icon_title; ?>">
            <span class="element-invisible"><?php print $forum-
>icon_title; ?></span>
          </div>
          <div class="name"><a href="<?php print $forum->link;
?>"><?php print $forum->name; ?></a></div>
          <?php if ($forum->description): ?>
            <div class="description"><?php print $forum->description;
?></div>
          <?php endif; ?>
          <?php print str_repeat('</div>', $forum->depth); ?>
        </td>
        <?php if (!$forum->is_container): ?>
          <td class="topics">
            <?php print $forum->num_topics ?>
            <?php if ($forum->new_topics): ?>
              <br />
              <a href="<?php print $forum->new_url; ?>"><?php print
$forum->new_text; ?></a>
            <?php endif; ?>
          </td>
          <td class="posts"><?php print $forum->num_posts ?></td>
          <td class="last-reply"><?php print $forum->last_reply ?></td>
        <?php endif; ?>
      </tr>
    <?php endforeach; ?>
  </tbody>
</table>

```

We see that there are two variables: the array **\$forums**, which contains information on all the forums and topics which will be displayed, and **\$forum_id**, which identifies the parent form of the forums contained in **\$forums**.

But, where did all these variables available for the template come from? Well, as we studied in previous sections, the **preprocessing functions** are responsible for assigning these values. Let's take a look at the content of the function **template_preprocess_forum_list(&\$variables)**: **F56.40**

```


/**
 * Process variables to format a forum listing.
 *
 * $variables contains the following information:
 * - $forums
 * - $parents
 * - $tid
 *
 * @see forum-list.tpl.php
 * @see theme_forum_list()
 */
function template_preprocess_forum_list (&$variables) {
    global $user;
    $row = 0;
    // Sanitize each forum so that the template can safely print the data.
    foreach ($variables['forums'] as $id => $forum) {
        $variables['forums'][$id]->description = !empty($forum->description) ? filter_xss_admin($forum->description) : '';
        $variables['forums'][$id]->link = url('forum/$forum->tid');
        $variables['forums'][$id]->name = check_plain($forum->name);
        $variables['forums'][$id]->is_container =
            !empty($forum->container);
        $variables['forums'][$id]->zebra = $row % 2 == 0 ? 'odd' : 'even';
        $row++;

        $variables['forums'][$id]->new_text = '';
        $variables['forums'][$id]->new_url = '';
        $variables['forums'][$id]->new_topics = 0;
        $variables['forums'][$id]->old_topics = $forum->num_topics;
        $variables['forums'][$id]->icon_class = 'default';
        $variables['forums'][$id]->icon_title = t('No new posts');
        if ($user->uid) {
            $variables['forums'][$id]->new_topics =
                forum_topics_unread($forum->tid, $user->uid);
            if ($variables['forums'][$id]->new_topics) {
                $variables['forums'][$id]->new_text =
                    format_plural($variables['forums'][$id]->new_topics, '1 new', '@count
new');
                $variables['forums'][$id]->new_url = url('forum/$forum->tid',
                    array('fragment' => 'new'));
                $variables['forums'][$id]->icon_class = 'new';
                $variables['forums'][$id]->icon_title = t('New posts');
            }
            $variables['forums'][$id]->old_topics = $forum->num_topics -
$variables['forums'][$id]->new_topics;
        }
        $variables['forums'][$id]->last_reply = theme('forum_submitted',
            array('topic' => $forum->last_post));
    }
    // Give meaning to $tid for themers. $tid actually stands for term id.
    $variables['forum_id'] = $variables['tid'];
    unset($variables['tid']);
}


```

F56.40**Preprocessing function**

Example of a preprocessing function belonging to the Forum module. It prepares the variables which will later be available in the template.

Finally, remember that if we want to modify the template of a module, we have to do so by copying the file into the theme directory of the site. In this case, we copy **forum-list.tpl.php** to the folder **/sites/all/themes/example_theme/templates**, and there we make the appropriate changes, which would affect how forum lists and containers appear.

For a deeper understanding of this unit, an in-depth study of the **Forum** module is recommended.

Review as well **Example 50.3**, in which we implemented theme functions of the module **Nodes Forcontu**, with the corresponding calls to **theme()** from **hook_view()**, in order to display the resulting final HTML of node elements

hook_init() function

The **hook_init()** function allows modules to carry out tasks prior to page loading. A common use related to themes is loading of CSS or JS files. **F56.41**

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_init/7

F56.41

hook_init()

The implementation of hook_init() allows modules to carry out tasks prior to page loading, such as adding css and js files.

```
/**  
 * Implements hook_init().  
 */  
function theming_forcontu_init() {  
 drupal_add_css(drupal_get_path('module', 'theming_forcontu') .  
 '/theming_forcontu.css');  
 drupal_add_js(drupal_get_path('module', 'theming_forcontu') .  
 '/theming_forcontu.js');  
}
```

The **drupal_add_css()** function adds a CSS stylesheet:

http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_add_css/7

The **drupal_add_js()** function adds a JavaScript file:

http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_add_js/7

The **drupal_add_library()** function can add multiple files JavaScript and CSS at the same time.

http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_add_library/7

57 Using jQuery and Ajax

JavaScript can add dynamic presentation effects to a website. Effects like showing/hiding an element or showing a countdown timer that decrements automatically without a page load are possible thanks to JavaScript. In Drupal it is possible to add specialized JavaScript code, and we can also make use of jQuery, a library of functions integrated into core that facilitates the use of JavaScript.

JavaScript is a client-side language, but in combination with **Ajax**, background asynchronous communication with the server is possible. This communication with the server allows for changes to be made in certain page content areas without having to reload the entire page.

Comparative D7/D6

In general we can state that the changes between Drupal 6 and Drupal 7, especially where Ajax is concerned, are important. Although the functioning is very similar, the API functions have been updated and improved. Es muy parecido, las funciones de la API han sido revisadas y mejoradas. The functioning of Ajax forms has also been changed, with the addition of the `#ajax` property which simplifies the process of adding Ajax to any form element.

In this unit we will introduce the concepts of jQuery and Ajax in Drupal. It is not the intent of this unit to study the programming languages that make up JavaScript and jQuery, but rather cover some of the methods for integrating these languages into our website. Therefore, if you do not have prior knowledge of JavaScript and/or jQuery, it may be necessary to study additional resources in which the use of these languages is covered more fully.

Unit contents

57.1 Introduction to JavaScript in Drupal	382
57.2 Introduction to jQuery	384
57.3 jQuery libraries in core.....	393
57.4 Ajax.....	397



57.1

Introduction to JavaScript in Drupal

JavaScript can add dynamic presentation effects to a website. Effects like showing/hiding an element or showing a countdown timer that decrements automatically without a page load are possible thanks to JavaScript. In Drupal it is possible to add specialized JavaScript code, and we can also make use of **jQuery**, a library of functions integrated into core that facilitates the use of JavaScript.

In Drupal it is possible to add specialized JavaScript code, and we can also make use of **jQuery**, a library of functions integrated into core that facilitates the use of JavaScript.

jQuery allows for the manipulation of **DOM** (Document Object Model) elements. The DOM is an API which provides access to the objects which make up HTML and XML pages. By manipulating these objects, it is possible to modify the content, structure, and style of HTML and XML documents. Moreover, it is also possible to react to events (clicking or moving the mouse over an element, pressing a key, loading a page, etc.).

JavaScript is a client-side language, but in combination with **Ajax**, background asynchronous communication with the server is possible. This communication with the server allows for changes to be made in certain page content areas without having to reload the entire page.

In this unit we will concentrate on programming JavaScript/Ajax by means of **jQuery** and the **Ajax framework of Drupal**.

Module jQuery Update

Drupal 7 is distributed with jQuery 1.4.4 and jQuery UI 1.8.7. When a desired functionality requires a later version, we can install the module **jQuery Update**, which updates jQuery to the latest available versions.

The **jQuery Update** module is available at:

http://drupal.org/project/jquery_update

To use the module, simply activate it and check in Drupal's **Status Report** that the version of jQuery has been updated.

Adding JavaScript to Drupal

There are several options for adding JavaScript files in a theme:

- Registering the file in the configuration file of the theme (**.info**). Keep in mind that in Drupal 7 it is necessary to register the JavaScript file for it to be included in the page load. JavaScript added by the theme will be executed after code added by core and other modules.

```
scripts[] = myscript.js
```

- We can also add JavaScript libraries in the file **template.php** by calling the **drupal_add_js()** or **drupal_add_library()** functions.

In the following example we use the page preprocessing function, HOOK 'html', to load the myscript.js library.

```
function example_theme_preprocess_html(&$variables) {
  $options = array(
    'group' => JS_THEME,
  );
  drupal_add_js(drupal_get_path('theme', 'example_theme').
    '/myscript.js', $options);
}
```

The function **drupal_add_library()** allows the addition of a library together with all its dependencies. For example, the system library vertical-tabs adds all the functionality required to show vertical tabs, which includes the files vertical-tabs.js and vertical-tabs.css, both in the /misc directory.

```
function example_theme_preprocess_html(&$variables) {
  drupal_add_library('system', 'drupal.vertical-tabs');
}
```

The **drupal_add_library()** function is available at:

http://api.drupal.org/api/drupal/includes--common.inc/function/drupal_add_library/7

As we stated in Unit 56, if we want to add JavaScript libraries in a module, we can implement **hook_init()** and use any of the functions referred to in this section. We can also load libraries directly in the return function of a page, referenced from **hook_menu()**, if the libraries are only used in that page.

Closing JavaScript code

It is a good practice to wrap JavaScript code in a (closure) function such that the scope of its variables is limited, and thereby avoiding accidentally overwriting other global variables.

```
(function () {
  // JavaScript code
})();
```

Passing the \$ parameter to this function, we are able to use function **\$()**, an alias for **jQuery()**, internally, which is what allows us to work with jQuery.

```
(function ($) {
  // jQuery code. $ can be used without conflicts
  console.log($.browser);
})($);
```

jQuery API

All necessary information about the functions available in jQuery can be found in its API: <http://api.jquery.com/>

57.2

Introduction to jQuery

In this section we will make a short introduction to the use of jQuery in Drupal. We've already stated that it is not within the scope of this course to offer full JavaScript and jQuery training, and we recommend consulting additional information and tutorials to supplement this unit.

`$(document).ready()`

By means of the `.ready()` jQuery method, the system checks that the structure of the page has been loaded and is ready for use, thereby avoiding that references are made to elements which are not yet defined.

<http://api.jquery.com/ready/>

As a general rule we will always use this syntax before adding custom jQuery code. **F57.1**

```
(function ($) {
  $(document).ready(function () {
    // jQuery code
  });
})(jQuery);
```

F57.1

Method `.ready()`

The `.ready()` method checks that the page has been loaded and is ready for use.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Executing jQuery code in Drupal

In order to execute the following within the Drupal site, we can implement the module **jQuery Forcontu** (`jquery_forcontu`) with the following structure:

- **hook_menu()**. We will implement the `hook_menu()` function within `jquery_forcontu.module`, registering each example page via a single callback function. **F57.2**

```
/***
 * Implements hook_menu().
 */
function jquery_forcontu_menu() {
  $items = array();
  $items['jquery_forcontu/example1'] = array(
    'title' => 'Example 1',
    'page callback' => 'jquery_forcontu_example1',
    'access callback' => TRUE,
  );
  $items['jquery_forcontu/example2'] = array(
    'title' => 'Example 2',
    'page callback' => 'jquery_forcontu_example2',
    'access callback' => TRUE,
  );
  return $items;
}
```

F57.2

jQuery Menu

We will implement the jQuery Menu module in order to include the examples and activites included in this unit. We will register the URLs of the example pages in `hook_menu()`.

- **Callback Functions.** In each callback function we will load the example js library (and css if necessary). The function will return the resulting HTML of the example in `$output`. **F57.3**

```
function jquery_forcontu_example1() {
  drupal_add_js(drupal_get_path('module', 'jquery_forcontu') .
  '/js/jquery_forcontu_example1.js');

  $output = '<a href="http://www.forcontu.com" target="_blank">Go
  to forcontu.com</a>';
  return $output;
}
```

- **Files .js.** We will create a.js file for each example and save them in the js directory within the module. **F57.4**

```
// file js/jquery_forcontu_example1.js

(function($) {
  $(document).ready(function() {
    $("a").click(function(event) {
      alert("You have clicked on a link...\\nThe page
           forcontu.com will open in a new window.");
    });
  })(jQuery);
}
```

- **Clear cache.** Every time we upload the module with new examples, we should clear site cache. It may also be necessary to update the page with Control + F5.

F57.3**Callback Function**

Each URL has an associated callback function, where we load the js library and will write the HTML code.

F57.4**.js file**

In the.js file we will include the jQuery code, which will be executed on the HTML elements of the page created in each example.

Element Selection

jQuery includes many methods for referencing a page element with which we will later interact. The complete list of selectors is available at:

<http://api.jquery.com/category/selectors/>

Some of these selectors are:

- **Element ID ("#id").** Selects the element with the indicated ID. In any given page the ID value of an element should be unique. If the value is repeated, the document will be considered invalid. Leaving this consideration aside, the selector will only return the first matching element which it finds.

<http://api.jquery.com/id-selector/>

In the following example the element with id="block1" is searched for and receives a value in its "border" attribute, by means of the .css() method.

JQuery code:

```
$("#block1").css("border", "3px solid red");
```

HTML code:

```
<div id="block1">...</div>
```

- **Element class ("class")**. Returns all elements with the indicated class. Actions will affect all the matching elements.

<http://api.jquery.com/class-selector/>

In the following example, all elements with the class="block" are found, assigned a value to their "border" attribute. This takes place using the .css() method.

JQuery code:

```
$(".block").css("border","3px solid red");
```

HTML code:

```
<div class="block">Block 1</div>
<div class="block">Block 2</div>
```

- **Tag name ("element")**. Returns all the elements with the tag name given in "element".

<http://api.jquery.com/element-selector/>

In the following example all <div> tags are found, regardless of id and/or class values assigned to them. The element will not be returned.

JQuery code:

```
$( "div" ).css("border","9px solid red");
```

HTML code:

```
<div id="block1" class="block">Block 1</div>
<div id="block2" class="block">Block 2</div>
<span>Block 3</div>
```

- **Attribute Value (name="value")**. Returns all elements which have a "name" attribute with the value of "value".

<http://api.jquery.com/attribute-equals-selector/>

In the following example all input elements whose "value" attribute contains the value "Monday" are returned.

JQuery code:

```
$('input[value="Monday"]').parent().css("color", "red");
```

HTML code:

```
<label>
  <input type="radio" name="weekday" value="Monday" />Monday
</label>
<label>
```

```

<input type="radio" name="weekday" value="Tuesday" />Tuesday
</label>
<label>
  <input type="radio" name="weekday" value="Wednesday" />Wednesday
</label>

```

There are other selectors which allow for attribute-based selection by making different types of comparisons:

- **[name |= "value"]**. The attribute name has the value "value" or a string which begins with "value-".
<http://api.jquery.com/attribute-contains-prefix-selector/>
- **[name *= "value"]**. The value of the attribute name contains the string "value" (substring).
<http://api.jquery.com/attribute-contains-selector/>
- **[name ~= "value"]**. The value of the attribute name contains the word "value". The word should be at the beginning or end of the chain and be separated from other words by spaces.
<http://api.jquery.com/attribute-contains-word-selector/>
- **[name != "value"]**. The value of the attribute is not equal to "value". This also returns elements which do not have the attribute defined.
<http://api.jquery.com/attribute-not-equal-selector/>
- **[name ^= "value"]**. The value of the attribute begins with the string "value".
<http://api.jquery.com/attribute-starts-with-selector/>
- **[name \$= "value"]**. The value of the attribute ends with the string "value".
<http://api.jquery.com/attribute-ends-with-selector/>

- **All elements ("*")**. The selector "*" selects all page elements.
<http://api.jquery.com/all-selector/>
- **Element type**. In these selectors we can use the short version, `$('button')`, which is equivalent to `$('[type=button]')`.

Some of the type selectors we can use are:

- **:button**. <http://api.jquery.com/button-selector/>
- **:checkbox**. <http://api.jquery.com/checkbox-selector/>
- **:file**. <http://api.jquery.com/file-selector/>
- **:image**. <http://api.jquery.com/image-selector/>
- **:password**. <http://api.jquery.com/password-selector/>
- **:radio**. <http://api.jquery.com/radio-selector/>
- **:reset**. <http://api.jquery.com/reset-selector/>
- **:submit**. <http://api.jquery.com/submit-selector/>
- **:text**. <http://api.jquery.com/text-selector/>

- **Element order:** even, odd, first element, last element, etc. Some of these selectors include:

- o **:even.** Selects even elements, for example, when traversing a table. Keep in mind that as a general rule, elements are indexed beginning with 0 (0, 1, 2, 3, 4, etc.), which means that the selection of even elements (0, 2, 4, etc.) actually returns odd elements (first element, third element, etc.).
<http://api.jquery.com/even-selector/>
- o **:odd.** Similar to :even, but with odd elements 1, 3, 5, etc.).
<http://api.jquery.com/odd-selector/>
- o **:first.** Returns the first element.
<http://api.jquery.com/first-selector/>
- o **:last.** Returns the last element.
<http://api.jquery.com/last-selector/>

In the following example we select elements whose tag is "tr" (table row), applying the following styles: **F57.5**

- In all rows the text color will be white (#FFFFFF).
- In even-numbered rows (0,2,4), the background will be red.
- In odd-numbered rows (1,3,5), the background will be green.
- The first row will have a blue background. In this case the background of row 0 is overwritten.

JQuery code:

```
$( "tr" ).css("color", "#FFFFFF");
$( "tr:even" ).css("background-color", "red");
$( "tr:odd" ).css("background-color", "green");
$( "tr:first" ).css("background-color", "blue");
```

HTML code:

```
<table border="1">
  <tr><td>Row #0 (first)</td></tr>
  <tr><td>Row #1</td></tr>
  <tr><td>Row #2</td></tr>
  <tr><td>Row #3</td></tr>
  <tr><td>Row #4</td></tr>
  <tr><td>Row #5</td></tr>
</table>
```

F57.5

jQuery Example

Example of selecting elements of a table tabla in function of their order (:even, :odd, :first and :last)

Row #0 (first)
Row #1
Row #2
Row #3
Row #4
Row #5

CSS Modification

One of the actions we can apply to page elements is to change their CSS attributes. Some of the methods in the jQuery API to do this type of change include:

- **.css()**. When only the name of a CSS property is passed, this returns the value of the property for the first element of the selection:
.css(propertyName)

```
var color = $("div.left").css("background-color");
```

If besides the property name a value or set of values is also passed, these values will be applied to the style and elements indicated:
.css(propertyName, value) and **.css(map)**

```
$( "p" ).css( "color" , "#000000" );
$( "#block1" ).css({ 'background-color': '#ffe' , 'border': '5px solid #ccc' });
```

<http://api.jquery.com/css/>

- **.addClass()**. Allows the addition of one or more classes, separated by spaces, to the class="" attribute of the element.
<http://api.jquery.com/addClass/>

```
$( "p" ).addClass( "class1 class2" );
```

- **.removeClass()**. Removes one or more classes, separated by spaces, from the class="" attribute of the element.
<http://api.jquery.com/removeClass/>

```
$( "p" ).removeClass( "oldclass1 oldclass2" );
```

These two functions are often used together to substitute one or more classes. For example:

```
$( "p" ).removeClass( "oldclass1 oldclass2" ).addClass( "class1" );
```

You will find more methods for working with CSS and attributes in the jQuery API, within the CSS and Attributes categories.

- <http://api.jquery.com/category/css/>
- <http://api.jquery.com/category/attributes/>

Events

Next we will consider some event-related methods. When one of these events occurs, the indicated actions will be carried out.

The available events can be reviewed at:

<http://api.jquery.com/category/events/>

- **.change()**, when a form value changes.
<http://api.jquery.com/change/>
- **.click()**, when an element is clicked.
<http://api.jquery.com/click/>
- **.error()**, when a JavaScript error is returned.
<http://api.jquery.com/error/>
- **.focus()**, when the focus or selected element is established. For example, in a form, the focus is the active field at any given moment.
<http://api.jquery.com/focus/>
- **.blur()**, when an element loses the focus.
<http://api.jquery.com/blur/>
- **.focusin, .focusout()**. Equivalent to .focus() and .blur(), respectively, with the difference that they also check the contents in the selected elements.
<http://api.jquery.com/focusin/>
<http://api.jquery.com/focusout/>
- **.hover()**, when the cursor passes over an element.
<http://api.jquery.com/hover/>
- **.keydown()**, when the user presses a key for the first time. For example, when writing within a text box begins, the event will execute only one time, when the first key is pressed.
<http://api.jquery.com/keydown/>
- **.keypress()**, when the user presses a key. The event is repeated with every key pressed.
<http://api.jquery.com/keypress/>
- **.mousedown(), .mouseenter(), .mouseleave(), .mousemove(), .mouseout(), .mouseover(), .mouseup()**. These functions offer different events related to mouse operations (move the cursor, drag and drop, etc.).
<http://api.jquery.com/mousedown/>
<http://api.jquery.com/mouseenter/>
<http://api.jquery.com/mouseleave/>
<http://api.jquery.com/mousemove/>
<http://api.jquery.com/mouseout/>
<http://api.jquery.com/mouseover/>
<http://api.jquery.com/mouseup/>
- **.scroll()**, when an element is scrolled.
<http://api.jquery.com/scroll/>
- **.select()**, when text is selected within an element.
<http://api.jquery.com/select/>
- **.submit()**, when a form submission is attempted.
<http://api.jquery.com/submit/>
- **.toggle()**. This event allows for a different action each time an element is clicked (default behavior)
<http://api.jquery.com/toggle-event/>

In the following example, any list element (``) will change color when clicked. After the first click it will change to blue. If we click again it will change again, this time to red, and with a third click, to green. At this point the cycle will begin again.

```
$("li").toggle(
  function () {
    $(this).css("color", "blue");
  },
  function () {
    $(this).css("color", "red");
  },
  function () {
    $(this).css("color", "green");
  }
);
```

We also want to draw attention to this function:

- **`event.preventDefault()`**, which prevents the default action for a determined event. For example, by calling this function we can avoid the default action of a clicked link, namely, the page load of the new URL.

Effects

The jQuery API also includes a set of methods that add dynamic effects to page elements. The complete list of effects can be found at:

<http://api.jquery.com/category/effects/>

- **`.animate()`**. Generates animations by modifying certain CSS properties.
<http://api.jquery.com/animate/>
- **`.delay()`**. Adds a delay to the animation, expressed in milliseconds.
<http://api.jquery.com/delay/>
- **`.fadeIn()`, `.fadeOut()`**. Both effects have to do with element opacity. The fadeIn() effect makes the element appear (less to greater opacity), while fadeOut() makes it disappear (greater to less opacity).
<http://api.jquery.com/fadeIn/>
<http://api.jquery.com/fadeOut/>
- **`.hide()`, `.show()`**. These effects hide or show an element. In both cases the duration of the effect can be specified as a parameter in milliseconds or by using the values 'slow' (600 ms) or 'fast' (200 ms).
<http://api.jquery.com/hide/>
<http://api.jquery.com/show/>
- **`.slideDown()`, `.slideUp()`**. The .slideDown() effect shows the element appearing from above. The .slideUp() effect hides the element, making it disappear from top to bottom.
<http://api.jquery.com/slidedown/>
<http://api.jquery.com/slidedup/>
- **`.stop()`**. Stops the animation which is being executed on the element.
<http://api.jquery.com/stop/>

In the following example we will combine some of the methods studied to this point. The layer identified as `#box1` will be shown or hidden by means of the defined links (`#hide` and `#show`). In order to hide the layer, we use the `.fadeOut()` effect, which makes the layer disappear by reducing its opacity. To show the layer we use the `.slideDown()` effect, which makes it appear from top to bottom.

F57.6

JQuery code:

```
$ (document) .ready(function() {
    $("#hide") .click(function(event) {
        event.preventDefault();
        $('#box1') .fadeOut(2000);
    });

    $("#show") .click(function(event) {
        event.preventDefault();
        $('#box1') .slideDown(3000);
    });
});
```

HTML code:

```
<div id="box1" style="background-color: green; color:#fff;
padding:10px;">
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tellus purus, hendrerit nec porta id, commodo sed sem. Donec scelerisque luctus arcu quis dignissim. Integer egestas velit eget neque sodales a feugiat ligula suscipit. Ut eros libero, blandit et porttitor eget, pharetra quis massa. In pharetra tincidunt dolor vitae auctor. Integer vitae velit molestie sapien vehicula consectetur ut id elit. Etiam tristique, risus at faucibus pharetra, augue nulla venenatis augue, vel semper turpis ante eu arcu. Donec sed orci ligula.</p>
</div>

<p>
<a href="#" id="hide">Hide the layer</a> |
<a href="#" id="show">Show the layer</a>
</p>
```

F57.6

jQuery effects

Example of effects created with jQuery.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tellus purus, hendrerit nec porta id, commodo sed sem. dignissim. Integer egestas velit eget neque sodales a feugiat ligula suscipit. Ut eros libero, blandit et porttitor eget, pharetra quis massa. In pharetra tincidunt dolor vitae auctor. Integer vitae velit molestie sapien vehicula consectetur ut id elit. Etiam tristique, risus at faucibus pharetra, augue nulla venenatis augue, vel semper turpis ante eu arcu. Donec sed orci ligula.

[Hide the layer](#) | [Show the layer](#)

jQuery Libraries in core

57.3

In this section we will analyze the JavaScript and jQuery libraries included in Drupal core. Drupal includes libraries related to its own functions and to other plugins which although generic, come packaged in core code.

We will start by looking at some of these generic plugins.

Accordion

This allows us to format page elements like an accordion, such that when one element is expanded, the remainder of the elements are collapsed.

The complete description of the plugin can be found at:

<http://docs.jquery.com/UI/Accordion>

As an example, we have created the function **jquery_forcontu_accordion()**, which applies to the URL **jquery_forcontu/accordion**, as registered in **hook_menu()**. **F57.7**

```
function jquery_forcontu_accordion() {
  drupal_add_library('system', 'ui.accordion');
  drupal_add_js(drupal_get_path('module', 'jquery_forcontu') .
    '/js/jquery_forcontu_accordion.js');

  $output = '<div id="accordion">
    <h3><a href="#">Section 1</a></h3>
    <div>
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing
      elit.</p>
    </div>
    <h3><a href="#">Section 2</a></h3>
    <div>
      <p>Vivamus nec nisl vitae lorem molestie elementum. </p>
    </div>
    <h3><a href="#">Section 3</a></h3>
    <div>
      <p>Quisque ultricies viverra hendrerit.</p>
    </div>
  </div>';
  return $output;
}
```

F57.7

jQuery Libraries

The system includes a set of jQuery libraries which can be used via a call to **drupal_add_library()**. In the same way we can add external libraries and use them in our site.

In order to load the plugin we use the **drupal_add_library()** function, which adds the complete library (**ui.accordion**). Later we add the customary .js file, with the particular configuration which makes use of the plugin in page elements (**jquery_forcontu_accordion.js**). **F57.8**

```
(function ($) {
  $(document).ready(function () {
    $('#accordion').accordion();
  })
})(jQuery);
```

F57.8

ui.accordion

Method call **.accordion()** once is available in the library.

The accordion effect which takes place when section headers are clicked is demonstrated in the **Figure**. **F57.9**

F57.9**Example: Accordion**

Example of the use of the Accordion library. When a title is clicked, its content is expanded and the rest of the elements collapse.

Accordion**Datepicker**

Adds a pop-up calendar for choosing dates, which we can use in textfield form fields.

<http://docs.jquery.com/UI/Datepicker>

We will define the form in the URL `jquery_forcontu/datepicker`.

F57.10**F57.10****Datepicker**

Using jQuery, adds a popup calendario for choosing a date.

