eZ publish basics

Created 26/11/2003

Table of contents

eZ publish basics

This chapter describes the basic terms, models, structures and building blocks of eZ publish. In order to understand and to be able to use eZ publish correctly, a person must know the basics. This is important. The reader should devote some time to this chapter. It is a good idea to make notes of the nomenclature and the techniques that are described here. This will make it easier to grasp the amazing possibilities of eZ publish.

The internal structure of eZ publish

This section describes the internal structure of eZ publish by presenting an brief overview of the different software-layers of the system. In addition, an overview and explanation of the directory structure is also presented.

Directory structure

The eZ publish root directory is divided into multiple subdirectories. Each subdirectory is dedicated to a specific part of the system. The subdirectories contain a collection of logically related files. The structure is clean and should be easy to understand.

bin	Contains various Perl and shell scripts. These scripts are mainly used for maintenence purposes.
design	Contains all design related files such as templates, banner/site-images, stylesheets, etc.
doc	Contains documentation and change logs.
kernel	Contains all the kernel files such as classes, views, data types, etc. This is where the core of the system resides. Only experts should tamper with this part.

lib	Contains the general purpose libraries. These libraries are collections of classes that perform various tasks. The kernel makes use of these libraries.
settings	Contains dynamic, site specific configuration files.
share	Contains static configuration files such as code pages, locale information and translation files.
var	Contains cache files and logs. In addition, it also contains images and files (site specific content). Obviolusly, this directory will grow in size.

Kernel, libraries and modules

eZ publish is a complex, fully object oriented application written in the PHP language. The system basically consists of three parts:

- Modules
- Kernel
- Libraries

The libraries
The libraries are the main building blocks of the system. These are highly reuseable general purpose PHP classes. The libraries are in no way dependent on the eZ publish kernel. People looking for general PHP libraries should take a look in the "lib" folder within the root directory of an eZ publish installation. Appendix A contains a complete list and a short description of the currently available libraries.

The kernel
The eZ publish kernel can be described as the system core. It takes care of all the low level functionality like content handling, workflows, version control, access control, etc. Basically, the kernel is a collection of engines, it builds upon and makes use of the libraries.

The modules
A module can be thought of as a collection of functionality. It may be described as an interface to a an engine inside the kernel. Each module has a collection of functions that take care of various tasks. Appendix B contains a complete list and a short description of the currently available eZ publish modules.

Content and design

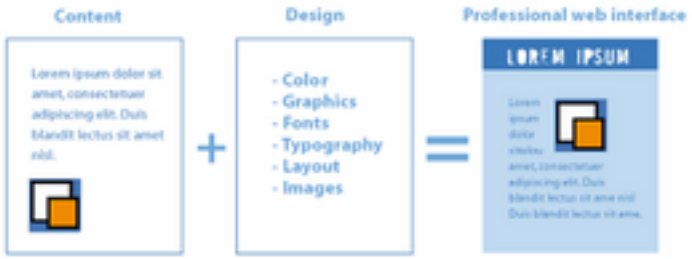
In the world of eZ publish, content and design is separated.

By content we mean information that is to be organized and stored in some structure (for later retrieval). It may be the actual contents of a news article (title, ingress, body, images), the attributes of a person (name, birthday, address, facial photo) and so on.

Obviously, the information stored in a content structure must be presented somehow, preferably in a way that is easily understood by humans. While content means "raw data", design is all about the way the data is visually presented. When talking about design, we're talking about the things that make up a nice interface: style sheets, banner images, layouts, etc.

The separation of content and design

The following diagram illustrates the elegant separation of content and design - and how these things can be combined to form a professional interface.



This distinction, and the system's ability to handle it elegantly, is one of the most important qualities of eZ publish. The separation of content and design opens up an entire array of possibilities that simply cannot be achieved otherwise. To name a few:

- Content authors and designers can work separately without conflicts.
- Content can be published easily in multiple formats.
- Content can easily be transferred and re-purposed.
- Global redesigns/changes can be applied by simple modifications.

