CGIKit User Guide

Contents

Install

System Requirements

- Operating system like UNIX
- Ruby 1.8.0 or later

The following can speed up CGIKit.

- mod_ruby

Installing CGIKit

To install CGIKit, use make command or copy libraries to directory under unix path variable.

List 1-1: Installing with install.rb

```
% tar xzf cgikit-xxx.tar.gz
% cd cgikit-xxx
% ruby install.rb config
% sudo ruby install.rb install
```

List 1-2: Installing with copying libraries

```
% tar xzf cgikit-xxx.tar.gz
% cd cgikit-xxx
% sudo cp lib/* /usr/local/lib/ruby/lib/site_ruby/1.8
```

Loading Libraries

You must load `cgikit.rb` to use CGIKit. Load with "`require`" if you installed, or add path of directory for CGIKit before loading.

List 1-3: Loading installed CGIKit

```
require 'cgikit'
```

List 1-4: Loading without installing CGIKit

```
# Path for CGIKit
$LOAD_PATH << './cgikit'
require 'cgikit'
```

Architecture

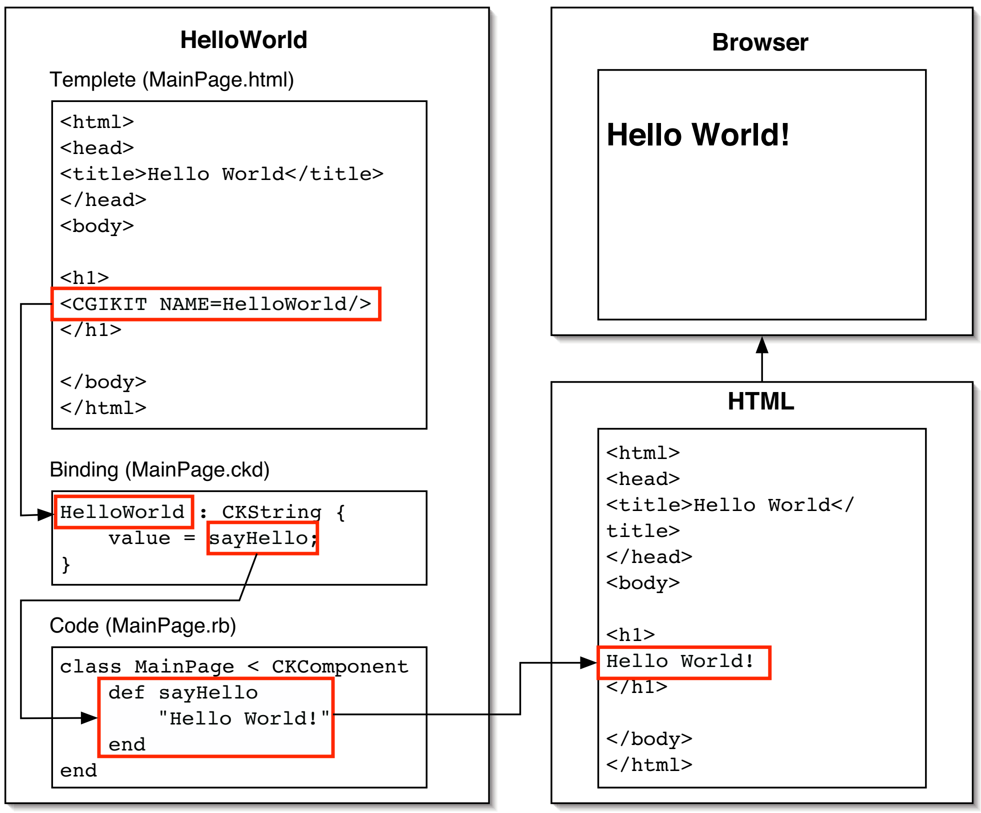
Basic Concept

In CGIKit, web pages are created by components, which uses some elements. So, elements and components play an important role.

Component is consisted of template, ckd file and Ruby script. By using these three parts, components are changed to HTML. In CGI programs, it often occurs that HTML is embedded in program. This archittttecture makes it possible to separate View(template) and Logic(Ruby script).

Element is something like HTML tag peculiar to CGIKit. Elements are used in template of components. The elements used in template are outputed as HTML by binding the elements to Ruby's methods or objects.