```
/**
 * Implements hook_menu().
 */
function jquery_forcontu_menu() {
  $items = array();

  $items['jquery_forcontu-datepicker'] = array(
    'title' => 'Example: Datepicker',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('jquery_forcontu_datepicker'),
    'access callback' => TRUE,
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

In the function which defines the form we have added the system library **ui.datepicker**, using **drupal_add_library()**. This time, in place of including the jQuery code which uses the library in a separate file, we've used the "inline" format, which allows us to write the code directly in the call to **_add_js()**. **F57.11**

F57.11**ui.datepicker**

In this case we have included the jQuery code in the function that displays the form, using the 'inline' method of the function **drupal_add_js()**.

```
function jquery_forcontu_datepicker($form_state) {
  drupal_add_library('system', 'ui.datepicker');
  drupal_add_js('
    (function($) {
      $(document).ready(function() {
        $('#edit-mydate').datepicker();
      });
    })(jQuery);
  ', 'inline');

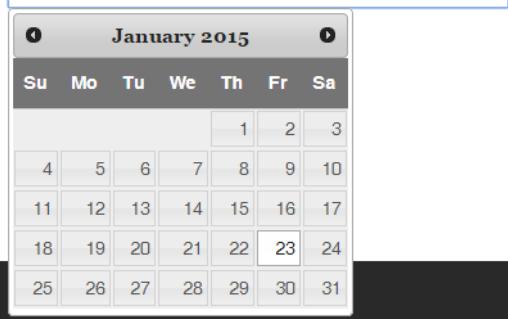
  $form['mydate'] = array(
    '#type' => 'textfield',
    '#title' => t('Select date'),
  );
  return $form;
}
```

As we studied in the previous section, we can reference a form element in a variety of ways. In this example we have used the element ID, keeping in mind that the ID assigned by the system to the element is #edit-mydate. **F57.12**

Example: Datepicker

Select date

01/23/2015



F57.12

Example: Datepicker

In this Figure we see the collapsible calendar which allows for date selection. The date value chosen will be sent to the associated form field.

Other libraries

We will briefly describe other libraries, which we can use in our code in the same way we have done in the previous examples. In the jQuery API you can find examples of how each one can be used.

- **Dialog** (`ui.dialog`). Creates a floating window (or dialog box) with a title and content area. This window can be moved, resized, and closed.
<http://docs.jquery.com/UI/Dialog>
- **Draggable** (`ui.draggable`). Makes an element draggable/movable by the mouse. This functionality is typically used with elements that allow releasing the element (the Droppable plugin). This technique is known as Drag & Drop.
<http://docs.jquery.com/UI/Draggable>
- **Droppable** (`ui.droppable`). Defines an element which allows draggable objects to be dropped. It is used, then, with the Draggable plugin.
<http://docs.jquery.com/UI/Droppable>
- **Progressbar** (`ui.progressbar`). Adds a progress bar, which displays the percent completion of any executing process. Keep in mind that the bar does not calculate time and percentage automatically, but that we must program the various states through which the bar will progress, in function of the tasks which have been completed.
<http://docs.jquery.com/UI/Progressbar>
- **Resizable** (`ui.resizable`). Makes an element resizable.
<http://docs.jquery.com/UI/Resizable>
- **Selectable** (`ui.selectable`). Makes list elements in selectable. Selections can be made directly via mouse clicks, using Control to select several non-contiguous elements, etc.
<http://docs.jquery.com/UI>Selectable>

- **Sortable** (`ui.sortable`). Allows the order of elements to be interactive through dragging and dropping.
<http://docs.jquery.com/UI/Sortable>
- **Tabs** (`ui.tabs`). Creates individual tabs for each element.
<http://docs.jquery.com/UI/Tabs>

Effects

In Drupal core there are also some jQuery effects included. Some of these are:

- **Bounce** (`effects.bounce`). This effect moves the element several times, moving it horizontally or vertically according to the specified configuration.
<http://docs.jquery.com/UI/Effects/Bounce>
- **Explode** (`effects.explode`). Makes the element explode into several pieces. It can be used as an effect that hides an element or can be used to show it.
<http://docs.jquery.com/UI/Effects/Explode>
- **Fold** (`effects.fold`). Folds the element as if it were a sheet of paper.
<http://docs.jquery.com/UI/Effects/Fold>
- **Pulsate** (`effects.pulsate`). Displays the element intermittently, altering its opacity and the number of pulses as indicated.
<http://docs.jquery.com/UI/Effects/Pulsate>

String translations

We know that the function `t()` allows strings included in code to be translatable via the translate interface. In the same way, we can include translatable strings in JavaScript code, by using the function `Drupal.t()`.

In the example we add the string "Advanced search" to the end of the element `#block1`, using the `.append()` method. The string will be available for translation just like any other string added using the function `t()`. F57.13

F57.13

Drupal.t()

The `Drupal.t()` function allows for the translation of strings that appear in JavaScript.

It is similar to `t()`.

```
(function ($) {
  $(document).ready(function () {
    $("#block1").append(Drupal.t('Advanced search'));
  });
})(jQuery);
```

Ajax

57.4

Using Ajax technology we can cause a page already loaded in the browser to continue communicating with the server in an asynchronous way, interchanging information in the background which can be displayed in the page without an entire page reload being necessary.

We have already seen the use of Ajax in Drupal in some of the modules we have used. For example, we can construct a view with a list of elements and a pager by using Ajax. When moving from one page to another, the content of the new page is obtained by an Ajax call, which consults the database and presents the information requested without reloading the page as a whole.

Drupal behaviors

We have already begun to work with Ajax, and should take the time to introduce the concept of **Drupal behaviors**.

Up until this point, we have used the `.ready()` method to check that the page has been loaded and to execute our code.

```
(function($) {
  $(document).ready(function() {
    // JQuery code
  });
}) (jQuery);
```

This method can cause a negative side effect: since the code is only executed when the page is loaded, it will not be applied to new elements which may be added later, dynamically. This problem is especially relevant when we use Ajax, given that it often requests new data from the server.

Let's consider a simple example. The following jQuery code adds the CSS class `"red-text"` to all `<p>` tags. We have previously defined the class in a.css file `(.red-text { color: red; })`.

All the actions defined in the `.ready()` event are executed as soon as the DOM is complete. First we add the new class to all `<p>` tags, but this action will only affect the tags defined in the. Because of this, when we later add additional HTML code with the `.append()` method, this content will not receive the class which the rest of the pre-existing paragraphs did in fact receive. **F57.14**

JQuery code:

```
(function($) {
  $(document).ready(function() {
    //Add the class "red-text" to all <p> tags
    $("p").addClass("red-text");

    //Add a new paragraph. This <p> tag will not receive the new class.
    $("#maintext").append("<p>Vivamus ut velit nisi, id ultricies
sem. Pellentesque tincidunt commodo neque et egestas.</p>");
  });
}) (jQuery);
```

HTML code:

```
<div id="maintext">
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce
tellus purus, hendrerit nec porta id, commodo sed sem.</p>
</div>
```

F57.14

Without Drupal Behaviors

Case in which the use of Drupal Behaviors is necessary.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tellus purus, hendrerit nec porta id, commodo sed sem.

Vivamus ut velit nisi, id ultricies sem. Pellentesque tincidunt commodo neque et egestas.

By using **Drupal behaviors** we can define behaviors for certain page elements, so that at any moment the behaviors can be executed anew on new elements added to the DOM.

In the following example we "encapsulate" the action which adds the class "red-text" to <p> tags in **Drupal.behaviors.addRedtext**, in order to be able to reuse it when new elements are added to the page.

Now, after a new element is added to the page, we should call the `Drupal.attachBehaviors()` function, which will be responsible for adding the correct behaviors to the element. F57.15

JQuery code:

```
(function ($) {
  Drupal.behaviors.addRedtext = {
    attach: function (context, settings) {
      $("p", context).addClass("red-text");
    }
  };

  $(document).ready(function () {
    // Add a new paragraph. This <p> tag will not receive the new class.

    $("#maintext").append("<p>Vivamus ut velit nisi, id ultricies
sem. Pellentesque tincidunt commodo neque et egestas.</p>");

    //Execute behaviors
    Drupal.attachBehaviors();
  });
})(jQuery);
```

HTML code:

```
<div id="maintext">
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce
tellus purus, hendrerit nec porta id, commodo sed sem.</p>
</div>
```

F57.15

With Drupal Behaviors

Previous example corrected using Drupal Behaviors.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tellus purus, hendrerit nec porta id, commodo sed sem.

Vivamus ut velit nisi, id ultricies sem. Pellentesque tincidunt commodo neque et egestas.

The function `Drupal.attachBehaviors(context, settings)` can receive context and settings parameters.

The context parameter is used to pass the function the element which has been

added. When we do not specify a specific context, the function will use the entire document as the context.

In our example we could limit the context to the element which contains the new paragraph: `Drupal.attachBehaviors($("#maintext"))`. In this way, only the `<p>` tags contained within the element `#maintext` would be looked at, the CSS class would only be added to the tags which do not yet have it.

When we do not give specific configuration settings for the current context in the `settings` parameter, the global object `Drupal.settings` will be used.

In Drupal 7 behaviors are defined by two functions, one when the content is added to the page (`attach`), and one when it is removed (`detach`). F57.16

```
function ($) {
  Drupal.behaviors.exampleModule = {
    attach: function (context, settings) {
      // When content is added to the page
    },
    detach: function (context, settings) {
      // When content is removed from the page (optional)
    },
  };
})(jQuery);
```

F57.16

Drupal Behaviors general structure

The general structure for implementing a behavior in `Drupal.behaviors`. The `attach` method is required, but the `detach` method is optional.

The `Drupal.attachBehaviors()` and `Drupal.detachBehaviors()` functions are defined in `misc/drupal.js`.

Using Ajax in Drupal

In order to include Ajax in our site we will use the functions of Drupal's **Ajax framework**, defined in the core file `includes/ajax.inc`.

By using Ajax we can dynamically update certain elements in the HTML page, requesting the information from the server when a given event occurs (for example when a button or link is clicked). The new information is provided by the server in the background, and in the page by reloading only the element that requires change, without having to reload the entire page.

Drupal's Ajax framework offers a set of PHP functions which provide complete Ajax event management. All of these functions are available at:

<http://api.drupal.org/api/drupal/includes--ajax.inc/7>

We can use the Ajax framework provided by Drupal in two ways:

- **Integrating Ajax with forms.** By using the Form API we can incorporate Ajax events in form elements, through the use of the `#ajax` property.
- **Integrating Ajax with links or buttons.** We can also use Ajax by adding classes to buttons or links. As we will see, by adding the class 'use-ajax' to a link enlace, the link will be treated as an Ajax call. In the same way, we can use buttons by applying the class 'use-ajax-submit'.

In either case, when the system receives an Ajax call, the server executes the requested actions and creates a **command array**. These commands will be converted to a JSON object, which is returned to the client for evaluation and parsing.

The commands generally correspond to JavaScript/jQuery methods. For example, the command "css" will indicate to the client that it should use the jQuery method .css(). As we have already learned, the .css() method allows CSS styles to be applied to the indicated element. The command will include in its definition array the following fields: command name ('command'), element or elements where the command applies ('selector'), and values which will be applied ('argument').

Each command will be converted to a JavaScript command object, which will be applied in the client according to the methods defined in Drupal.ajax. For example, the command 'css' is implemented by the **Drupal.ajax.prototype.commands.css()** function defined in misc/ajax.js.

We will create the module **Ajax Forcontu** (ajax_forcontu), where we will implement the following examples.

Ajax in links (without forms)

It is possible to use Ajax from a link, such that when the link is clicked, a function is executed via Ajax, which obtains information from the server. The information obtained, which in example will be the **server time**, will be displayed in the indicated HTML element, and obviously without reloading the entire page.

To begin, we register two URLs: **F57.17**

- '**ajax_forcontu/ajax_link**'. This first URL is the page which contains the link and where the content obtained by Ajax will be displayed. The function responsible for displaying the page content with the enlace will be ajax_forcontu_link().
- '**ajax_forcontu_link_callback**'. This registers the return function which makes the Ajax call. The link with the class 'use-ajax' should point to this URL.

F57.17

URL Registration

In hook_menu() we register two URLs with their corresponding functions, one to show the content and the other which does the lookup using Ajax.

```
/***
 * Implements hook_menu().
 */
function ajax_forcontu_menu() {
  $items = array();

  // Using Ajax without forms. Class 'use-ajax'.
  $items['ajax_forcontu/ajax_link'] = array(
    'title' => 'Ajax Example with link ("use-ajax" class)',
    'page callback' => 'ajax_forcontu_link',
    'access callback' => TRUE,
    'weight' => 9,
  );
  // Return function to issue an Ajax call to the server.
  $items['ajax_forcontu_link_callback'] = array(
    'page callback' => 'ajax_forcontu_link_response',
    'access callback' => 'user_access',
    'access arguments' => array('access content'),
    'type' => MENU_CALLBACK,
  );
  return $items;
}
```

The **ajax_forcontu_link()** function creates the page with the link which makes the Ajax call. It also includes the HTML code of the element which will later be replaced by the new content returned by Ajax.

In the page where the link resides, we should include the **drupal.ajax** system library by using the function **drupal_add_library()**. As we will see later, when we use Ajax in forms, this will no longer be necessary, since the system will load the library automatically.

We have created the link using the function `l()`. Although the URL registered for the Ajax call is **ajax_forcontu_link_callback**, we have added the parameter **/nojs/**. If JavaScript is enabled in the browser, the Ajax library is responsible for eliminating this fragment added to the URL. In this way we can test within the Ajax return function if the content should be returned via Ajax or without JavaScript, as normal content, reloading the page. In the link we have also added the attribute `class="use-ajax"`, a fundamental element that means that the link will be interpreted as an Ajax call.

Finally, we add the element `<div id="time"></div>`, which will be replaced by the content returned by the Ajax function. **F57.18**

```
function ajax_forcontu_link() {
    // We should include the library "drupal.ajax".
    drupal_add_library('system', 'drupal.ajax');

    $output = "<div>Click on the link to obtain the server
time.</div>";

    // We create a link with the special class 'use-ajax'
    // /nojs will be removed from the URL if javascript is enabled.
    $link = l(t('Click here'), 'ajax_forcontu_link_callback/nojs/',
        array('attributes' => array('class' => array('use-ajax'))));

    $output .= "<div id='time'></div><div>$link</div>";
    return $output;
}
```

F57.18**ajax_forcontu_link()**

In this function we load the drupal.ajax library and create a link with the class 'use-ajax'. The system will interpret the link as an Ajax lookup, executing the call to the function in the background.

The callback function which executes the Ajax function test first if the call type is Ajax (`type=='ajax'`). Within the function we generate an array **\$commands** with all the Ajax commands which will be returned to the client. **F57.19**

In this example we use the function **ajax_command_replace()**, in which we indicate that the element with the ID '#time' will be replaced by the value of `$output`. Note that the function does not replace only the content of the element, but rather the element in its entirety, including the `<div id="time"></div>` tags, and we will therefore include the tags within the new string `$output`.

The `ajax_command_replace()` function creates the 'insert/replaceWith' command, which on the client side will correspond to a call to the jQuery method `.replaceWith()`.

The complete list of available commands, together with the corresponding jQuery, can be found at:

<http://api.drupal.org/api/drupal/includes%21ajax.inc/7>

Finally, we construct the page `$page` with the added commands and make a call to the function **ajax_deliver(\$page)**, which will package and send the result as an Ajax response (in JSON format).

F57.19**ajax_forcontu_link_responce()**

This function is responsible for making the request to the server in the background and returning the result to the browser, in JSON format.

```
/*
 * Callback Function.
 *
 * If javascript is not enabled, the value 'nojs' will be passed
 * in the parameter $type. Otherwise, $type == 'ajax'.
 *
 * @return
 *   If $type == 'ajax', return an array of Ajax commands.
 *   Otherwise return the content by loading the page.
 */
function ajax_forcontu_link_response($type = 'ajax') {
  if ($type == 'ajax') {
    $output = "<div id='time'" . t("Content obtained from the
server via AJAX: ") . date("H:i:s") . '</div>';

    $commands = array();
    $commands[] = ajax_command_replace('#time', $output);

    $page = array('#type' => 'ajax', '#commands' => $commands);
    ajax_deliver($page);
  } else {
    $output = t("Content returned by loading the page without Ajax.");
    return $output;
  }
}
```

In the **Figure F57.20** we see the text returned by the server when the link is clicked.

F57.20**Example: Ajax with a link**

Example of using Ajax in a link with the class 'use-ajax'.

Example: Ajax with a link (class "use-ajax")

Click on the link to obtain the server time.

Content obtained from the server via AJAX: 03:57:01

[Click here](#)

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Ajax link with render array

The **ajax_forcontu_link()** function which creates the link that makes the Ajax call can also have been composed via a render array. In the following example we demonstrate the alternative function, which could be substituted without having to modify the function that returns the Ajax response. **F57.21**

F57.21**Ajax link with render array**

As an example, we again show the previous function, but this time creating the link as a render array. In this case we use the '#ajax' property to indicate that we are dealing with an Ajax call, which means we do not need to include the class 'use-ajax'.

```
function ajax_forcontu_link () {
  $build['my_div'] = array(
    '#markup' => $output = "<div>Click on the link to obtain the
server time.</div><div id='time'></div>",
  );
  $build['ajax_link'] = array(
    '#type' => 'link',
    '#title' => t('Click here'),
    '#href' => 'ajax_forcontu_link_callback/nojs/',
    '#id' => 'ajax_link',
    '#ajax' => array(
      'wrapper' => 'time',
      'method' => 'html',
    ),
  );
  return $build;
}
```

The '#ajax' property of the link within the render array is enough to indicate that the link will act as an Ajax call, which means that including the class 'use-ajax' is not necessary.

Ajax in forms

In the module **Ajax Example** (`ajax_example`), included in **Examples for Developers**, you will find several examples of forms that use Ajax.

In this unit we will use the example '`ajax_example_dependent_dropdown`' to show how Ajax is used in forms. This is one typical example in which two **selection lists which depend on one another** are displayed. The option list of the second list will depend on what option was selected in the first.

In this concrete example, in the first list the instrument type is selected (string, wind, percussion, etc.). The second list will be reloaded with the instruments of the type selected in the first list. [F57.22](#)

Dependent dropdown

Instrument Type

Brass 

Brass Instruments

Trumpet 

- Trumpet
- Trombone
- French Horn
- Euphonium

[F57.22](#)

Example: Ajax in forms

Example of applying Ajax to forms. The drop down lists are related, such that the second list will load a set of instruments, obtained from the server, as a function of the selection made in the first list.

The steps for creating, validating, and sending the form have already been studied in [Unit 47](#), so in this unit we will only explain the differences that Ajax brings to the form.

We begin by registering the URL where we will display the form. In this case it is not necessary to register a URL for the Ajax function, as we would need do in the case of an Ajax link. [F57.23](#)

```
/***
 * Implements hook_menu().
 */
function ajax_example_menu() {
  $items = array();

  // Reloads a list as a function of a selection.
  $items['examples/ajax_example/dependent_dropdown'] = array(
    'title' => 'Dependent dropdown',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('ajax_example_dependent_dropdown'),
    'access callback' => TRUE,
    'weight' => 4,
  );

  // ...
  return $items;
}
```

[F57.23](#)

Registering the URL

We begin by registering the URL responsible for displaying the form.

In the function which defines the form we will add two select elements: [F57.24](#)

- '**dropdown_first**'. Shows a first list of elements, obtained by `_ajax_example_get_first_dropdown_options()` function. In this example it corresponds to the list of types of musical instruments. This element is what is responsible for making the Ajax call, by using the

'#ajax' property.

In the array '#ajax' we have defined three values:

- 'event'. Event which fires off the call to the Ajax function. In a select element the default event is 'change'. Since this is the event we are going to use, it is not necessary to define the 'event' field.
 - 'callback'. Return function that the Ajax call will perform.
 - 'wrapper'. Identifier (ID) of the element which acts as the container for the Ajax content.
- 'dropdown_second'. Shows a dynamic options list based on the value selected in the first list (\$select). The function _ajax_example_get_second_dropdown_options() will be responsible for returning the option list as a function of the \$select parameter, which includes the value of the option selected in the first list.

By means of the '#prefix' and '#suffix' attributes, we enclose the HTML of the field in a <div> tag, in which the ID se corresponds to the value indicated in the 'wrapper' field of the previous '#ajax' attribute.

