Manual de Usuário

ref.: A92D001

Índice

Introdução

No princípio, foi criado um framework bem simples para atender a necessidade dos programadores em termos de facilidade de uso, rapidez no desenvolvimento e também a redução do processo de aprendizagem do próprio framework por parte de equipes de programadores com elementos que tinham pouca ou até nenhuma experiência com frameworks. O nome desse primeiro projeto foi, sugestivamente: Simple; "simples", em inglês.

O Simple foi criado depois de muita frustração no uso de outros frameworks que apresentavam diversos problemas:

- Tempo de aprendizado muito grande;
- Alta complexidade dificultando a usabilidade;
- Padronização muito rígida dificultando a inclusão de bibliotecas externas;
- Alto consumo de recursos do servidor (memória, processamento);
- Excesso de abstração em bancos de dados gerando incompatibilidades entre gerenciadores de bancos;
- Uso de expressões próprias em substituição ao PHP que obriga o aprendizado de um "Frameworkês";
- Documentação em inglês;

Esses outros frameworks vêm de muitos anos de desenvolvimentos e procuram se manter coesos com os princípios iniciais de seus desenvolvimentos. Isto não é tão bom quanto parece. A evolução do próprio PHP, as novas tecnologias, métodos, normas (etc.) são adaptadas de forma arcaica nas novas versões destes frameworks, tentando manter a compatibilidade com o modelo inicial. O resultado é sempre ruim pois essas modificações são verdadeiros "gatilhos". O que deveria ser feito, realmente, é uma reformulação completa nesses frameworks, mesmo com a perda de compatibilidade.

Na criação do Simple, procuramos fugir de qualquer conceito e padronização que pudesse "engessar" o framework, dificultando a adoção futura de novas idéias, métodos, tecnologias e a própria evolução da linguagem PHP. Quanto a esse último item, pensamos que o framework não deve substituir a linguagem original (PHP) com outros comandos ou funções que já existam na própria linguagem, como acontece na maioria dos outros frameworks. Damos total prioridade à linguagem em detrimento as necessidades do framework. O framework deve ser uma ferramenta para o PHP e não o inverso.

O Simple é modular. Sua estrutura é pensada como um núcleo que provê as funcionalidades básicas para a criação de um "berço" para a aplicação desenvolvida.

Esse mesmo "berço", então, será acrescido de infinitos blocos de expansão, de forma dinâmica e inteligente, para dotar o framework de todas as funcionalidades necessárias a mais exigente aplicação. Quando falamos em "dinâmica e inteligente" queremos dizer que o programador não deve perder

tempo se preocupando com as operações de inclusão dos módulos. Esta tarefa deve ser do núcleo do Simple ou provido pelo próprio bloco de expansão.

Como exemplo disso podemos citar as bibliotecas de conexão à banco de dados: Quando for solicitado o acesso ao banco de dados o framework, somente nesse momento, adiciona o bloco necessário a conexão e ao banco de dados especificado, automaticamente. Para o programador, basta fazer a consulta ao banco de dados (veja mais sobre isso em "Carregamento Automático").

Depois de termos alcançado com muito êxito os objetivos e do grande sucesso do framework, quando cópias do Simple foram usadas por vários outros programadores e até grupo de programadores em empresas, nos sentimos motivados a prosseguir no desenvolvimento e aperfeiçoamento que nos levaram a publicação de um site específico para o Simple.

Até aqui o Simple era considerado um projeto em desenvolvimento: um BETA!

Quando resolvemos dar ao Simple o merecido status de "pronto para uso", fizemos um longo e minucioso trabalho de pesquisa em seus componentes, acrescentando as atualizações, correções de erros, sugestões de outros programadores, padronização da distribuição, etc. Depois de um árduo trabalho encontramos em nossas mãos um projeto realmente maduro, transpassando em muito as espectativas originais, porém, conservando os objetivos de rapidez, pouco consumo de recursos do servidor e principalmente facilidade de uso e aprendizado. Neste momento resolvemos trocar o nome provisório (Simple) para NEOS que significa Novo (neo) Simple.

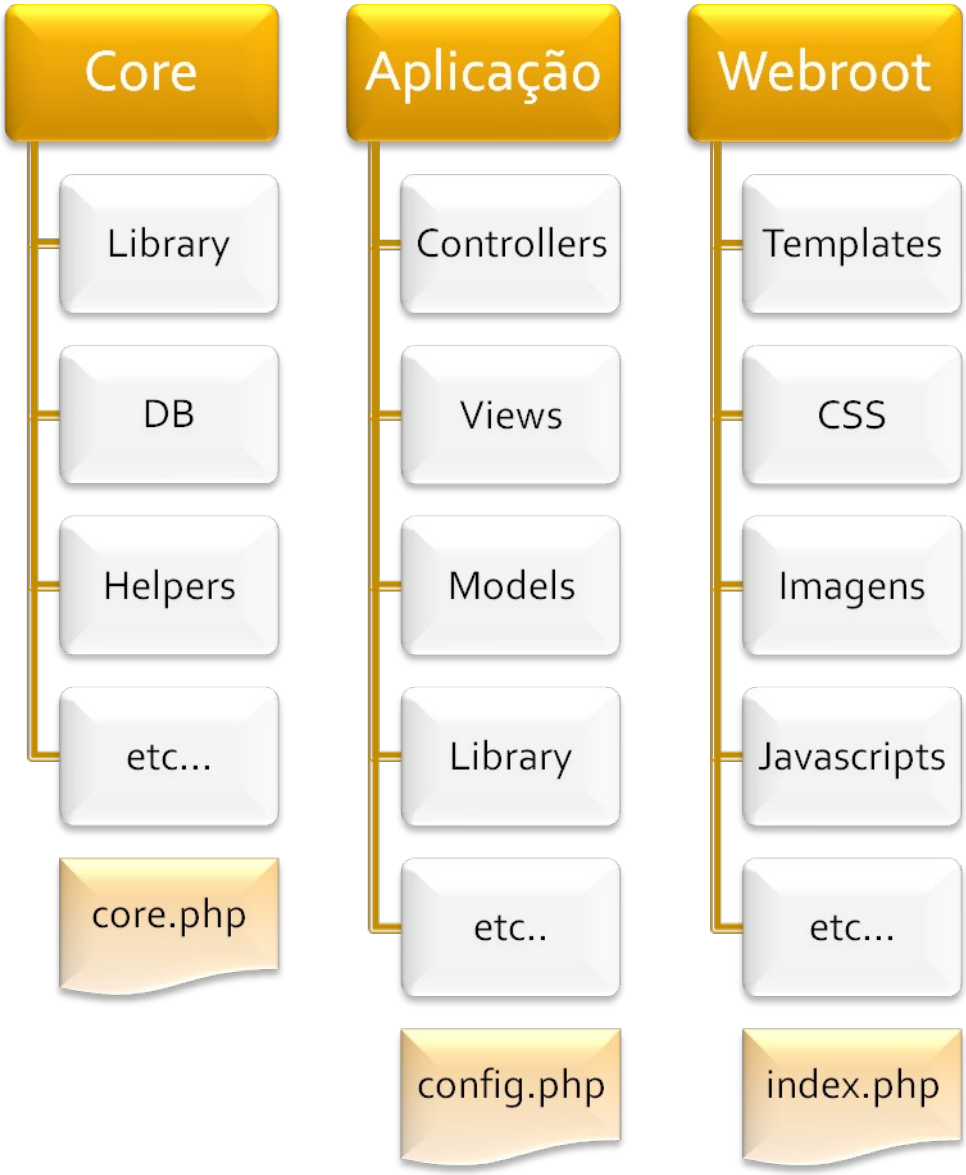
Dentre os aperfeiçoamentos adicionados, o NEOS conta agora com uma rotina de Instalação de Nova Aplicação. Esta rotina pode ser chamada com apenas duas linhas de instruções e cria todos os arquivos e pastas necessários além dos dados de conexão a banco de dados. A rotina também traz uma aplicação inicial, de exemplo, para facilitar principalmente os que estão começando a usar o framework, além da possibilidade da criação de suas próprias Aplicações Iniciais.

Outra questão importante sobre o NEOS é a de que seu núcleo pode ser modificado para atender a qualquer necessidade do programador e essa modificação (CoreMod) pode ser publicada no site do framework para acesso pelos outros programadores, livremente. Uma documentação e exemplo de uso acompanham a "CoreMod", para facilitar a utilização.

Acreditamos que você, ao ler esse pequeno manual e fazer as primeiras experiências com o NEOS, notará o grande e positivo investimento que fez. Depois disso, a adoção do NEOS como seu principal framework, será a escolha natural.

Estrutura

Antes de começarmos a usar o framework, vamos conhecer melhor a sua estrutura. O NEOS pode ser dividido em três blocos distintos: Core, Aplicação e Webroot.



Devido à grande flexibilidade esperada de um framework, esses blocos podem estar reunidos em uma única pasta ou espalhados em locais diferentes sem qualquer prejuízo para a aplicação desenvolvida.

Blocos Estruturais

Vamos ver o que cada um desses blocos representa para o funcionamento do framework.

Core

O core é o bloco que contém o núcleo do NEOS.

O arquivo "core.php" é o coração do NEOS. Este arquivo contém os "mecanismos" responsáveis pela inteligência do framework; é o núcleo onde os demais recursos serão acoplados para formar uma estrutura de tamanho praticamente infinito, com todos os recursos requeridos pela aplicação. E isto é feito de forma dinâmica mediante a necessidade atual da aplicação. É assim que, dependendo dos recursos requeridos no momento, o NEOS pode variar de tamanho, requerendo somente o que é estritamente necessário em recursos (memória, processamento...) do sistema onde está rodando.

No Core também encontramos, organizados em subpastas, os recursos (ou bibliotecas) para a expansão do core, bibliotecas do próprio desenvolvedor, bibliotecas de terceiros (baixados do site do NEOS, por exemplo), helpers (funções de ajuda) além dos conectores para banco de dados.

Todas as bibliotecas (classes) do núcleo, para efeito de organização, devem começar com o prefixo "NEOS"; como em "neos_Template", "neos_Status" ou "neos_Erros", por exemplo. As demais podem ter qualquer configuração ou convenção para os nomes com uma única restrição: o arquivo que contenha uma determinada classe ou função deve ter seu nome igualado ao nome do recurso contido e em letras minúsculas, seguido da extensão ".php".

Por exemplo: um arquivo que contenha a classe "Email" deve ser nomeado com "email.php", ou seja, o nome do arquivo deve ser o mesmo nome do recurso que contem porém com os caracteres em letras minúsculas, seguido pela extensão ".php", mesmo em sistemas Unix (Linux).

