Sitemgr user manual

Michael Totschnig

Table of Contents

Introduction **3**
User manual **6**
Administrator manual **10**
Template designer manual **13**
Module developper manual **13**

Introduction

This introductory section describes some concepts, you will need to understand if you want to get the most out of sitemgr. If you are just interested in a quick installation introduction go to section Installation

Template

Sitemgr builds web sites from templates. Those are stored in the directory sitemgr/sitemg-site/templates. Each of them has a directory of its own. The file main.tpl defines the template of the whole site, there can be other files that define code for separate areas of the page or for specific modules. Templates contain four kinds of variables. These are explained here since the types 2,3 and 4 can be used both in the template as in the page content a module generates.

1. {contentarea: areaname} These define where administrator and contributor provided content go. The names and the number of contentareas a template defines are almost arbitrary.
2. {modulename(?arguments)}These let you hardcode a call to a module into a template. Arguments are in the HTTP GET query string syntax (?key1=value1&key2=value2). At the moment, if you use this type of variables, you have to urlencode the query string yourself. Future versions of sitemgr might provide a simpler notation. For example,{lang_block] creates the select box for multilingual sites.
3. {?(phpgw | sitemgr:)link} This lets you create links either to other pages of your website, or to phpgroupware applications:
    a. Links to sitemgr either start with '?sitemgr:' or only with '?'. You can either link to a page with {?page_name=downloads] and [?page_id=4}, to a category's index (?category_id=5), to the site index (?index), to the site's table of contents (?toc). Just [?] or {?sitemgr:} links to the administrator defined home page.
    b. Links to phpgw start with '?phpgw:' . You have to name the application followed by a comma, and optionnally arguments the application can interpret. For example {?phpgw:/addressbook,order=n_given&sort=ASC}.
4. {variable} Finally there are some simple variables you can use to retrieve metainformation about the page or about the user's context:
    a. {title} the page title
    b. {subtitle} the page subtitle
    c. {sitename}the sitename the administrator has chosen for the site
    d. {sitedescription} the sitedescription
    e. {user}the user's account name

Module

The main function of a sitemgr module is to generate dynamic web site content. To make the development of new modules, and the use of modules easy and convenient, sitemgr defines an "abstract" class module which each module should extend. This parent of all modules, provides one essential service: It hooks the module into the content managment interface and permits the editing of the module's arguments that determine what content will be generated. Thus in order to create a new module, all you have to do, is to extend the abstract super module, define the modules

arguments, and write a get_content function that returns the content that should be displayed on the website. More on this in the chapter aboutModule developpment.

Argument/Content

A module can be seen as a transformation of input arguments into generated content. It is important to understand that the input arguments can be of completely different kinds. On the one hand there can be arguments that are as close as possible to the generated content. For example the html module's only argument is named "html-content" and in normal circumstances it is not transformed at all but handed over as is to the page generation engine. On the other hand arguments can play any conceivable role in the generation of content that is driven by data coming from other phpgroupware applications. They can be used to select between different categories of content, they can choose a certain format of presentation, they can function as search terms, etc.

Properties

A module can define properties. Properties are accessible to the modules get_content function in the same way as arguments, but they differ from them in two ways:

- Properties are edited by the site administrator. Their intended role is to put certain constrains on the workings of the module, so that a site administrator can enforce certain rules. For example the standard html module defines a property called striphtml. If this boolean property is set the module replaces all html in the generated content with entities.
- Properties are not defined with respect to a certain generated content block, but are defined either on a site-wide level or on the level of specific categories. Additionally you can differenciate on each scope properties for each content area. When a content block is generated, properties are searched for in a cascading way, i.e. when the block is specific to a specific page and contentarea and the module's properties are not defined for the combination of the contentarea and the page's category, then properties defined for higher scopes are looked for in a certain order. More on this in the chapter aboutAdministration.