F57.24

Form Function

Function which implements the form. The first list of the form will be what is responsible for activating the Ajax call with its 'change' event. This happens when the selected element changes.

```
function ajax_example_dependent_dropdown($form, &$form_state) {
  // Obtain the list of options for the first select.
  $options_first = _ajax_example_get_first_dropdown_options();

  // in the variable $selected is stored the selected value,
  // which will be passed as a parameter to the second list
  $selected = isset($form_state['values']['dropdown_first']) ?
    $form_state['values']['dropdown_first'] : key($options_first);

  $form['dropdown_first'] = array(
    '#type' => 'select',
    '#title' => 'Instrument Type',
    '#options' => $options_first,
    '#default_value' => $selected,
    // Build the Ajax call, using the '#ajax' property.
    '#ajax' => array(
      // in a select element, the default event
      // is change
      // 'event' => 'change',
      'callback' => 'ajax_example_dependent_dropdown_callback',
      'wrapper' => 'dropdown-second-replace',
    ),
  );

  $form['dropdown_second'] = array(
    '#type' => 'select',
    '#title' => $options_first[$selected] . ' ' . t('Instruments'),
    '#prefix' => '<div id="dropdown-second-replace">',
    '#suffix' => '</div>',
    '#options' => _ajax_example_get_second_dropdown_options($selected),
    '#default_value' => isset($form_state['values']['dropdown_second'])
      ? $form_state['values']['dropdown_second'] : '',
  );
  $form['submit'] = array(
    '#type' => 'submit',
    '#value' => t('Submit'),
  );
  return $form;
}
```

In the Ajax return function it will only be necessary to return the field that corresponds to the second element (\$form['dropdown_second']), so that it can be re-rendered in the page. **F57.25**

```
/** 
 * Ajax Return Function
 */
function ajax_example_dependent_dropdown_callback($form,
$form_state) {
  return $form['dropdown_second'];
}
```

Finally we present the functions that have been used to obtain the values of each list. Although we have used static values, defined directly in an array within the function, the most typical scenario is that the data is obtained from the database at this point. **F57.26**

```
/** 
 * Option list for the first list
 */
function _ajax_example_get_first_dropdown_options() {
  return drupal_map_assoc(array(t('String'), t('Woodwind'),
    t('Brass'), t('Percussion')));
}

/** 
 * Option list for the second list. The array returned
 * depends on the value of $key.
 */
function _ajax_example_get_second_dropdown_options($key = '') {
  $options = array(
    t('String') => drupal_map_assoc(array(t('Violin'), t('Viola'),
      t('Cello'), t('Double Bass'))),
    t('Woodwind') => drupal_map_assoc(array(t('Flute'),
      t('Clarinet'), t('Oboe'), t('Bassoon'))),
    t('Brass') => drupal_map_assoc(array(t('Trumpet'),
      t('Trombone'), t('French Horn'), t('Euphonium'))),
    t('Percussion') => drupal_map_assoc(array(t('Bass Drum'),
      t('Timpani'), t('Snare Drum'), t('Tambourine'))),
  );
  if (isset($options[$key])) {
    return $options[$key];
  }
  else {
    return array();
  }
}
```

F57.25**Ajax Return Function**

In the Ajax return function only the changed form element will be returned, indicating to the system that it should be updated.

F57.26**Additional functions**

Here we see the functions used to return the elements belonging to both lists. The first function is "static" and is only executed once, when the page is loaded. The second function will be called each time that we change the selection made in the first list, obtaining the information from the server.

Although in this example the list contents are obtained directly from the array, we could instead make a database query within the function in order to obtain the needed data.

The '#ajax' property

We have already seen that a form element can include the '#ajax' property, which adds an event to establish Ajax communication with the server.

http://api.drupal.org/api/drupal/developer!topics!forms_api_reference.html/7#ajax

The '#ajax' property is an array which can have the following values:

- **#ajax['callback']**. Name of the Ajax return function. The value returned by this function can be HTML content, a render array (as in the last example) or an Ajax command array.
- **#ajax['effect']**. Specifies the effect which will be used when the content is added from the Ajax request. By default the value is 'none', while other possible values are 'fade' and 'slide'.

- **#ajax['event']**. Event which will fire off the Ajax request. Any jQuery event can be used, although usually the event is related to the type of form element. Some of the default values include:
 - o 'mousedown', for **submit**, **imagebutton**, and **button** elements.
 - o 'blur', for **textfield** and **textarea** elements.
 - o 'change', for **select** elements.
- **#ajax['keypress']**. If this is set to TRUE, the event defined in #ajax['event'] will also be executed when the element is selected (has the focus) and the ENTER key is pressed.
- **#ajax['method']**. Refers to the method of insertion of the HTML returned by Ajax in the defined container (#ajax['wrapper']). The default value is 'replace', but other values like 'after', 'append', 'before' and 'prepend' are also possible.
- **#ajax['path']**. Path of the Ajax return function. Usually this value is not used, as defining the function in #ajax['callback'] suffices, without having to register it via hook_menu().
- **#ajax['prevent']**. The default value is 'click', in order to avoid a form submission while an Ajax event is underway.
- **#ajax['progress']**. The type of icon or progress bar which is displayed while waiting for the Ajax response.
- **#ajax['trigger_as']**. Indicates which submit element will be used to submit the form when the event is triggered in an element which by default does not make a form submission.
- **#ajax['wrapper']**. Defines the ID of the HTML element which is used to display the content returned by Ajax function. It is important to keep in mind that the entire element is substituted, not just its content, which means that the new content should include the ID value of the element. We will generally use a container element like <div> to display the dynamic content returned by the Ajax function.

As we have already stated, you can find more examples of Ajax usage in , the module **Ajax Example** (ajax_example), included in **Examples for Developers**.

58 Features

The **Features** module allows you to group features and reuse them. Basically, the features are groups or objects that can be installed and turned on and off in a single step.

For example, we can create a feature that contains a content type (with additional fields) and a view related to the type of content. When you install and activate this feature in the site, a file will be created automatically for all of the elements that contain the content type and view.

The use of features is especially useful when we develop several portals that, although different, share certain functionalities; a blog, a forum, a virtual store, and so on. Without features we'll have to install and configure all modules from scratch. However, with the use of features we can add functional groups in a quick and simple form.

Comparative D7/D6

The function of Drupal 7 features is similar to that of Drupal 6, but we must take into account the new units and components compatible with Drupal 7.

Unit contents

58.1 Introduction to Features	408
58.2 Create and install Features.....	409
58.3 Integration with Drush	420
58.4 Additional Modules.....	421

58

58.1 Introduction to Features

The **Features** module allows you to group features and reuse them. Basically, the features are groups or objects that can be installed and turned on and off in a single step.

For example, we can create a feature that contains a content type (with additional fields) and a view related to the type of content. When you install and activate this feature in the site, a file will be created automatically for all of the elements that it contains including the content type and view.

It is important to know that the features are used to export and import the structure of these objects but not the content. Therefore we can not include the nodes, users, etc. in the feature.

The **Features** module is available at:

<http://drupal.org/project/features>

The use of features is especially useful when we develop several portals that, although different, share certain functionalities; a blog, a forum, a virtual store, and so on. Without features we'll have to install and configure all modules from scratch. However, with the use of features we can add functional groups in a quick and simple form.

Although its name is sometimes translated as Characteristics, to avoid confusion, we will use the original name of the module, Features.

Compatibility

We can only add elements to a feature in compatible modules. Some of the elements that we can include are:

- Views
- Rules
- Context
- Content types
- Definition of fields
- Dependencies with other modules
- Menus and menu links
- Image Styles
- Text Formats
- Permissions and roles
- Taxonomies

Other modules that can be integrated with features are as follows:

Drupal Reset Module

The **Drupal Reset** module removes all the tables and files from the site, reverting to the installation process (install.php). It will be especially useful during the development of **Features** and **Installation profiles** (Unit 59).

The **Drupal Reset** module is available at:

http://drupal.org/project/drupal_reset

Create and install Features

58.2

Once we have installed and enabled the module, we can manage the features from:

Administration ⇒ Structure ⇒ Features

The **Manage** tab shows the available features. To activate one of the available features, we have to select it (selection box) and **Save configuration**. If activation is possible, the status will change to **Enabled**. **F58.1**

FEATURE	SIGNATURE	STATE	ACTIONS
Date Migration Example Examples of migrating with the Date module Umet dependencies: migrate (Missing)	http://drupal.org/7.x-2.7	Disabled	Recreate

URL Features

/admin/structure/features

F58.1

Manage Features

The Manage tab shows the features available.

The features are installed using the same procedure as the modules, and will be listed in the administration area for the modules, within the group features. We can activate a feature from both the administration area for the modules and from the administration area for the features.

In both cases, the feature may not be activated if you do not meet all of its dependencies. **F58.2**

ENABLED	NAME	VERSION	DESCRIPTION	OPERATIONS
<input type="checkbox"/>	Date Migration Example	7.x-2.7	Requires: Date (enabled), Date API (enabled), Date Repeat API (disabled), Date Repeat Field (disabled), Features (enabled), Migrate (missing)	
<input checked="" type="checkbox"/>	Features	7.x-2.0	Provides feature management for Drupal. Required by: Date Migration Example (disabled)	Help Permissions Configure

F58.2

Installation Features

As with any module, the feature may not be activated if you do not have all the required modules.

Using the **Recreate** link we can edit the Feature, add new elements, and re-download (**Download feature**). The changes will not apply if we do not upload the feature to the server by overwriting the files from the previous version. **F58.3**

F58.3**Recreate Feature**

Editing the Feature.

The screenshot shows the 'Edit' view for a feature named 'Date Migration Example'. The 'GENERAL INFORMATION' section contains fields for 'Name' (Date Migration Example), 'Machine-readable name' (date_migrate_example), 'Description' (Examples of migrating with the Date module), 'Package' (Features), and 'Version' (7.x-2.7). The 'COMPONENTS' section lists various dependencies and components, each with a 'Select all' checkbox. The components listed include DEPENDENCIES (29), FIELD BASES (8), FIELD INSTANCES (12), FORMATOS DE TEXTO (3), IDIOMAS (2), MENU LINKS (70), MENÚS (5), PERMISOS (77), ROLES (2), TAXONOMÍA (2), and TIPOS DE CONTENIDO (2). A legend at the bottom right indicates 'Normal', 'Changed', and 'Conflict' status.

Create a Feature

Using the **Create Feature** tab we started to create a new feature. The fields that we will have to complete are: **F58.4**

- **Name and Machine name.**
- **Description.** Short Description to help users to know what features and items will be available to enable the feature.
- **Version.** You can include the version number (7.x-1.0).
- **URL of update XML.** Allows you to indicate the URL where you can control the available versions of the feature. If you do not use this option, you will leave the field empty.
- **Edit components.** List of components that can be incorporated into the feature. The list of components will depend on the module that is compatible with features installed on the site.

Home > Administration > Structure > Features

MANAGE **CREATE FEATURE** **SETTINGS**

GENERAL INFORMATION

Name
List of events
Machine name: list_of_events [Edit]
Example: Image gallery (Do not begin name with numbers.)

Description
List of events using custom content type and views
Provide a short description of what users should expect when they enable your feature.

Package
Features
Organize your features in groups.

Version
7.x-1.0
Examples: 7.x-1.0, 7.x-1.0-beta1

ADVANCED OPTIONS
Download feature

COMPONENTS
Expand each component section and select which items should be included in this feature export.

Search Clear Select all

- ▶ **CONTENT TYPES (2) (node)**
- ▶ **DEPENDENCIES (32) (dependencies)**
- ▶ **FIELD BASES (3) (field_base)**
- ▶ **FIELD INSTANCES (7) (field_instance)**
- ▶ **LANGUAGES (1) (language)**
- ▶ **MENU LINKS (70) (menu_links)**
- ▶ **MENUS (5) (menu_custom)**
- ▶ **PERMISSIONS (77) (user_permission)**
- ▶ **ROLES (1) (user_role)**
- ▶ **TAXONOMY (1) (taxonomy)**
- ▶ **TEXT FORMATS (3) (filter)**

LEGEND
 Normal
 Changed
 Auto detected
 Conflict

F58.4**Create a Feature**

To create a new feature we must indicate the name, system name, description and version.

Add components

As discussed in the previous paragraph, a feature may include different types of elements: Views, Rules, Rules Contexts, Content Types, Definition of fields, Dependencies on other modules, Menus and menu links, Image styles, Text formats, Permissions and roles, Taxonomies, etc.

The list of components will depend on the modules compatible with features installed on the site. **F58.5**

Editar componentes

Tipos de contenido: node

Campos: field
Dependencias: dependencies
Enlaces de menú: menu_links
Estilos de imagen: image
Formatos de texto: filter
Menús: menu_custom
Permisos: user_permission
Roles: user_role
Taxonomía: taxonomy

Tipos de contenido: node

Views: views_view

F58.5**Edit components**

To add components which we will include in the feature. The components must have been previously created on the site.

For example, to select **Content types**, we can add any types of content available on the site to the feature regardless of whether these have been added by the core, other modules, or if we have manually created them.

The block on the right shows all the elements added to the feature. For example, when adding a content type, additional fields are also added.

Also, the system automatically detects the required modules and they are added to the list of dependencies. **F58.6**

F58.6

Add content type

To the right show all added components.

We can also add dependencies manually from the component, **F58.7**

F58.7

Dependencies

To add components including the dependencies with other modules. We can also add additional components.

Download and install a Feature

When you have added all of the components of the feature, you can download it by clicking on **Download feature**. **F58.8**

F58.8

Download Feature

Once we have added components, we will download the Feature from Download Feature.

The downloaded file will have the name *name-versión.tar*. For example:

`list_of_events-7.x-1.0.tar`

To install this feature we must decompress the file and upload the content in the modules folder. It is recommended to save all the features in a common folder. For example:

`/sites/all/modules/custom/features`

The files generated will depend on the components included in the feature:

- list_of_events.module
- list_of_events.info
- list_of_events.features.inc
- list_of_events.features.field.inc

As you can see, the feature includes the files .module and .info because all the effects are treated as a module.

Once it is uploaded, we can enable the feature from the administration area of modules or features. As is the case with the modules, in order to activate the feature, all the required modules should be available.

Practical Case 58.1 Create a Feature

In this Practical Case study we will create a feature called **Events**, where we will define:

- A content type, **Event**, with the fields:
 - o **Title**.
 - o **Description**. Text field.
 - o **Event type**. Another phrase for taxonomy.
 - o **Image**. Image file type..
 - o **Date**. Date Field of the event. We will use only the start date of the event.
 - o **Highlight**. Boolean field (YES/NO).
- Two new **image styles**:
 - o **image_event_node**. Style used to display the node.
 - o **image_event_lists**. Style used to display content lists in view.
- To view upcoming events (`proximos_eventos`) showing the list of events:
 - o Page. Displays all the events.
 - o Block. Displays only the salient events.
- A **Highlight event** rule, that will be executed when you create or update an event. The rule will check the start date of the event, so that if it will occur in less than 2 days, it will be marked as a prominent.
- An **event manager role** with permissions to edit the contents of the event type.

To test the functioning of features, we will reinstall Drupal once they have been created and export the feature. We'll do this case study in a **new subdomain**, installing Drupal from scratch.

Step 1. Initial Installation

First, we must perform a new Drupal installation by following these steps:

- Install Drupal.
- Install and activate the required modules: Chaos tools, Date (Date, Date API and Date Popup), Drupal Reset, Entity (Entity API, Entity token), Features, Rules (Rules, Rules UI) and Views (Views, Views UI).

Step 2. Creation of the content type Event

We will create the content type Event from

Administration ⇒ Structure ⇒ Content types ⇒ Add content type

We will add the following fields. You can apply the additional configuration to each field based on what you think it should be, while respecting the minimum specified below:

- **Event type (field_event_type)**. First we will have to create a new field called Event type. Once created, we will return to field management to add the field **Term reference**. We will use the control **Check boxes/radio buttons**, to select multiple items.
- **Image (field_event_image)**. You can only upload an image. In the next step we will modify the image style.
- **Date (field_event_date)**. As a control, we will use Pop-up calendar. You can select the start date of the event (year, month, and day).
- **Destacado (field_event_highlight)**. Will be a Boolean field, with **Check boxes/radio buttons** (values NO and YES disabled and enabled respectively). The default value will be disabled (NO).

Step 3. Creation of image styles

We will create two image styles:

- **image_event_node**. **Scale** effect with width 400px, enabling you to extend the image.
- **image_event_lists**. **Scale and cut** effect (100x100px).

When we return to edit the **Event** content type, we will modify the presentation of the Image field to use the defined image style, **image_event_node**.

Step 4. Creation of the field Upcoming Events

We will create the field **Upcoming Events**, with the name **upcoming_events** (it edits and corrects the system name). The field will show the contents of the Event type. We will create two presentations:

- **Page** (titled **Upcoming events**), with the URL **upcoming-events**. The events will be displayed in list format HTML. Ten items per page will be displayed. The fields to be displayed are; Title (bound to the node) and date.
- **Block** (titled **Featured events**). The events will be displayed in HTML list format, with the same fields and presentation as the type page. Five elements per page will be displayed. In the block we will filter the contents and show only the upcoming events.

To test the fields created, we can create contents for the Event. Keep in mind that the contents will not be exported along with the feature, so that will be lost when reinstalling the site.

Step 5. Creation of the rule

We will create the rule **Highlight event (highlight_event)**, along with **Before saving content**. We will see that the content of this event type and the event will be displayed two days before. If the conditions are fulfilled, main content field will be set to YES.

F58.9

The screenshot shows the configuration of a reaction rule named "Highlight event".

- Events:** A table with one row: "Before saving content of type Event".
- Conditions:** A table with one row: "Data comparison" (operator: lower than, value: +2 days).
- Actions:** A table with one row: "Set a data value" (value: Yes).
- SETTINGS:** A section with a "Save changes" button.

F58.9

Practical case 58.1 Create rule

An example rule to highlight an event.

Step 6. Create the roles and permissions

Next, we will create the role of **events manager** and we will assign the permissions needed to manage the content of **Event type**:

- Event: Create new content
- Event: Edit own content
- Event: Edit any content
- Event: Delete own content
- Event: Delete any content

Step 7. Create the Feature

Once we have created all these elements, the next step is to create the Feature and add the components.

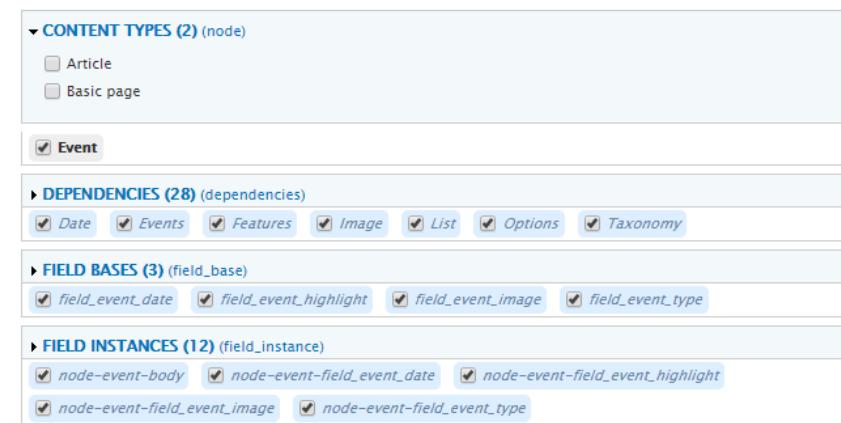
URL Create Feature
</admin/structure/features/create>

Administration ⇒ Structure ⇒ Features [Create Feature]

- **Name:** Events
- **Description:** Event content type and other associated items.
- **Version:** 7.x-1.0

Using **Edit components** we will be selecting the previously created elements:

- **Content types: node.** We will select **Event**. They will automatically be added to the fields created for the event. **F58.10**



The screenshot shows the 'Edit components' section of the Feature creation page. It includes three expandable sections: 'CONTENT TYPES (2) (node)', 'DEPENDENCIES (28) (dependencies)', and 'FIELD BASES (3) (field_base)'. Under 'CONTENT TYPES', 'Event' is selected. Under 'DEPENDENCIES', several modules are checked: Date, Events, Features, Image, List, Options, and Taxonomy. Under 'FIELD BASES', fields like field_event_date, field_event_highlight, field_event_image, and field_event_type are selected. Under 'FIELD INSTANCES (12) (field_instance)', fields like node-event-body, node-event-field_event_date, node-event-field_event_highlight, node-event-field_event_image, and node-event-field_event_type are selected.

- **Image Styles:** We will select **image_event_lists** and **image_event_node**. **F58.11**



The screenshot shows the 'Image Styles' section of the Feature creation page. It displays two checked options: 'image_event_lists' and 'image_event_node'.

- **Views: views_view.** We will select the field **Upcoming events**. This includes all the events we have created in the field. **F58.12**

A screenshot of the Views configuration interface. It shows a list of views with 'Upcoming events (upcoming_events)' selected. The interface includes tabs for 'FIELDS' and 'FIELDS & FILTERS'.

F58.12**Practical Case 58.1****Add views**

We add the view Upcoming events.

- **Configuration of Rules: rules_config.** We will select the rule **Destacar Evento**. **F58.13**

A screenshot of the Rules Configuration interface. It shows a list of rules with 'Highlight event' selected. The interface includes tabs for 'RULES' and 'RULES & CONDITIONS'.

F58.13**Practical Case 58.1****Add rules**

We add the rule Upcoming events.

- **Roles: user_role.** We will select **event manager**. **F58.14**

A screenshot of the Roles configuration interface. It shows a list of roles with 'event manager' selected. The interface includes tabs for 'ROLES' and 'ROLES & PERMISSIONS'.

F58.14**Practical Case 58.1****Añadir roles**

We add the role Event Manager.

- **Dependencies.** In addition to the dependencies automatically detected, we can add other additional dependencies, for example, Rules UI and Views UI.

Once you have added all the components, we generated in the Feature click on **Download feature**.

The system generates a compressed file **events-7.x-1.0.tar**, containing the following files within the folder **events**:

- events.features.field.inc
- events.features.inc
- events.features.taxonomy.inc
- events.features.user_role.inc
- events.info
- events.module
- events.rules_defaults.inc
- events.views_default.inc

Keep in mind that the feature is not saved in the Features administration area. At this time it will only be available downloaded files.

Step 8. Reuse the feature

To check the functioning of the Feature created, we need apply it to a site with the same components. Therefore we can install it on another Drupal web site or, as we will do in this case study, reinstall the site.

To reinstall the site we will use the **Drupal Reset** module, which will delete all the tables and files on the site, going back to the installation process (install.php). **Don't use Drupal Reset without making a backup of your site, both the database and the file structure.**

When reinstalling the site, we will lose the Event content, the associated images, the image styles, etc.

To install the **feature Events**, first we upload the module folder to the folder **/sites/all/modules**. It is recommended that you upload all the features to a common folder different from the rest of modules. For example:

- /sites/all/modules/**custom/features**

In this way, the **Events** feature will be available in:

- /sites/all/modules/**custom/features/events**

Now we can enable the feature from the module administration area or from the features administration area by first activating the features module. We'll use the second option.

The **Events** feature can only be activated if it is available in all of the modules it depends on (even though they are not activated). **Save settings**, in the module will enable the required modules. **F58.15**

F58.15

Practical Case 58.1

Reuse Feature

Once again we upload the feature, Events, which will be available in the features administration area.

The screenshot shows the 'Features' administration page. On the left, there's a sidebar with 'Migrate Examples'. The main area has a table with two rows. The first row is for 'Date Migration Example', which is disabled. The second row is for 'Events', which is overridden. A 'Save settings' button is at the bottom.

FEATURE	SIGNATURE	STATE	ACTIONS
Date Migration Example	http://drupal.org/7.x-2.7	Disabled	Recreate
Events	Event content type and other related elements	Unavailable 7.x-1.0	Overridden Recreate

By clicking on the state of the Feature, we will be able to see the detail of dependencies and plugins. In this column some of them can appear **Overwritten**. This means that the elements have been modified through the administration area. If we want to return them to their original state, with the Feature configuration set, we will select the components and we will click (**Revert components**). **F58.16**

Home > Administration > Structure > Features

Features VIEW RECREATE

GENERAL INFORMATION

Name
Events
Example: Image gallery (Do not begin name with numbers.)

Machine-readable name *
events
Example: image_gallery
May only contain lowercase letters, numbers and underscores. Try to avoid conflicts with the names of existing Drupal projects.

Description
Event content type and other related elements
Provide a short description of what users should expect when they enable your feature.

Package
Features
Organize your features in groups.

Version
7.x-1.0
Examples: 7.x-1.0, 7.x-1.0-beta1

COMPONENTS
Expand each component section and select which items should be included in this feature export.

Search Clear Select all

- ▶ **CONTENT TYPES (2) (node)**
 Event
- ▶ **DEPENDENCIES (26) (dependencies)**
 Events Views Chaos tools Date Features
 Image List Options Taxonomy
- ▶ **FIELD BASES (3) (field_base)**
 field_event_type field_event_date field_event_image
 field_event_highlight
- ▶ **FIELD INSTANCES (12) (field_instance)**
 node-event-body node-event-field_event_date
 node-event-field_event_highlight node-event-field_event_image
 node-event-field_event_type
- ▶ **IMAGE STYLES (0) (image)**
 image_event_lists image_event_node
- ▶ **LANGUAGES (1) (language)**
- ▶ **MENU LINKS (93) (menu_links)**
- ▶ **MENUS (6) (menu_custom)**
- ▶ **PERMISSIONS (85) (user_permission)**
- ▶ **ROLES (1) (user_role)**
 event manager
- ▶ **TAXONOMY (1) (taxonomy)**
 Event type
- ▶ **TEXT FORMATS (3) (filter)**
- ▶ **VIEWS (0) (views_view)**
 Upcoming events (upcoming_events)

LEGEND
 Normal Changed Auto detected Conflict

F58.16**Practical Case 58.1****View Feature**

Detail of the Events Feature, with it's dependencies and components.