O Core do NEOS deve preferencialmente ser instalado em uma pasta fora do "root" do servidor web (por questão de segurança) e referenciado pelo "include_path" do PHP para evitar um eventual acesso indesejado aos scripts do núcleo. Apesar disso, nada impede que o Core seja instalado numa subpasta da própria aplicação ou ainda misturado aos recursos do bloco "Aplicação", como veremos mais a frente.

Aplicação

Toda a lógica PHP da aplicação deve ser encontrada neste bloco. O NEOS usa o modelo de desenvolvimento MVC, onde uma aplicação estará dividida em três partes:

- Model : modelagem de transação (de dados);
- View : a parte visual e de integração com o usuário da aplicação;
- Control : toda a lógica de controle da aplicação.

Na prática, para não aprofundarmos no assunto e considerando que existem muitas referencias sobre o modelo MVC disponíveis, no Model criamos os objetos que formam os modelos de transação da aplicação; todos os procedimentos de acesso aos bancos de dados.

O View contém as visualizações que a aplicação mostrará ao usuário do site (aplicação). É basicamente composto por arquivos "html", porém alguns desenvolvedores costumam dotá-los de alguma lógica (PHP) de apoio. A extensão dos arquivos deve ser ".html" para que o NEOS reconheça e carregue corretamente. Você poderá usar as neosTags para carregar dados, blocos, módulos, (etc) principalmente pelos designers que não tem conhecimento em programação e, apesar da extensão "html", comandos do PHP funcionarão normalmente nestes arquivos.

No Controller, "controlamos" a aplicação. Aqui você encontrará toda a lógica (PHP) usada para controlar a aplicação, usando os recursos dos models e views além das bibliotecas, helpers e outros recursos disponíveis.

No NEOS, diferentemente de outros frameworks, o Controller "extends" a classe principal do núcleo. Isso significa que o Controller e , conseqüentemente o programador, tem o controle total do framework estendido; o Controller passa a ser o núcleo do sistema. Isso parece lógico, não é? Mas, até hoje, não entendi porque outros frameworks não procedem assim!

Neste bloco do Framework (Aplicação) destacamos o arquivo "config.php". Este arquivo, como o próprio nome indica, trás todas as configurações do framework. Nele determinamos entre outras coisas, o layout do framework, os dados de conexão aos bancos de dados, etc.

O bloco Core do NEOS contem um arquivo de configuração padrão para todas as aplicações em um determinado servidor. Neste caso, onde se estaria usando o bloco Core de forma compartilhada para todas as aplicações, um arquivo de configuração em cada aplicação deverá conter apenas os itens específicos desta aplicação; a aplicação corrente. Como exemplo, se precisamos configurar um banco de dados específico para uma aplicação, somente o item "banco de dados" deverá ser descrito no arquivo de configuração da aplicação. Abra o arquivo "config.php" para um maior entendimento de seus parâmetros ou leia a publicação "Configurando o NEOS" que pode ser baixada do site do framework.

Este bloco do NEOS pode conter também, organizados em pastas, bibliotecas para expansão do NEOS, bibliotecas de recursos diversos (classes), funções de apoio (helpers) e outros recursos que o desenvolvedor julgar necessário (você pode criar diretórios e arquivos sem restrição). Estes recursos dispostos neste bloco, serão exclusivos desta aplicação. O core do NEOS, quando estiver adicionando algum recurso ao núcleo do framework (como conectores de banco de dados, classes de apoio, funções, etc.) primeiro procurará nas pastas da aplicação. Não encontrando o recurso solicitado, procurará nas pastas do núcleo (Core). O mais correto é colocar todos os recursos (classes, etc.) no Core do framework; ficando disponível para todas as aplicações.

Webroot

Este bloco contém os arquivos externos, de acesso público. Aqui você deve encontrar as imagens, arquivos Flash (swf), folhas de estilos (CSS), arquivos de JavaScript e outros similares.

Neste bloco temos, como elemento predominante, um arquivo "índex.php" que funciona como "bootstrap" do framework. Isso significa que qualquer solicitação feita por um usuário (ou link) será re-direcionada para este arquivo pelo servidor web, conforme configurado num arquivo ".htaccess",

usando o módulo rewrite do servidor web. Se a solicitação for de um dos recursos listados acima (imagens, javascript, etc.) o servidor apenas atende a solicitação, enviando o arquivo solicitado. Caso contrário, através do arquivo "índex.php" a solicitação passa para o framework que, depois de decodificar a solicitação, chamará o controller adequado.

Se você nunca usou um framework antes talvez não esteja entendendo como isso acontece. Pense que, em um formato "normal", aqui teríamos um script para cada função da sua aplicação. Mas no caso do framework, o arquivo "index.php" funciona como um funil! Todas as solicitações são redirecionadas (módulo rewrite e arquivo ".htaccess") para este arquivo. O que o framework faz é detectar qual ação foi solicitada e, então, carregar o controller correspondente. Normalmente, no final do processo, o framework retorna uma view (ou outra ação de saída...).

Outros Conceitos

Além da estrutura organizacional do framework, estudada no tópico anterior, precisamos conhecer mais alguns conceitos importantes do funcionamento do NEOS. Vamos aprender como o NEOS trabalha com as URLs e também das várias possibilidades de organização dos diretórios e da localização dos blocos estruturais em seus projetos. O NEOS, devido a sua grande flexibilidade, nos proporciona uma variedade de LAYOUTs que visam atender as necessidades específicas de cada projeto.

Url

O NEOS usa um formato de acesso em suas urls baseado em segmentos. Isto facilita os mecanismos de busca e ao próprio usuário em detrimento ao habitual "query string". Note que tanto o arquivo "index.php" quanto o arquivo ".htaccess" com o módulo rewrite são igualmente importantes neste processo. Se o seu servidor não tiver um módulo de redirecionamento (rewrite) o funcionamento correto do framework não ocorrerá. Você será obrigado a incluir o "index.php" nas solicitações de páginas ao servidor.

Uma url como "http://meusite/minha-pagina" teria que ser digitada assim: "http://meusite/index.php/minha-pagina".

Isso não deve te preocupar pois, atualmente, praticamente todos os servidores possuem algum recurso de redirecionamento compatível com o rewrite do Apache.

Numa url os segmentos serão interpretados pelo NEOS da seguinte maneira:

Dada a url: "http://meusite/controller/função/argumento/argumento2/argu..."

Os seguintes seguimentos podem ser notados:

- controller : nome do controller que será carregado pelo NEOS;
- função : método ou função do controller que deverá ser invocado;
- argumentos : argumentos ou dados passados ao método invocado.

Se você preferir as "query strings" o NEOS também conseguirá funcionar da forma esperada.

Dada a url: "http://meusite/index.php?c=controller&f=função&arg=argumentos..."

- c : esta variável (GET) passará o nome do Controller;
- f : esta variável (GET) passará o método ou função do controller;
- arg : ou qualquer outra variável (GET) será passada ao método invocado, como um array.

O nome das variáveis podem ser trocadas no arquivo "config.php", nos itens "$cfg->get_ctrl" e "$cfg->get_func".

O NEOS ainda pode funcionar com variáveis passadas por POST nas mesmas condições descritas acima. Neste caso, quando a url não contiver seguimentos e não forem encontradas as variáveis em formato GET, o NEOS tentará encontrar o controller, função e argumentos usando o método POST. Para configurar o nome das variáveis modifique os itens "$cfg->post_ctrl" e "$cfg->post_func".

Em todos os casos, o NEOS indicará o método usado para a decodificação da url na variável "$neos_metodo".

Mascaramento

Você pode modificar uma solicitação de endereçamento, seja por URL, GET ou POST, usando mascaramentos.

Para criar um mascaramento crie um novo item de configuração com a seguinte sintaxe:

```
  $cfg->mask [ 'comando' ]  =  array( 'controller' , 'function' , 'args' ,
'args...' );

  Onde:
  comando :     O primeiro segmento da URL ou o controller em GET e POST;
  controller :  O controller de destino;
  function :    A função de destino;
  args :        Pseudos argumentos passados a função de destino;

  Além dos pseudos argumentos, todos os demais segmentos da URL (e variáveis
GET ou POST quando este for o método usado) serão passados em seguida à
função de destino.
```

Com o mascaramento podemos criar "rotas" específicas sem a necessidade de usar a convencional indicação de controller+function+args.

Como exemplo vamos considerar a seguinte máscara:

```
  $cfg->mask ['manual'] = array ( 'manuals' , 'select_book' );
```

Terá o seguinte efeito:

```
http://www.meu_site.com/manual/javascript
//será interpretado como:
http://www.meu_site.com/manuals/select_book/javascript
```

Lembre-se que esta é apenas uma representação figurativa – na verdade não será feito um redirecionamento da url, mas sim, um roteamento para o controller e função configurados.

Views Estáticas

O NEOS aceita um tipo de endereçamento estático quando você não precisa criar um controller para apenas chamar uma view, onde não se pretenda a execução de qualquer código PHP: uma view estática.

Neste caso a view é apenas um arquivo html simples que pode conter, por exemplo, um texto com a licença de uso do site, uma tela de abertura, uma tela (simples) de login, um formulário , etc.

Para usar views estáticas você deve criar um diretório com o nome "statics" dentro do seu diretório de views. Salve suas views estáticas neste diretório com a extensão ".html".

Para acessar uma view estática, o usuário do site deve digitar o nome completo da view estática com ou sem a extensão que pode ser ".html", ".htm", ".php" ou ".neos". O nome da view não deve ser igual ao de um controller da sua aplicação pois o NEOS dará prioridade ao controller.

Finalmente, para que a view seja exibida, acesse o arquivo "config.php" da sua aplicação e inclua (ou modifique) o seguinte ítem:

```
$cfg->static_view = true;
```

Lembre-se:

1. Views estáticas não podem conter comandos do PHP e neosTags. Estes não funcionarão e ainda serão enviados ao navegador do visitante sem alterações.
2. Por segurança o NEOS não habilita as "statics views" por default..

NameSpace

Bem, não se trata realmente de namespace como se usa no PHP. Acontece que, para evitar conflitos entre os recursos intrínsecos do framework e de sua aplicação, resolvemos prefixar todas as variáveis, librarys, helpers, (etc) e demais recursos do NEOS com a palavra "neos". Então, evite usar em sua aplicação variáveis, constantes, classes e outros recursos começados com "neos...". A menos que pretenda se referir aos recursos do core!

Para exemplificar, a classe de status do framework se chama "NEOS_status" e não "Status" como era de se esperar. A variável que contém as definições das views a serem carregadas no final da execução do NEOS se chama "$_neosViews".

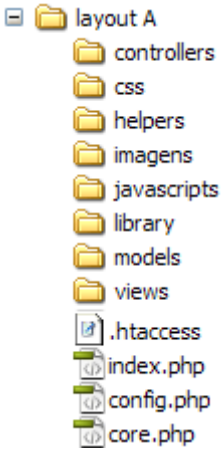
Neste último exemplo vemos uma outra questão: reservamos o caractere "_" (underline ou sublinhado) para as funções de apoio (helpers) do núcleo (core) do NEOS e variáveis (propriedades) da superClasse (sugestão do Filipe Dutra). Este caractere também é usado, dentro de uma classe controller, para indicar uma função restrita; que não pode ser chamada externamente (por URL, GET ou POST).