Blocks, content areas and scope

There are three ways a module can generate content that will be displayed on a web page:

1. A template can contain hardcoded calls to modules that will generate content visible on each page. Thus you could put dynamic content that is visible across the whole site on a place of its own directly into the template. But beware that restrictions defined by the administrator also apply to these hardcoded module calls.
2. A template defines several content areas. This is one important differenced between the modularized sitemgr and previous versions where there were was only one place where contributor edited content went (page_content) and two places for administrator configured blocks (right_blocks, left_blocks). Now templates can define any practical number of content areas with arbitrary names. And there is no more much difference between central and peripheric areas. All can show administrator and contributor provided block content.With one exeption: The name "center" is special in that special pages that get generated by special URLs like "?toc" and "?index" or "?category_id=number" always put there page content (which is internally generated by nothing else

than a module) into this content area. If your template does not have a content area called "center" these special URLs won't work. Content areas can display module output, as edited by the page's contributors. We refer to each output of a module as a block. Here is another important difference to how sitemgr used to work: Until now there was a sharp distinction between page content which replaces the template variable page_content, and side blocks which got defined in a special file called blockconfig and replaced the template varialbes right_blocks and left_blocks. Now from the perspective of the page generation engine there is no more any difference between content areas, all display blocks of content generated by modules. The blocks one content area displays can be defined on different levels of scope: There are site wide blocks that are visible across the whole site, category wide blocks, that are visible on pages that belong to the category or any of its children, and finally are page blocks that define what distinguishes the page from other pages, and normally will be that what you'd call the page's main content.

3. The block content generated by a module can contain template variables of the same type as those that can be hardcoded. This is mostly useful for modules like the html module, where the contributor specified argument is nearly identical to the generated content. Thus a contributor can embed module calls inside the content of a certain block. This is done only once without any recursion, i.e. if a embedded module call returns itself a template variable it is not parsed and processed again.

Versioning

Sitemgr handles block content in versions. This means that you can have different versions of one block at the same time, but only one of them is visible on the website. This allows for working on several changes to the website and viewing them in a draft mode, before commiting them all at once to the production site. Sitemgr distinguishes five states for versions:

draft:

A draft version will not appear at the website at all. Draft versions are here for content you are not yet sure about.

prepublished:

A prepublished version will appear on the website if you view it in draft mode. A prepublished version is registered for publication, and it will change to published state, once you commit it.

published:

A published version is visible on your website both in production mode and draft mode.

preunpublished:

You put a version into preunpublished mode, if you want to remove it or replace it, but want to do this only by commiting other changes in the same time.

archived:

Archived versions will no longer appear anywhere but a special archive interface that allows you to reactivate archived content.

There are no versions of categories or pages, but they do can be in any of these five states. This means you can create a new category or a new page, but it will not appear on the website until you commit it.

Only one version of a content block can be in published, prepublished or preunpublished state at a time. There is one exeption: a prepublished and preunpublished version can exist together. When you commit the change, the prepublished version will become published and the preunpublished version will get archived.

Transformer

The architecture for sitemgr modules provides for the distinction between some form of raw content a module produces and the way it should get displayed on the web site, with the future possibility to plug other display types into the same framework. The idea is that the raw content of a module gets passed through different transformers, possibly even several transformers in a sequence at the time of content generation. Additionally this provides for a use of modules outside of sitemgr, for example remote retrieval of information with something like XML-RPC.

At the moment, a module does not need to use transformers on its own, it can directly generate html content, but if it does, sitemgr provides for an easy way to chain different transformers together. Thus a module can be constructed in different layers of functionality. For example a module's get_content function could return data in XML, and it defines a XSLT transformation to reorganize this data, and a second transformer for creating a user interface for display.

Transformers are also used on the level of the site template, insofar as each contentarea can have an associated transformer, which wraps the different content blocks into a common display format. This does the same thing as the file sideblock.tpl in former versions of sitemgr.

Translations

Sitemgr in its new modularized architecture continues to be fully multilingual. It is very simple to have a new module use this feature. All that is needed is to use a special flag in the definition of the module's arguments. All arguments that have this flag will be stored in a language specific database table, and can be translated in the translation manager, very similar to the way category and page definitions could already be translated to several languages.

User manual

Manage categories and pages

Create a new category

to be completed

Edit a category

to be completed

Delete a category

to be completed

Create a new page

to be completed

Edit a page

to be completed

Editing page content

There are two interfaces for creating and editing content blocks. The first one is called "content manager" and works inside the phpgw interface, the second works in interaction with the generated web site and we call it "editing mode".

The content manager

The interface for creating content blocks is the same on each level of scope, besides that when editing blocks on a lower level you can see all the blocks that have been defined on a higher level, and will be displayed on the website together with the blocks you are editing. I will refer to this interface as the content manager.