58.3 Integration with Drush

Drush Commands

Features includes Drush commands to perform operations similar to those made through the user interface. Some of the commands listed are:

- **features.** Includes a list of all the available features and their state.
Syntax: 'drush features'
- **features-export.** Generates the code for a new feature to the list of components.
Syntax: 'drush features-export [name feature] [components list]'
- **features-update.** Updates the code for a feature by adding the changes that have been made to it.
Syntax: 'drush features-update [name feature]'
- **features-revert.** Reverts the components of a feature to its original state, depending on the configuration established in the module code.
Syntax: 'drush features-revert [name feature]'
- **features-diff.** Shows the differences between the components of the feature in the database and their original values in the code module. For this, it uses the **Diff module**.
Syntax: 'drush features-diff [name feature]'

Additional Modules

58.4

The following modules implement features that can be added to the site by following the procedures in this unit.

Featured news feature module

The module **featured news feature** is available at:

http://drupal.org/project/featured_news_feature

It adds a **Feature** that includes the following components: **F58.17**

- A new type of **content** with an associated image. Designed to add news.
- A **view** that displays the list of news published. Highlights are shown with their image in the header of the view. This view will be available at the URL /news.

News

Featured news



Britain's Andy Murray 'serious' about tennis strike threat



Dutch tulips impounded by Romania



Welsh mine rescue: One of four at Gleision mine dead

F58.17

Featured News module feature

The module adds a Feature with a new content type and a view that displays a list of news.

Other news

[Italy's sovereign debt rating cut by S&P on growth fear](#)

Tuesday, September 20, 2011 - 09:34

S&P cut its rating by one level to A from A+, adding that the outlook for the country was "negative".

It cited fears over Italy's ability to cut state spending and bring its...

[Dinosaur feather evolution trapped in Canadian amber](#)

Friday, September 16, 2011 - 13:39

Eleven fragments show the progression from hair-like "filaments" to doubly-branched feathers of modern birds. The analysis of the 80-million-year-old...

[Argentina marks 'Night of the Pencils'](#)

Friday, September 16, 2011 - 13:38

Thirty-five years ago, one of most notorious episodes of abuse committed during military rule in Argentina took place - the abduction of 10 students by security forces in the city of La Plata near...

[S Korea holds North defector 'in poison-needle plot'](#)

Friday, September 16, 2011 - 13:37

Mr Park is an anti-Pyongyang activist involved in sending propaganda leaflets to the North. Named only as An, the arrested man is reported to be a former commando in his 40s who defected to the...

Events calendar feature module

The module **Events calendar feature** is available at:

http://drupal.org/project/events_calendar_feature

It adds a **Feature** that includes the following components: **F58.18**

- A **Event content type** with a date field that includes the start and end date of the event.
- Create a field (URL /events) that lists the events. The field allows you to set the parameters of a particular date.
- Create an **Events calendar** field with a block that shows the calendar of events.

F58.18

Calendar Events module feature

The module adds a Feature with a Event content type and a view that shows the events. Also a general calendar block.

The screenshot shows a calendar interface for August. Below it, two event details are listed: 'Event 3' and 'Event 2'. Each event has a timestamp, author information, and a 'Read more' link.

Event	Date	Author	Actions
Event 3	Tue, 08/16/2011 - 11:43	Submitted by admin	Read more Log in or register
Event 2	Tue, 08/16/2011 - 11:43	Submitted by admin	body

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Enterprise Blog module

The **Enterprise Blog** module adds a set of functionalities to create a complete system of blogs. The module is available at: **F58.19**

http://drupal.org/project/enterprise_blog

Among the additional modules required, we found:

- http://drupal.org/project/enterprise_base
- <http://drupal.org/project/realname>
- http://drupal.org/project/save_draft
- <http://drupal.org/project/scheduler>
- <http://drupal.org/project/strongarm>
- <http://drupal.org/project/token>
- http://drupal.org/project/views_bulk_operations
- http://drupal.org/project/block_views

F58.19

Enterprise Blog module

The module adds Features to create a system of blogs.

The screenshot shows the 'Features' configuration page. It displays a table with one row for the 'Enterprise Blog Content' feature. The table columns are: FEATURE, SIGNATURE, STATE, and ACTIONS. The 'FEATURE' column contains 'Enterprise Blog Content'. The 'SIGNATURE' column contains 'http://drupal.org/7.x-1.0-rc1'. The 'STATE' column is 'Default'. The 'ACTIONS' column contains a 'Recreate' button. At the bottom, there is a 'Save settings' button.

FEATURE	SIGNATURE	STATE	ACTIONS
Enterprise Blog Content	http://drupal.org/7.x-1.0-rc1	Default	Recreate

Once you have installed the **Feature**, we will have a content type available called **Blog Post** (enterprise_blog) to create blog entries, which can be published in the URL **/blog**. **F58.20**

We can also activate the blocks: Blog categories (shows the categories), Blog archive (shows the number of entries per month) and Blog posts (with the latest published entries).

The screenshot shows the Drupal 7 blog module interface. At the top, there are two tabs: 'Blog' and 'Inicio'. Below the tabs, there's a 'Recent Posts' block listing three posts: 'Eum Melior Quibus', 'Augue Eligio Gravis', and 'Cogo Melior'. There's a 'more' link at the bottom of this block. To the right of the posts is a sidebar with a 'Blog' heading, 'Add blog post', and 'Administer blog post' links. Below the sidebar is a 'Blog Archive' block showing a list of months from October 2014 to January 2014, with a 'next' and 'last' link. The main content area displays two blog posts: 'Eum Melior Quibus' and 'Augue Eligio Gravis'. Each post has a small thumbnail image composed of colored squares (blue, green, yellow, pink) and a brief excerpt. A 'Read more' link is located at the end of each post excerpt.

F58.20
/blog
View blog and blocks generated by the module.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

59 Installation profiles

Distributions or installation profiles help to streamline the installation process for web portals, which is very useful for developers. With the distributions, it is possible to define which modules and themes will be activated during the installation process.

It is also possible to configure the modules enabled and modify the variables of Drupal; in addition to working with the functions of the Drupal API and even performing direct queries on the database.

In this unit we will look at how to schedule installation profiles. Also we will describe some of the profiles available for installation on drupal.org.

Comparativa D7/D6

Installation profiles

Drupal 7 has greatly improved the definition of system installation profiles. When incorporated into de file .info, it will no longer be necessary to implement specific functions to define the profile and its dependencies.

It has also simplified the process of implementing additional tasks, so that it is no longer necessary to control each task form within, the next task to be executed.

Unit contents

59.1 What are installation profiles?	426
59.2 Definition of the installation profile	427
59.3 Additional installation tasks	431
59.4 Actions during the profile installation	440
59.5 Contributed distribution profiles	442



59.1 What are installation profiles?

When we have developed several projects with Drupal we realize that there are certain modules and configurations that are always used and that we have to install again and again, so the start of each project is heavy and repetitive. A distribution, also known as an installation profile, is a set of instructions that will allow us to define which modules and options we want to activate during the installation process of Drupal. In this way we can create Web portals without the need to start from scratch, gaining time and productivity.

Like the modules and topics, the installation profiles can also be shared with the community. In this URL you will find installation profiles available at drupal.org:

<http://drupal.org/project/installation+profiles>

Installation profiles allow you to mount thematic sites. For example, we can find specific installation profiles to deploy a digital newspaper, a corporate website, a virtual classroom, an online shop, etc.

Once more we invite you to complement the content of this unit by downloading and analyzing the code of some of these installation profiles.

Installation profiles included in Drupal 7

The first example of installation profiles are found in the distribution of Drupal 7 itself. In the first step of the installation of Drupal, we are prompted to select one of the two installation profiles available; **Standard** or **Minimal**. F59.1

F59.1

Installation Profiles

Includes two Drupal installation profiles: Standard and Minimal.

Select an installation profile



 Standard
 Install with commonly used features pre-configured.

Minimal
 Start with only a few modules enabled.

► Choose profile

- [Choose language](#)
- [Verify requirements](#)
- [Set up database](#)
- [Install profile](#)
- [Configure site](#)
- [Finished](#)

[Save and continue](#)

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
 This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

These installation profiles can be found in the folder **/profiles**, where we also upload new installation profiles.

Drupal Reset module

The **Drupal Reset** module removes all the tables and files of the site back into the installation process (install.php). It will be especially useful during the development of the **installation profiles**.

The **Drupal Reset** module can be found at:

http://drupal.org/project/drupal_reset

Definition of the installation profile

59.2

Where installation profiles are stored

The installation profiles can be found in the file **/profiles**. In the original version of Drupal 7 this folder contains two types of installation profiles, **standard** and **minimal**. It also includes a third installation profile, called **testing**, that has been defined as hidden and will not display during Drupal installation.

When we create our own installation profiles, we will be adding them to the folder **/profiles**. We can also add the installation profiles contributed by the community that are available. During the installation process, if Drupal finds multiple installation profiles available, it will ask us first which of them we want to install.

File Structure of the installation profile

An installation profile typically consists of the following files and folders:

- file **.info**. Definition file of the installation profile. Its content is similar to the file .info for modules and themes.
- file **.profile**. Is the file where you implement the tasks to be carried out during the installation. It is equivalent to the file .module from the modules, and we will be able to use most of the functions available in the API for Drupal.
- file **.install**. Allows you to use **hook_install()** to insert content in the database during installation.
- folder **/modules**. In this folder we can add modules that will be available during installation. Generally we only use this folder to add modules created specifically for the distribution. The rest of modules required will be registered as dependencies, so that will be requested during the installation.
- folder **/themes**. In this folder we can add themes that will be available during installation. Generally we only use this folder to add themes specifically created for the distribution.
- folder **/translations**. In this folder we will add the core translations, necessary for the installation process to perform in another language (for example in Spanish).

File .info

As an example, for this unit we are going to create an installation profile called **Profile Forcontu (profile_forcontu)**. We begin by creating the folder **/profiles/profile_forcontu** and the file **profile_forcontu.info**, in which we will begin defining the characteristics of this installation profile.

In the file .info we will register some parameters defining the installation profile, already studied with the modules: **name**, **description**, **core**, **dependencies (dependencies[])**, and **files (files[])**.

F59.2

profile_forcontu.info

Definition file for the installation profile.

```
name = Profile Forcontu
description = Profile installation example.
core = 7.x
dependencies[] = block
dependencies[] = color
dependencies[] = comment
dependencies[] = contextual
dependencies[] = dashboard
dependencies[] = help
dependencies[] = image
dependencies[] = list
dependencies[] = menu
dependencies[] = number
dependencies[] = options
dependencies[] = path
dependencies[] = taxonomy
dependencies[] = dblog
dependencies[] = search
dependencies[] = shortcut
dependencies[] = toolbar
dependencies[] = overlay
dependencies[] = field_ui
dependencies[] = file
dependencies[] = rdf
dependencies[] = pathauto
dependencies[] = webform
files[] = profile_forcontu.profile
```

Remember that, for correct viewing, files must be encoded in **UTF-8 sin BOM** (Byte-order mark).

To see how the installation works, we can create the file **.profile** (initially empty), upload the profile folder for the server installation (/profiles) and run the installation of Drupal from the browser. As shown in the **Figure F59.3**, Drupal will ask us which installation we want to use.

F59.3

Selection of the installation profile

Once the installation profile is uploaded to the folder /profiles, we can select it in the first step of the Drupal installation process.

Select an installation profile

Standard
Install with commonly used features pre-configured.

Minimal
Start with only a few modules enabled.

▶ Choose profile

Profile Forcontu
Perfil de instalación de ejemplo.

Choose language Verify requirements Set up database Save and continue

Language Selection (optional)

In the **Beginning Level** we study how to perform an installation in other language. For the new installation to be performed in any language, it is necessary to upload the file **.po** corresponding to that language to the folder /translations of the installation profile.

For our example (in Spanish):

- /profiles/profile_forcontu/translations/**es.po**

We can also directly upload the language file core, using the name given by localize.drupal.org.

- /profiles/profile_forcontu/translations/**drupal-7.32.es.po**

By doing this we can continue the installation in the installed language.

F59.4
F59.4

Language Selection

If we added the translation file in the folder translations, we will be able to continue the installation in other languages.

During installation, the translation feature strings of Drupal are not yet available **t()**, but if it is possible to use the function **st()**, that will make direct use of the files **.po**. therefore, the translations that we write in the archives of the installation profile (**.profile**) must be translated with **st()**, including the strings with your translation in the file **es.po**.

<http://api.drupal.org/api/drupal/includes--install.inc/function/st/7>

Required Modules

In the example above we have indicated, through the vector **dependencies[]**, what core modules we want to activate during the installation. These modules are available in the Drupal distribution files, so it is not necessary to download them.

In addition to core modules, we want to install other additional modules, **Webform** and **Pathauto**.

```
dependencies[] = pathauto
dependencies[] = webform
```

These modules are not available within the distribution files, so you will have to be download and upload them to the appropriate folder (for example, /sites/all/modules) during the installation process. The system will inform us of this requirement in the step, **Verify requirements**. F59.5

F59.5

Problem of requirements

The installation cannot continue if the system does not find all the modules required by the Distribution.

Requirements problem



✓ Choose profile
✓ Choose language
▶ Verify requirements

Web server	Apache/2.2.22 (Debian)
PHP	5.3.29-1~dotdeb.0
PHP register globals	Disabled
PHP extensions	Enabled
Database support	Enabled
PHP memory limit	256M
File system	Writable (<i>public</i> download method)
Unicode library	PHP Mbstring Extension
Settings file	The <code>./sites/default/settings.php</code> file exists.
Settings file	The settings file is writable.
✗ Required modules	Required modules not found. The following modules are required but were not found. Move them into the appropriate modules subdirectory, such as <code>sites/all/modules</code> . Missing modules: Pathauto, Webform

Check the error messages and [proceed with the installation](#).

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
 This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

The requested modules must be downloaded and uploaded to the appropriate folders (/sites/all/modules, /sites/default/modules, etc). This is the only way we can overcome the problems with the requirements and continue with the installation of the distribution.

At this point we can ask ourselves if the distribution should already contain the downloaded modules (en /profiles/profile_forcontu/modules). You must keep in mind that Drupal modules are updated continuously. If we include all the modules previously downloaded, it is very possible that the distribution could be outdated. Therefore, it is advised not to include the modules, but download them at the time of installation to ensure that we install the latest versions.

If the distribution is for personal use, as it is for our development projects, it will be easier to keep it up to date, with all the modules you have already downloaded. Note also that the modules that are not updated may also be updated after the installation.

Additional installation tasks

59.3

`hook_install_tasks()`

By implementing `hook_install_tasks()` we will be able to add additional tasks.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_install_tasks/7

The function `hook_install_tasks()` returns an array of additional tasks. Each additional task will have an associated return function, where we program the actions to be implemented.

The additional tasks will be executed, according to the specified order, between the steps **Configure site** and **Complete translations**:

- Choose profile
- Choose language
- Verify requirements
- Set up database
- Install profile
- Configure translations
- Configure site
- **Additional task 1**
- **Additional task 2**
- **Additional task n**
- Finish translations
- Finished

In the array of definition of the additional tasks, the primary key identifies the task and corresponds to the name of the return function . Also, it may include the following fields:

- **'display_name'**. Descriptive name of the task, as it will be displayed in the list of tasks or steps during installation. We will use the `st()` function to allow your translation. If we do not give a value in this field, the task will run when you tell it to, but will not display in the browser.
- **'display'**. Boolean value to indicate whether the task will be displayed (TRUE) or not (false) in the browser. This value takes precedence over 'display_name'. This parameter can be useful when we want to show or not show the task depending on certain conditions.
- **'type'**. The task can be one of these three types:
 - o **'normal'**. Is the general behavior and the default value. The task will have a return function that can return a HTML output.
 - o **'batch'**. The task defines a process for batch execution.
 - o **'form'**. The task will display a form to collect additional data. The actions of the task will run to send the form after it's validation.
- **'run'**. Constant that indicates how the task will run. The most common values are:
 - o `INSTALL_TASK_RUN_IF_NOT_COMPLETED`. Indicates that the

task will run once during the installation process. It is the default value.

- o `INSTALL_TASK_SKIP`. Indicates that the task will not run, passing control to the next task. It can be useful when we want the task to be displayed in the browser but only run under certain conditions.
- **'function'**. Allows you to indicate a different function than the default function that corresponds to the array index. It can be useful when we want to run different functions depending on certain conditions.

In the following example outlines two additional tasks, whose functions we will define later. **F59.6**

- **Additional options.**
- **Send email to admin.**

F59.6

Additional installation tasks

From `hook_install_tasks()` it is possible to define new installation tasks.

```
/***
 * Implements hook_install_tasks().
 */
function profile_forcontu_install_tasks() {
  $tasks = array();

  //Requests additional information through a form
  $tasks['profile_forcontu_settings_form'] = array(
    'display_name' => st('Additional options'),
    'type' => 'form',
  );

  //Sends an email to the admin
  $tasks['profile_forcontu_send_email'] = array(
    'display_name' => st('Send email to admin'),
  );

  return $tasks;
}
```

The new tasks are displayed in the browser during installation. **F59.7**

F59.7

Additional tasks

Additional tasks are displayed in the left column, just after the task of configuring the site.

Choose language

English (built-in)

Spanish (Español)

Save and continue

✓ Choose profile

▶ Choose language

Verify requirements

Set up database

Install profile

Configure site

Additional options

Send email to admin

Finished

Parameter \$install_state

Each function receives a parameter by the array reference **\$install_state**, that provides information on the current status of the installation process.

If the task is 'normal', the declaration of the function only takes the parameter **&\$install_state**, of the form: **F59.8**

```
/***
 * Return function of task 'profile_forcontu_send_email'
 */
function profile_forcontu_send_email(&$install_state) {
  //...
}
```

F59.8

**Return function
'normal'**

Shows the declaration of the function.

If the task is the 'form' type, first declare the parameters of the form: **F59.9**

```
/***
 * Form of task 'profile_forcontu_settings_form'
 */
function profile_forcontu_settings_form($form, &$form_state, &$install_state) {
  //...
}
```

F59.9

**Return function
'form'**

Shows the declaration of the function.

Referring to the function **install_state_defaults()** we can learn about the elements of the array **\$install_state**:

http://api.drupal.org/api/drupal/includes--install.core.inc/function/install_state_defaults/7

- **'active_task'**. The current task.
- **'completed_task'**. The last task completed.
- **'database_tables_exist'**. Once Drupal is installed the value will be TRUE.
- **'forms'**. Array of forms for different tasks. The key of the array corresponds to the identification of the task.
- **'installation_finished'**. When the installation is finished the value will be TRUE.
- **'interactive'**. Indicates that the installation is interactive. By default, the value is FALSE.
- **'locales'**. Array with the available languages for the installation.
- **'parameters'**. Array with parameters for the installation.
- **'parameters_changed'**. Indicates that the parameters have changed.
- **'profile_info'**. Information profile of the selected installation.
- **'profiles'**. Array with the available installation profiles.
- **'server'**. Array of server variables, which will be replaced in the array `$_SERVER`.
- **'settings_verified'**. The value will be TRUE when you check that the connection to the database is correct.
- **'stop_page_request'**. By setting the value to TRUE the page finishes loading.
- **'task_not_complete'**. By setting the value to TRUE we are indicating that the task must be executed again.
- **'tasks_performed'**. List of tasks that have been executed.

For the task to function you can consult the array **\$install_state** to make decisions based on any of their fields, or you can perform modifications on the array. The changes in **\$install_state** will be available in the next running tasks.

Return Function 'normal'

By executing the task '**profile_forcontu_send_email**', you will send an email to the site developer. If the task has been defined as 'type' => '**normal**', it will be sent to your return function, which corresponds to the identification of the task, **profile_forcontu_send_email()**. **F59.10**

F59.10

Return function 'normal'

Example of a function for a task of 'normal' type. The function sends an e-mail to the site administrator, so it also requires the implementation of `hook_mail()`.

```
/**
 * Return function of task 'profile_forcontu_send_email'
 */
function profile_forcontu_send_email(&$install_state) {
    //Title
    drupal_set_title(st('Send email to admin'));

    //Site administration email
    $site_mail = variable_get('site_mail');
    $to = $site_mail;
    $from = $site_mail;

    global $language;
    $params = array();

    drupal_mail('profile_forcontu', 'email_admin', $to, $language,
    $params, $from);

    $output = '<p>' . st('An email to the author of the distribution
has been sent (%email).', array('%email' => $site_mail)) . '</p>';

    // Link to the next step of the installation process
    $output .= '<p><a href="'. url('install.php', array('query' =>
$install_state['parameters'])) .'>'. st('Click here to continue')
.'</a></p>';

    return $output;
}

/**
 * Implements hook_mail().
 */
function profile_forcontu_mail($key, &$message, $params) {
    $language = $message['language'];

    $variables = array(
        '!site-name' => variable_get('site_name', 'Drupal'),
    );

    switch ($key) {
        case 'email_admin':
            $message['subject'] = t('Site installed: !site-name',
                $variables, array('langcode' =>
$language->language));
            $message['body'][] = t("The site !site-name has been
successfully installed.", $variables, array('langcode' =>
$language->language));
            break;
    }
}
```

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

As we studied in **Unit 52**, to send an email first, we choose `hook_mail()`, and then we specialize and send the email with the `drupal_mail()` function.

We have also used the function `drupal_set_title()`, that allows you to generate a title for the current page.

http://api.drupal.org/api/drupal/includes!bootstrap.inc/function/drupal_set_title/7

Finally, we have generated a link to the next installation step using as base URL **install.php**. And the parameters of the installation state, available in **\$install_state['parameters']**.

Figure F59.11 shows the message displayed when the task runs, including the link that will allow us to move forward in the installation process.

Send email to admin

An email to the author of the distribution has been sent (fran.gil@forcontu.com).
[Click here to continue](#)

- ✓ Choose profile
- ✓ Choose language
- ✓ Verify requirements
- ✓ Set up database
- ✓ Install profile
- ✓ Configure site
- ▶ Send email to author

Finished

F59.11

Task: Send email to admin

The task sends an email to the administrator and displays a message. It also adds a link that will take us to the next step of the installation.

Return Function 'form'

The task '**profile_forcontu_settings_form**' had been defined as 'type' => '**form**', So the return to function must correspond with a form function. As in the previous case, the function will have the name the task identifier, **profile_forcontu_settings_form()**.

Along with the main function of the form, we must also implement the function of the form when it is sent, **_submit()** and optionally, the validation function, **_validate()**. F59.12

```
/*
 * Form of task 'profile_forcontu_settings_form'
 */
function profile_forcontu_settings_form($form, &$form_state, &$install_state) {

  // Sample form with required text field.
  $form['example_text'] = array(
    '#type' => 'textfield',
    '#title' => st('Testing text'),
    '#default_value' => '',
    '#size' => 45,
    '#maxlength' => 45,
    '#required' => TRUE,
    '#description' => st('Example of using forms in installation tasks. This value is stored in a variable in the database.'),
  );

  $form['continue'] = array(
    '#type' => 'submit',
    '#value' => st('Continue'),
  );
}
```

F59.12

Return Function 'form'

The function associated with the task type 'form' generates a form, with a submit button.

We must also implement the send function and validation of the form.

```

        return $form;
    }

/**
 * Submit function
 */
function profile_forcontu_settings_form_submit($form,
&$form_state) {
    //if you need Access to $install_state declare it as global.
    global $install_state;

    // stores the value of a form field
    // in a variable of the database
    variable_set('profile_forcontu_example_text',
$form_state['values']['example_text']);
}

/**
 * Validation function
 */
function profile_forcontu_settings_form_validate($form,
&$form_state) {
    // Additional validation
}

```

Figure F59.13 shows the form for this step. The form makes the value of a text field mandatory. The value entered will be stored in a variable database, from the send function (submit) of the form.

F59.13

Task with form

The Figure shows the form generated by the return task function. When you click continue, the content introduced will be saved and it will continue with the next task.

hook_install_tasks_alter()

The list of installation tasks added by the system can be found in the function **install_tasks()**, in the file, **install.core.inc**.

http://api.drupal.org/api/drupal/includes!install.core.inc/function/install_tasks/7

The tasks that are running by default during the installation are;

- '**install_select_profile**'. Choose profile.
- '**install_select_locale**'. Choose a language.
- '**install_load_profile**'. Task not visible.
- '**install_verify_requirements**'. Check requirements.
- '**install_settings_form**'. Configure database.
- '**install_system_module**'. Task not visible.
- '**install_bootstrap_full**'. Task not visible.
- '**install_profile_modules**'. Install profile.
- '**install_import_locales**'. Configure translations.
- '**install_configure_form**'. Configure site.
- Tasks added by the installation profile.
- '**install_import_locales_remaining**'. Complete translations.
- '**install_finished**'. Finished.