Resumindo: a menos que queira referenciar recursos do Core do NEOS, não inicie suas variáveis, constantes, classes, funções (etc) com a palavra "neos" ou "_neos". Ainda, em suas classes controller, não inicie uma função com o caracter "_", a menos que queira que o NEOS ignore esta função nas requisições de acesso externo (URL – segmentos, GET ou POST).

Layouts

Considerando os vários blocos estruturais descritos anteriormente e a capacidade de configuração do NEOS, podemos organizar o framework em vários layouts distintos, conforme a necessidade do projeto e a preferência do desenvolvedor.

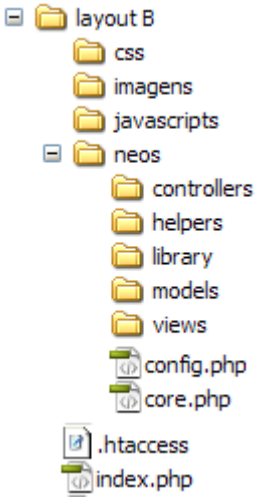
Entre várias possibilidades, destacamos três layouts:



O layout 'A' tem todos os blocos, descritos nos tópicos anteriores, misturados na raiz do site. Este layout é mais indicado para aqueles programadores não acostumados com frameworks ou para aplicações mais simples.

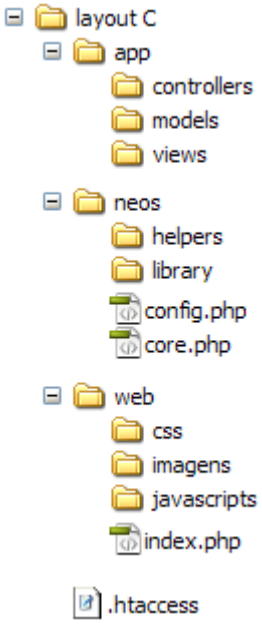
A vantagem é que a visualização dos recursos utilizados é imediata.

A desvantagem é com relação a segurança, principalmente por deixar expostos os arquivos do core. Se você instalar o NEOS com o núcleo (core) compartilhado, as pastas "library", "helpers" e os arquivos "core.php" e "config.php" não serão necessários. Estarão na pasta compartilhada do núcleo. Neste caso é viável, do ponto de vista da segurança, a utilização deste layout.



No layout 'B' temos o bloco "Webroot" disponível na raiz do site (como no caso anterior) porém, o bloco "Aplicação" e o bloco "Core" estão misturados na pasta "neos".

Se você estiver usando o core compartilhado (em outra pasta), somente o bloco "Aplicação" deve ficar na pasta "neos".



O layout 'C' é o mais organizado.

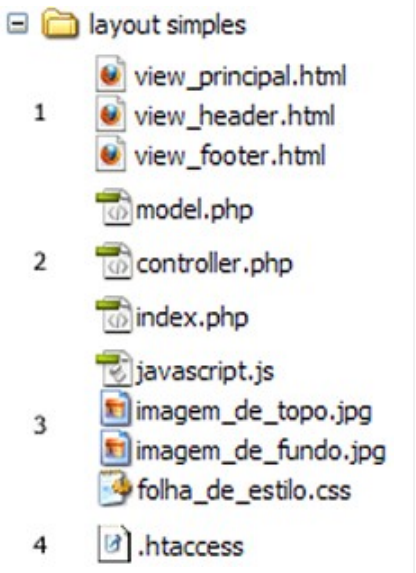
O bloco "Webroot" esta na pasta "web". O arquivo ".htaccess", na raiz do site (na parte inferior da imagem), direciona as requisições para a pasta "web".

O bloco "Aplicação" está na pasta "app".

O Bloco "Core" está na pasta "neos". Caso você tenha instalado o core do NEOS em uma pasta separada, compartilhada, basta eliminar a pasta "neos", mostrada na imagem ao lado.

Você está livre para usar um dos layouts propostos ou criar seus próprios layouts. Você pode, ainda, modificar o instalador do NEOS para acrescentar seus layouts. Basicamente o que você precisa é criar uma estrutura de diretórios adequada e modificar o arquivo "config.php", indicando a localização dos diversos componentes do NEOS (controllers, models, views, core, etc).

Para entendermos melhor as possibilidades de estruturação de uma aplicação com o NEOS, vemos abaixo uma aplicação extremamente simples, que nem mesmo organiza seus recursos em pastas. Estamos considerando, neste caso, que a aplicação esta utilizando uma instalação compartilhada do core do NEOS. Então, somente os blocos "Aplicação" e "Webroot", estão representados.



Na Imagem ao lado vemos:

1. Arquivos das views. Foi usado um prefixo "view" para melhorar a organização;
2. Controller, model e o arquivo index (bootstrap). Pode ser usado um prefixo para organizar os controllers e models;
3. Outros recursos do web site;
4. Arquivo de segurança e redirecionamento do servidor.

Para evitar um acesso indevido (direto) aos arquivos dos controllers e models, coloque o seguinte no inicio de cada arquivo:

```
<?php if (!defined('URL')) exit(); ?>
```

Para mais detalhes sobre layouts e configurações leia a publicação "Configurando o NEOS" que pode ser baixada do site do NEOS.

Carregamento Automático

Você não precisa se preocupar em carregar CLASSES e FUNÇÕES quando está programando com o NEOS. Diferentemente de outros frameworks, o NEOS possui um carregador automático que procura e carrega as classes e funções (helpers) quando forem solicitadas no script.

Tomamos como exemplo o seguinte caso: Eu tenho uma função que uso muito em meus scripts e que faz uma série de "limpezas" em dados recebidos de um formulário ou de uma requisição em Ajax. A função "_escape()", então, foi armazenada como um helper, na pasta "Helpers" de minha instalação do Core. Para acessar essa função eu deveria seguir os seguintes passos:

1. Antes de usar a função, eu precisaria saber onde está o arquivo com a função;
2. Incluir o arquivo com um comando "include";
3. Finalmente chamar a função.

```
If ( file_exists ('caminho-para-o-arquivo-com-o-recurso-desejado') )
{
      include 'caminho-para-o-arquivo-com-o-recurso-desejado';
      $variavel = _escape( $_POST['dados'] );
}
```

Com o NEOS você precisa apenas chamar a função, mesmo que não tenha sido incluída. O NEOS inclui para você automaticamente:

```
$variavel = $this->_escape( $_POST['dados'] );
```

Além de funções também é possível carregar classes de forma automática. Para isso procedemos normalmente, como se todas as classes de que precisamos já estivessem disponíveis (incluídas).

```
//carregando a classe 'Mail' e chamando o método 'enviar()'
$mail = new Mail();
$mail->enviar($argumentos);
```

Entre a primeira e a segunda linha, o NEOS localiza o arquivo que contenha a classe solicitada, carrega-o e processa a próxima linha. O programador não precisa se preocupar com mais nada.

Isso é muito interessante também para a optimização do consumo de recursos do servidor: O NEOS trabalha de forma inteligente, adicionando somente o que é realmente necessário e no momento oportuno.

O carregador automático procura novas funções na pasta de helpers de sua aplicação e em seguida na pasta "Helpers" do core. Para as classes, primeiro o NEOS procura o recurso na pasta "Library" (ou o nome que você configurou para esta pasta) da aplicação. Se não encontrar, procura na pasta "Library" do core. Assim, é possível que uma função ou classe seja alterada para atender a uma especificação da aplicação corrente; o recurso alterado será armazenado na pasta correspondente da aplicação e será chamado prioritariamente.

Alternativamente você pode carregar os helpers e classes da forma antiga, usando os métodos apropriados, mostrados na seção "Funções e Objetos".

Instalação

O NEOS é muito fácil de instalar. Os parâmetros de configuração (pré-configurados) devem estar de acordo com a maioria dos servidores e aplicações mais comuns.

É só descompactar e usar!

Pré-requisitos

O NEOS foi otimizado para funcionar com os seguintes itens:

- Servidor web com módulo rewrite (Apache ou IIS com Isapi_Rewrite);
- PHP 5.2 ou superior (5.3 para usar versões PHAR);
- Banco de dados atualizado (Oracle, Mysql, Postgres, etc);

Dependendo das necessidades de sua aplicação, algumas bibliotecas e módulos extras devem ser ativados. Tanto para o PHP quanto para o Servidor Web (Zlib, oci8_11g, MySql, gd2...).

Também baseado nas necessidades de seu projeto, novas bibliotecas de expansão para o NEOS podem ser baixadas do site do NEOS ou de qualquer outro repositório de classes espalhados pela web (ex.: www.phpclasses.org). Você pode usar até mesmo bibliotecas de outros framework, se estes estiverem instalados em seu servidor. Para a maioria dos casos uma pequena modificação pode resolver algum problema de compatibilidade (raro!).

O Núcleo ou Core do NEOS pode ser instalado em uma pasta (diretório) qualquer em seu servidor. Este Core será acessado por todas as aplicações que usarem o NEOS como framework. Existe ainda a possibilidade de instalação do NEOS para uma única aplicação. Neste caso os arquivos que compõem o core do NEOS podem ser alojados em uma pasta específica, em meio a organização de diretórios da aplicação (como visto no tópico Layout, anteriormente).

Aplicação de Exemplo

Entre os arquivos para download no site, você encontrará uma aplicação de exemplo do NEOS (app.zip). Esta aplicação pode ser usada em conjunto com a vídeo-aula "Criando uma Aplicação do Zero", para estudar o funcionamento do framework e também para iniciar sua própria aplicação, pois contém todos os recursos necessários para o funcionamento do framework. Atenção, porém, para a versão do CORE (A6EN001) que pode ser ligeiramente diferente (antiga) da versão tratada por este manual (A92D001).

Para instalar, apenas descompacte o arquivo em seu servidor web e "rode" o site com o seu navegador.

A aplicação de exemplo usa um banco de dados MySql para armazenar alguns dados – na verdade apenas para mostrar como o NEOS acessa bancos de dados. Então, crie um banco de dados com as seguintes especificações:

```
Usuário:      'neos';
Senha:        '123456';
```

Agora use o script de instalação contido na raiz do site ("neos.sql") para criar o banco e a tabela. Se a aplicação não conseguir acessar seu banco de dados, verifique as configurações no arquivo "site1/app/config.php".

Você pode configurar a aplicação de exemplo para outro banco de dados, outro usuário e senha e até mesmo outro tipo de banco de dados (ex.: Oracle) bastando reconfigurar o arquivo "sute1/app/config.php".