In each content manager, there is a section for each content area, where you can add blocks selected from a menu of all available modules. Once you have added a new block, it appears amidst all other blocks of the content area. Those defined on higher level scopes are only displayed, those pertaining to the scope (site-wide, category or page) you are managing are editable.

Editing mode

In order to use editing mode, you must add a site-wide administration block to your website (viewable only by registered users). This block has a dropdown menu where you can switch between production mode (viewing the website in its public state), draft mode (viewing the website in the prepublished state, i.e. how it will look after you commit any pending changes) and edit mode. In edit mode, you will see the same content as in draft mode, but in front of each content block you will see a link "Edit block". Activating this link will pop up a new window for editing it.

Editing a block

In both the content manager and the editing window that opens in editing mode, the interface for editing the block is the same. There are three standard interface elements you can edit for each block:

Title

: Each module defines a default title for blocks it generates. The block title is not necessarily displayed in each content area. It depends on the block transformer defined for the content area by the site template you use. Here you can override the default title with a customized title that will only be used by the block you are editing

Seenby

: You can control the visibility of each block for the different user roles defined by sitemgr: site administrators, phpgroupware users (which include site administrators) and the anonymous user. Evidently you can also make the block visible for everyone which is the default.

Sortorder

Here you can change the order in which the blocks are displayed by defining a different integer for each block.

When you first create a content block, sitemgr automatically creates a version for it. For each version, you have to edit the module specific arguments, each of which is preceded by an explanatory label. The input elements for arguments can be of different types: checkboxes, selectboxes, textareas, textfields. In the case of text input, you can use the same template variables as are used in the site template. But be aware that this only works if the module puts the same variable into its output, which need not necessarily be the case.

Even if not all blocks may make sense for each content area, the content manager does not impose on itselft any constraints on which modules you use for which area (Administrators can restrict the available modules for categories and content areas). For example you can use modules that are optimized for side block areas in the central area and you could use a simple html content block on side areas. The following modules are shiped with sitemgr:

administration

simply creates a link to sitemgr's administration interface. Thus you probably would not want to make it visible for anonymous user's but only for phpgw users or adminstrators.

amazon

shows ads for books on the Amazon website. There is a comment in the modules source file about how it works.

appdir

a demonstration of how sitemgr's architecture can be used to realize a simple application for creating a directory of information. It should be adaptable to other needs. At the moment, you need PHP's XSLT extension to use this module.

bookmarks

lets you show content from phpgw's bookmarks application

calendar

produces a calendar where each days links to phpgroupware's calendar application

currentsection

produces an index of the pages in the current section.

download

you can use this module for creating links to files you store with phpgw's filemanager

filecontents

this module lets you include the content of a file on your webserver into your website

forum

another demonstration of what sitemg's modules are intended for: This module displays the discussions of phpgroupware's forum application on the web site. It does not permit to post, since to implement this, in my humble opinion, the forum application has to be redesigned slightly.

galerie

creates a picture galery. You have to name both the filesytem path and the URL to a directory, where images that have a common filename, and are numbered beginning with one, are stored. Once the directory has been found, you can edit a subtext for each image.

google

displays a form for querying the google website.

hello

a simple “Hello world” module used below in this manual.

html

probably the most important module, since it plays the role, formerly the simple page content area had, ie. you add html content to the page.

index_block

a index of the whole site, formatted for peripheric areas of your web site.

index

is automatically used with the index GET parameter. You probably would not have to use this block otherwise.

lang_block

displays a select box for changing the user’s site language.

login

displays a login block

news

publishes news you edit with phpgroupware’s news_admin application. You can choose a category to display.

redirect

makes the page redirect to another URL. This is useful, if you want an entry in your menus that links to a page outside your sitemgr site. If you use this module you shouldn’t put any other blocks on the same page.

sitetree

displays a tree like menu to the website.

toc_block

the site’s table of contents, formatted for peripheric areas of your web site.

toc

is automatically used with the toc GET parameter. You probably would not have to use this block otherwise.

xml

is only a demonstration of how a module could serve XML content stored on your webserver. It does not do anything useful besides creating a browser from one file to the other. Files have to be named like images in the galerie module. You need PHP’s xslt extension to use this module.