The function **hook_install_tasks_alter()** permits you to modify an array of installation tasks (\$tasks), including the tasks added by the system.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_install_tasks_alter/7

In the following example, the installation profile eliminates some of the tasks performed by the system by default, so that they will not be implemented during the installation of Drupal.

```
/**
 * Implements hook_install_tasks_alter().
 */
function demo_profile_install_tasks_alter(&$tasks, &$install_state) {
  // Stores some system tasks
  $install_bootstrap_full = (array) $tasks['install_bootstrap_full'];
  $install_finished = (array) $tasks['install_finished'];

  // Deletes these tasks
  unset(
    $tasks['install_system_module'],
    $tasks['install_bootstrap_full'],
    $tasks['install_profile_modules'],
    $tasks['install_import_locales'],
    $tasks['install_configure_form'],
    $tasks['install_import_locales_remaining'],
    $tasks['install_finished']
  );
  // Adds a custom task
  $tasks['demo_profile_form'] = array(
    'display_name' => st('Choose snapshot'),
    'type' => 'form',
    'run' => INSTALL_TASK_RUN_IF_NOT_COMPLETED,
  );
  // Recovers the following tasks
  $tasks['install_bootstrap_full'] = $install_bootstrap_full;
  $tasks['install_finished'] = $install_finished;
}
```

F59.14

hook_install_tasks_alter()

Allows you to modify the tasks to be performed during the installation of the profile.

In this example the following tasks will be executed;

- 'install_select_profile'.
- 'install_select_locale'.
- 'install_load_profile'.
- 'install_verify_requirements'.
- 'install_settings_form'.
- 'demo_profile_form'.
- 'install_bootstrap_full'.
- 'install_finished'.

We must keep in mind that the call to **hook_install_tasks_alter()** is performed once you have selected the installation profile, so that the first task will always run('install_select_profile').

Modify the configuration form of the site

Implementing **hook_form_alter()** we can modify any form of the site ([section 47.5](#)). During the installation process of the profile, we can use **hook_form_alter()** to interact with the configuration form on the site, **\$form_id** es '**install_configure_form**'.

In the following example we made the following changes on the fields of the configuration form on the site: [F59.15](#)

- The name of the admin user will have the value 'admin'. Also, it may not be modified (#disabled = TRUE).
- It sets the default value of the time zone, and is not displayed to the user (#access = FALSE).
- Spain is set as default country, but you are allowed to select another country.

F59.15

Configuration Form

We can modify the configuration form of the site implementing **hook_form_alter()**, which allows you to interact with any form of the site.

```
/**
 * Implements hook_form_alter().
 */
function profile_forcontu_form_alter(&$form, $form_state,
$form_id) {
  if ($form_id == 'install_configure_form') {
    // The administrator user will be 'admin' and can't be changed
    $form['admin_account']['account']['name']['#value'] = 'admin';
    $form['admin_account']['account']['name']['#disabled'] = TRUE;

    // The default timezone is set to Europe/Madrid
    // and won't be shown to the user
    $form['server_settings']['date_default_timezone']['#access'] = FALSE;
    $form['server_settings']['date_default_timezone']['#value'] = 'Europe/Madrid';

    // The default country is Spain but can be modified
    $form['server_settings']['site_default_country']['#default_value'] = 'ES';
  }
}
```

Figure F59.16 shows the configuration form of the site with the changes.

Configure site



- ✓ Choose profile
- ✓ Choose language
- ✓ Verify requirements
- ✓ Set up database
- ✓ Install profile
- ▶ Configure site

[Additional options](#)

[Send email to author](#)

[Finished](#)

F59.16**Modified configuration form**

Shows the changes that have been made in the site configuration form.

SITE INFORMATION

Site name *

Site e-mail address *

Automated e-mails, such as registration information, will be sent from this address. Use an address ending in your site's domain to help prevent these e-mails from being flagged as spam.

SITE MAINTENANCE ACCOUNT

Username *

 admin
Spaces are allowed; punctuation is not allowed except for periods, hyphens, and underscores.

E-mail address *

Password *

 Password strength:

Confirm password *

SERVER SETTINGS

Default country

Spain

▼

Select the default country for the site.

59.4

Actions during the profile installation

Now let's look at some actions that we can perform during the installation of this distribution.

Translation Strings

As we have already discussed, during the profile installation, we will use for the translation strings **st()**, to replace **t()**.

<http://api.drupal.org/api/drupal/includes--install.inc/function/st/7>

If we are not sure if a translation can be done with **t()**, we can use the function **get_t()**, that returns the name of the translation function that this available, giving preference to the function **t()**.

http://api.drupal.org/api/drupal/includes--bootstrap.inc/function/get_t/7

F59.17

Function get_t()

```
$t = get_t();
$translated = $t('translate this');
```

hook_install()

In the file **.install()** we can implement **hook_install()**, and within this function we can perform any of the operations studied during the course;

- Create text formats.
- Select the default theme of the site.
- Select the topic of administration.
- Activate blocks by assigning them to specific regions.
- Create content types and fields.
- Assign values to variables or create new variables.
- Insert or update values in the database.
- etc.

In the file **.install** we can also use the function **st()** to translate strings.

A good example can be found in the installation profile **standard**, included with the Drupal distribution.

When defining new installation profiles we should take into account that many elements are created from the **standard_install()** function. For example, the text formats Full HTML and Filtered HTML you create at this point, so if you don't have to include this code in your profile. These text formats are not created on the site.

In order to avoid replicating the code included in the **standard_install()**, it is recommended that you use **hook_install()** from our profile, and part of the standard profile: **F59.18**

```
/***
 * Implements hook_install().
 */
function profile_forcontu_install() {
  // Run the installation of standard profile
  include_once DRUPAL_ROOT . '/profiles/standard/standard.install';
  standard_install();

  // Other operations
}
```

F59.18**standard_install()**

A function called, `standard_install()`, which runs the `hook_install()` implementation of the Standard core profile.

Establishing a theme during the installation

It is possible to change the default theme during the installation process. In the following example we use the default theme **Garland**, included in the core. We will do so during the `hook_install()`. **F59.19**

```
/***
 * Implements hook_install().
 */
function profile_forcontu_install() {
  // Run the installation of standard profile
  include_once DRUPAL_ROOT . '/profiles/standard/standard.install';
  standard_install();

  // Sets Garland as default theme
  db_update('system')
    ->fields(array('status' => 1))
    ->condition('type', 'theme')
    ->condition('name', 'garland')
    ->execute();

  variable_set('theme_default', 'garland');
}
```

F59.19**Establish a theme**

Code to set a default theme during the installation of the profile.

Integration with Features

In the same way, the modules must be installed during the installation of this distribution. We can also add **Features**, that must be included in the modules folder of the installation profile.

We will add these **features** to the vector `dependencies[]` in the file `.info`, Without forgetting the unit with the module 'features'. **F59.20**

```
dependencies[] = features

dependencies[] = feature1
dependencies[] = feature2
```

The integration of features in the installation profiles makes it easy to create compatible objects with Features in the site, already studied in Unit 58. Look at; Rules of Rules Contexts, Content types, Definition of fields, Dependencies with other modules, Menus and menu links, Image styles, Text formats, Permissions and roles, Taxonomies, etc.

F59.20**Features**

We can add features by adding them to the array `dependencies[]`.

59.5

Contributed distribution profiles

To conclude this unit we will look at some examples of installation profiles contributed by the community. We will be able to find these and many other installation profiles at;

<http://drupal.org/project/installation+profiles>

When an installation profile is downloaded, it can be done in two different ways;

- **Profile folder.** To decompress the downloaded file you will see that it only includes the folder corresponding to the installation profile. To install this profile first upload the latest version of Drupal 7, and then upload the folder to the profile folder /profiles.
- **Full Distribution.** Some profiles also include all the core files of Drupal. In this case we upload the entire contents of the file, which already includes the installation profile within the folder /profiles.

Corporative site

Corporative site is a site that includes a good number of useful modules to create a corporate portal. It also includes some features to add additional functionality.

The site is available at:

http://drupal.org/project/corporative_site

During the installation process, the system we will request the necessary modules that will be installed in the folder /sites/all/modules. **F59.21**

F59.21

Modules required

During the installation of the profile, the system will tell us which modules are necessary. We will be unable to continue the installation until we have uploaded all the required modules to the server.

✗ Required modules	Required modules not found.
<p>The following modules are required but were not found. Move them into the appropriate modules subdirectory, such as <code>sites/all/modules</code>. Missing modules: Advanced_help, Better_exposed_filters, Block_class, Ctools, Calendar, CKeditor, Context, Context_ui, Context_layouts, Date, Diff, Email, Empty_page, Humanistxt, Features, Imce, Libraries, Link, Megamenu, Nodequeue, Pathauto, Transliteration, Print, Search404, Sharethis, Site_map, Special_menu_items, Strongarm, Token, Views, Views_ui, Views_slideshow_cycle, Views_slideshow, Fb_social, Twitter, References, Field_group, Search_config, Shadowbox</p>	

Figure **F59.22** shows the appearance of the site once it is installed. You must keep in mind that, although the account profile had many functionalities, you will have to configure them so that they are available.

F59.22

Corporative Site

State of the site once installed the installation profile. We will have to configure the modules installed to improve the layout of the site.

Welcome to your site

Corporative Site comes with many modules and features to speed up the creation of your site.

Below is a list with the location of some of the resources enabled by default:

- This block's template is located at /profiles/corporative_site/modules/custom/welcome.tpl.php. It has been added to the frontpage by the [Homepage Context](#).
- The frontpage has been defined to be 'home' at [Site information](#). It uses an empty page so you can easily add blocks to it. You can edit or add empty pages at [Empty page callbacks](#).
- The sidebar blocks are loaded by a site wide context. You can edit it at the [General Context settings](#). Click on 'Blocks' when being at that page so you can choose which blocks should be listed in all pages.
- The Facebook fan box can be configured at [Fan box settings](#). You can specify there your Facebook page.

OpenPublish

OpenPublish is a site aimed at the creation of a digital newspaper. It allows you to add news to different categories, comment on the content published, share on social network, etc.

OpenPublish is available at:

<http://drupal.org/project/openpublish>

The downloaded file already contains all the installation files of Drupal, and we only need to upload the content and begin the installation by selecting the installation profile OpenPublish. The Figure shows a screenshot of the default look of the site. Keep in mind that the version for Drupal 7 is still an alpha version.

F59.23

OpenPublish

State of the site once installed in the installation profile. It corresponds to the version of Drupal 6, because the version of Drupal 7 is still in development.

Open Publish Journal

Blogs »

April 15, 2010 11:43 AM
A Clash Over Unpasteurized Milk Gets Raw

LAURA LANORD ET AL.
Advocates of fresh-from-the-farm unpasteurized foods tout "raw" milk as the ultimate health food, claiming it is rich in disease-fighting nutrients and healthy enzymes that are lost in pasteurization.

April 15, 2010 12:07 PM
A Downside of Organ Donation

Multimedia »

VIDEO
Obama: A Day of Great Progress

President Obama said this evening that he will push hard for "strong sanctions" against Iran for its continued development of a nuclear program.

Commerce Kickstart

Commerce Kickstart is a layout that facilitates the creation of a virtual store with **Drupal Commerce**. The site includes the required modules and the possibility to generate example content.

Commerce Kickstart can be found at:

http://drupal.org/project/commerce_kickstart

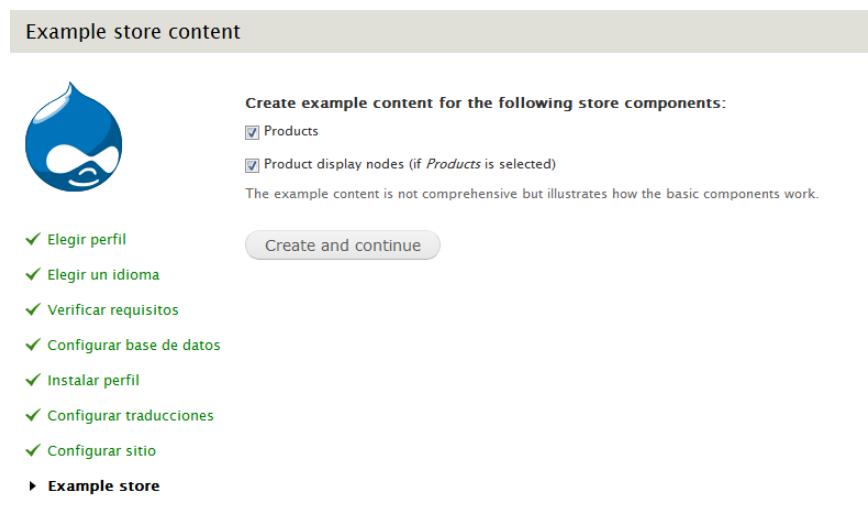
It is also a full distribution, so we will only have to upload the files and start the installation.

During the installation **F59.24** we can indicate whether we want to generate examples.

F59.24

Commerce Kickstart

During the Kickstart installation we can indicate whether we want to generate example content.



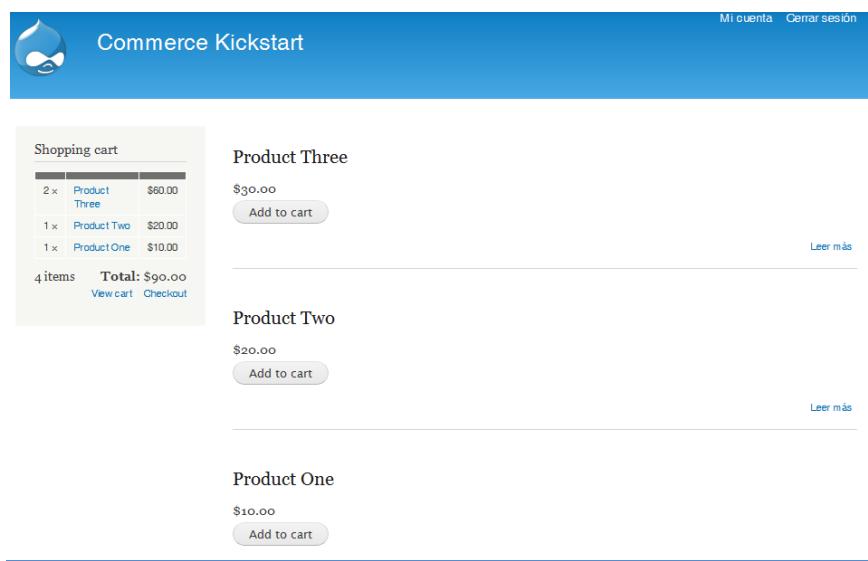
The screenshot shows the 'Example store content' configuration screen. It features a large blue Drupal logo icon. To its right, there's a section titled 'Create example content for the following store components:' with two checked checkboxes: 'Products' and 'Product display nodes (if Products is selected)'. Below this, a note states: 'The example content is not comprehensive but illustrates how the basic components work.' On the left, a vertical list of tasks with green checkmarks indicates completed steps: 'Elegir perfil', 'Elegir un idioma', 'Verificar requisitos', 'Configurar base de datos', 'Instalar perfil', 'Configurar traducciones', 'Configurar sitio', and 'Example store'. A 'Create and continue' button is located next to the task list.

Once we have installed the profile, we can directly simulate an example of a product purchase which will be added to the basket. For more information on how to use **Drupal Commerce**, look at **Unit 36** of the Intermediate Level. **F59.25**

F59.25

Commerce Kickstart

State of the site once installed in the profile. The installed base will facilitate the implementation and configuration of the virtual store with Drupal Commerce.



The screenshot shows the Commerce Kickstart website. At the top, there's a header with a user account link ('Mi cuenta') and a 'Cerrar sesión' button. The main content area starts with a 'Shopping cart' summary table:

Shopping cart		
2 x	Product Three	\$60.00
1 x	Product Two	\$20.00
1 x	Product One	\$10.00
4 items	Total:	\$90.00
View cart Checkout		

Below the cart, there are three product cards:

- Product Three**: Price \$30.00, with an 'Add to cart' button and a 'Leer más' link.
- Product Two**: Price \$20.00, with an 'Add to cart' button and a 'Leer más' link.
- Product One**: Price \$10.00, with an 'Add to cart' button and a 'Leer más' link.

60 Contributing to the Drupal Community

The community of users has been instrumental in the growth and continuous improvement of Drupal. In this last unit of the Advanced Level we explain the different ways to actively participate in the Drupal community, to report errors or incidents in the modules, to create and share modules, to create and use patches to help with the translation of Drupal, etc.

Comparative D7/D6

As Drupal has evolved, also have the tools that give you support. For example, now the version control system is Git, which was introduced in 2011. Therefore, in this unit we cannot talk about differences between the versions, but about a continuous development of the software and the tools used.

Unit contents

60.1 The Drupal Community	446
60.2 Reporting Issues	448
60.3 Sharing a project in Drupal.org	451
60.4 Create and apply patches	460
60.5 Create new versions of a project	464
60.6 Assist in Drupal translation	466

60

60.1 The Drupal Community

The community of users has been instrumental in the growth and continuous improvement of Drupal. The ways to participate in the community are very diverse; to help other users that are starting with Drupal, organize events, translate modules, report errors, etc.

In this link you will find an initial guide (in English) on the different ways to participate in the community.

<http://drupal.org/getting-involved-guide>

We begin this unit with a brief introduction to the events, associations and meetings for the Drupal community.

Events

There are different types of events related to Drupal and organized by the community depending on where you are located:

- **DrupalCon.** Is the international Drupal event . It is an event conducted entirely in English, which brings together people from all over the world. To date, this has been organized twice a year, once in the United States and another time in Europe, but it is being extended to other continents. DrupalCon 2012 will be in Denver (United States) and Munich (Germany).
- **DrupalCamp.** DrupalCamp is an event that generally has a wide audience and is held in the language of the host country. For example, Spain celebrates the Drupalcamp Spain.
- **DrupalCamp Spain.** The first DrupalCamp Spain was held in 2010 in Barcelona where the Spanish Association of Drupal (AED) originated. Since then the AED provides the necessary support to continue the annual celebration of the Drupalcamp (Seville 2011, Madrid 2012).
- **Drupal Developer Days.** This event is aimed towards Drupal developers. It is an international event on a lesser scale than the DrupalCon but includes developers from other countries. In 2011 the event was held in Brussels (Belgium) and in 2012 it will be held in Barcelona (Spain), with the support of the AED.
- **Drupal Day.** This event is celebrated in Spain and is promoted by the AED. This is an event for a single day whose objective is to create a meeting place for Drupal professionals. In 2011 was held in Barcelona and in 2012 will be held in Valencia.
- **Drupal Summit Latino.** It is an event organized by the Latin American community of Drupal that brings together Drupal enthusiasts from all over Latin America in different countries. In 2011 the first event was held in Lima (Peru) and in 2012 it was held in Guadalajara (Mexico).

These are just some of the events that take place around Drupal. In your local community ask for upcoming events in your area why not attempt to organize an informal meeting to meet other users and professionals of Drupal.

Drupal Associations

- **Drupal Association** (<https://association.drupal.org>). Is the international association of Drupal, whose mission is to grow Drupal. The Drupal Association accepts two types of partners, individuals and organizations. In both cases, in addition to your contribution, you get a badge or insignia that you can use on your web blog, etc.

Online Groups

There are many ways to get in contact with other members of the community; either to search for information, to make connections, resolve doubts, organize events, etc. Some of these ways include:

- **Drupal.org** (<http://drupal.org/>). It is the primary Drupal site written in English. As a user on drupal.org it is important to use many of the tools that are included on this site.
- **Drupal Forums** (<http://drupal.org/forum>). Are the main source used to request support from other users, publish professional services, etc. Most of the forums are conducted in English.
- **Drupal Groups** (<http://groups.drupal.org/>). You can join any group that interests you. Also, you can find international groups in English, but also local groups in Spanish (Spanish association, Drupalcamp Spain Madrid, etc.).
- **Google Groups** Within the Google groups we emphasize the Drupal group in Spanish. It is used primarily to conduct technical consultations.
- **Pages and Groups on Facebook.** On Facebook you'll find many pages and groups related to Drupal. We emphasize the **Drupal group** (in Spanish) and the **Drupal page** (in English).
- **Twitter.** On Twitter you can also follow many Drupal professionals. #Drupal is the official user of drupal.org. The hashtags used most often by the community are #drupal (international) and #drupal_es (in Spanish).
- **IRC.** The full list of channels to communicate via IRC are available at: <http://drupal.org/node/679904>. Some of the most important channels are: #drupal-support, #drupal and #drupal-es, the latter is in Spanish.
- **Linkedin.** In this professional network you will find groups who talk about Drupal. In the group **Drupal España** you will find a demand for Drupal developers.
- **Drupal Hispano** (<http://drupal.org.es/>). It is the meeting point of the Spanish-speaking community. On this page you will find documentation in Spanish; forums, announcements, etc.
- **Planet Drupal** (<http://drupal.org/planet>). It is a page where content is added to different Drupal blogs.

60.2 Reporting Issues

From the official website of each project (module, theme, installation profile, etc.) we can contact the developers who maintain the site through the **Issues**. Taking into account that all communication must be in English. The incident reports are displayed in the right hand column. **F60.1**

F60.1**Issues**

Each project has its own queue of issues, where we can find the errors reported by other users or publish the errors found. Always check the issue already published before you make a new query.

Download & Extend

[Download & Extend Home](#) [Drupal Core](#) [Modules](#) [Themes](#) [Translations](#) [Installation Profiles](#)

Devel

[View](#) [Version control](#) [Revisions](#)

Posted by moshe weitzman on September 28, 2003 at 4:44pm

A suite of modules containing fun for module developers and themers ...

Devel

- Helper functions for Drupal developers and inquisitive admins. This module can print a summary of all database queries for each page request at the bottom of each page. The summary includes how many times each query was executed on a page (shouldn't run same query multiple times), and how long each query took (short is good - use cache for complex queries).
- Also a `dprint_r($array)` function is provided, which pretty prints arrays. Useful during development. Similarly, a `ddebug_backtrace()` is offered.
- much more. See [this helpful demo page](#).

Generate content

Accelerate development of your site or module by quickly generating nodes, comments, terms, users, and more.

Devel Node Access (DNA)

View the node access entries for the node(s) that are shown on a page. Essential for developers of node access modules and useful for site admins in debugging problems with those modules.

Performance Logging

This module was removed from devel in 7/2011. It has been revived over at the [performance project](#). Please use its [issue queue](#) for discussion.

Compatibility Note:

The Devel module will not work with the old and buggy Zend optimizer 3.2.8 and below (and possibly 3.2.x in general). More info at [this comment and the rest of that issue](#).

Downloads

Recommended releases

Version	Downloads	Date	Links
7.x-1.2	tar.gz (181.78 KB) zip (229.17 KB)	2011-Jul-22	Notes
6.x-1.26	tar.gz (115.96 KB) zip (154.57 KB)	2011-Jul-22	Notes

Development releases

Version	Downloads	Date	Links
8.x-1.x-dev	tar.gz (186.27 KB) zip (234.15 KB)	2012-Mar-04	Notes
7.x-1.x-dev	tar.gz (186.34 KB) zip (234.26 KB)	2012-Mar-04	Notes
6.x-1.x-dev	tar.gz (116.72 KB) zip (155.37 KB)	2011-Sep-17	Notes

Maintainers for Devel

[salvis](#) - 264 commits
last: 1 week ago, first: 3 years ago

[moshe weitzman](#) - 963 commits
last: 13 weeks ago, first: 9 years ago

[Dave Reid](#) - 20 commits
last: 1 year ago, first: 2 years ago

[kbahey](#) - 96 commits
last: 1 year ago, first: 3 years ago

[drewish](#) - 12 commits
last: 1 year ago, first: 3 years ago

[View all committers](#)

[View commits](#)

Issues for Devel

To avoid duplicates, please search before submitting a new issue.

[Advanced search](#)

All issues

258 open, 1384 total

Bug reports

110 open, 793 total

[Subscribe via e-mail](#)

Resources

[Look at screenshots](#)

[View project translations](#)

Development

[View pending patches](#)

[Repository viewer](#)

[View commits](#)

[Report a security issue](#)

[View change records](#)

Before publishing any issue it is highly recommended that you find out if your issue is already published in the issues queue. This prevents duplicates and, above all, check if it is already solved. **F60.2**

Download & Extend

Download & Extend Home Drupal Core Modules Themes Translations Installation Profiles

Devel

Issues for Devel

Create a new issue Advanced search E-mail notifications

Search for: [] Status: [- Open issues -] Priority: [- Any -] Category: [- Any -] Version: [- 7.x issues -]

Component: [- Any -] Search

Summary	Status	Priority	Category	Version	Created	Assigned to	Last updated	Replies	
Notice/Warning after content was generated using devel-generate.	active	major	support requests	7.x-1.2	devel_generate	1	22 hours 4 min	20 hours 41 min	1
Command to delete content created by devel generate new	active	normal	support requests	6.x-1.9	devel_generate		3 days 4 hours	3 days 4 hours	
Keep querylog files new	active	normal	feature requests	7.x-1.x-dev	devel	3	6 days 17 hours	4 days 9 hours	3 new
Taxonomy terms aren't randomised when generating nodes with devel_generate updated	needs review	normal	bug reports	7.x-1.x-dev	devel_generate	3	5 weeks 10 hours	1 week 2 days	1 new
Users generated with invalid e-mails new	active	normal	bug reports	7.x-1.2	devel_generate	3	1 week 5 days	1 week 3 days	3 new

F60.2**Issue queue**

List of issues for a project. We can filter the issues by category or version of the module and we can indicate the string to search for.