Compartilhado

Para a instalação de um único Core, que será compartilhado por todas as aplicações no mesmo servidor, siga os passos abaixo:

1. Baixe o arquivo "core.zip"; descompacte-o em uma pasta qualquer do sistema de arquivo do servidor. Não precisa ser na pasta "root" do servidor web. Você também poderá optar pelo arquivo "core.phar", mais compacto e seguro pois usa o modelo de Arquivo do PHP (PHAR). Neste último caso não é necessário descompactar – basta colocar o arquivo dentro da pasta escolhida.

2. Abra o arquivo "php.ini" do seu servidor, localize o item "include_path" e inclua a pasta criada anteriormente. Por exemplo, se a pasta tiver o nome de "Framework" e estiver usando um servidor Windows seria assim:

```
include_path = ".;c:\Framework\"
```

Veja mais detalhes sobre 'include_path' na documentação do PHP ou no próprio arquivo "php.ini".

Caso você pretenda fazer referência direta ao core do NEOS, sem usar o "include_path", pode ignorar o passo 2. O "include_path" facilita as referencias aos componentes do core do NEOS pois, desta forma, não precisamos saber exatamente onde o NEOS está instalado, cada vez que acessarmos seus recursos.

Por exemplo:

Vamos supor que configuramos o 'include_path' com: 'C:\Web\Frameworks\PHP\'.

```
Com o "include_path":     include 'NEOS/core.php';
Sem o "include_path":     include 'C:\Web\Frameworks\PHP\NEOS\core.php';
```

Atenção!

Para evitar confusão com um possível arquivo com um mesmo nome é recomendável sempre incluir uma subpasta quando for usar o 'include_path'.

Veja este quadro (baseado no exemplo anterior):

```
C:\Web\Frameworks\PHP\NEOS\core.php
//para arquivo PHAR
C:\Web\Frameworks\PHP\NEOS\core.neos

//No php.ini, configuramos o include_path:
include_path = .;C:\Web\Frameworks\PHP\;phar://C:/Web/Frameworks/PHP/NEOS/core.neos
```

Quando for usar...

```
include 'NEOS/core.php';
//para arquivo PHAR
include 'core.php';
```

Arquivos PHAR

Este é um tipo de arquivo nativo do PHP que possui algumas vantagens em comparação aos scripts convencionais. PHAR significa "PHP Archive" e pode ser usado em versão TAR, ZIP ou como um script convencional (texto editável).

Depois de muitos testes com esse tipo de arquivo pudemos notar que a sua utilização traz, por um lado, um consumo levemente maior de memória e, por outro lado, uma enorme compensação em performance. Ambos os 'efeitos' causados pelo mesmo motivo: trata-se de um arquivo comprimido e que precisa ser 'descomprimido' na memória do servidor. Mas, acontece que o PHP é muito inteligente no gerenciamento de memória e essa ocupação a mais de memória é compensada pelo fato de o PHP reaproveitar o arquivo que já esta na memória para todas as futuras solicitações feitas ao mesmo arquivo. Ou seja, o PHP descompacta somente uma vez e mantém os arquivos na memória para futuras utilizações.

Usando os arquivos na memória (que é muito mais rápida que o HD), estes rodam muito mais rapidamente que os scripts convencionais.

Além da velocidade, ganhamos em segurança e espaço no HD. Um arquivo TAR, ZIPADO (gzip – tgz) ocupa menos espaço em disco e não pode ser modificado – para fazer uma alteração em qualquer arquivo compactado é preciso recriar todo o arquivo PHAR.

Depois de muitos testes optamos por usar a compactação Tar+Gzip, tornando o arquivo 'não executável'. Em nossos testes, num servidor bem modesto em termos de memória e processamento, o resultado foi muito positivo e a diferença em consumo de recursos foi praticamente nula em relação a versão convencional do core do NEOS. Por esses e outros motivos recomendamos o uso de arquivos PHAR não apenas no core do NEOS, mas também em seus arquivos PHP da aplicação – claro que somente na fase de 'produção', quando sua aplicação passar por todos os teste e realmente não precisar mais de modificações.

Se você não conhece o tipo PHAR do PHP leia o manual do PHP e tenha certeza que sua versão seja a 5.3 (ou superior) do PHP. Arquivos PHAR funcionam bem a partir da versão 5.2 (com a extensão PHP_PHAR ativada), porém, para manipular os arquivos (criar, editar, etc) será melhor usar a versão 5.3 em diante. A partir da versão 5.3 o PHP usa os arquivos PHAR nativamente; sem a necessidade de ativar extensões. Além disso, as extensões (bibliotecas) zlib e bzip2 podem ser úteis para os trabalhos de compressão/descompressão dos arquivos.

Até o momento da edição deste manual, tenho visto artigos na internet informando que sites como o Facebook e Yahoo utilizam arquivos PHAR em seus servidores por motivos de velocidade e segurança.

Criando uma Aplicação

Baixar a aplicação de exemplo e usá-la para iniciar uma aplicação é uma forma rápida de começar a trabalhar com o NEOS. Principalmente se estiver pretendendo usar o NEOS em apenas uma aplicação em seu servidor web.

Mas, se pretende instalar o framework para várias aplicações (ou sites) em um mesmo servidor, você terá duas opções. Você pode usar o Instalador Automático (somente a partir da versão A9xx) para a criação de suas aplicações ou criar os arquivos e pastas manualmente.

Compartilhando o mesmo núcleo, as aplicações terão somente os blocos "Web" e "Aplicação", estudados anteriormente, tornando a criação manual um processo viável.

Vamos ver, nos próximos tópicos, como proceder em ambos os casos.

Automaticamente

Como já mencionado anteriormente, o NEOS possui uma rotina de instalação automática de novas aplicações que está disponível somente a partir da versão A9xx em diante; e pode ser usada da seguinte forma:

1. Crie uma pasta para a sua nova aplicação no "root" do servidor web. Esta pasta deve ter acesso completo de escrita e leitura ou "0777" em servidores Unix. Caso contrário o instalador não conseguirá criar e copiar os arquivos necessários e apresentará uma mensagem de error. No final da instalação, recomendo deixar esta pasta (e subpastas) com as restrições normais. Vamos assumir que o nome da pasta criada seja "site".

2. Nesta pasta crie um arquivo "índex.php" (...também com permissão "0777" ou "rw"...). Use um editor qualquer para adicionar o seguinte conteúdo;

```
<?php
@$cfg->app = dirname(__FILE__);
Include 'NEOS/core.php';
```

Se você ignorou o item 2 da instalação do núcleo do NEOS ou seja, não configurou o "include_path", use o conteúdo mostrado abaixo para o arquivo "index.php". Onde 'caminho-da-instalação' deve ser substituído pelo caminho completo da pasta onde esta o Core (compartilhado) do NEOS.

```
<?php
@$cfg->app = dirname(__FILE__);
Include 'caminho-da-instalação/NEOS/core.php';
```

3. Acesse a pasta criada no primeiro passo usando o seu navegador. Se você não configurou um "virtual host" para a aplicação, estará digitando algo parecido com isso em seu navegador: *http://localhost/site/*. "Localhost" deve ser substituído pelo endereço adequado em seu servidor (ex.: *http://192.168.3.34/site/*).

4. Uma tela da Ajuda Dinâmica do NEOS aparecerá informando que o controller "inicial" não foi encontrado. Isso ocorre porque ainda não criamos um controller de inicialização para a aplicação. Procure nesta tela uma referencia ao "CORE SERVICE" e clique neste link. Você tem que configurar o acesso ($cfg->admin_url, $cfg->admin_user, etc) no arquivo "config.php" do Core do NEOS, antes (veja o quadro abaixo!). Por segurança este item vem comentado...

5. Faça login e siga as instruções que aparecerão na tela para definir os parâmetros necessários;

6. Ao clicar sobre o botão "Salvar" o instalador criará os arquivos e pastas necessários. Então, ao acessar novamente (recarregar a página ou clicar no link apropriado...) a aplicação recém criada aparecerá.

Por questão de segurança, configure a rota (url), o controller, usuário e senha para acesso ao CORE SERVICE no arquivo de configurações do Core do NEOS. A senha deve ser criptografada em MD5.

Descomente o ítem "$cfg->admin_url" para tornar este serviço disponível.

```
$cfg->admin_user          = 'neosAdmin';
$cfg->admin_pass          = MD5('123456');
$cfg->admin_url           = 'neoscoreadmin';
$cfg->admin_controller    = 'control.php';
```

Lembre-se: Sua versão do NEOS deve ser igual ou superior a A9xx para usar o CORE SERVICE.

Manualmente

Criar manualmente sua aplicação é um pouco mais trabalhoso que o processo anterior. Porém, não será tão difícil assim, depois de entendermos cada arquivo e pastas necessários ao funcionamento de uma aplicação com o NEOS.

Para este tópico vamos usar o layout 'C' como padrão e vamos considerar que o bloco "Core" esteja numa pasta externa (compartilhado) e que você tenha configurado o "include_path" do seu PHP para acessar os arquivos do NEOS.

Diretórios

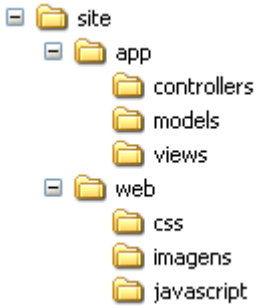
Criamos neste passo uma nova pasta no diretório "root" de seu servidor web com o nome de "site". Em seguida, criamos mais duas subpastas: "app" e "web". Respectivamente para os blocos "Aplicação" e "Webroot".

Dentro da pasta "app" criamos as subpastas "controllers", "models" e "views".

Dentro da pasta "web" criamos as pastas necessárias para a nossa aplicação: imagens, css, javascript, etc.

Todos estes nomes de pasta são apenas sugestões. Você pode usar o nome que quiser, não esquecendo de configurar os nomes no arquivo "config.php" da sua aplicação. Os nomes sugeridos são defaults do NEOS – não precisaremos mexer na configuração.

Deve ficar parecido com isto:



Arquivos ".htaccess"

Vamos criar agora os arquivos ".htaccess" para prover segurança e redirecionamentos adequados para cada pasta.

Na pasta principal do site (raiz) o arquivo tem o seguinte conteúdo:

```
<IfModule mod_rewrite.c>
      RewriteEngine On
      #se estiver usando um virtual host, descomente o item abaixo
      #RewriteBase /
      RewriteRule    ^$ web/    [L]
      RewriteRule    (.*) web/$1 [L]
</IfModule>
```

Dentro da pasta "app", crie um arquivo ".htaccess" com o seguinte conteúdo:

```
Deny From All
```

Na pasta "web", crie outro arquivo ".htaccess" com o conteúdo:

```
<IfModule mod_rewrite.c>
      RewriteEngine On
      #se estiver usando um virtual host, descomente o item abaixo
      #RewriteBase /web/
      RewriteCond %{REQUEST_FILENAME} !-d
      RewriteCond %{REQUEST_FILENAME} !-f
      RewriteRule ^(.*)$ index.php/$1 [QSA,L]
</IfModule>
```

Aplicação (controller e view)

Vamos criar um controller e três views bem simples, somente para servir de exemplo.