Handling versions

For each content block you can create a new version simply by clicking on the “Create new version link”. Once you have several versions of a block, you can change their status at any time. If you want to immediately publish a block, you choose “published”. If you want to register a version for being published at the next commit, you choose “prepublished”. A version that is “preunpublished” is visible on the website, but will be archived at the next commit. Thus if you want to replace some content at the next commit, you have to put the published version into “preunpublished” and create a new version that you put into “prepublished” state. Archived versions will be no longer visible in the content managment interface, but will not be deleted from your database. They can be reactivated in the “Manage archived content” interface.

Manage translations

to be completed

Commit changes

In this interface you see a list of all categories, pages and blocks that are in prepublished or preunpublished state. You can return to any of them for further editing or changing their status, and you can choose between them for commiting. Prepublished content will go public, preunpublished content will go to the archive.

Manage archived content

to be completed

Administrator manual

Installation

1. Once you have the sitemgr directory inside your phpgroupware install you can setup it like any other phpgroupware application with the setup program (http://yourmachine/phpgw-path/setup/). You should also install the sitemgr-link application, which is is inside sitemgr and has to be moved up in the directory hierarchy.

   ```
   cd sitemgr
   mv sitemgr-link ..
   ```

   and then install sitemgr-link with setup

2. Log in to phpGroupWare as an admin and create an anonymous phpgw user and assign it a password. The only app (I assume) that they should have access to is sitemgr-link. sitemgr-link is a dummy application that redirects phpGW users to the generated site.
3. Users who you wish to see sitemgr (aka contributors) must be given acces to sitemgr, users who you want to be able to link to the sitemgr site from phpGW must be given rights to sitemgr-link. The easiest way to do this is to go to User groups and give groups permissions to use the applications.

4. The sitemgr-site is the directory that serves the dynamic web site. It is located inside sitemgr and works without moving it somewhere else. But it can be located anywhere. For example, you could put it in /var/www/html. You could make the root location of your web server point to it, if you wish (ie, http://yourmachine/ refers to /var/www/html/sitemgr-site if you moved the directory, or to /path/to/phpgroupware/sitemgr/sitemgr-site if you did not). Make a mental note of the directory where you put it and the url that it is accessed by.
5. In the sitemgr-site directory is a file called config.inc.php. You only have to edit it if you moved the directory. Change the value of the line

   ```
   'phpgw_path'            => '../../',
   ```

   so that the value of $sitemgr_info['phpgw_path'] is '/path/to/phpgroupware/'
6. You can handle different websites with sitemgr. The first thing to do is to define your new website. Sitemgr defines a default website upon installation, but you will probably have to redefine it. Log in as a phpgw administrator and go into phpgroupware's admin application and choose "Define websites" in the sitemgr section. You should see the default website listed, and can choose to edit it. A website is defined by the following values:

Site_name

: This is not used on the website, but only helps identify a website inside the administration interface.

Filesystem_path_to_sitemgr-site_directory

: If you did not move the sitemgr-site directory above you can leave this unchanged.

URL_to_sitemgr-site

: You can also leave this unchanged if you want to access your public website with a url that looks like http:/your.phpgw.url/sitemgr/sitemgr-site. If you want a different URL, you either have to move the sitemgr-site directory or to use an alias or a virtual server in your webserver's configuration.

Anonymous_user's_username

: the account name you created above.

Anonymous_user's_password

: and the corresponding password

Site_administrators

: Here you choose the users and/or groups that should have administrator rights for the website. They do not have to be adminstrators in phpgw's sense.

7. You can know log in as on one of the user's you gave administrator rights for the website.. Go to the sitemgr application and select "Configure SiteMgr". Here you can set different values that affect how your site will be presented.

Site_name

: is used mainly for metadata

Site_description

is used mainly for metadata

Default_home_page

here you can select the page that will show up first one your website. Evidently there won't be any choice until you create some pages.

Template_select

lets you choose between the different site designs that come with sitemgr or any you will create yourself.

Site_languages

If you want a multilingual site, you have to set them here. This will let you use the translation interface for translating your content and you can put a selectbox on your website where it's visitors can choose between different languages.