In the queue of issues there are not only published errors; we can also publish tasks, requests for new functionality or even request for help or technical support. You should always keep in mind that the people who look at your projects do it in an arbitrary way so you don't have support on demand and you should not wait around for a response to your query.

Create a new issue

To send an issue report using **Create a new issue**. Remember that the issue must be written in English, by completing the following fields: **F60.3**

- Information about the project:
 - o **Project.** Name of the project. If we are creating the incidence from the project page we will be unable to change this value.
 - o **Version.** Select the version of the project where you have found the issue.
 - o **Component.** The component refers to the category within the project. The available values may vary from one project to another, but the default values are; code, documentation, miscellaneous and user interface.
- Information about the issue:
 - o **Category.** The available categories are:
 - bug report.
 - task.
 - feature request.
 - support request.
 - o **Priority.** Select the priority of the issue. Keep in mind that the priority must be established based on the type of error found and not so much on the particular needs you may have in your project.

- o **Assigned.** If it is a job that you're going to do yourself, you can FWS the task from this option.
- o **Status.** The initial state will be active. The moderator of the project may change this state.

- Details of the issue:

- o **Title.** Include a title that describes the issue..
- o **Description.** Describe, in detail, the issue. Include the code that you have created in case there is an error.
- o **Issue tags.** Classify the issue.
- o **File attachments.** Add the files that you consider necessary to explain the issue. For example; a screenshot with the error, etc.

F60.3

Create an issue

Before creating an issue we must check that there is not a similar discussion in the queue of issues. Add to the issue all the information that you consider necessary to ensure that other users understand the problem reported. Remember that all the communication must be made in English.

Post new comment

[Edit issue settings](#)

Note: changing any of these items will update the issue's overall values.

Issue title: *

Notice/Warning after content was generated using devel-generate.

Project: *	Version: *	Component: *	Assigned:
Devel	7.x-1.2	devel_generate	Unassigned
Category: *	Priority:	Status:	
support request	major	active	

Descriptions of the [Priority](#) and [Status](#) values can be found in the [Issue queue handbook](#).

Comment:

B I ABC H2 H3 H4

• Web page addresses and e-mail addresses turn into links automatically.
 • Allowed HTML tags: <h2> <h3> <h4> <h5> <code> <blockquote> <q> <cite> <sup> <sub> <p>
 <dl> <dt> <dd> <a> <u> <i> <table> <tr> <td> <th> <tbody> <thead> <tfoot> <colgroup> <caption> <hr>
 • Lines and paragraphs break automatically.
 • You may post code using <code>...</code> (generic) or <?php ... ?> (highlighted PHP) tags.
 • Project issue numbers (ex. #12345) turn into links automatically.
 • Only images hosted on this site may be used in tags. Others are replaced with an error icon.

[More information about formatting options](#)

[File attachments](#)

Changes made to the attachments are not permanent until you save this post.

Attach new file:

No se ha ...n archivo

The maximum upload size is 3 MB. Only files with the following extensions may be uploaded: jpg jpeg gif png txt xls pdf ppt pps odt ods odp gz tgz patch diff zip test info po pot psd. Only files ending in .patch or .diff will be sent for testing. For patches that should not be tested by the testbot (for example, patches that do not work with the current API version in the issue), add "do-not-test" before the .patch. For example "drupal.do_nothing_555555-do-not-test.patch"

Once you have submitted the issue report, be patient. You will receive an email each time a user posts a comment on your issue.

If you want to see comments posted on an issue report made by another user, simply post a comment with the text "subscribing". In this way you will indicate that you are interested in the topic and you will receive any new comments by email.

Sharing a project in Drupal.org

60.3

The philosophy of the Drupal community is "don't compete, collaborate". Because of this, before you create a new project (module, theme, or installation profile), see if there is a similar project that was already created. If so, it would make sense to collaborate with the designer of the existing module by helping to maintain and broaden it instead of starting over from scratch.

You can see all of the instructions for sharing a project at Drupal.org:
<http://drupal.org/node/7765>

Documentation of a module

Share and maintaining a project in Drupal.org goes far beyond sharing the code in the repository of modules. When you begin your project everything should be well documented; including some of the following files:

- **README.txt** (mandatory). Description of the project and instructions for use and installation.
- **INSTALL.txt**. It is an optional file but is necessary if the installation instructions are extensive. In this case it should be separate from the README.txt.
- **Project page**. On this page of the project we will include a summary of the project. We should not dwell on leaving complete information from the file README.txt, or from the documentation page of the project.
- **CHANGELOG.txt**. Registered version changes of the project..
- **Documentation page of the project on Drupal.org**. The project may have a section in the documentation area of Drupal.org. You can include the information contained in the file README.txt and INSTALL.txt.

Coding Standards

The project must follow the **coding standards** already studied in this course level. For this reason it is recommended that you use the **Coder module** and check that your coding is correct.

Translation

Remember that the project must be in English; the names of the variables and functions, the comments in the code, etc. If you wish to include the translation of the module to spanish, generate the translation and share it in localiza.drupal.org.

Git: project repository and version control

The best place to upload and share a related project with Drupal is in the official repository of Drupal.org. If your module is not available in this repository, it may not have its page on Drupal.org and therefore it will not be grouped with the rest of modules. We should also bear in mind that, for a project that is in the official repository it must be licensed under the GNU GPL version 2 or higher, which is the same license that is distributed with Drupal.

If the module has components (for example a library) that cannot be distributed with this license, you can share the base module, detailing in the instructions how to download and install the additional components.

The repository of files used in Drupal is **Git**, which is a version control system that allows the free publication of files in a distributed manner. Developers can work on a complete copy of a module from the repository sharing the posted changes.

To use the repository we must first gain access to Git. We will do it from the tab Git access, within the edition of our user account on Drupal.org. We must indicate the username of Git and accept the terms of use.

You will find more information about the request for access to Git here: **F60.4**

<http://drupal.org/node/1047190>

F60.4

Access to Git

Git from the access tab, in the edition of our Drupal.org account, we may request access to the repository of Drupal projects.

forcontu

Dashboard Your Posts Your Commits Your Issues Your Projects Your Groups Profile

View Edit Notifications SSH keys

Account **Git access** Drupal Personal information Work My newsletters E-mail addresses

Note: You will not be able to use Git until you have selected a Git username. A suggestion has been provided for you, based on your username. **Note that once chosen, your Git username cannot be changed.**

You will not be able to use Git unless the checkbox consenting to the terms of service is checked.

Git username
Choose a username to use for Git. This will be your SSH user when authenticating to Drupal.org with Git, and Drupal.org will use it to generate your personalized sandbox URIs. **Once chosen, your Git username cannot be changed.**

Desired Git username:

Acceptable characters are ANSI alphanumerics (A-Z, a-z, 0-9), periods, underscores, or dashes. A suggested username is provided, based on the acceptable characters in your Drupal username and the availability of Git usernames.

Git access agreement

To use Drupal's version control systems you must agree to the following:

- I will only commit [GPL V2+-licensed](#) code and resources to Drupal code repositories.
- I will only commit code and resources that I own or am permitted to distribute.
- I will cooperate with the [Drupal Security Team](#) as needed.
- I have read and will adhere to the [Drupal Code of Conduct](#).
- I agree to the [Drupal Code Repository Terms of Service](#).

I agree to these terms

Save

Installation and configuration of Git

In order to use Git we have to install it on our own equipment. Download the latest version at: <http://git-scm.com/>.

Once installed, to access the application **Git Bash** we will open the console or command line from Git, which uses the usual commands of Linux (ls, cd mkdir, etc.).

From the command line we need to establish the user.name and user.email. It is important that the email indicated corresponds to the email account of our Drupal account because the accounts will be linked through this field.

```
git config --global user.name "User name"
git config --global user.email user@example.com
```

Once you have changed these settings, we can check if the parameters have been added to the command Git with:

```
git config -l
```

The user's password will also be the Drupal.org password for Git.

The process of installing and configuring Git is detailed in:

<http://drupal.org/node/1022156>

Create a test project (**Sandbox**)

Drupal allows the creation of projects in **Sandbox** mode or development. This will allow us to experiment with the code of a module or topic before attempting to share it with the rest of the community as a complete project (Full Project).

<http://drupal.org/node/1011196>

Although the module is in Sandbox mode, the project will be accessible via the repository of modules by selecting in the filter state **All the projects** or only the projects in **Sandbox**.

The fundamental difference between a complete project and a project in Sandbox is that in the second case it is not possible to publish versions of the same project. If another user wants to view the code for the project they can only download it using Git.

Prior to having access to the repository, we need to create a project in development (sandbox) from the link "Add a new project" in: **F60.5**

<http://drupal.org/node/1068950>

Creating a sandbox project

[View](#) [Edit](#) [Revisions](#)

Last updated July 15, 2011.

Once you have obtained basic Git access on Drupal.org, you can create a sandbox project as follows:

1. [Add a new project](#).
2. Fill out the form. Make sure the **sandbox** check box is selected. (It will be force-checked if you have not yet received permission to promote your sandbox projects to full projects.)
3. Click the **Save** button. Drupal creates and loads a page for your new project.
4. Click the **Version control** tab near the top of the new project page for instructions on how to start committing code to your sandbox repository.

The project will appear in the [My Projects](#) page in your dashboard.

F60.5

Sandbox project

We can create test projects or Sandbox, which will be available in the repository but that are considered experimental.

To create a project we will select the type of project (module, theme, installation profile, etc.), the categories it is related to, the state of maintenance and development and, finally, the title and description of the project. **F60.6**

F60.6

Create project

To create a project we select its type (module, theme, installation profile, etc.), the category, the status of maintenance, development, the title, and description of the project (in English).

Create Project

Project categories:

Project type: *

Modules
 Themes
 Theme engines
 Installation profiles
 Drupal.org projects
 Drupal core

Note: Translation projects have been moved and are now actively maintained at localize.drupal.org.

Modules categories:

- Administration
- Commerce/Advertising
- Community
- Content
- Content Access Control
- Content Construction Kit (CCK)
- Content Display
- Database Drivers
- Developer
- Drush
- E-commerce
- Evaluation/Rating
- Event
- Examples

Vocabularies:

Maintenance status: *

Actively maintained

Development status: *

Under active development

For definitions of maintenance and development statuses, see [Maintenance of your project](#).

Project information:

Project title: *

Forcontu Test Project

Sandbox

Description: *

B I ABC H2 H3 H4 @ code PHP

Test project for educational purposes.

[Split summary at cursor](#)

We have created the project **Forcontu Test Project**, which will allow us to test the repository. The module is available at: **F60.7**

<http://drupal.org/sandbox/forcontu/1483800>

F60.7

Project Page

Page of a project in sandbox. The project is automatically added to the Git repository of Drupal.

forcontu's sandbox: Forcontu Test Project

[View](#) Version control Edit Maintainers



- The Git repository for this project has been enqueued for creation. It should be available in a few seconds.
- Project *Forcontu Test Project* has been created.

Posted by [forcontu](#) on March 15, 2012 at 2:39pm

Experimental Project

This is a [sandbox project](#), which contains experimental code for developer use only.

Test project for educational purposes.

[Apply for full project access](#)

Categories: [Actively maintained](#), [Under active development](#), [Modules](#), [Examples](#)

In the **Version control** tab for each project we are shown the instructions for downloading from the repository. As we have not yet uploaded the code of the draft, the instructions that are displayed initially explain the steps to follow to get the project to the repository. **F60.8**

Forcontu Test Project

Forcontu Test Project

[View](#) [Version control](#) [Edit](#) [Maintainers](#)

This page gives the essential Git commands for working with this project's source files.

Empty Sandbox Repository

Setting up this repository for the first time:

You will be prompted to enter your Drupal.org password after the last step (and any time you make requests from Drupal.org) if you have not uploaded an SSH key or if your SSH key fails. See [Authenticate with Git on Drupal.org](#) for details.

```
mkdir forcontu_test_project
cd forcontu_test_project
git init
echo "name = Forcontu Test Project" > forcontu_test_project.info
git add forcontu_test_project.info
git commit -m "Initial commit."
git remote add origin forcontu@git.drupal.org:sandbox/forcontu/1483800.git
git push origin master
```

F60.8**Version Control of the project**

The instructions in the version control tab are used to upload the project to the repository.

A note about your sandbox and function names:

Contributed modules on Drupal.org typically use their project's shortname as the prefix to the module's functions. Sandbox shortnames are numbers, which are not valid function names. If you intend to release your code for Drupal.org someday, choose a prefix that isn't being used on Drupal.org already. There is, unfortunately, no guarantee it will still be available when you're ready to release, but you'll save a step if it is. To check, type [http://drupal.org/project/\[the_name_you_want\]](http://drupal.org/project/[the_name_you_want]).

When you've completed these steps, [refresh this page](#) for further direction.

Note: You have created a local repository. You will not need to complete the One-Time clone on the next page unless you delete your local repository or set up on a different machine. Then, you'll need to perform the One-time setup, as will co-maintainers.

It is recommended that you create a root folder for the repository where we will store all the projects synchronized with the repository of Drupal.org.

In our case we have created the local folder **h:/workspace/drupal.org**, where we will create the projects uploaded (or downloaded) in the repository of Drupal.org. Within this folder we have added the folder for the test project, **forcontu_test_project**, which already includes the files **.info**, **.module** and **.install**.

To synchronize the folder of the module with the repository we get inside the folder from the command line of Git. Use the usual Linux commands to navigate to the folder. We will follow the instructions on the project page taking into account that we have already created the folder for the project. **F60.9**

F60.9**Commands to move the project to the repository**

The Figure shows the commands that run from the Git console to create the project and upload it to the local repository, Drupal.org, so that they are synchronized.

```
$ cd forcontu_test_project/
$ git init
Initialized empty Git repository in
h:/workspace/drupal.org/forcontu_test_project/.git/
$ git add forcontu_test_project.info
$ git add forcontu_test_project.module
$ git add forcontu_test_project.install
$ git commit -m "Initial commit."
[master (root-commit) e9b7b18] Initial commit.
 3 files changed, 379 insertions(+), 0 deletions(-)
create mode 100644 forcontu_test_project.info
create mode 100644 forcontu test project.install
create mode 100644 forcontu_test_project.module

$ git remote add origin forcontu@git.drupal.org:sandbox/forcontu/1483800.git
$ git push origin master
The authenticity of host 'git.drupal.org (140.211.10.6)' can't be
established.
RSA key fingerprint is
69:35:9f:1b:a8:26:ad:6d:cc:93:29:25:6e:d7:25:2b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'git.drupal.org,140.211.10.6' (RSA) to
the list of known hosts.
forcontu@git.drupal.org's password:
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 2.70 KiB, done.
Total 5 (delta 0), reused 0 (delta 0)
To forcontu@git.drupal.org:sandbox/forcontu/1483800.git
 * [new branch]      master -> master
$
```

Once inside the folder the command "**git init**" initializes the repository for this project in particular, by creating the folder .git (hidden folder). Even if the folder contains files, the repository is considered initially empty having to add each file using the command "**git add**".

Once we have added the files with "**git commit**" we created a first version (commit) of the project.

With "**git remote add origin**" we establish the relationship between the local repository and the repository in Drupal.org.

Once we establish this relationship, with "**git push origin master**" we will be climbing up the local changes (established in the initial commit) to the repository on Drupal.org. At this point **authentication is required** for access to the repository, that can be done two ways:

- **Defining the SSH key.** This allows the identification to be done directly through the keys defined by the local computer. The process of creating SSH keys are detailed in <http://drupal.org/node/1027094>.
- **Drupal.org password.** Facilitating the password for access to the repository, which corresponds to that of our user in Drupal.org.

In our example, we used the second method. Once properly authenticated, the files will be uploaded to the server version so that any user can download them.

If we go back to the **Version control** tab of our project, we see that the instructions have changed. These instructions, which are common to all projects, permit any user in the community to collaborate on the project, upload changes, patches etc. **F60.10**

Forcontu Test Project

[View](#) [Version control](#)

This page gives the essential Git commands for working with this project's source files.

Version to work from: * [Show](#)

- Update Notice: See [Git instructions updates](#) for a record of updates to these instructions.
- Last updated: May 12, 2011

One-Time Only

Setting up this repository locally for the first time

```
git clone --branch master http://git.drupal.org/sandbox/forcontu/1483800.git
forcontu_test_project
cd forcontu_test_project
```

Not working for you? See [Troubleshooting Git clone](#).

Routinely

The headings below are not sequential. What you choose to do depends on where you are in your process.

Checking your repository status

To see what you will commit by running `git commit` and what you could commit by running `git add` before running `git commit`.

```
git status
```

Update changes to the draft

If we modify any of the files on the local version of the draft, we will have to upload the changes to the repository. The action of publishing changes to the project is called **commit**.

We started with the command "git add -A", that looks like, in the folder (and subfolders) all the changes that have been made in the project.

Then we define the commit with **git commit - m "Description of the version"**. Every time we submit a change, it should identify the change we made. If a change is made by another user, the developers of the module can decide whether or not to implement the change in the next version of the module. In this URL you will find more information about the format **commit**: <http://drupal.org/node/52287>. The shorter version is:

Issue #[\[issue number\]](#) by [\[comma-separated usernames\]](#): [Short summary of the change].

F60.10

Instructions for other users

Once we have uploaded the project to the repository, the instructions shown in the Version control tab will be used for any user to download the project, create patches, etc.

Finally, to save the commit to the server version, we will execute the command, "git push -u origin master". **F60.11**

F60.11**Upload changes to the repository**

Commands used to upload changes made in the project to the repository.

```
$ git add -A
$ git commit -m "Issue #12 by forcontu: Added module dependencies."
[master a12ae35] Issue #12 by forcontu: Added module dependencies.
 1 files changed, 1 insertions(+), 0 deletions(-)

$ git push -u origin master
forcontu@git.drupal.org's password:
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 347 bytes, done.
Total 3 (delta 2), reused 0 (delta 0)
To forcontu@git.drupal.org:sandbox/forcontu/1483800.git
  e9b7b18..a12ae35  master -> master
Branch master set up to track remote branch master from origin.
```

The commits files uploaded to the repository can be seen from the project page in the right hand column (View commits). **F60.12**

F60.12**Commits**

Each change uploaded to the repository is called commit. From the project page we can see the list of commits of the project.

The screenshot shows the project page for 'forcontu's sandbox: Forcontu Test Project'. It includes tabs for View, Version control, Edit, and Maintainers. The View tab is active. Below the tabs, it says 'Posted by forcontu on March 15, 2012 at 2:39pm'. A section titled 'Experimental Project' describes it as a sandbox project for developer use only. It links to the Version control tab for full directions and states it's a test project for educational purposes. Below this, there's a link to 'Apply for full project access'. Under 'Categories', it lists 'Actively maintained', 'Under active development', 'Modules', and 'Examples'. On the right side, there are sections for 'Maintainers for Forcontu Test Project' (listing 'forcontu' with 2 commits, last updated 18 min ago) and 'Issues for Forcontu Test Project' (with a search bar and a link to 'Advanced search').

In our example we will look at the commit and the second initial commit carried out, both with the corresponding description. **F60.13**

F60.13**List of commits**

The list of commits includes all of the changes made to the project. That is why the commit should include a short description of the published change.

Commits for Forcontu Test Project**Forcontu Test Project: March 16, 2012 10:38**

Commit **a12ae35** on **master**
by **forcontu**

Issue #12 by forcontu: Added module dependencies.

Forcontu Test Project: March 16, 2012 9:34

Commit **e9b7b18** on **master**
by **forcontu**

Initial commit.

Issues management

Once we have published the draft, other users may send issues through the issues queue. With each issue we can maintain an active conversation with other users, always in English, to find the source of the issue or the possible solutions. **F60.14**

Forcontu Test Project

Issues for Forcontu Test Project

[Create a new issue](#) [Advanced search](#) [Statistics](#) [E-mail notifications](#)

Search for	Status	Priority	Category	Component			
<input type="text"/>	- Open issues -	- Any -	- Any -	- Any -	<input type="button" value="Search"/>		
Summary	Status	Priority	Category	Component	Replies	Last updated	Assigned to
Needs views dependency	active	major	bug reports	Code	1	1 min 21 sec	1 min 21 sec

Subscribe with RSS 

F60.14

Issues management

Our project also has its own queue of issues, where other users may publish errors, to ask for help, etc.

Once the issue is settled, change your status to **fixed**, or any of the appropriate available statuses. If the correction of the issue required a commit, it should be noted in the descriptive text of the commit; the number of issues, the users that have participated, and their identification and solution. **F60.15**

Forcontu Test Project

Issues for Forcontu Test Project

[Create a new issue](#) [Advanced search](#) [Statistics](#) [E-mail notifications](#)

Search for	Status	Priority	Category	Component			
<input type="text"/>	- Any -	- Any -	- Any -	- Any -	<input type="button" value="Search"/>		
Summary	Status	Priority	Category	Component	Replies	Last updated	Assigned to
Needs views dependency	closed (fixed)	major	bug reports	Code	1	12 sec	forcontu

Subscribe with RSS 

F60.15

Close issues

It is appropriate to manage the issue, closing those that are already solved.

The issues that have not had activity for 2 weeks will automatically close, going to the closed state (**fixed**). At any time we can reopen the issue and continue working in it.

60.4 Create and apply patches

Download other projects

If we wish to collaborate on a modification in the code of another project, first we will have to download it in our local repository of Git.

The instructions to download another project are found on the page of each projects **Version control** tab. [F60.16](#)

[F60.16](#)

Download project

In the version control tab there are instructions on how to download other projects from the repository.

One-Time Only

Setting up this repository locally for the first time

```
git clone --branch master http://git.drupal.org/sandbox/forcontu/1483800.git
forcontu_test_project
cd forcontu_test_project
```

Once you run the command "**git clone**", the latest version of the module can be obtained from the remote repository. To download the files we have not needed log in, we only do that to upload files. [F60.17](#)

[F60.17](#)

Commands to download project

Git commands to download the project. Once downloaded you will get access to all files from the local computer.

```
$ git clone --branch master
http://git.drupal.org/sandbox/forcontu/1483800.git forcontu_test_project
Cloning into 'forcontu_test_project'...
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (8/8), done.

$ cd forcontu_test_project/
$ ls
forcontu_test_project.info  forcontu_test_project.install
forcontu_test_project.module
```

What is a patch?

A **patch** is the result of evaluating the differences between two versions of the same file (or a set of files). This result is generated with the command "**git diff**" and is stored in the form of a text file, with the extension **.patch**, that will have this type of structure: [F60.18](#)

[F60.18](#)

Patch

Example of file **.patch**. A patch shows the differences between the original file and the new file.

```
diff --git a/token_example/token_example.tokens.inc
b/token_example/token_example.tokens.inc
index 585dcea..b06d9d6 100644
--- a/token_example/token_example.tokens.inc
+++ b/token_example/token_example.tokens.inc
@@ -13,8 +13,8 @@ function token_example_token_info() {
    // second is the user's default text format, which is itself a 'format' token
    // type so it can be used directly.

-   // Comment in the original file that will be deleted
+   // New lines of code that were not in the original file
+   // and will be added when the patch is applied
 $info['types']['format'] = array(
   'name' => t('Text formats'),
   'description' => t('Tokens related to text formats.'),
```

The first few lines identify the file that is being modified, for example: token_example/tokem_example.tokens.inc.

```
diff --git a/token_example/token_example.tokens.inc
b/token_example/token_example.tokens.inc
index 585dcea..b06d9d6 100644
--- a/token_example/token_example.tokens.inc
+++ b/token_example/token_example.tokens.inc
```

The lines that begin with --- indicate they are replaced by the line with +++.

The next line indicates that the change is within the function token_example_token_inf(), above line 13.

```
@@ -13,8 +13,8 @@ function token_example_token_info() {
```

The lines that begin with - will be deleted.

```
- // Comment in the original file that will be deleted
```

Lines starting with + will be added to the file at the point indicated.

```
+ // New lines of code that were not in the original file
+ // and will be added when the patch is applied
```

The lines that do not include any symbol will remain the same as the original file.

The final file will not display the previous labels (+++, ---, +, -, etc.), they are only used to indicate changes in the patch.

Creating a patch with Git

The patches in Drupal are required when you discover an error in a module, and its modification is required. We could find ourselves in one of two situations, depending on whether we act as a provider or consumer of the patch.

- We have found an error in a module and we want to give the solution so that it is incorporated into the next version of the module. In this case we will **create a patch** that we will share through the queue of issues of the module.
- A user has contributed a patch, through the issue queue, which has not yet been included in the next version of the module. If we want to correct the error we will have to **apply the patch** in the module of our website.

The commands to create a patch are listed in the tab **Version control** of the module that we are changing. **F60.19**

F60.19**Create a patch**

The instructions to create a patch with Git are included in the version control tab of the module, although they are the same in all modules.