Crie um arquivo com o nome de "inicial.php" na pasta **"app/controllers"** e cole o seguinte conteúdo:

```
      <?php

1     class Inicial extends NEOS
      {

2        function index()
         {
3              $this->_viewVar('titulo','Site em Construção');
3              $this->_viewVar('mensagem','Para mais informações acesse
               www.neophp.tk');

4              $this->_view('head');
4              $this->_view('site');
4              $this->_view('footer');
         }
      }
```

Vamos entender o conteúdo deste arquivo:

1. A classe "Inicial" deve estender (extends) a superclasse NEOS;
2. A função default neste caso ("index");
3. A função "_viewVar ()" cria uma variável somente disponível para as views;
4. A função "_view()" carrega as views indicadas.

Crie também três arquivos na pasta **"app/views"** com os seguintes nomes e conteúdos:

head.html:

```
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title><neos var="titulo"/></title>
  <link href="<neos:url />css/css.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
```

Na sexta linha vemos uma tag XHTML específica do NEOS: "<neos var="titulo" />". Durante a renderização da view, o NEOS substituirá esta tag pelo conteúdo da variável indicada. O NEOS possui uma variedade de "neosTags" que podem ser muito útil para o desenvolvimento do design das views e principalmente quando se esta usando uma classe de template.

A tag "<neos:url/>" (sétima linha) também é uma "neosTag" e contém o endereço base do site. Neste exemplo não usaremos o arquivo CSS; esta linha é somente para exemplificar o uso da neosTag e pode ser eliminada.

site.html:

```
  <div class="pagina">

        <h1>Site em Construção!</h1>

        <neos var="mensagem" style="color:#F00; font-weight:bold; font-
size:12px" />

  </div>
```

footer.html:

```
  </body>
  </html>
```

Webroot

Por último, vamos criar o nosso bootstrap. Para isso criamos um arquivo com o nome de "index.php" na pasta "web". Colocamos o seguinte conteúdo:

```
  <?php
  @$cfg->app = '../app/';
  @$cfg->web = dirname(__FILE__);
  include 'NEOS/core.php';
```

Vamos analisar cada linha deste arquivo:

1. Configuramos a localização da pasta "app" (bloco "Aplicação");
2. Indicamos a localização da pasta "web" (bloco "Webroot");
3. Incluimos o CORE do NEOS (bloco "Core"). Lembre-se que estamos usando o include_path! Se você não quiser (puder), pode indicar a localização do CORE diretamente.

Agora já temos o mínimo para a nossa aplicação funcionar!

Abra o seu navegador e digite:

```
http://localhost/site/
```

Como **não** foi indicado um controller e função na url digitada, o NEOS assume o default. Você também poderia ter feito a seleção diretamente na url de duas formas (como vimos nos tópicos anteriores):

```
http://localhost/site/inicial/index
ou
http://localhost/site/index.php?c=inicial&f=index
```

No controller foram chamadas três views ou três partes (fatiamento) que formarão o arquivo de visualização final. Porém, é possível fazer isso de outra forma: usando neosTags. Desta forma o fatiamento ficará a cargo do designer da página html; o que é mais comum.

No controller localize as seguintes linhas:

```
$this->_view('head');
$this->_view('site');
$this->_view('footer');
```

Retire as linhas que chamam as views "head" e "footer"; deixe somente a principal ("site").

Veja:

```
$this->_view('site');
```

Na view "site" modifique o arquivo site.html, acrescentando as neosTags para carregamento de views (subviews). O arquivo ficará assim:

site.html:

```
  <neos type="view" name="head" />
  <div class="pagina">

        <h1>Site em Construção!</h1>

        <neos var="mensagem" style="color:#F00; font-weight:bold; font-
size:12px" />

  </div>
  <neos type="view" name="footer" />
```

As demais views não precisam ser alteradas e o resultado visual será o mesmo.

Usando Banco de Dados

Para usarmos banco de dados numa aplicação é necessário apenas configurar os dados de acesso ao banco de dados e indicar o gerenciador.

Para a aplicação criada no tópico anterior podemos proceder da seguinte forma:

Configurando o Banco de Dados

Se você já configurou corretamente o arquivo "config.php" global que se encontra na pasta do Core do NEOS, não é preciso fazer mais nada neste passo. Porém, se quiser fazer uma configuração específica para essa aplicação, crie o arquivo "app/config.php" e adicione as linhas de configuração do seu banco de dados – por exemplo:

```
<?php
//Setando o Banco de Dados default
$cfg->default->db         = 'mysql';

//Configurações do Banco de Dados
$cfg->db->mysql->driver   = 'mysql';
$cfg->db->mysql->host     = 'localhost';
$cfg->db->mysql->user     = 'neos';
$cfg->db->mysql->pass     = '123456';
$cfg->db->mysql->database = 'site';
$cfg->db->mysql->char     = 'latin1_swedish_ci';


//Outro Banco usando PDO (NEOS_DBO)
//opcional: $cfg->db->my_pdo->driver  = 'pdo';
$cfg->db->my_pdo->dsn     = 'mysql:host=localhost;dbname=site';
$cfg->db->my_pdo->user    = 'neos';
$cfg->db->my_pdo->pass    = '123456';
```

Fazendo uma Consulta

Para fazermos uma simples consulta, podemos usar o comando "_db()" do NEOS. Este comando pode ser chamado dentro de um controller, um model ou qualquer classe carregada pelo NEOS.

Para exemplificar podemos analisar o seguinte trecho de código:

```
1     if($q=_db('SELECT * FROM TABELA');)
      {
2       foreach($q as $row)
        {
3          echo 'Campo = '.$row->CAMPO;
        }

      } else { echo 'A consulta resultou vazia…'; }
```

1. Nesta linha fazemos uma consulta simples ao banco de dados e a variável "$q" armazena o resultado. A classe de conexão padrão do NEOS retorna o resultado como um array contendo um objeto para cada campo da tabela consultada. O índice do array indexa cada linha do resultado. Fazemos uma checagem ("if") para saber se houve algum resultado;
2. Uma função "foreach" faz a varredura dos resultados;
3. Mostramos o conteúdo do campo de nome "CAMPO";

Uma forma mais simples e direta, caso eu tenha certeza de que a consulta retornará algo, seria assim:

```
1     $q=_db('SELECT * FROM TABELA');

2     echo 'Campo = '.$q[ 0 ]->CAMPO;
3     echo 'Campo = '.$q[ n ]->CAMPO;
```

1. Como no exemplo acima, fazemos a consulta.
1. Mostramos o resultado para o primeiro "CAMPO".
2. Mostramos os demais "CAMPOS" (n = 1, 2, 3, etc.) .

Podemos saber quantos resultados a consulta retornou da seguinte forma:

```
1     echo count ( $q );
//ou
2     echo $this->_db->num_rows;
```

1. Podemos usar o comando "count" do PHP ou;
2. Obtemos o mesmo valor consultando o conector de banco de dados.

Obtenha mais informações sobre bancos de dados consultando o tópico sobre o método "_db()".

Funções e Objetos

Um pequeno número de funções e alguns objetos podem ser usados pelo desenvolvedor para acessar e controlar alguns parâmetros do framework. O NEOS foi projetado para ser o mais simples possível, assim sendo, utiliza este conjunto bem resumido de comandos próprios, deixando o mais para os comandos da própria linguagem PHP. Isto também economiza muitas páginas de manual, melhorando a curva de aprendizado do framework.

Objetos

Configuração

Todas as configurações necessárias ao funcionamento do NEOS podem ser encontradas no objeto com o nome "$cfg". Este objeto pode ser acessado de três formas:

- Diretamente usando a sintaxe "$this->_cfg", de dentro de um controller;
- Fazendo o objeto virar global com a sintaxe "global $cfg;" no inicio de uma função ou método de uma classe qualquer;
- Usando o método global "_neos('cfg')", que retorna o objeto em qualquer parte do NEOS (classes, controllers, models, etc).

Exemplos:

```
//no controller
$this->_cfg->app = 'C:/www/site/app/';

//usando o comando global
global $cfg;
$cfg->app = 'C:/www/site/app/';

//usando o método global
_neos('cfg')->app = 'C:/www/site/app/';
```

Para uma melhor compreensão dos dados configurados neste objeto consulte a publicação "Configurando o NEOS", que pode ser baixada do site do NEOS.

Métodos

Abaixo temos uma listagem dos comandos específicos do NEOS Framework:

```
Métodos da SuperClasse

_view();
_viewVar();
_helper();
_model();
_controller();

Métodos Globais

_db();
_goto();
_load();
_helper();
_setmark();
_modulo();
_pegatag();
```

Métodos da SuperClasse

Os métodos da Super classe (classe NEOS) só podem ser chamados a partir do controller, usando a sintaxe:

$this->nome-do-método(parâmetros);

Método _view()

O método "_view" indica qual view o NEOS deverá carregar na saída; no final da execução do script. O método pode ser chamado mais de uma vez, conforme o número de views que se queira mostrar.

Sintaxe:

$this->_view(nome-do-arquivo, variáveis, nome-da-view, template, retorna);

- nome-do-arquivo : caminho e nome do arquivo da view a partir do diretório padrão das views ($cfg->view); não é preciso indicar a extensão ('.html').
- variáveis : Um array, cujos índices correspondem ao nome de uma variável disponível dentro do escopo da própria view.
- nome-da-view : você pode atribuir um nome para essa view usando este parâmetro; caso contrário, será atribuída uma numeração a partir de zero (0). O nome é muito importante quando trabalhamos com templates, para indicar em qual lugar deverá a view ser mostrada.
- template : string com o nome do template desejado.

- retorna : se você quiser apenas obter o conteúdo do arquivo da view (sem visualizar a view) indique o valor TRUE para este parâmetro. Somente o parâmetro "nome-do-arquivo" será considerado – nenhum processamento será feito na view.

O método retornará FALSE em caso de erro e disparará uma mensagem de erro para o mecanismo de tratamento de erro habilitado no momento.

Exemplos:

```
$this->_view ('splash');
```

A view 'splash' será mostrada no final da execução do NEOS.

```
$view = $this->_view ('splash','','','',TRUE);
```

O conteúdo da view (arquivo) será carregado na variável "$view".

```
$dados['titulo']='Titulo do Site';
$this->_view ('splash', $dados, 'Splash', 'MeuTemplate');
```

A view 'splash' será mostrada; a variável 'titulo' (se existir esta variável na view) será substituída por "Titulo do Site"; o nome da view para o NEOS será 'Splash' e usará o template 'MeuTemplate'.