Templates

eZ publish uses templates as the fundamental unit of site design. A template is basically an extended HTML file that describes how some particular type of content should be visualized. The extension is a powerful set of eZ publish specific template functions and operators. For instance, a template might dictate that a page should appear with blue borders, the site's title bar on the top, and then a handful of content elements in the middle of the page. When the page is accessed, it then becomes the content management systems' job to "flow" the content into the appropriate places in the template.

Storage

eZ publish structures and stores content inside a database. This is true for all content except images and raw/binary files. These are stored on the filesystem to gain speed. Everything that has to do with the design (template files, CSS files, banner/site-images, etc.) is stored on the filesystem.

Content management in eZ publish

Every file that is created, whether it is a document, financial transaction, report, memo, video file, image, purchase order, data back-up or customer contact - is classified as "content".
The role of a content management system is to organize every single element of this content - regardless of its type and complexity. This provides organizations with a structured, automated yet flexible solution allowing information to be freely distributed and instantly updated across various communication channels such as the world wide web, intranets and miscellaneous front and back-end systems.

Within eZ publish, all content is managed through a graphical user interface called the *administration interface*. This interface is described in detail in "The administration interface" chapter.

This section describes how eZ publish structures and manages content.

A note about object oriented technology

Superficially, object-oriented means nothing more than looking at the world in terms of objects. In real life, people are surrounded by several objects: furniture, cars, pets, humans, etc. Each of these objects have traits that we use to identify them. This is also the way content is described and managed within eZ publish, through abstraction and objects.

The content structure

Unlike other content management systems, eZ publish does not make use of a predefined "one-size-fits-all" content model. Instead, it allows the site administrator to create his/her own content structure using a unique object oriented approach. This makes it easier to structure, store, retrieve and present custom data. Instead of desperately trying to fit data into some predefined, rigid structure, we simply create a structure that fits our data. However, is it unlikely that everybody needs to create a custom content structure. This is why eZ publish comes with a set of ready-to-use structures and datatypes. It is also possible to modify and extend the built in structures. In other words, eZ publish offers both ready-to-use and custom content structure. This is one of the features that make eZ publish an extremely flexible and successful system.

Datatypes

A datatype is the smallest possible entity of storage. eZ publish comes with a collection of fundamental datatypes that can be used to build powerful and complex content structures. In addition, it is possible to create custom datatypes for special needs. Some of the default datatypes are: text string, XML text, image, binary file, price, etc. Appendix C contains a complete list of the built-in datatypes.

The content class

A content class is simply a definition of an arbitrary data structure. A content class doesn't store any actual data. As mentioned earlier, eZ publish comes with a set of predefined, ready-to-use content classes such as "folder", "article", "user account",
"image", etc. These content classes are carefully designed to fit the most common generic/everyday tasks of a CMS. In addition, it is possible to create custom content classes using the administration interface. The content classes (both custom and built-in) may be easily modified and/or extended at any time. Appendix D contains a complete list of the built-in content classes.

Content class attributes
Each content class is made up of attributes, hence a collection of attributes define the actual data structure of a content class. An attribute can be a text string, an integer, a float, an image, etc. The type of the attribute is determined by the datatype that is chosen for that specific attribute. In other words: each content class is made up of one or more datatypes. Content classes can be built and modified easily using the administration interface.