8. After installation sitemgr does not immediately know about all available modules. The setup routine only installs the modules html,index and toc that every site will need. In order to register all availabe modules select "Manage site-wide module properties" from the administrative menu, and then follow the "Register new modules" link.
9. That's it. Go to the Outline manager ("Manage categories and pages), add a category or three and check who can view and edit them, then add a page or three to each category, create some content for each page. You can also have content that is visible everwhere on your website ("Manage site-wide content") or on all pages that belong to a category ("Manage categories and pages" => "Manage category wide content"). Take care to put all categories and pages and blocks you create into published state, otherwise they will not be visible on the website.
10. Go view your recently created site by clicking on the sitemgr-link application. Voilà!

Maintenance

As a site administrator, you have three privileges/responsibilies:

1. Define site wide content blocks. These are edited in the same interface as category and page specific blocks.
2. Choose permitted modules: If you choose "Manage site-wide module properties" from sitemgr's main menu, you will see several rows which all contain four elements: a select box, a button labelled "Select allowed modules', another select box and a button labelled "Configure module properties". The first select box and its adjacent button in each row permits to choose lists of permitted modules. The first row defines a master list for the whole site which is used when no more specific lists are found for content areas or categories. The following rows each pertain to the different content areas of your site template. Here you can choose to allow some modules for one content areas, and other modules for another one. In the category manager, there is a button "Manage Modules" associated with each category. There you can use the same interface to define lists specific to one category (and all its subcategories, unless overriden). Again you have the choice between one list that pertains to all content areas, and specific lists for each category. When sitemgr has to find a specific value of permitted lists in a given context (a given contentarea in a given category) the following algorithm is used: First it aks for a value defined for the

pair contentarea/category. If none is defined, it climbs up the category hierarchy until the site wide value, each time looking for a value defined for the pair contentarea/category. If none can be found, it returns to the given category the search started from, and now asks for values defined for the given category but independent from the contentarea. If there is still none defined, it repeats the same traversal up the category hierarchy. This means that by simply defining one global master list of permitted modules, you can configure the whole site, if you do not need more fine grained control. The lists of permitted lists are never merged, if you define one list for a given context, this list is used in this context.

3. Define module properties: The lookup algorithm for module properties is exactly the same as for the lists of permitted modules. For each module you can set properties for the whole site, for content areas, for categories, or for combinations of content areas and categories. You access the property editor from the same page where you choose the list of permitted modules. You just use the second select box in each row. By selecting one module and clicking the "Configure module properties" button, you will open a interface for editing the module's properties which ressembles the interface for editing module arguments in the content manager. Be aware that only some modules define properties.

Template designer manual

One main idea behind sitemgr's modularized architecture is that all dynamic content on the website is produced by a module. This permits for a very structural way of defining functionality and for an easy way of extending functionality. With respect to former versions of sitemgr, this means that the templates have to be slightly modified:

- The whole page template is now stored in a file main.tpl in the template's directory.
- The variables page_content, left_blocks,right_blocks have to be replaced by content areas: {contentarea:center}, {contentarea:left}, {contentarea:right}. Only the contentarea center has a special semantics, since it is the hardcoded value for the display of table of contents and side index. All other contentareas can have arbitrary names, and you can have any practical number of them.
- A contentarea serves to display the content blocks, the site administrator and contributors define. Each contentarea can have its own way of wrapping html code around each content block. This at the moment defined in a class that implements the transformer interface, i.e. defines a function apply_transform($title,$content).This class' name is areaname_bt (for blocktransformer) and it is stored in file areaname_bt.inc.php inside the template directory. The function apply_transform just has to wrap the desired html around the content. It is free to ignore the title, for example the block title does not necessarily make sense in a page's central content area. A block transformer could apply other transformations to the content, but this would probably have counter-intuitive effects on your page's contributors[1].
- Other than that a template directory can contain template files that are specific to a certain module. For example the news module uses a file newsblock.tpl which is a standard API template. It is up to a module's developpers what kind of templates he wants to use. We propose to use a namespace for these module specific template files. For example a template used by module 'news' would go into a subdirectory 'modules/news' in each template directory. If the module does not find a template it needs in the site template's directory, it should look for a default template file, for example in "default/modules/news'.
- You can hardcode module calls into the template if your site should have the same dynamic content on one specific place.