Método _viewVar()

Use este método para indicar variáveis para todas ou uma específica view.

Sintaxe:

$this->_viewVar(variável, valor, nome-da-view);

- variável : nome da variável como aparecerá para a view (ou views). Também pode ser usado um array aos moldes do método "_view".
- valor : o conteúdo da variável indicada no parâmetro anterior. Se pretender usar um array para o parâmetro anterior, este item deve ser ignorado ou preenchido com uma string vazia ('').
- view : o nome da view setado no método "_view". Caso não seja indicado, esta variável estará disponível para todas as views.

Exemplos:

```
$this->_viewVar ('titulo', 'Titulo do Site');
```

Disponibiliza a variável 'titulo' para todas as views carregadas. Seu valor (ou conteúdo) será "Titulo do site".

```
$this->_viewVar ('titulo', 'Titulo do Site', 'header');
```

A variável 'titulo' estará disponível somente para a view nomeada como "header";

```
$dados['titulo']='Titulo do Site';
$dados['mensagem']='Este site está em construção!';
$this->_viewVar( $dados );
```

As variáveis armazenadas no array "$dados" estarão disponíveis para todas as views.

```
$dados['titulo']='Titulo do Site';
$dados['mensagem']='Este site está em construção!';
$this->_viewVar( $dados, ' ', 'header' );
```

O mesmo do exemplo anterior, porém, a variável será 'visível' somente na view 'header'.

Método _helper()

Carrega uma função armazenada em um arquivo "helper" (de ajuda) e disponibiliza no contexto do controller atual.

Sintaxe:

$this->_helper(nome-do-arquivo, argumentos, *pack);

- nome-do-arquivo : indique o nome do arquivo que contenha a função desejada e que esteja armazenada na pasta apropriada para os helpers.
- argumentos : opcionalmente você pode passar argumentos para o helper usando este campo (array) e obterá a saída do helper chamado.
- *pack: os helpers podem ser agrupados (como o caso das neosTags) em um 'pack' ou subpasta. Se for o caso, deve ser indicado o pack aqui.

*PACK só funcionará na chamada ao método de forma global, usando a sintaxe: _helper(); (sem o "$this->").

Depois de chamar este método a função estará disponível e pode ser acessada com a seguinte sintaxe:

$this->nome-da-função(argumentos);

Este método é a forma tradicional para adicionar um helper ao sistema. Veja também a forma automática que deve ser usada preferencialmente.

Método _model()

Para o NEOS os models são classes comuns, porém, para facilitar a organização de seu projeto, os models devem ser armazenados na pasta específica para os models ($cfg->model).

Sintaxe:

$model = $this->_model(nome-do-model , retorna);

- $model : Se " retorna " for FALSE (ou não for indicada), o model, indicado em "nome-do-model" será retornado nesta variável. Caso contrário retorna TRUE.
- nome-do-arquivo : nome do arquivo onde esta armazenado o model. A extensão não precisa ser indicada.
- retorna : Se for FALSE ou não estiver indicado (default = false) o método retorna o model. Caso contrário retornará TRUE e o model indicado será criado na variável " model " da superclasse.

Exemplos:

```
$meu_model = $this->_model( 'meu_model' );
$meu_model->meu_metodo( argumentos ... );
```

Primeiro carregamos o model "meu_model". Em seguida chamamos o método "meu_metodo" da classe model recém adicionada.

```
$this->_model ('meu_model', TRUE );
$this->model->meu_metodo( argumentos ... );
```

O mesmo exemplo acima porém agora o NEOS fará referencia ao model através da variável "model". Neste caso, somente um model pode ser carregado por vez.

Método _controller()

Pode parecer estranho a principio mas, o NEOS, admite o carregamento de mais de um controller ao mesmo tempo. Isto se dá pela forma aberta como o NEOS trabalha com sua estrutura. Para o NEOS uma classe é apenas uma classe, não importando se estamos tratando de um controller, um model ou qualquer outra classe. Isso também amplia as possibilidades de criação de controllers, dando ao desenvolvedor a opção de criar controllers com métodos (funções) genéricos, utilizáveis para outros controllers. É importante notar que um controller normal deve entender a superclasse ('class Controller extends NEOS {...}'); já um controller de apoio NÃO deve estender a superclasse.

Sintaxe:

$this->_controller('nome-do-controller');

- nome-do-controller : indica o nome do arquivo que contenha o controller de apoio desejado sem a necessidade de incluir a extensão. Convencionalmente, o nome do arquivo deve ser o mesmo nome do controller que contiver, em letras minúsculas e com a extensão '.php'.

Exemplo:

```
$this->_controller( 'Utils' );
$this->Utils->método(argumentos...);
```

O NEOS carregará o arquivo que contenha o controller 'Utils' e inicializará a classe automaticamente. Em seguida é possível acessar os recursos do controller 'Utils' diretamente.

Métodos Globais

Estes métodos estão disponíveis em qualquer escopo do NEOS. Isto inclui controllers, models, helpers ou qualquer classe usada por seu script.

Método _db()

Este é o método usado para acessar e carregar uma classe de conexão a banco de dados. No NEOS a classe de conexão a banco de dados somente será carregada se este método for invocado.

Sintaxe:

$objeto = $this->_db(parâmetro 1, parâmetro 2, parâmetro 3, método, conector);

- $objeto : Toda consulta a banco de dados deve retornar um objeto com os dados no formato específico da função indicada (consulte a documentação do conector). Caso não retorne dados terá o valor FALSE.
- parâmetros : dependendo do método invocado, pode ser de qualquer tipo PHP válido (string, array, objeto, etc.).
- método : método da classe de conexão ao banco de dados indicada (ex.: query, insert, update, etc). O método default é "query".
- conector : alias (apelido) da conexão configurada previamente. Caso não seja indicado será usado o default, indicado no arquivo de configuração do NEOS como "$cfg->default->db".

Para os conectores de banco de dados padrões do NEOS, estes retornarão um array contendo objetos, como resultado de uma consulta. Este array terá um índice numérico representando cada linha do resultado (no caso de um "SELECT"). Para cada linha o array conterá objetos para cada campo da tabela, cujo nome será exatamente o nome do campo, na tabela.

Exemplos:

```
$q = _db('SELECT * FROM TABELA');
```

A variável "$q" receberá um objeto correspondendo ao resultado da consulta. Como o método default é o "query" e o conector é definido no "config.php", então neste exemplo estamos invocando uma consulta simples (query) no banco de dados default.

```
$dados['campo1'] = 'valor1';
$dados['campo2'] = 'valor2';
$query = _db( 'TABELA',$dados,'', 'insert', 'mysql' );
```

Neste exemplo estamos inserindo dois campos na tabela 'TABELA' usando o conector 'mysql'. Os detalhes de conexão estão no arquivo de configuração do NEOS.

```
$ret=_db ( 'SELECT * FROM TABELA' );
if($ret){
      foreach( $ret as $row){
        echo '<br />Campo1: '.$row->campo1;
        echo '<br />Campo2: '.$row->campo2;
      }
}else{
      echo 'Não foi possível mostrar os campos ou a TABELA está vazia!';
}
```

No exemplo acima fazemos uma consulta, checamos se teve algum resultado e em seguida mostramos os resultados. O conector de banco de dados é o default, configurado no "config.php" e a função default é "query" (consulta simples).

O NEOS trás conectores pré-instalados compreendendo funções básicas para os bancos de dados disponíveis, além da possibilidade de usar o PDO (PHP Data Object) através da classe NEOS_DBO. Você deve instalar suas próprias classes de conexão, baixar do site do NEOS ou usar qualquer classe de terceiros. É necessário somente que a classe usada tenha alguma compatibilidade com as convenções do método "_db()". Em último caso você poderá usar sua classe como uma biblioteca comum, chamando os métodos de sua classe diretamente, sem usar o método "_db()".

Para cada método (ou função) disponível, os parâmetros são diferentes. Isso tanto para as classes de conexão do NEOS quanto para a classe NEOS_DBO, que usa o PDO.

Método	Parâmetro 1	Parâmetro 2	Parâmetro 3
query	Script SQL	-	-
insert	O nome da tabela	Array com os dados	-
update	O nome da tabela	Array com os dados	Cláusula WHERE
*prepare	Script SQL	Nome da pilha de execução	-
*bind	Nome da pilha de execução	Nome do campo	Valor
*execute	Nome da pilha de execução	-	-

Os itens marcados com asterisco (*) estão presentes somente nas classes nativas do NEOS.

Para o NEOS_DBO, como esse retorna um objeto PDO, todos os métodos do PDO estarão disponíveis conforme a documentação do PHP (query, bind, prepare, etc). Usar a classe NEOS_DBO é mais fácil do que usar o PDO nativo do PHP, pois, está integrada as configurações do framework e é totalmente compatível com o método "_db()", apresentado neste tópico.

Método _goto()

O método "_goto" é muito útil quando queremos redirecionar o framework para uma nova localização (url interna).

Sintaxe:

_goto (ulr, método, código-http);

- ulr : a url de destino. Deve ser um seguimento do próprio site: "controller/função/argumentos...".
- método : dois métodos estão disponíveis: "Reflesh" e "Location". O último é o default.
- código-http : um código de redirecionamento HTTP válido. O default é '302'.

Exemplo:

```
_goto ( 'controller/start' );
```

O NEOS redireciona para a função "start", do controller "controller".

```
_goto ('redirecionado/ofline' , 'reflesh', '301' );
```

Redireciona para o controller e função indicados no primeiro parâmetro usando o método "reflesh" e faz um redirecionamento provisório, indicado pelo código HTTP '301'.

Método _load()

É a forma "não automática" ou direta de chamar uma classe qualquer. Preferencialmente use a forma automática.

Sintaxe:

_load(nome-da-classe);

- nome-da-classe: o nome da classe começando com letra maiúscula.

Um arquivo que contenha uma classe deve ter seu nome igual ao nome da classe que contém, seguida da extensão ".php". A classe deve ter o primeiro caractere em letra maiúscula.

O método _load() procura a classe nas pastas: models, librarys, drivers, core/Library e core/Library/DB. Nesta ordem.

Método _helper()

O método _helper() é a forma "não automática" e global (sem o "$this->") de carregar um helper. Veja o método da superclasse de mesmo nome. A principal diferença é a de que podemos chamar este método de qualquer lugar, dentro do framework.

Prefira sempre a forma automática de carregar seus helpers.

Método _setmark()

Durante um processo de depuração dos seus scripts, talvês você queira tirar amostras de tempo, valor de variáveis e arquivos carregados pelo seu framework. Se for esse o caso, use esta função global para isso.

Sintaxe:

_setmark(nome-da-marca, arquivos, variáveis);

- nome-da-marca: pode ser dado um nome para ajudar a visualização dos dados; caso não seja indicado, será mostrado um número seqüencial (zero (0) é o benchmark inicial do framework, no arquivo do Core do NEOS);
- arquivos: setado (true) lista os arquivos carregados até o momento, no arquivo de logs (default false);
- variáveis: você pode indicar uma variável (ou array) que terá seu valor "amostrado" neste momento, no script.