Patching**Getting ready to create or apply patches**

If you have not already cloned the repository, follow the directions above for setting up this repository in your local environment. Be sure you are on the branch you wish to patch, then ensure it is up-to-date with the following command:

```
git pull origin master
```

Creating a patch

For most improvements, use the following command after making your changes:

```
git diff > [description]-[issue-number]-[comment-number].patch
```

For more complex improvements that require adding/removing files, work over the course of multiple days, or collaboration with others, see the [Advanced patch workflow](#).

Applying a patch

Download the patch to your working directory. Apply the patch with the following command:

```
git apply -v [patchname.patch]
```

To avoid accidentally including the patch file in future commits, remove it:

```
rm [patchname.patch]
```

It is important to use an appropriate name for the patch. A good practice is to use the following:

[module]-[description]-[issue-number]-[comment-number].patch

For example, a patch for the **Pathauto** module related to the titles and that is going to be published in the issue commentary should be the named:

pathauto-titles-12345-95.patch

In this page you will find more information about good practices of issuing a patch.

<http://drupal.org/patch/submit>

To create a patch first modify the file or files in the project that we have previously downloaded from the repository. The command "**git diff**" is responsible for generating the patch with the differences: **F60.20**

F60.20**Create a patch with Git**

Instructions on how to create a patch with Git.

```
$ git diff > forcontu_test_project-dependencies-1485186-2.patch
```

```
$ ls
forcontu_test_project-dependencies-1485186-2.patch
forcontu_test_project.install
forcontu_test_project.info
forcontu_test_project.module
```

The patch will be generated in the module folder. In our example we have modified the file **forcontu_test-project.info** to eliminate the dependency on the module **views** and add dependencies with **panels** and **Pathauto**. **F60.21**

```

diff --git a/forcontu_test_project.info
b/forcontu_test_project.info
index f38e595..aab5a2c 100644
--- a/forcontu_test_project.info
+++ b/forcontu test project.info
@@ -2,5 +2,6 @@ name = "Forcontu Test Project"
description = 'Test project for educational purposes'
core = 7.x
files[] = forcontu_test_project.module
-dependencies[] = views
+dependencies[] = panels
+dependencies[] = pathauto
package = My Modules
\ No newline at end of file

```

F60.21**Patch created**

Example of a patch created with Git.

Attach a patch to a issue

The way to share a patch, with other users or with the maintainer of the project to take into account in the following versions of the module, is through the issue queue.

When we attach a patch to an issue we must change the status to "**Needs review**" or "**Needs work**". This way, the patch will be reviewed by the system for patches. This system, which does not apply in the projects in Sandbox, checks whether the patch has been successfully generated, saving the people doing the project maintenance extra work. **F60.22**

#8	Posted by skottler on July 4, 2011 at 5:34pm			
Status:	needs work	» needs review		
I've updated the patch to meet the new specs.				
Attachment	Size	Status	Test result	Operations
multiple_delete-870404-4.patch	1.29 KB	Idle	PASSED: [[SimpleTest]]: [MySQL] 0 pass(es).	View details Re-test

F60.22**Publish a patch**

Patches are published as attachments to an issue.

Apply a patch

Applying a patch with git is very simple. First we will download the version of the module that the patch has been created for. The module can be downloaded either from the module page or from the repository using Git.

Then we will download the patch file (file .patch) in the module folder. The patches are usually found in the issue comments of a module.

From the Git console, write the command: **F60.23**

```

$ git apply -v [patch-name.patch]
$ rm [patch-name.patch]

```

F60.23**Apply a patch**

It is recommended that you remove the patch once it's applied (command "rm").

Once the patch is applied we upload the modified module to our web site.

In general the patches should not be applied directly to a server that is currently in production or, at least, be sure to make appropriate precautions for a backup of the site.

60.5 Create new versions of a project

When we share a project, we must commit to respond to issues submitted by other members of the community and publish new versions when necessary.

To generate a new version of a module, we have to take into account that it may already be used by other users. This implies that the new version should include the appropriate code to update the database without affecting the content created in the web site.

Function hook_update_N()

The function **hook_update_N()** allows you to make an update, that will be executed during the process of updating the module when the system is running **update.php**.

http://api.drupal.org/api/drupal/modules--system--system.api.php/function/hook_update_N/7

Each time we use the function we need to specify the value of N, using 4 digits, with the following structure ABCC.

- **A.** The first digit corresponds to the version of Drupal: 5, 6, 7, 8, etc.
- **B.** The second digit corresponds to the version of the module. For example, for the version 5.x-1.* the value is 1, and for version 5.x-2.* the value is 2.
- **CC.** Sequence, starting at 00, that indicates the order of the updates.

F60.24

Functions for updating the project

Implementing `hook_update_N()` we can add chained updates a project database.

```
function example_module_update_5100 () {
  //...
}

function example_module_update_5200 () {
  //...
}

function example_module_update_6200 () {
  //...
}

function example_module_update_6201 () {
  //...
}

function example_module_update_7200 () {
  //...
}

function example_module_update_7202 () {
  //...
}
```

In this example, we have implemented the following updates: **F60.24**

- `example_module_update_5100()`. Updates the database to version 5.x-1.*.
- `example_module_update_5200()`. Updates the database to version 5.x-2.*.

- `example_module_update_6200()`. First update of the database for the version 6.x-2.*.
- `example_module_update_6201()`. Second update of the database for the version 6.x-2.*.
- `example_module_update_7200()`. First update of the database for the version 7.x-2.*.
- `example_module_update_7201()`. Second update of the database for the version 7.x-2.*.

The implementation of the update functions shall be carried out sequentially from the version of the installed module. According to the previous example, if we are doing a migration from a site in Drupal 5 to Drupal 7, the module includes all the steps to upgrade the database to the latest version of Drupal 5 and then to Drupal 6 and finally, to Drupal 7. In this case we should apply all the update functions.

If we subsequently installed a new version of the module, only the update functions that have not been implemented before will be taken into account.

Add items in the database

When an update of the module will be added or elements of the database (tables or fields) edited, we must add the changes to the implementation of `hook_schema()`, in addition to creating the function `hook_update_N()` to make the change. **F60.25**

```
/**
 * Adds a field to a table.
 */
function example_module_update_7100() {
  $new_field = array(
    'type' => 'varchar',
    'description' => 'New Col',
    'length' => 20,
    'not null' => FALSE,
  );
  db_add_field('mytable', 'newcol', $new_field);
}

/**
 * Adds a new table.
 */
function example_module_update_7101() {
  $schema['mytable2'] = array(
    // table definition
  );
  db_create_table('mytable2', $schema['mytable2']);
}
```

F60.25

Adding items to the database

Upgrade features allow you to add or modify the items in the database. These changes should be reflected in the deployment of `hook_schema()`.

The elements added to the database should be included in the implementation of `hook_schema()` of the module.

60.6

Assist in Drupal translation

Another way to collaborate with the community is providing and reviewing Drupal translations in both the core and contributed modules.

The translations of all the versions of Drupal are managed at localize.drupal.org. Although it is not required to register with the site to download the translations of the core and contributed modules, when you sign in you can join a translation group and then the import and export translation option will be enabled.

Access to localize.drupal.org is performed with the same user drupal.org. Once the account has been created, we can join the translation group for any language. Just look for the button **Join** on the main page of the language. **F60.26**

F60.26

Language translation page

Page for the translation of a language. We can join a translation group by clicking the **Join** button. To be able to join we must be recorded at the site, taking into account that you are the same user of Drupal.org.

This screenshot shows the 'Portuguese, Brazil overview' page on localize.drupal.org. At the top, there's a navigation bar with tabs: Download & Extend Home, Drupal Core, Modules, Themes, Translations (which is selected), and Installation Profiles. Below the navigation, there's a sub-navigation for 'Portuguese, Brazil overview' with tabs: Overview (selected), Board, and Translate. The main content area starts with a section titled 'Download & Extend' with a table showing top downloads for Drupal core. To the right, there's a 'Top contributors' table and 'Translation statistics'. A red box highlights the 'Join' button in the 'Portuguese, Brazil team' section.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Once we find a translation group, we can see the tabs **Translate**, **Import**, and **Export**. **F60.27**

F60.27

Translate Tabs , Import and Export in localize.drupal.org

Once we have registered and joined the translation group for a language, we can access the tabs Translate, Import and Export.

This screenshot shows the 'Spanish overview' page on localize.drupal.org. The navigation bar and sub-navigation are identical to the previous screenshot. The main content area starts with a section titled 'Spanish overview' with a table showing top contributors. A red box highlights the 'Translate', 'Import', and 'Export' tabs in the sub-navigation. To the right, there's a 'Top contributors' table and 'Translation statistics'.

Using the **Translate** tab **F60.28** we will be able to work together by adding translations of the core or contributed modules. The translations will be added to a translation queue until the moderators or the translation managers review and approve them. Only when they have been approved, the translations added will be available for download in the corresponding file .po.

We have a complete search engine to locate the strings to translate, such as those belonging to a particular module. By clicking **Reveal more filters** it will show us more search options.

Translate to Spanish

The screenshot shows the 'Translate' tab interface. At the top, there are tabs for Overview, Board, Translate (which is selected), Import, and Export. Below the tabs are several filter options: Project (radio buttons for All or a specific project), Release (dropdown for All), Context (dropdown for All), Status (dropdown for <Any>), Submitted by (dropdown for All), Contains (text input field), String ID (text input field), and Limit (dropdown for 10). There is also a 'Clear all filters' button. Below the filters is a page navigation bar with links from 1 to last. The main area is divided into 'Source text' and 'Translations' sections. Under 'Source text', there are three entries: '# flickr', '# photos', and '# sets', each with a link to 'Show related projects'. Under 'Translations', there are three entries for each source text entry, each with a status indicator (green circle with a dot), the translated string, and the date and user who made the translation. There are also edit icons next to each translation entry.

F60.28

Translate tab

From the Translate tab we can collaborate incorporating translation suggestions.

The translations to be incorporated will be validated by a moderator, before becoming part of the translation of the core or module.

Using the **Import** tab **F60.29** we can import the file .po to the project. Doing this will add all suggestions and new translations that we have incorporated into this file. This can be useful if we have translated strings in our site, using the **Translate interface** tool, and we want to collaborate by adding them as translation suggestions.

Download & Extend

The screenshot shows the 'Import' tab interface. At the top, there are tabs for Download & Extend Home, Drupal Core, Modules, Themes, Translations (which is selected), and Installation Profiles. Below the tabs is a sub-tab for Import. There are tabs for Overview, Board, Translate, Import (which is selected), and Export. A section titled 'Gettext .po file with translations:' has a file input field and a 'Examinar...' button. A note below says 'The maximum allowed size for uploads is 20 MB.' Below this is a section titled 'Attribute import to:' with two radio buttons: 'Yourself (forcontu)' (selected) and 'Multiple contributors'. A note below says 'When importing the result of teamwork, If the imported translations were worked on by a team of people, it is common courtesy to not attribute them to your personal username to ensure you do not claim credit for work of the whole team.' At the bottom is a large 'Import' button.

F60.29

Import tab

From the **Import** tab we can help with the translation of modules by uploading many translation files in .po.

Using the **Export** tab **F60.30** we will be able to select the project we want to get the latest available translation. We will get it in a file .po. If we select the Drupal core project we can download the latest version of the translation from the main system.

The really interesting thing about this method is that we can download the translation suggestions, which are translations that have not yet been approved by a moderator. This will allow us to gain a more complete translation (in amount of translated texts). In this case though, it could contain translation errors.

F60.30**Export tab**

In the **Export** we can download a **.po** file with the latest translations available for any project. Add an additional option we can add translation suggestions to the file pending approval.

Download & Extend

Download & Extend Home Drupal Core Modules Themes Translations Installation Profiles

Export Spanish translations

Overview Board Translate Import Export

Project: * Drupal core

Release: * All releases merged

Add Spanish translations

Add Spanish suggestions
For untranslated strings, the first suggestion will be exported as a fuzzy translation. All other suggestions are exported in comments.

Verbose output
Verbose files are ideal for desktop editing, compact files are better for download size.

Export Gettext file

Better download options
Our export and import features are designed for translators who'd like to contribute while on the road/plane or are just more comfortable in an editor on their desktop. If you are just interested in downloading translations, there are much better options, detailed on the [downloads page](#).

Spanish team Admins
Jose Reyero
adiaz
arhak
develCuy

Request moderation permissions

Each group has its own rules for moderating translations. The moderators are fundamental for reviewing and approving translations submitted by other users. If you want to participate as a moderator of the translations, you have to send an application to any of the administrators of the group. In the Spanish translations group you can write a comment on this page by contacting:

<http://localize.drupal.org/node/115>

A

Configuration of the hosting and necessary tools

In this annex we present a short introduction to the tools that will be necessary for managing the hosting. Reference will only be made to the functionalities that are useful for managing the website with Drupal.

Hosting dashboard

A.1

In this annex we present a short introduction to the tools that will be necessary for managing the hosting. Reference will only be made to the functionalities that are useful for managing the website with Drupal.

When we contract a Hosting service, the provider will give us access to a **Dashboard** from where we will be able to administer the different functionalities available, depending on the hosting plan contracted.

Some typical actions that we will be able to do from the dashboard are:

- Check the status of our account: disk space occupied, monthly bandwidth transfer, number of e-mail accounts, number of sub-domains, FTP accounts, Databases, etc.
- Register domains and sub-domains.
- Create FTP accounts.
- Create MySQL users and databases, and administer them using phpMyAdmin.
- Create e-mail accounts with the domains available.
- Configure Cron tasks that will be run automatically every now and then.

There are different dashboard applications for hostings, of which the most well-known are **cPanel** and **Plesk**. The application available will be the one facilitated by the service provider.

In this section we will study some functionalities of the dashboard with the **cPanel** software, although these are fully applicable to other types of dashboards.

Dashboard access

Generally access to the dashboard will be done from the browser via an URL of the www.example.com/cPanel type, with www.example.com being the main domain of our website. This piece of data, along with the user name and password, will be provided by the service provider.

In **Figure FA.1** we see the typical form for access to cPanel.

FA.1

cPanel. Access form

cPanel access form.

Reaction: Regions

Disable theme regions when the context is active.



The dashboard is divided into two main sections. The column on the left usually shows the status of the hosting and the resources used and available. The column on the right shows all of the functionalities available, grouped together by categories. The available settings will vary depending on the services contracted.

FA.2

FA.2

cPanel. Overview

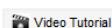
Overview of the cPanel dashboard.



Add domains and sub-domains

From the **Add on domains** setting we can add new domains to the hosting server, which we will have acquired beforehand via a domains provider. When an add on domain is created, we have to state the folder that this domain will point to. A main FTP account will be created along with the domain, with the user name and password that are indicated.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Addon Domains

An addon domain allows visitors to reach a subdomain of your site by typing the addon domain's URL into a browser. This means that you can host additional domains from your account, if allowed by your hosting provider. Addon Domains Subdomains are relative to your account's home directory. The icon signifies your home directory which is /home/cursod7.

Create an Addon Domain

New Domain Name:	<input type="text" value="example.com"/>	
Subdomain/FTP Username:	<input type="text" value="example"/>	
Document Root:	<input type="text" value="public_html/example.com"/>	
Password:	<input type="password" value="*****"/>	
Password (Again):	<input type="password" value="*****"/>	
Strength (why?):	Strong (70/100)	Password Generator
Add Domain		

Hint: This feature must be enabled for your account before you can use it. Addon domains will not function unless the domain name is registered with a valid registrar and configured to point to the correct DNS servers.

FA.3**cPanel. Add a domain**

When a domain is added, an FTP account will be automatically created for accessing the folder that is indicated.

In order for the domain to begin functioning, we have to indicate this to the hosting server. This is done by means of the name servers or DNS. In **Figure FA.4** we see an example of the registration of DNS servers associated with the domain. This configuration is not applied using the dashboard, but rather in the administration area of the domains provider where we have acquired ours.

The DNS names (typically 2) and their corresponding IP addresses must have provided by the hosting provider.

Servidores de nombre DNS

Nombre	<input type="text" value="ns123.exampledns.net"/>
IP	<input type="text" value="67.212.155.221"/>
Nombre	<input type="text" value="ns124.exampledns.net"/>
IP	<input type="text" value="67.212.155.222"/>

[Modificar](#)**FA.4****cPanel. Point to the domain**

We will use the name servers or DNSs to point to the domain of our hosting.

Once a domain has been added, we can also create subdomains (such as for example, web1.example.com, web2.example.com, etc.). We will add these from the Subdomains setting, indicating the name of the subdomain, the associated domain and the folder that this will point to. **FA.5**

Subdomains

Subdomains are URLs for different sections of your website. They use your main domain name and a prefix. For example, if your domain is cursod7.aprendedrupal.es a sub-domain of your domain might be support.cursod7.aprendedrupal.es.

Subdomains are relative to your account's home directory. The icon signifies your home directory which is /home/cursod7.

Create a Subdomain

Subdomain :	<input type="text" value="web1"/>	.	<input type="text" value="example.com"/>	
Document Root :	<input type="text" value="public_html/web1"/>			
Create				

FA.5**cPanel. Create sub-domains**

We can add sub-domains to any available domain or sub-domain that has previously been created.

Once a domain or subdomain has been created, the files uploaded in the corresponding files will be available via the URLs:

- <http://www.example.com>
- <http://web1.example.com>

Create FTP accounts

An FTP account allows us to access the files of particular folders of the server, by using a FTP software (such as **FileZilla**). To create an FTP account we have the **FTP Accounts** setting available, where we will have to state the user name, the password and the folder that he will have access to. Optionally, we can also indicate a quota or limit on the disk space that the account created will use. **FA.6**

FA.6

cPanel. FTP Accounts

The FTP accounts will enable us to manage the files of the server (upload and download files).

FTP Accounts

FTP accounts allow you to access your website's files through a protocol called FTP. You will need a third-party FTP program to access your files. You can log into via FTP by entering

Add FTP Account

Login: course @cursod7.aprendedrupal.es ✓
 Password: ✓
 Password (Again): ✓
 Strength (OK (48/100)) Password Generator
 Directory: /home/cursod7/public_html/course ✓
 Quota: 200 MB ✓
 Unlimited
 Create FTP Account

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
 This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Create MySQL databases

From the **MySQL Databases** setting **FA.7** we can create databases and the users associated with these. **FA.8**

FA.7

cPanel. Create databases

Create New Database

New Database: cursod7_bdcourse ✓
 Create Database

FA.8

cPanel. Create database user

To access a database it is necessary to create a MySQL user, with the right permissions.

MySQL Users

Add New User

Username: cursod7_course ✓
 Password: ✓
 Password (Again): ✓
 Strength (why?): Strong (74/100) Password Generator
 Create User

Add User To Database

User: cursod7_course Database: cursod7_bdcourse Add

The next step will be Add User to database, which means that we will give permissions to the user so that he can access that database, stating the exact privileges that he will have with respect to this. **FA.9**

<input checked="" type="checkbox"/> ALL PRIVILEGES	
<input checked="" type="checkbox"/> ALTER	<input checked="" type="checkbox"/> CREATE
<input checked="" type="checkbox"/> CREATE ROUTINE	<input checked="" type="checkbox"/> CREATE TEMPORARY TABLES
<input checked="" type="checkbox"/> CREATE VIEW	<input checked="" type="checkbox"/> DELETE
<input checked="" type="checkbox"/> DROP	<input checked="" type="checkbox"/> EXECUTE
<input checked="" type="checkbox"/> INDEX	<input checked="" type="checkbox"/> INSERT
<input checked="" type="checkbox"/> LOCK TABLES	<input checked="" type="checkbox"/> REFERENCES
<input checked="" type="checkbox"/> SELECT	<input checked="" type="checkbox"/> TRIGGER
<input checked="" type="checkbox"/> UPDATE	

Make Changes

FA.9**cPanel. Assign permissions to the database user**

When the database is assigned to the user we have to state which privileges he will have over it.

Practical Case 31.1 Adding Action

In the SUBJECT field of mail, we may use replacement patterns.

Any available variable may be used, using brackets [variable].

Once the user privileges have been granted, this will be ready to make use of the databases, whether directly or in any application that makes use of databases, such as is the case with Drupal.

Enable the Cron

We can create tasks that will periodically run commands from the **Cron jobs** setting. Of the two methods available, **Standard** and **Advanced**, we will only see the **Standard** option in this course, which is sufficient to enable and configure the Drupal cron. **FA.10**

Cron Jobs

Cron jobs allow you to automate certain commands or scripts on your site. You can set a command or script to run at a specific time every day, week, etc. For example, you could set a cron job to delete temporary files every week to free up disk space.

Warning: You need to have a good knowledge of Linux commands before you can use cron jobs effectively. Check your script with your hosting administrator before adding a cron job.

Figure A.11 shows the form for configuring a cron task. We must firstly enter the command that is going to be run.

Generally speaking, the Linux curl command is used to load an URL, and hence the full command for running the cron of our site (www.example.com) will be:

curl http://www.example.com/cron.php

We will then select the time interval that we want for running the cron on our site.

In **Figure FA.11** this has been configured to run every day at 3:05 in the morning. As far as possible it is recommended that the cron is run at times at which the site activity level is low, for example in the early hours of the morning.

FA.10**cPanel. Cron Mode**

We will only see the Standard cron configuration mode.

Practical Case 31.1 Adding Action

Lastly, we will indicate that the mail is sent from the current user in the FROM field.

Practical Case 31.1 Adding Action

Lastly, we will indicate that the mail is sent from the current user in the FROM field.

FA.11**cPanel**

The curl command allows us to run any URL of the system. To configure the cron, it is sufficient to state which periods we want to run it in.

Practical Case 31.1**Adding Action**

An e-mail will be sent to the node author where the comment has been published.

We will find the variable that has the node author's e-mail in the TO field.

The screenshot shows the 'Cron Email' section of the cPanel interface. At the top, there is a link to 'Send an email every time a cron job runs.' Below this, the 'Add New Cron Job' form is displayed. The 'Common Settings' dropdown is set to 'Every 5 minutes (*/5 * * * *). The 'Minute' field is set to '/5' and 'Every 5 minutes (*/5)' is selected. The 'Hour', 'Day', 'Month', and 'Weekday' fields all have '*' selected and 'Every hour (*)', 'Every day (*)', 'Every month (*)', and 'Every weekday (*)' are selected respectively. There is a large empty 'Command:' input field at the bottom. A 'Add New Cron Job' button is located below the command field.

If the **curl** command does not function on your hosting, check which command you should use with your provider, because this may vary depending on the operating system and the configuration of the server.

Once the cron has been enabled, check in the site **status report** that the cron is being run correctly, in the set time periods.

A.2

Managing databases with phpMyAdmin

From the dashboard it is possible to access the management of the databases created using the **phpMyAdmin** software. To enlarge upon the information about this tool, please consult the official page of **phpMyAdmin** at <http://www.phpmyadmin.net>. **FA.12**

The main settings that we will find in **phpMyAdmin** are:

- **List of databases and tables** (in the left-hand column). This makes it possible to select a particular database and/or table.
- **Structure**. If a database is selected, this displays the set of tables that makes it up. If one particular table is selected, it displays the fields and items that make it up.
- **Query**. This allows us to navigate through the data or records contained in the selected table.
- **SQL**. This allows us to launch SQL sentences (select, insert, update, delete, etc.).
- **Search**. Help for searching for data contained in the table, using different filters related to the fields of the table.
- **Insert**. Form for inserting data or logs into the table.
- **Export**. This makes it possible to export a database or a table. In both cases it is possible to export from the structure, from the data, or both. To make a database backup we will carry out an export of both the structure and of the data. The resulting file is a file with SQL sentences, which can be used in order to Import, into the same database or into another one.
- **Import**. This allows us to import an item and data from previous exports (SQL files).
- **Operations**. This allows us to carry out additional operations on a database or on a table (change name, copy, etc.).

- **Clear.** Deletes all of the data from a table.
- **Delete.** Deletes a table (structure and data).

The screenshot shows the phpMyAdmin interface with the database 'bdcursoso' selected. The 'Structure' tab is active. A table named 'node' is listed under the 'Table' column. To its right, there is a column titled 'Action' containing several icons: 'Browse', 'Structure', 'Search', 'Insert', 'Empty', and 'Drop'. Below the table list, there is a message: 'Showing rows 0 - 14 (~15 total) (Query took 0.0004 sec)'.

In **Figure FA.13** we see the content of the **node** table of Drupal, once the contents have been created on the site. To access the content of the table we have first selected the database, then the **node** table and lastly the **Query** tab.

The screenshot shows the phpMyAdmin interface with the 'Query' tab selected. The SQL query displayed is: `SELECT * FROM 'node' LIMIT 0 , 30`. Below the query, there are various configuration options: 'Show : 30' (row(s) starting from row # 0), 'horizontal' mode, and '100' cells. There is also a 'Sort by key: None' dropdown and a '+ Options' button. The results table displays eight rows of data, each with columns: nid, vid, type, language, title, uid, status, and created. The data includes articles like 'About us', 'First article', 'Second article', 'Headline 1', 'About me', and a poll question. At the bottom of the table, there are edit, inline edit, copy, and delete links for each row.