Module developper manual

Sitemgr's parent module class, defines all the important functionality a module needs in its lifetime. Thus creating a new module can be as easy as creating a class that extends the standard module, defines the module's arguments if there are any, and has a function get_content that produces the module's content. Let's start with "Hello World".

```
<?php
class module_hello extends Module
{
    function module_hello()
    {
        $this->arguments = array(
            'name' => array(
                'type' => 'textfield',
                'label' => 'The person to say hello to'
            )
        );
        $this->title = "Hello world";
        $this->description = "This is a simple sample module";
    }
    function get_content($arguments,$properties)
    {
        return lang('Hello') . ' ' . $arguments['name'];
    }
}
```

Once your module is registered[2] and added to the list of permitted modules for some context, users can create blocks from this module: They will see in the content manager a textfield where they edit the argument name, and once the block is activated, it will generate the hello phrase on the website. Easy, isn't it?

Now let's examine more in detail how the standard module is constructed. This will help you understand in what way you can extend it to create more powerful modules. It defines the following functions:

add_transformer($transformer)

This function adds a transformer class to the module's transformer chain, so that when a block is generated from this module, its content will be passed through $transformer. This function is automatically called for block transformers, but you can use it on your own, if you want to separate in your module raw content from different forms of output. There is only one function a transformer class has to provide:

apply_transform($title,$content)

A transformer that is not a block transformer should normally ignore the title argument, and construct its return value from the content argument.

set_block($block,$produce=False)

This function is called by the content manager (with $produce=False) and by the page generation (with $produce=True) for each content block, so that the module knows everything about the block it has to edit or generate (above all its context and its arguments). If your module overrides this function, it should always call the parent class' set_block function first with parent::set_block($block). If you want to configure your module with respect to the block, you can do this here. This is also the place where your module should add the transformers it needs for generating output. For example:

```
function set_block($block)
{
    parent::set_block($block);
    if ($produce)
    {
        $this->add_transformer(new my_transform());
    }
}
```

get_properties()

This function looks up the value of the module's properties for the context of a block. There should not be much reason to override this function.

get_user_interface()

This function is responsible for creating the interface you use in the content manager when you edit a module's arguments. If you want this interface to show more than your module's arguments, youcan override this function. It must return an array of interface elements where each element is an array with two values associated to the keys label and form. You can even dynamically construct arguments, sitemgr's sample gallery module shows how to do this.

get_admin_interface($defaults)

This function creates the interface for editing module properties, it works in a similar way to get_user_interface.

get_translation_interface($fromblock,$toblock)

This function creates the interface for the translation manager. If your module makes use of sitemgr multilingual feature, and you have overriden get_user_interface, you'll probably have to override this function too.

build_input_element(($input,$default,$elementname)

This is a helper function for above functions. If you override one of above functions you can use build_input_element in the same way as the parent module does.

validate(&$data)

This function is called when a module's arguments are edited. The parent class simply returns true. When you override this function, you can alter the arguments, a reference to which is passed to the function, but you can also return false, and set the module's validation_error variable. In this case, the arguments will not be saved and the validation error is displayed to the user. For example we could add the following lines to our hello module:

```
function validate(&$data)
{
    if (preg_match("/[[:upper:]]/",$data['name']))
    {
        $data['name'] = strtolower($data['name']);
        $this->validation_error = "Name has been translated to lower case";
    }
    return true;
}
```

This would make sure that the module argument name would always be lowercase.

get_content(&$arguments,$properties)

This is the function every module needs. It produces the module's content. It is passed two arrays, one with the arguments for the block the module is generating, and the other with the properties that apply for the block's context. At the moment there is no constraint on what type of date the get_content function returns. It can be html, xml, an array, etc. But if it does not return html, you have to provide a transformer capable to produce html from the data get_content produces. The arguments are passed as a reference, because the get_content function can change them, and they can get stored automatically as session variable. This is because the parent module provides one other service: Your module can rely on an automatic handling of HTTP GET, POST and COOKIE variables, and of session variables. All you'd have to do is to define the arrays $this->get, $this->post, $this->cookie and $this->session. All members of these variables will be fetched from the GET, POST or COOKIE parameters, or from session variables and stored for you in the $arguments array. The entries of $this->session additionnaly will be stored after get_content returns to get_output. This can be very useful if you want your module to remain in a stable state while the user interacts with other modules on the same page.