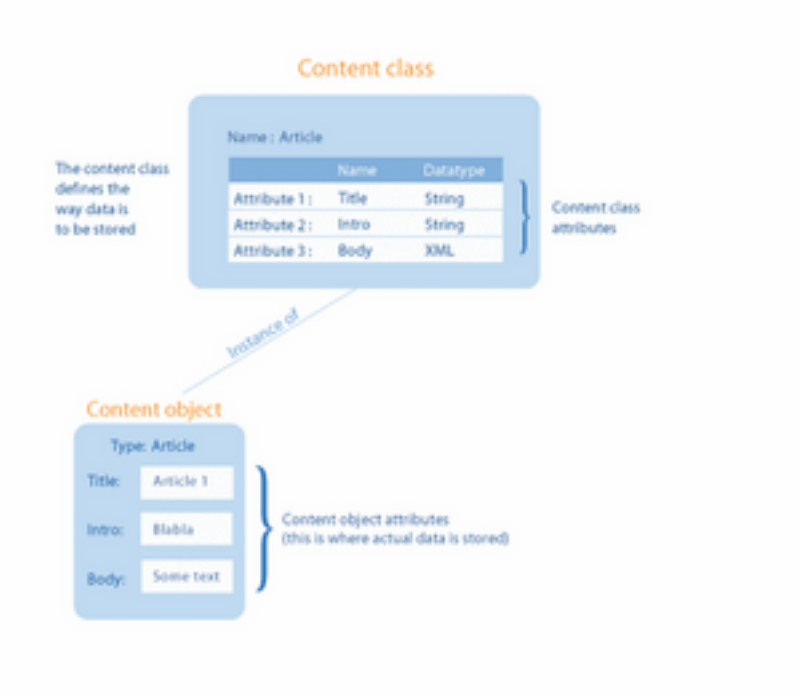
Example
Lets assume that we would like to store information about fruits. The information we wish to store is the name of the fruit, its weight and color. We could easily create and build a content class called "Fruit" with three attributes: name, weight and color. The string datatype could
be used to represent the name and the color while the weight attribute could be represented using the integer datatype:

```
Content class: Fruit

             Attribute name    Datatype
-------------------------------------
Attribute 1: Name              Text
Attribute 2: Weight            Integer
Attribute 3: Color             Text
```

The content object

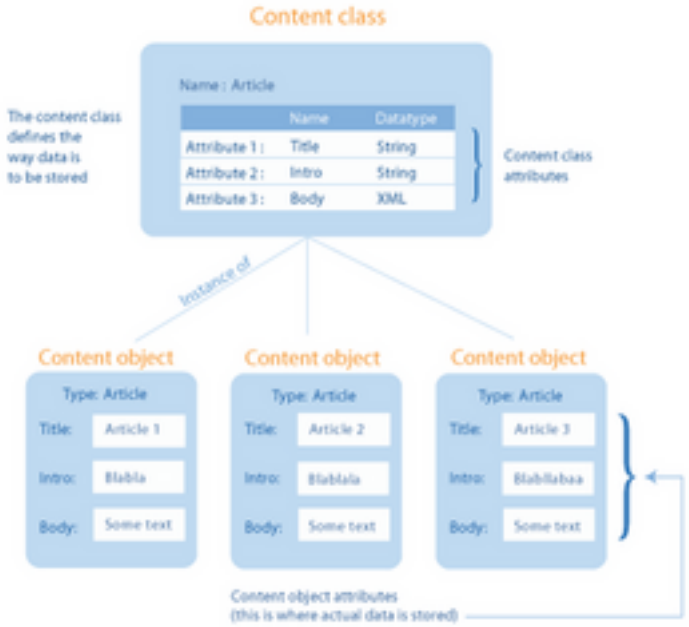
A content object is an instance of a content class. A content class simply defines the structure of content objects using content class attributes. It is the content objects themselves that contain actual data/information. Once a content class is defined, several content objects of that type can be created. Multiple objects of the same type are used to store similar data/information, for instance: multiple article objects are used to store different news articles.



Content object attributes
Each content object consists of several independent attributes. These attributes are defined by content class attributes. Inside a content object, all data is encapsulated in one or more content object attributes.

Relation between datatypes, classes and objects

The following diagram summarizes and illustrates the relation between datatypes, a content class and a collection of content objects. The content objects are instances of the same content class; which basically means that they are the same type, but they contain different content/data/information.