Este método depende de dois fatores para funcionar: da classe NEOS_Status (incluída no Core do NEOS) e da configuração do ítem "$cfg->status";

Somente a quantidade de memória, a mémoria de pico e o tempo gasto até este momento serão mostrados na barra de status do framework. Os demais itens (arquivos e variáveis) serão indicados num arquivo de log da sua aplicação (se tiver configurado o ítem "$cfg->logfile").

Configure o ítem "$cfg->status" com as palavras "file" e "display" para, respectivamente gravar um arquivo de log e mostrar os dados na barra de status.

Exemplo:

```
$cfg->status = 'display'; //default do NEOS
 $cfg->status = 'file';
 $cfg->status = 'displayfile'; ou $cfg->status = 'filedisplay';
```

Método _modulo()

Este é um método especifico para trabalhar com CMS e Templates. Quando invocado, chama uma classe especialmente criada para pequenas tarefas de código auxiliar e rotineiro.

Para entendermos melhor, imagine um site que possua um menu e este deve aparecer em todas as páginas do site. Normalmente você deveria criar este menu repetidamente; para cada página.

Se este menu for estático, até que não seria uma tarefa difícil! Porém, imagine que seja um menu dinâmico, dependente de vários fatores como: em que página está, em que região, que tipo de usuário, etc.

Para facilitar a vida do desenvolvedor e até mesmo melhor a organização e visualização do código por outros desenvolvedores, seria melhor separar o menu em um módulo e fazer a sua programação apenas uma vez.

O método "_modulo" chama, então, a classe responsável que retorna o bloco já processado. A classe de módulo tem total acesso ao framework (métodos, objetos, configurações, banco de dados, etc.).

Este método foi projetado para funcionar com a neosTag "modulo" e/ou uma classe de template. O uso do método fora deste contexto não tem muito sentido.

Sintaxe:

$modulo = _modulo(nome-do-modulo , start);

- nome-do-modulo : nome do arquivo que contenha o módulo desejado, em letras minúsculas. Este também será o nome do módulo (classe) para o NEOS.
- start : Se não for indicado ou for FALSE (default) o NEOS somente carrega o arquivo do módulo. Se for TRUE o NEOS carrega o arquivo, cria a classe (new modulo...), chama e retorna os valores da função "start()" do módulo.
- $modulo : recebe o módulo se " start " for TRUE.

Exemplo:

```
_modulo ( 'menu' );
$mod = new $modulo;
echo $mod->start ( argumentos ... );
```

Para a classe de template default do NEOS, por padrão, a função "start" do módulo deve ser a principal. Acima, carregamos o módulo "menu" e mostramos (echo) diretamente na tela – apenas como exemplo.

```
echo  _modulo( ' menu ', TRUE );
```

O mesmo efeito do exemplo anterior, usando neosTags (arquivo da view):

```
<neos type="modulo" name="menu" />
```

Método _pegatag()

Este é outro método específico para os 'mecanismos' internos do NEOS no tratamento de views e templates. Este método é usado para localizar determinadas TAGs XML nos arquivos de views e templates. Se você pretende criar sua própria classe de template este método será de muita utilidade.

Sintaxe:

$array = _pegatag (xhtml, ponteiro, tipo, tag);

- $array : um array com os dados encontrados:
  - $array ['tamanho'] : tamanho (bytes) total do conteúdo xhtml (xml);
  - $array ['inicio'] : ponteiro para o inicio do bloco encontrado;
  - $array ['final'] : ponteiro para o final do bloco encontrado;
  - $array [- atributos -] : todos os atributos da tag;

- xhtml : uma string contendo os dados xhtml (ou xml) a ser pesquisado (conteúdo do arquivo da view).
- ponteiro : a posição (em bytes) a partir da qual o conteúdo (xhtml) será pesquisado. O default é o inicio do conteúdo.
- tipo : RESERVADO - não usado nesta versão do NEOS.
- tag : o nome da tag (XML) pesquisada. A tag default é "neos".

Exemplo:

```
1    $xhtml = file_get_contents ( 'arquivo-da-view-a-ser-analisado' );
2    $array = _pegatag ( $xhtml );
3    print_r ( $array );
```

1. Carregamos o conteúdo do arquivo a ser analisado – uma view ou arquivo de template;
2. Chamamos o método "_pegatag ";
3. Neste exemplo, usamos o comando "print_r" do PHP para visualizar o resultado.

Sqlite DB

O NEOS (versões superiores a ACxx) possui um banco de dados sqlite que pode, opcionalmente, ser usado tanto para guardar as configurações e dados do framework, quanto para sua aplicação.

Para usar o banco de dados é necessário que sua versão do PHP seja maior ou igual a 5.1 e que as extensões PDO e SQLITE sejam habilitadas no arquivo 'php.ini'.

O NEOS procurará o banco de dados (arquivo "neos.db") na pasta da aplicação atual ($cfg->app). Caso não encontre tentará criar o banco de dados neste local. Se estiver criando o banco de dados, todas as configurações default serão copiadas para tabelas deste banco (uma forma "à quente" de reconfigurar o DB). Se o arquivo já existir e tiver as tabelas de configuração do NEOS, estas serão usadas prioritariamente. Mesmo que exista um arquivo "config.php" para esta aplicação.

A vantagem para o NEOS de usar um banco de dados em lugar de um arquivo "config.php" é a de que os dados alterados podem ser recuperados em uma nova sessão. Por outro lado, deve-se ter um cuidado redobrado para evitar tentativas de acesso indevido (invasão, hackers...); aliás, como se teria de qualquer forma, quando se usa um banco de dados.

Outra vantagem notória é a de que, se sua aplicação precisar de um pequeno banco de dados para manipular um grupo não muito complexo de tabelas, poderá usar o banco do NEOS. Com isso, ganhamos duas vezes: não precisamos configurar nada externo (mysql, oracle, password, username, host...) e conseguimos muito mais velocidade para o conjunto. O Sqlite é extremamente rápido!

As rotinas que trabalham com o banco de dados do NEOS tem, por convenção, um campo nomeado como 'ID' (que deve existir em todas as tabelas) que serve de "ROWID" para todos os processos (creat, insert, update, etc). Quando estiver criando uma nova tabela não deve indicar um campo com o mesmo nome (ID); o NEOS criará este campo automaticamente e este será auto-incrementável. Outra convenção é a de se usar caracteres em letras maiúsculas, tanto para as tabelas quanto para os campos e nunca começar seus nomes com o caractere "_" (underline ou sublinhado), reservado no NEOS para as funções globais e da classe NEOS_DB_TABLE.

Você poderá usar comandos do SQLITE convencionais para manipular este banco de dados ou usar os comandos listados abaixo. O NEOS, na inicialização do BD, cria um objeto da classe NEOS_DB_TABLE para cada tabela do SQLITE. Assim, é possível manipular as tabelas de forma O² (Orientado a Objeto), diretamente. Para acessar este objeto, use o método "_neos()", acessível em qualquer elemento do NEOS (classes, controllers, models, etc) e que retorna diretamente o objeto.

Antes de usar o banco de dados é necessário setar no "config.php" do CORE do NEOS:

```
$cfg->use_db = true;
```

COMANDOS:

CAMPO:

```
_neos()->TABELA->CAMPO(ID)
```

Retorna o valor do CAMPO, da linha atual (ID). Se ID não for indicado usará o ultimo ID setado. Onde se vê CAMPO deve ser trocado pelo nome real do campo requerido (assim como em TABELA).

UPDATE:

```
_neos()->TABELA->CAMPO(ID,VALUE)
```

Muda o valor do CAMPO, da linha ID, para o valor VALUE.

INSERT:

```
_neos()->TABELA->_insert(ARRAY)
```

Insere uma nova linha na tabela com os dados do ARRAY (ARRAY[CAMPO] = VALUE).

DELETE:

```
_neos()->TABELA->_delete(ID)
```

Apaga uma linha da TABELA indicada pelo ID. Se não for indicado, usará o último valor de ID.

CLEAR:

```
_neos()->TABELA->_clear()
```

Limpa a TABELA (apaga todos os dados).

LIST:

```
_neos()->TABELA->_list(LEN,START)
```

Retorna um array numérico contendo um objeto para cada linha encontrada na TABELA, começando em START e limitado (em tamanho :P) por LEN.

DESTROY:

```
_neos()->_destroy('TABELA')
```

Deleta (destrói!) a TABELA.

CREATE:

```
_neos()->_create('TABELA',CAMPOS)
```

Cria uma TABELA com os campos indicados no CAMPOS (CAMPOS[NOME]=VALOR).

Por exemplo:

```
$array[ 'PRIMEIRO' ]      = 'varchar(50)';
$array[ 'SEGUNDO' ]       = 'integer';
$array[ 'TERCEIRO' ]      = 'text';

_neos()->_create( 'TABLE' , $array );
```

SQL_QUERY:

```
_neos()->_query('SQL')
```

'Roda' a query SQL no banco de dados e retorna um array numérico contendo um objeto para cada linha retornada da consulta.

No exemplo, CAMPO é o nome do campo retornado; como um objeto:

```
$q = _neos()->query('SELECT * FROM TABELA');
if($q){
      foreach($q as $row){
        echo $row->CAMPO;
        echo $row->CAMPO2;
        echo ...
      }
}
```

Lembre-se que esses comandos e funções só funcionam para versões a partir de ACxx. Nas versões anteriores o NEOS usará apenas os arquivos "config.php".

Expansibilidade

Devido a estrutura modular do NEOS, não é possível dizer que esta seção termina aqui. Uma infinidade de recursos podem ser adicionados: novas funções, objetos, classes, conectores de banco de dados, módulos de conexão com Google Maps, Facebook, etc... (veja a seção: "Caregamento Automático").

Todos os novos recursos podem ser instalados como helpers ou librarys, sejam os disponibilizados pelos desenvolvedores do NEOS como também pelos outros usuários do framework (temos um seção para divulgar os recursos desenvolvidos pelos usuários no site do NEOS).

E isto se estende a módulos, templates e até mesmo a aplicações inteiras que podem ser instaladas junto a sua aplicação (controllers, models, views, css, javascripts...).

A modularidade e expansibilidade do NEOS é realmente infinita!

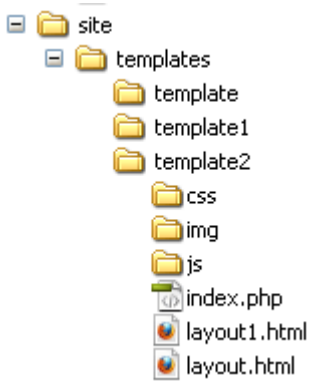
Templates

O NEOS trás uma classe de template bem simples, porém, muito eficiente. Você pode usar esta classe como base para desenvolver sua própria classe ou substituí-la por uma classe de template qualquer.