FA.12

phpMyAdmin

Main view once a database has been selected.

Practical Case 31.1

Adding Action

We can also use replacement patterns in the MESSAGE field of mail.

Practical Case 31.1

Adding Action

We can also use replacement patterns in the MESSAGE field of mail.

A.3

Upload and download files via FTP

The software that enables us to upload and download files to a remote server is known as **FTP (File Transfer Protocol)**. There are many alternatives, and they all work in practically the same way. Here we will analyse how the **FileZilla** software works, as this is freeware that is widely used.

FileZilla is available for downloading at <http://filezilla-project.org/>, by using the *Download FileZilla Client* link.

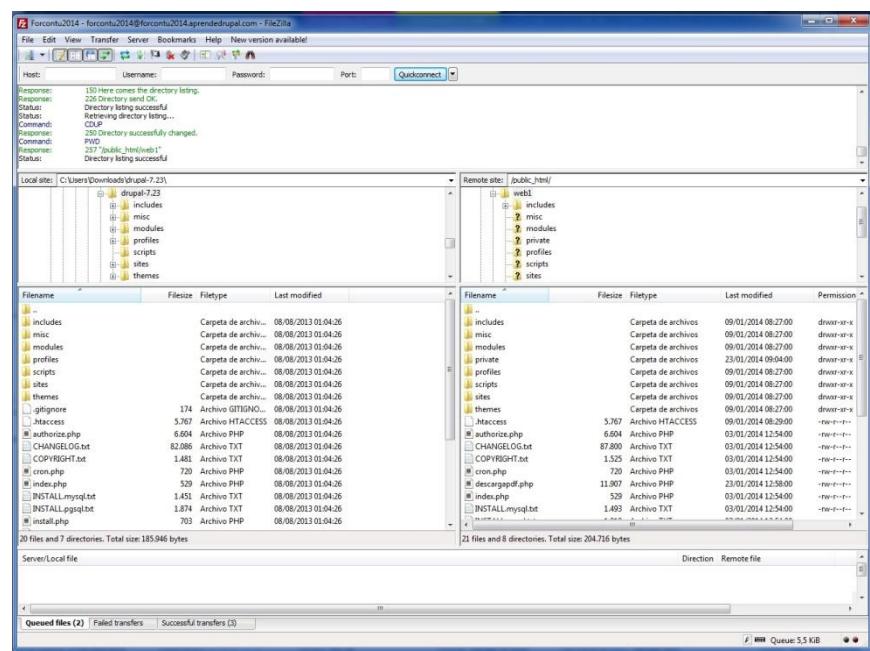
In **Figure FA.14** we see an overview of the software. The central part of the window is split in two. On the left we see the files and folders of our operator (local) and on the right we see those belonging to the server. We can browse through both structures, and move the files and folders from one site to another one.

FA.14**FileZilla**

FileZilla is a piece of freeware for uploading or downloading files by FTP.

Practical Case 31.1**Testing the Rule**

The rule will be executed when the registered user publishes a comment. The node author will receive an e-mail like the one shown.

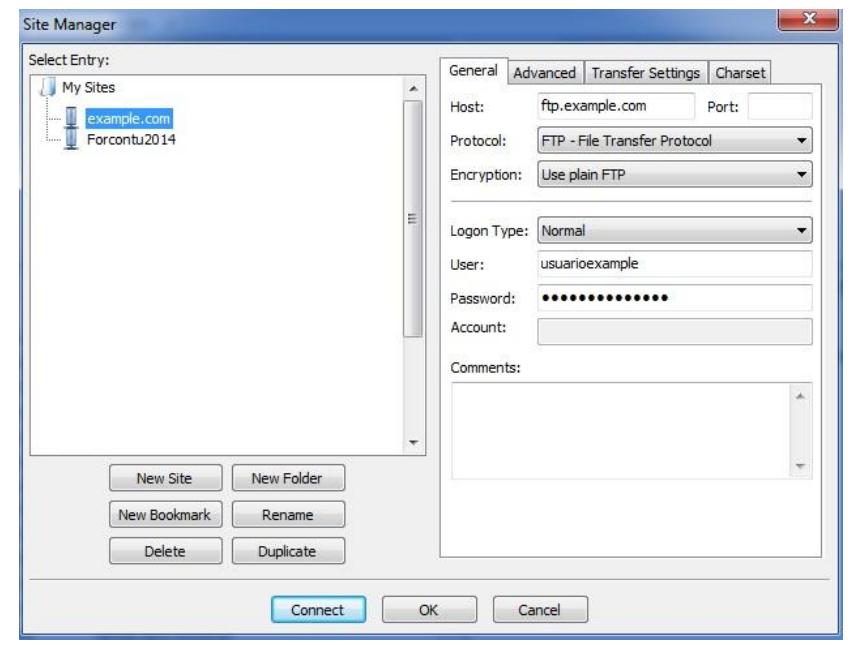


Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

Set up an FTP connection

In addition to the server's name or IP address, to set up a connection with an FTP server we need a valid user and password. In **section A.1** we saw how to create an FTP user to access the files of our hosting server.

From **File⇒Site manager** we can access the connections management window that enables us to create and save the usual connections. **FA.15**

**FA.15****FileZilla**

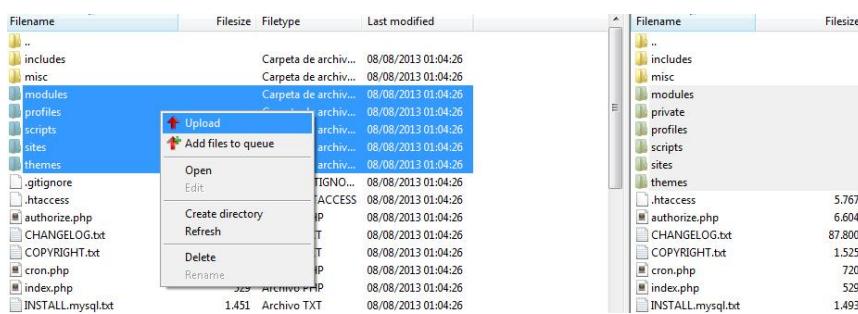
The **Sites manager** enables us to create and save the usual connections. **Filter rules**

We can filter by event or tag from the rules administration area.

Upload and download files

We will take the following steps to **upload files to the server** from our local computer: **FA.16**

- In the window on the right (server) we will move through the structure of folders till we reach the folder that we want to upload the files into.
- In the window on the left (local computer), we will move in the same way through the structure of folders till we locate the files that we want to upload to the server. We will then select the files and folders and by clicking on the right-hand button we will choose the **Upload** option.

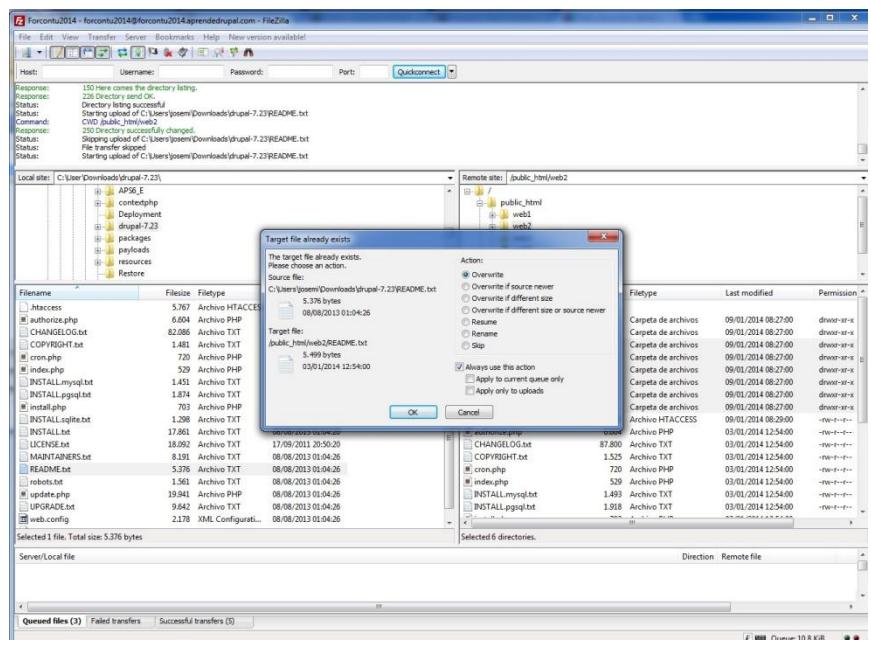
**FA.16****FileZilla**

To **Upload files** to the server, select the files to be uploaded on the left (local).

If the files or folders exist on the server, a window will be displayed that asks us whether we want to overwrite the files. The 'Always use this action' option will stop us being asked this question for every individual file. **FA.17**

FA.17**FileZilla**

Use the "Always use this action" option to overwrite multiple files.



Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

To **download files from server** to our local computer, we will take the following steps: **FA.18**

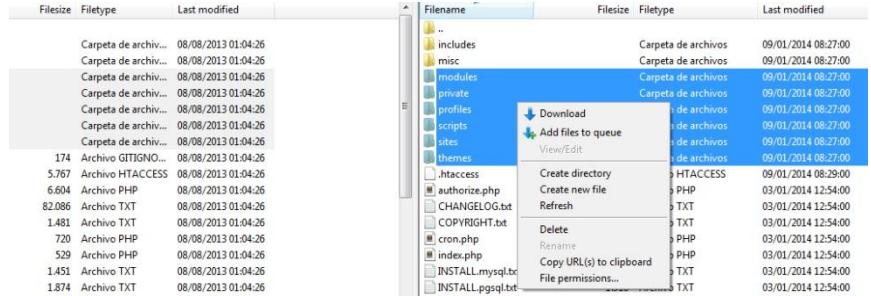
- In the window on the left (local computer), we will move through a structure of folders until we reach the folder that we want to download the files to.
- In the window on the right (server), we will move in the same way through the structure of folders till we locate the files that we want to download. We will then select the files and folders and clicking with the right-hand button we will choose the Download option.

FA.18**FileZilla**

To **Download files** from the server, select the files to be downloaded on the right (server).

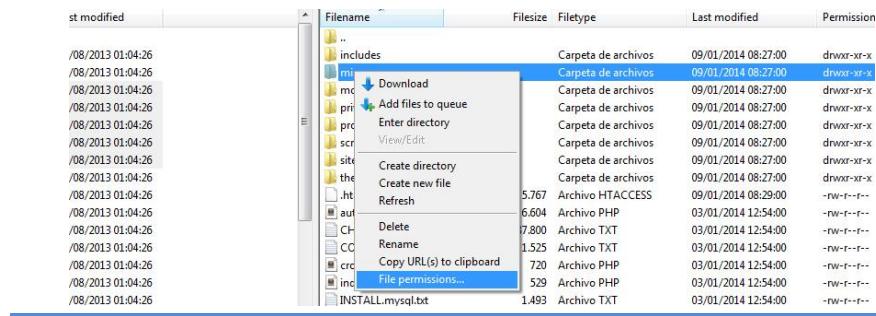
Login one time Module

The e-mail containing the one-time use access link that will be sent to the user, can be edited from Account Options



Modify the permissions of files and folders of the server

A value is shown on the server side, along with each file or folder, that relates to the permissions that are established (775, 644, etc). To modify the permissions of a file or folder, select it and click with the right-hand button, and then choose the **File permissions** option from the dropdown menu. **FA.19**

**FA.19****FileZilla**

Access File permissions to modify the file reading, writing or run permissions

Login one time Module

When sending a link to a user we will establish what page the user will be redirected to when accessing the site.

In **Figure FA.20** we see the permissions that can be assigned to a file or folder, which can be read, write or run. We have to take special care with the write permissions.

FA.20**FileZilla**

Special care must be taken with the writing permissions that we assign to the files. Drupal protects some files and folders, and so if we want to make any modification in these we will first have to modify the permissions using FileZilla.

Login one time Module

Configuration of the elements that will be displayed on the user's profile page.

Drupal protects certain folders and files of the system (such as /sites/default and /sites/default/settings.php) so as to stop it being possible for those files to be modified maliciously. If we want to create a folder within /sites/default, we first have to change the permissions to the default folder, in such a way that allows for reading (owner permissions and group permissions, generally speaking). Once the folder (for example, modules or themes) has been created, we will be able to change the permissions of the default folder again.

The same will happen when we want to modify the settings.php file. It will be necessary to modify the permissions of both the folder that contains it (default) and of the settings.php file itself. Once we have made the changes we will apply the initial permissions again.

A.4

Uncompress files with 7zip

During the course you will find that the files downloaded from drupal.org are compressed in the **.tar.gz** format. We can use any file compression and uncompression software to uncompress these files, such as WinZip or WinRAR. In this section we will see how to use **7zip**, a piece of freeware available at <http://www.7-zip.org/>.

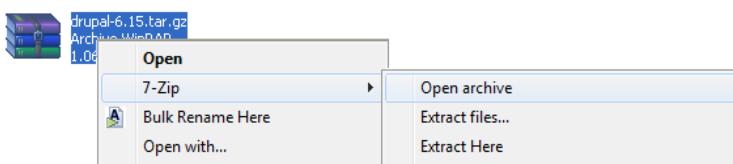
To uncompress a **.tar.gz** file, it is necessary to take account of the fact that this is a dual compression: first the files and folders are grouped together into one single file (**tar**) and then the resulting file (**gz**) is compressed. This is why we will have to do a dual uncompression.

FA.21**7zip**

Uncompress a **.tar.gz** file with **7zip**.

Login one time Module

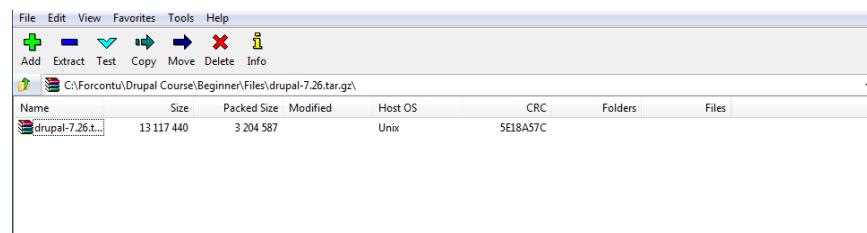
Users with permission to send access links will see the corresponding button on each users profile, along with the rest of the options that have been selected in the module configuration.



When the compressed file is opened, as shown in **Figure FA.21** (right click on the file and open with **7-zip** or first opening **7-zip** and selecting the file to decompress), the file will be uncompressed on one occasion, and giving the result of the **.tar** file. **FA.22**

FA.22**7zip**

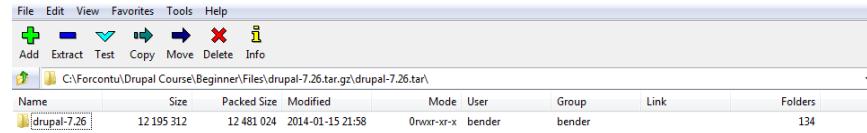
In the first unzipping we obtain the **.tar** file, which continues being a compressed file.



We will reach the end folder by double clicking on the **.tar** file. From this point we will browse through the structure of files and folders contained in the compressed file and uncompress or extract just the files that we want. To do this it is enough to select them and drag them to the folder we want (on our computer) or to click on the **Extract button**, indicating the destination folder.

FA.23**FA.23****7zip**

On the website, by double clicking on the **.tar** file we arrive at the content of this. To finish the unzipping we can use Extract or drag and drop the content into any folder of the system.



Editing text files with Notepad++

A.5

Notepad++ is a freeware text editor for Windows, available to be downloaded at the URL <http://notepad-plus.sourceforge.net>. It is a very useful editor for editing PHP, HTML and/or CSS files, because this shows in the syntax in colour, the line number and the nestings between items. It is moreover possible to choose the language in which the file is written from among a large range of options (PHP, CSS, XML, HTML, SQL, etc.).

Figure FA.24 shows us a CSS file, edited with **Notepad++**.

```

    /*C:\Forcontu\themes\style.css - Notepad++*/
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
style.css
73
74  p {
75      margin: 0.6em 0 1.2em;
76      padding: 0;
77  }
78
79  /* Default Links */
80
81  a:link,
82  a:visited {
83      color: #027AC6;
84      text-decoration: none;
85  }
86
87  a:hover {
88      color: #0062A0;
89      text-decoration: underline;
90  }
91
92  a:active,
93  a.active {
94      color: #5B95B6;
95  }
96
97  hr {
98      margin: 0;
99      padding: 0;
100     border: none;
101     height: 1px;
102     background: #5294c1;
103 }

```

FA.24

Notepad++

Notepad++ shows the structure of the file in diagrammatical file, colouring the key words depending on the type of the language: PHP, HTML, CSS, etc.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

B Links and resources of interest

In this annex you will find links to pages and resources of interest so as to properly follow this course and to learn Drupal in general.

forcontu.com

B.1

www.forcontu.com is the official page of the course. From this page, you will be able to download the materials, access the classrooms and obtain **additional resources**.

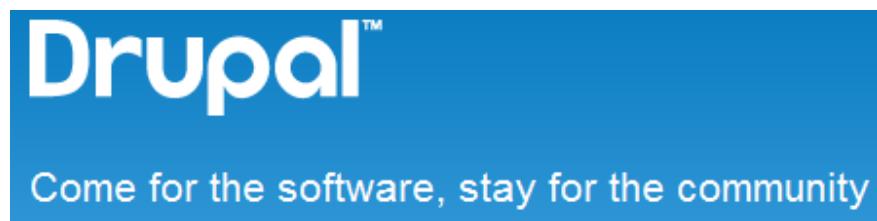


Furthermore, you will be able to **contract other services that are supplementary** to this book:

- **Supervised course.** By contracting the supervise course, you will have the continuous support of the tutors, who will guide you in carrying out the activities proposed and in your personal project. The customised course includes the student assessment and the certificate of **Expert in Drupal 7**, once you have passed this. In addition, you will be able to contract this by separate or grouped levels, depending on your needs and previous knowledge.
- **Continuous training.** You do not stop learning Drupal when the courses end. Sign up for continuous training so as to continue periodically receiving Activities, Practical Cases and all of the new features on new modules, themes and about Drupal in general. You get all of this with the same training focus as that applied in the course.
- **Consultancy.** If you consider that you need help to complete your website project or start a new project once you have finished the course, you will be able to contract the Consultancy service. This service is conceived of to help you, from a training focus, to make the best choice about modules and the configuration of your site, depending on the particular project that you are going to carry out. Your advisors will also help you to resolve the errors and conflicts that you may encounter when you install and configure new modules.
- **Website development.** At Forcontu, we have made a clear investment in sustainable development, and this is what we want clients to actively participate in the process of building their projects. In this way, we aim to convey to you the need to keep your project alive, adding new functionalities and innovations that are being adapted to the inevitable progress of the business and of the market, thus achieving an alignment between the business and the ITC tools that are implemented. If your company needs to develop a website project, do not hesitate to contact us.

B.2**Official Drupal page, drupal.org**

On the official Drupal page, drupal.org, you will find very many of the resources studied (in English) on the course and that are necessary to carry out the installation and enlargement of Drupal.



- **Latest Drupal version.** You will be able to download the latest Drupal version from the site home page.

- **Additional modules.** From the **Modules** section, you can consult and download all of the free modules that are available.

<http://drupal.org/project/modules>

- **Graphical themes.** You can consult and download all of the graphical themes available from the **Themes** section.

<http://drupal.org/project/themes>

- **Translations.** You can consult and download all of the available translations of the core from the **Translations** section (section B.6).

<http://localize.drupal.org>

- **Forums.** In the Drupal forums you will find discussions (in English) about different issues related to Drupal (installation, updating, development of modules and themes, etc.).

<http://drupal.org/forum>

B.3**drupalmodules.com**

The drupalmodules.com website is the site that is recommended for searching for and downloading Drupal modules (in English). Unlike with drupal.org, this site is completely focused on modules for Drupal and it incorporates filters and search engines that make it possible to locate modules more smoothly and in a more straightforward way. In addition, it incorporates graphics that show the functionalities, security, ease of use and documentation on a scale from 1 to 5 for every module and, in some cases, comments from the users. The repositories are synchronised, and the versions of the modules available on both sites are completely identical. **FB.1**

The screenshot shows the homepage of drupalmodules.com. At the top, there's a navigation bar with links like Home, Latest Reviews, Top Rated, Most Downloaded, Most Favorited, Category List, Rating Guide, Forum, and About. Below the navigation is a search bar titled "Search with Module Finder" with fields for Category (Content), Version (7.X), Title, Body, and a search button. To the right is a "Search Modules" section with a search input field and a "Restrict search to:" dropdown menu containing options for Drupal 6.x, 5.x, 7.x, Modules, and Reviews. Further down is a "User login" section with fields for Username and Password, and a "Log in" button. On the left, there's a "Date" module summary, a "Wysiwyg" module summary, and a sidebar with links for Users, Status, Categories, and Downloads.

FB.1**Repository of modules at drupalmodules.com**

The drupalmodules.com website is exclusively orientated towards the Drupal modules. A lot of modules can be found with very useful search and query tools.

Drupal Association**B.4**

The Drupal Association is an organization dedicated to helping the open-source Drupal CMS project flourish. We help the Drupal community with funding, infrastructure, education, promotion, distribution and online collaboration at Drupal.org.

FB.2

Funds to support these programs and the Association staff come from memberships, supporting partners, sponsorships, donations, and volunteers.

If you want to work together with the Drupal Association you can become a member from <https://association.drupal.org/>.

The screenshot shows the homepage of the Drupal Association. The header includes the "Drupal Association" logo and links for Blog, About, DrupalCon, Contact, DA Info, and Staff. Below the header is a search bar. The main content area features a video player with the title "Why Drupal Association Membership Matters - Video". To the right of the video are sections for "How Can You Help?", "Become a Member", "Donate Directly", "Make a Donation", and "Upcoming DrupalCons".

FB.2**Drupal Association**

Website of the *Drupal Association*.

B.5 Drupal Group on Google

Join the ***Google Drupal*** group, a meeting point for making technical enquiries and getting to know other members of the community. You will just need a Google account to be able to take part.

<http://groups.google.com/group/drupal>

FB.3

FB.3 Drupal Group

You will find the Drupal group in Google Groups.

Copyright 2011-2015 Forcontu S.L. All Rights Reserved. No part of this book may be reproduced or transmitted in any form.
This book was prepared exclusively for Nidhi Badani. Verification code: D7AVZPDFEN00037737007079

B.6 Translations of Drupal, localize.drupal.org

Localize Drupal (<http://localize.drupal.org/>) is a new website created by Drupal and that is wholly devoted to translating both the Drupal core and the additional modules shared by the community into many languages. The translations are also done in community, where the participants propose translations that are then accepted and incorporated into the site.

FB.4

FB.4 localize.drupal.org

The Localize Drupal website is a new Drupal website dedicated to the translation of Drupal and of the additional modules.

Drupal API, api.drupal.org

B.7

An API (Applications Programming Interface) is a library of functions that makes it possible to access particular data and functionalities of the system with no need to understand their internal structure. The Drupal API contributes communication functions with the system and with the core modules. The details of the Drupal API are available at api.drupal.org. Consult the **Advanced Level of the course.**

FB.5

The screenshot shows the Drupal API documentation for the function `drupal_get_path_alias`. The page has a blue header with the Drupal logo and navigation links for 'Drupal Homepage', 'Your Dashboard', 'Logged in as forcontu', and 'Log out'. Below the header, there's a 'Community Documentation' section with tabs for 'Docs Home' and 'API'. The main content area shows the function signature and its description: 'Given an internal Drupal path, return the alias set by the administrator. If no path is provided, the function will return the alias of the current page.' Below this, there are 'Parameters' sections for '\$path' and '\$path_language'. On the right side, there's a sidebar with a search bar labeled 'Search 7' and a 'Function, file, or topic: *' input field, along with a 'Search' button. Below the search bar is an 'API Navigation' section with links to 'Drupal 7', 'Constants', 'Classes', 'Files', and 'Functions'.

FB.5

api.drupal.org

The Drupal API is necessary to schedule Drupal modules. This will be exclusively studied on the Advanced Level of the course.

Drupal paid-for themes

B.8

In addition to the repositories of paid-for themes, it is possible to find sites that offer paid-for themes. Some of these sites are:

- **Template Monster**, <http://www.templatemonster.com/drupal-themes.php>
- **ThemeForest**, <http://themeforest.net/category/cms-themes/drupal>

Generic texts for tests

B.9

When we design a website it may be very useful to have generic texts of tests available that simulate actual texts with paragraphs, lists, etc.

One standard that is very widespread among website design and layout professionals on Internet is the text known as **Lorem Ipsum**. Through the www.lipsum.com page, it is possible to generate **Lorem Ipsum** text blocks with different characteristics (number of paragraphs, number of words, number of characters or lists).