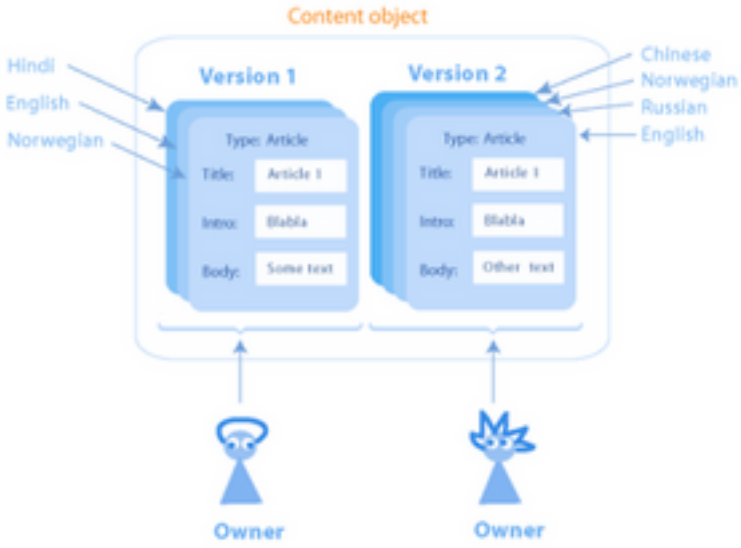


Content object versioning

The actual content that is stored inside a content object can exist in one or more versions. Each time the content is modified, a new version is created. The old version remains untouched. This is how eZ publish keeps track of changes made by various users. The versioning system makes it possible to revert/undo changes. Each version may belong to a different owner. Only one version of an object can be considered "published", this is called the current version. Using journalists and articles as an example, the following diagram presents a rough illustration of how the versioning system works.

Support for multiple languages

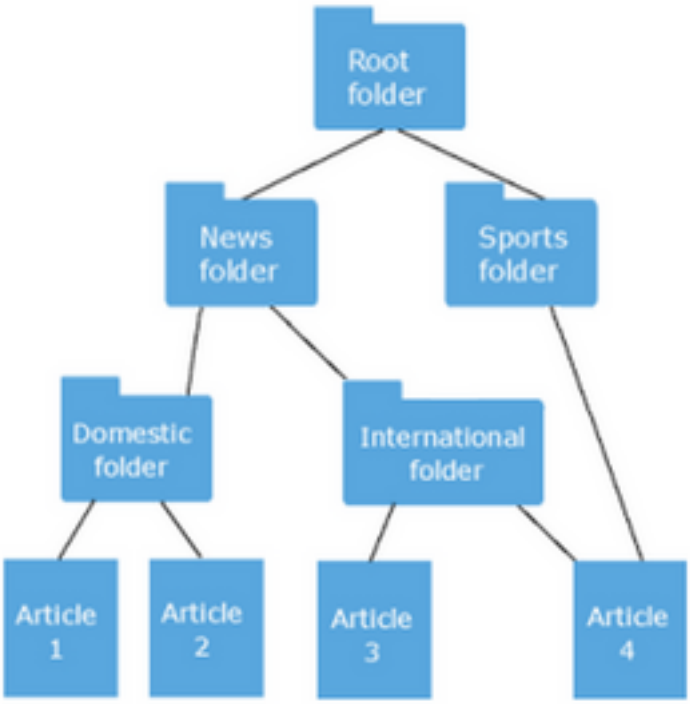
In addition to the versioning system, eZ publish also provides a powerful multi language solution. It can be thought of as the third dimension of a content object. Each version of some arbitrary content can exist in several languages. This is an extremely neat feature. It basically provides a generic, out-of-the-box multilingual framework that can be used with any kind of content. The diagram below illustrates how the multi language solution is built into the versioning system.