Para usar a classe e desenvolver seus templates você precisa entender algumas convenções da classe:

Estrutura dos Templates

No arquivo de configuração do NEOS encontramos um item que define a localização dos templates ($cfg->template_path), o template default ($cfg->default->template) e o endereço externo para o template ($cfg->template_url). A pasta contendo os templates deve estar localizada no bloco "Webroot" do framework e disponível externamente.



Na figura acima podemos ver que a pasta de templates contem uma série de subpastas (template, template1, template2, etc). Cada subpasta contém um template diferente.

Dentro da pasta de cada template, encontramos ainda, outras subpastas onde armazenamos os recursos necessários para o funcionamento do template. Imagens, javascript, folhas de estilo, arquivos flash e muito mais.

Quando um template esta configurado ($cfg->default->template) ou você definiu um template usando o método "_view()", o NEOS procurará pelo arquivo "index.php" da pasta do template selecionado. Se este arquivo não existir o NEOS aciona a classe template padrão do NEOS; caso contrário chama a classe contida neste arquivo que deve ter o nome de "template". O método invocado pelo NEOS é "get_layout()", para um ou outro caso.

Um arquivo nomeado como "layout.html" deve conter o template, propriamente dito. Este arquivo contém o html básico (desenvolvido por um designer, por exemplo) contendo, também, algumas tags específicas do NEOS: as "neosTags". Outros arquivos de layout podem ser criados, porém, a classe de template do NEOS usará somente o "layout.html", ignorando os outros. Para usar outros layouts no mesmo template é necessário algum seletor na classe do arquivo "index.php" ou usar outra classe de template que suporte.

NEOS Tags

Tanto nos templates como nas views, o NEOS reconhece um conjunto de tags especiais, usadas para uma série de comandos próprios do framework, dedicados ao tratamento de views. Os comandos PHP ainda estarão ativos, porém, não é recomendado o seu uso em arquivos de visualização. As neosTags facilitam muito o trabalho dos designers que estão normalmente acostumados com as tags comuns do HTML e não com comandos PHP.

Com as neosTags, o designer terá facilidade para carregar variáveis do framework, módulos e definir onde aparecerão as views dentro de um layout, usando tags muito parecidas com as tags do HTML convencional. Além disso, todos os atributos (style, align, class, id, etc.) são transportados para os blocos carregados pelas neosTags.

Sintaxe:

<neos atributo1="valor" atributo2="valor" ... />

Atenção!

Não se esqueça de fechar a tag com " /> ". Caso contrário o NEOS não detectará o final da tag, juntando o conteúdo da tag "neos" ao conteúdo da tag seguinte, no arquivo html analisado. Além disso, as regras para uso de tags em HTML devem ser observadas (não incluir espaços entre o símbolo "<" e o nome da tag, usar atributos padronizados, as tags devem estar em letras minúsculas, etc). Um erro de "simplexml_load_string()" (WARNING :: 2) pode ser disparado no caso de erro de sintaxe nas neosTags.

Tags

Estas são as tags especiais disponíveis nesta versão do NEOS.

- <neos:url /> : o NEOS substitui pela url base do site.
- <neos:charset/> : o NEOS substitui pelo valor indicado em $cfg->charset;
- <neos:template /> : o NEOS substitui pelo endereço base do template atual.
- <neos . . . /> : o NEOS substitui pelo conteúdo indicado pelos atributos (módulos, variáveis, views, etc).

Nas versões anteriores a A7xx as neosTags url e template não possuíam prefixo. Para padronização e evitar conflitos com futuras implementações do html, foi acrescentado o prefixo neos em todas as neosTags simples (as que não tem atributos...).

Atributos

- **var**: carrega o valor da variável indicada;
- **type**: define o tipo de recurso a ser carregado;
- **name**: nome do recurso a ser carregado;
- **style**, **class**, **id**, **title**, **align**, **etc**: esses atributos serão transportado para o bloco carregado;

Exemplos:

```
<neos var="titulo" />
```

O NEOS substitui esta tag pelo conteúdo da variável 'titulo' ($titulo).

Os exemplos a seguir funcionam somente com um template:

```
<neos type="modulo" name="menu" class="classe-menu" id="menu" />
```

Quando o NEOS interpretar esta tag (acima), carregará o módulo "menu" e colocará o conteúdo em substituição a neosTag original. Os atributos "class" e "id" serão transportados para o novo conteúdo, numa "div".

```
<neos type="area" name="principal" />
```

Carrega a view nomeada como "principal" em substituição a esta neosTag.

Outros "type" podem ser implementados, modificando a classe "NEOS_Template".

As tags "<neos:url />" e "<neos:template />" não precisam de atributos:

```
<form action="<neos:url />controller/função" > . . . </form>
```

Se o seu site tiver a seguinte url: "http://www.meu-site.com.br/" este será o conteúdo encontrado em substituição a neosTag "<neos:url />". É muito útil para carregar folhas de estilo, javascripts e outros arquivos com endereçamento direto. Isso porque, quando usamos seguimentos nas url, o navegador acaba interpretando a url incorretamente.

Por exemplo: a url "http://meu-site/controller/função" seria interpretada pelo navegador como o endereço base do site; não considerando que se trata de seguimentos e que o endereço base real é "http://meu-site/".

Para uma melhor familiarização com a forma como o NEOS trabalha com templates, aconselhamos dar uma boa olha na classe de template padrão do NEOS assim como os templates de exemplo encontrados nas distribuições do NEOS.

Uma grande vantagem das neosTags é a de que, numa implementação usando código PHP convencional na view, é preciso checar se a variável exite e somente depois usar um comando como o "echo". Com as neosTags, se uma variável (ou outro recurso) não existir o NEOS apenas não mostra – sem dar erro.

NeosTags Pack

Além das neosTags vistas anteriormente, novas neosTags podem ser instaladas sem a necessidade de um upgrade total do framework. Você pode instalar novas funções para as neosTags na subpasta "neostagspack" da pasta de helpers do seu NEOS. Da mesma forma que os helpers são carregados automáticamente, o NEOS, ao renderizar as views, encontrando uma nova neosTag, carrega automaticamente a função correspondente que esteja nesta subpasta.

Você pode desenvolver suas próprias neosTags ou baixar novos packs do site do NEOS.

Vamos ver algumas neosTags para termos idéia de seu potencial:

View

Carrega uma view em substituição a neosTag.

Sintaxe:

```
<neos type="view" name="nome-da-view" />
```

O "type" determina o tipo "view"

Em "nome-da-view" indicamos o nome da view a carregar em substituição a essa neosTag. Pode ser incluído o caminho para uma subpasta, se necessário. A extensão não precisa ser indicada.

Exemplo:

```
<neos type="view" name="header"/>
<body>

      … minha página normal ...

</body>
<neos type="view" name="footer"/>
```

Neste exemplo uma view "header" e outra "footer", provavelmente comum a todas as views, serão carregadas automaticamente nos locais respectivos.

List e Numlist

Estas neosTags servem para criar uma lista e uma lista numérica, respectivamente, a partir de um array fornecido. Alternativamente, é possível indicar um link para cada item da lista.

Sintaxe:

```
<neos type="list" var="array"/>
```

O "type" pode ser "list" ou "numlist"; respectivamente: lista e lista numérica.

O "array" tem o seguinte formato: $array[' link '] = ' item '. Normalmente 'link' deve ser numérico. Caso queira que um link seja criado para o item corrente (como em um menu...) indique-o aqui. Este link será relativo ao site. Em 'item' indicamos o valor a ser exibido na listagem.

Exemplo:

No controller definimos o array:

```
$array ['home']     = 'página inicial';
$array ['download'] = 'baixar arquivos';
$array [ ]          = ' --- sem link --- ';
$array ['contato']  = 'fale conosco';
```

Na view:

```
…
  <neos type="list" var="array"/>
…
```

Depois de renderizado:

```
…
  <ul>
   <li><a href="http://site.com/home">página inicial</a></li>
   <li><a href="http://site.com/download">baixar arquivos</a></li>
   <li> --- sem link --- </li>
   <li><a href="http://site.com/contato">fale conosco</a></li>
  </ul>
…
```

Select

Esta é uma neosTag para gerar um "select" automaticamente, a partir de um array.

Sintaxe:

```
<neos type="select" var="array" />
```

Em "type" definimos o tipo da neosTag (select); indicamos, então, o array com os dados a serem usados para criar o select.

A formatação deste array é: $array[' valor '] = ' label ';

Se for necessário indicar uma das "options" como selecionada (selected) torne o label em um array com apenas um elemento. Assim: $array[' valor '] = array (' label ');

Também é possível tornar o select em list (multiple). Para isso é preciso fugir um pouco a regra e indicar o atributo "multiple" da seguinte forma: <neos type="select" var="array" multiple=""/>. O convencional seria indicar apenas "multiple" sem um valor. Mas, para a neosTags, se for deixado sem um valor (sinal de igual e aspas), ocorrerá um erro de interpretação no XML.

Exemplo:

No controller:

```
$array ['home']          = 'página inicial';
$array ['download']      = array('baixar arquivos');
$array ['contato']       = 'fale conosco';
```

Na view:

```
…
   <neos type="select" var="array" class="pages"  multiple=""/>
…
```

Depois de renderizado:

```
…
   <select class=”pages” multiple>
    <option value=”home” >página inicial</option>
    <option value=”download” selected="selected"  >baixar arquivos</option>
    <option value=”contato” >fale conosco</option>
   </select>
…
```

Sobre o Manual

Este manual foi escrito para dar uma pequena idéia da simplicidade e facilidade de uso deste framework.

Se você pensava que seria muito difícil usar o NEOS, espero que estas poucas linhas tenham lhe mostrado o contrário. Se você está acostumado a programar em PHP já pode instalar e usar o NEOS em seus próximos projetos, sem a necessidade de aprender mais nada!

Mas, se ainda estiver com dúvidas ou deseja fazer comentários, sugestões e críticas entre em contato. Estamos ansiosos para te conhecer e ajudar no que for possível.

Se você gostou do nosso framework então seja um colaborador deste projeto. Ajude-nos no desenvolvimento colaborando com classes, scripts, funções e aperfeiçoando o NEOS como um todo. Todas as modificações ou colaborações serão publicadas no site oficial do NEOS juntamente com suas licenças (indicadas por você) para que outros usuários tenham acesso.

Contatos

Site oficial: http://neophp.tk

E-mail: prbr@ymail.com

Publicação

Esta publicação é disponibilizada mediante a licença GPL2.

Nome da publicação: Manual de Usuário

Autor: Paulo R. B. Rocha

Data da publicação: 09/2010

Código da publicação: **A92D001**