The variables you define in these arrays can be identical to those in $this->arguments. In this case, if they are defined in the HTTP session context, they will override the values the page contributor has defined for the page. But they can be different variables that do not need an initial value provided by the page contributor. Whereas $this->get,$this->cookie and $this->session are simple arrays enumerating the variable names, $this->post is special because it can contain the element definition in the same way as $this->arguments, which can be used to programatically construct the form elements.

Your module does not need to use this service, it could directly read HTTP variables. The advantage of using it is that it provides a namespace for each module, so that if different modules that use the same variable names are used on the same page, no problems occur. If you use this service you can construct URLS automatically with the modules link function (see below), and if you construct the user interface, you have to provide the correct form element names for this service to work. The build_post_element function can help you do this. For example lets extend our hello module, so that the site user can choose his own name. Since we can no longer rely on the validation that is automatically done on contributor provided input. we call validate from the get_content function.

```
<?php
class module_hello extends Module  {
    function module_hello()
    {
        $this->arguments = array(
            'name' => array(
                'type' => 'textfield',
                 'label' => 'The person to say hello to'
            )
        );
       $this->post = array('name' => array('type' => 'textfield'));
       $this->session = array('name');
       $this->title = "Hello world";
       $this->description = "This is a simple sample module";
    }
    function get_content(&$arguments,$properties)
    {
       $this->validate($arguments);
       return lang('Hello') . ' ' . $arguments['name'] . '<br><form ac-
tion="' .
           $_SERVER['REQUEST_URI'] . '" method="post">' .
               $this->build_post_element('name',lang('Enter a name') .
           '</form>';
    }
```

```
    function validate(&$data)
    {
       if (preg_match("/[[:upper:]]/",$data['name']))
       {
           $data['name'] = strtolower($data['name']);
           $this->validation_error = "Name has been translated to lower case";
       return true;
    }
}
```

build_post_element($key,$default=False)

You can use this function from your module's get_content function to construct form elements. This works with the argument definition you put into $this->post. If you do not provide a default the current blocks value for the argument will be filled in.

link($modulevars)

helps you construct URLS with GET parameters that use the service described above. modulevars is an array of variable values keyed on variable names.

find_template_dir()

if a module uses a different template (of whatever kind) for different site themes, this function can help finding the template directory inside the theme's directory, if it follows the namespace described above, or if it cannot be found name the default directory.

get_output($type='html')

This is the function that is actually called by the page generation engine, since it not only calls the module's get_content function, but makes sure that all transformers that have been added to the modules transformer_chain get called. For type argument is not really used at the moment, but future versions of sitemgr could be extended so that modules could produce output in different formats by specifying different transformers for each output type. Your module should not need to override get_output.

To summarize, there are the following requirements for a sitemgr module:

1. It is written as a class called module_name and extends the class Module. It must be put into a file called class.module_name.inc.php and put into the inc directory of any phpgroupware application.
2. Its constructor should define the following member variables:
    a. arguments: the module's arguments a contributor can edit in order to create content blocks from the module. Each argument needs to define a label and the type of input element used to edit it in the content manager. Parameters for these input elements (like size for textfields, cols and rows for textareas can be defined). Translatable arguments can be specially flagged with a i18n entry in the arguments definition array.
    b. properties: the module's properties the site administrator can edit in order to constrain or configure the functionnality of the module
    c. title: The module's default title that can be overriden for each content block.
    d. description: A short descriptive text, that is displayed in the content manager and module manager (for example when you put the mouse over an entry in the module select lists).

3. It needs a get_content function that has access to arguments and properties and produces the block content.
4. If the content returned by get_content is something different from HTML, the module has to define transformer classes, and should add them to the module's transformer chain. This can be done in the constructor, but the best place for it is the set_block function
5. The parent module class provides a user interface for editing module arguments. If a module needs a customized interface or wants to construct arguments dynamically, it can override the get_user_interface function.

These are the building blocks which should allow for some flexibility in constructing modules that make phpgroupware managed data visible on a sitemgr web site.

Notes

1. For compatibility with how sitemgr used to work, the file sideblock.tpl which uses the two variables block_title and block_content is still recognized as a template for a transformation of the two contentareas left and right. The transformer is automatically created by the template class. Thus you could use one file sideblock.tpl instead of the two files right_bt.inc.php and left_bt.inc.php.
2. Modules must be stored in the directory /path/to/phpgroupware/sitemgr/modules and be named class.module_modulename.inc.php in order that sitemgr can find them.