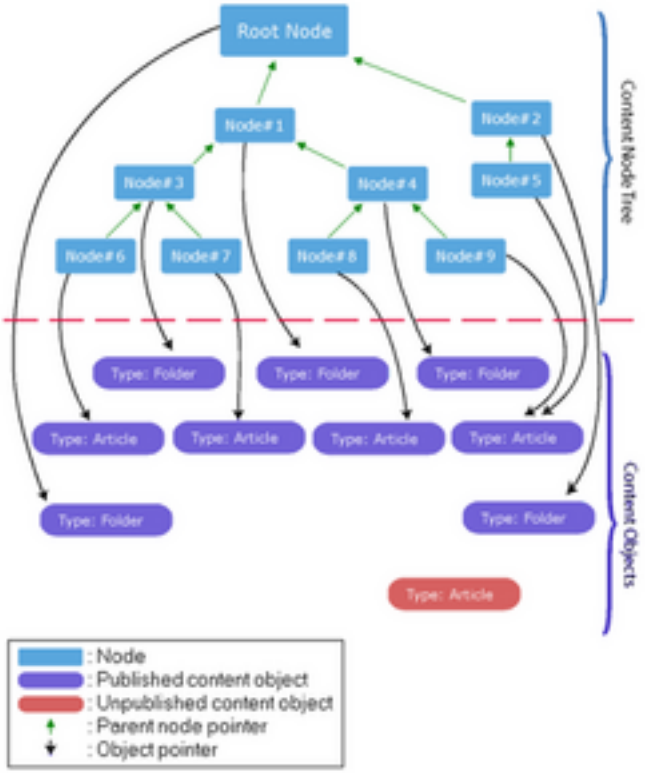
Objects, nodes and the content node tree

A content object is an instance of a content class. When the system is in use, new content objects are created on the fly. For instance, when a news article is composed, a new content object is created to store the actual contents of the article (the object will also take care of storing all the different versions and languages of the article). Obviously, the content objects can't just fly around in space. They have to be organized and structured in some way. This is where the nodes and the content node tree comes in.

A node can be thought of as an encapsulation of a content object. The content node tree is built up of nodes. It is a hierarchical tree structure representation of all content that belongs to a site. In other words, this is the mechanism that is used to organize every content object that is present in the system. A usual way of categorizing content within the tree is by making use of containers (for example folders) under which relevant content objects are placed (roughly in the same way as on a filesystem).



When using the administration interface to manage folders, articles, files, images, etc., the user inter acts with the content node tree. The tree can be easily used to generate a sitemap. The following diagram illustrates the relation between content objects and nodes.

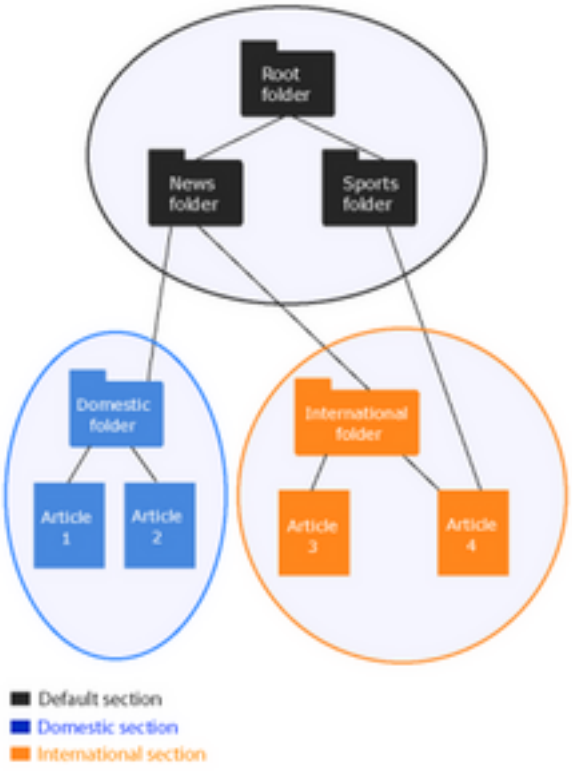


Each leaf in the content tree is a node. At the minimum, the tree consists of one node, which is

called the root node. This is the actual root folder under which all content is placed. Each node (except the root node) has two pointers, one that points to a parent node and one that points to a content object. A node can only point to one parent node and one content object. However, a content object can be pointed to by several nodes, which means that the same object may appear at different locations within the tree. An unpublished content object (draft) doesn't have a node associated to it. In other words, only published objects appear in the content node tree. Archived content objects will also lose their node and thus disappear from the tree. Because of the node-encapsulation of objects, any type of content object can be placed anywhere in the tree (but will still exist in the system). The number of levels in the tree is unlimited. Clearly, this is an extremely flexible and generic solution that can be used to structure any kind of custom content.

Sections

As mentioned in the previous section, the content node tree is a hierarchical tree representation of all content that belongs to a site. A usual way of categorizing content within the tree is by making use of collectors (for example folders) under which relevant content objects are placed (roughly in the same way as when organizing files on a filesystem). In addition to the hierarchical structure, the tree can be divided into logical sections. Sections are basically used to segment the content node tree.



Using sections makes it easier to:
Use custom templates
Control/limit access to content

The use of sections is implemented on the object level. Each object can only belong to one section. By default, an object belongs to the first/default section. The administration interface makes the use of sections an easy match by allowing sectioning to be done on the node level. Each time the section assignment of a node is changed, the section assignment of all subsequent children of that node will also be changed. Whenever an object is published, it will automatically inherit the section of the object's main location.

Locations

When describing the placement of an object, the previous section makes use of the term "location".

An object may be assigned to several nodes, thereby appearing at multiple places in the content tree. In other words, an object may have several *locations* within the content tree. A location is basically a synonym for a node (an encapsulation of an object).

Site management in eZ publish

eZ publish is capable of running multiple sites within the same installation. A site may be reached in different ways. Various designs can be associated with the different parts of a site. This section contains a brief explanation about how eZ publish manages sites and site interfaces.

Site

The term "site" encapsulates everything that belongs to a particular site:

- Configuration settings
- A database containing the content structure and actual content
- Content related files
- Design related files

Site interface

The content of a site can be displayed (and modified) in various ways. This is possible through the use of multiple site interfaces. A site interface basically determines which particular design to use when accessing a site in some way. At the minimum, each site has two interfaces: an administration interface and a user interface. A typical example would be that the site administrator uses the administration interface to build and modify the site. The users have only access to the user interface, which simply displays the content in a nice format. A site can have several different interfaces.

Site access

To distinguish between sites and site interfaces, eZ publish makes use of a generic solution called site access. A site access setting basically defines two things:

- How eZ publish recognizes the site/interface that is being accessed
- Miscellaneous configuration settings that will override the defaults

The most significant configuration overrides are the database and the design settings. By adding site accesses, it is possible to create new sites and/or add new interfaces to an already existing site.

eZ publish URLs

eZ publish is capable of handling two types of URLs:

System URLs
Virtual URLs

System URLs
A system URL is a raw eZ publish URL that contains miscellaneous information about the content/functionality that is being accessed. A system URL may look something like this: *http://www.example.com/content/view/full/44*

Virtual URLs
A virtual URL is a simplified version of an arbitrary system URL. A virtual URL is nicer, easier to remember and sometimes shorter than its corresponding system URL. Virtual URLs are often referred to as "nice URLs" or "URL aliases". A virtual URL may look
something like this: *http://www.example.com/recent_news*

Example

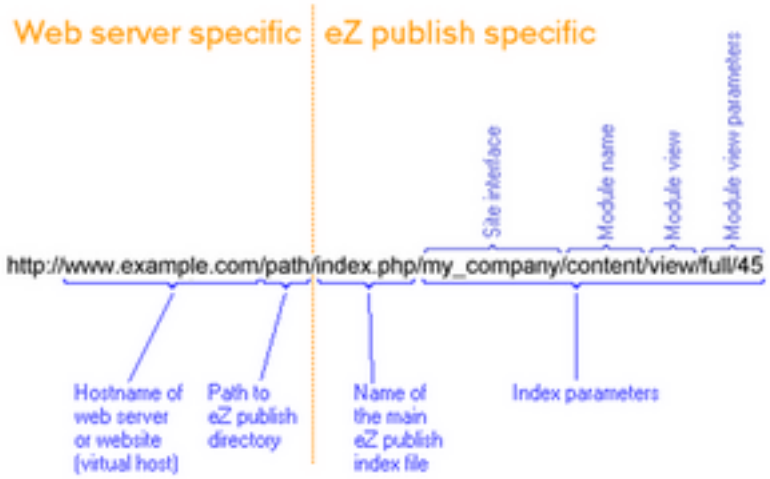
The links page of a site may be reached in two different ways:

Using a system URL: http://www.example.com/content/view/full/46
Using a virtual URL: http://www.example.com/links

The following subsections give further explanation of the two URL types.

System URLs

At first, a system URL may look intimidating and chaotic. However, this is not the case. All system URLs follow an elegant, well defined structure. The following illustration explains the various parts of a typical eZ publish system URL:

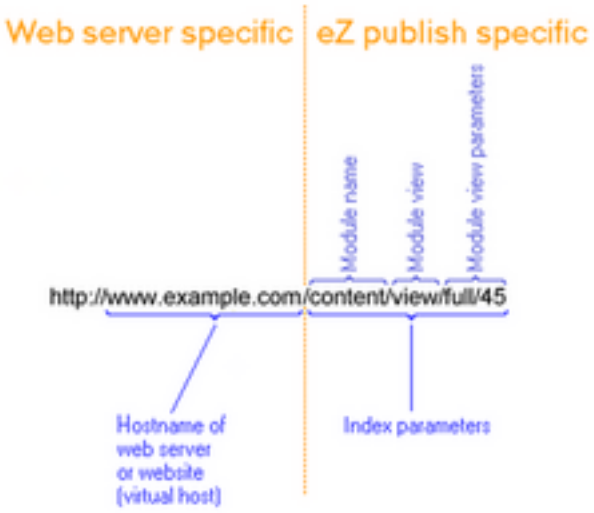


The URL in the illustration above reflects the most basic/default web server and eZ publish configuration.

The web server might be configured not to show the index file; in this case, "index.php" will simply disappear from all URL requests.

The default eZ publish site access configuration is *URL*, which means that the name of the site interface is incorporated into the URL requests themselves. If the site access configuration is *hostname*, then the site name will most likely be incorporated into the hostname (perhaps in the form of a subdomain). If the site access configuration is *port*, then a colon and a port number will be appended to the hostname. In both of the last cases (*hostname* and *port*), the name of the site interface will not be included in the URL.

The diagram below illustrates a stripped version (hidden index file, non-URL site access mode) of the previous aliased/nice URL.



Virtual URLs

A virtual URL is simply an alias for an arbitrary system URL. A virtual URL is nicer, easier to remember and sometimes shorter than its corresponding system URL. Virtual URLs are often referred to as "nice URLs" or "URL aliases".

Each content node is reachable using a virtual URL (eZ publish automatically generates URL aliases for content objects that are published). The virtual URL of a node (a published content object) is a simplified version of the node's name. The node's name is converted to lowercase, spaces are replaced with underscores and so on. Virtual URLs (and redirections, etc.) may also be added manually using the administration interface.

The following illustration explains the various parts of a typical virtual eZ publish URL:

The URL in the illustration above reflects the most basic/default web server and eZ publish configuration.

The web server might be configured not to show the index file; in this case, "index.php" will simply disappear from all URL requests.

The default eZ publish site access configuration is URL, which means that the name of the site interface is incorporated into the URL requests themselves. If the site access configuration is hostname, then the site name will most likely be incorporated into the hostname (perhaps in the form of a subdomain). If the site access configuration is port, then a colon and a port number will be appended to the hostname. In both of the last cases (hostname and port), the name of the site interface will not be included in the URL.

The diagram below illustrates a stripped version (hidden index file, non-URL site access mode) of the previous virtual/aliased/nice URL.

Summary

This chapter presented a brief introduction to... __FIX_ME__