Figure 2-1: Architecture



```
<html>
<head>
<title>Hello World</title>
</head>
<body>

<h1>
<CGIKIT NAME=HelloWorld/>
</h1>

</body>
</html>
```

```
HelloWorld : CKString {
    value = sayHello;
}
```

```
class MainPage < CKComponent
    def sayHello
        "Hello World!"
    end
end
```

```
<html>
<head>
<title>Hello World</
title>
</head>
<body>

<h1>
Hello World!
</h1>

</body>
</html>
```

Development in CGIKit is that of components by combining CGIKit elements and Ruby's methods/objects. CGIKit provides many varied elements. One shows Ruby object simply. Another judges its condition and decides whether it shows HTML or not.

Component

Component represents some part of web page (component sometimes represents the while web page). Depending on client's request, CGIKit creates components and converts them to HTML.

Composition

Component has 3 parts, **template**, **binding** and **code**. Template is HTML and Code is Ruby script. In binding (ckd file), you can determine types of the elements used in template and the behavior of the elements. According to this definitions, CGIKit binds template to code.

Normally, these files (HTML, ckd file and Ruby script) of component are located in the same directory whose name is the same as component's name. In addition to it, the names of HTML, ckd file and Ruby script are different only in their suffix.

For example, if you want to create "`MainPage`" component, you name HTML, ckd file, and Ruby script like this.

List 2-1: Directory for a component

```
HelloWorld.cgi
/MainPage
  MainPage.html
  MainPage.ckd
  MainPage.rb
```

Template (.html)

In template, you write normal HTML and CGIKit elements. You can locate elements as "`<CGIKIT>`". CGIKit tag is case-insensitive.

This is an example.

List 2-2: HelloWorld (MainPage.html)

```
<html>
<head>
<title>Hello World</title>
</head>
<body>

<h1>
<cgikit name="HelloWorld"></cgikit>
</h1>
```

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

CGIKit tag has only one attribute, "name". And, the value of "name" attribute is also used in ckd file. The value don't need to be enclosed by double quotation mark ("). Of course, you can enclose the value by the mark.

Tags that have no bodies can be empty tags. Using empty tags for elements like CKString, CKSubmitButton, etc., is easy to write CGIKit tags.

List 2-3: Empty tag

```
<cgikit name="HelloWorld"/>
```

Comment

Comment of HTML is interpreted because it is necessary to include something like CSS and Javascript. If you want to comment out some parts of a template, you use "`<!--- ... --->`". The format is like HTML comments, but with an additional hyphen.

Binding (.ckd)

In ckd file, you decide types of the elements and their behavior. Please be care of the grammar of ckd file. Its grammar is different from that of Ruby or HTML. In ckd file, all the attributes of the element are enclosed by braces, "{}", and an attribute is separated by semicolon, ";". You can omit semicolon with using return as term.

List 2-4: ckd file

```
name of element : type of element {
  attribute = value

  # semicolon
  attribute = value; attribute = value;
}
```

The value of the attribute is Ruby's method, string, number and true/false. When you use string in ckd file, you surround the value with single/double quotation mark.

List 2-5: HelloWorld (MainPage.ckd)

```
HelloWorld : CKString {
  value = sayHello;
}
```

In this ckd file, you declare that you use "`HelloWorld`" element whose type is CKString and that "`say_hello`" method of HelloWorld class is set to "`value`" attribute of "`HelloWorld`" element.

Literals

You can use these literals in binding files.

Table 2-1: Literals

Literal	Format
String	’abc’, "string", ...
Digit	1, 2, 3, 4, 5, ...
true/false, nil	true, false, nil, ...
Array	[array], [0], [1], ...
Hash	[key], [’abc’], ["string"], [0], [1], ...

Code (.rb)

Code is Ruby script. The name of component’s class must be the same as component’s name. Besides, the component’s class must inherit CKComponent.

List 2-6: HelloWorld (MainPage.rb)

```
class MainPage < CKComponent
  def sayHello
    "Hello World!"
  end
end
```

To initialize in component, use init() that initializing method without arguments instead of initialize().

Process of Binding

CGIKit binds elements in template to code. After binding, CGIKit shows HTML by converting the elements. Compoents and elements are changed to HTML like below.

1. CGIKit reads template of the specified component.
2. When "`<CGIKIT>`" is found, CGIKit searches the entry in ckd file which corresponds to the CGIKit tag in template. For example, if "`<cgikit name=foo>`" is found in template, CGIKit looks for the entry of "`foo`" in ckd file.
3. The Elements are bound to code.

4. The Elements are converted to HTML.

Accessor Method

You need not to define accessor methods for instance variables to use with elements. CGIKit uses accessors if defined, or accesses instance variables directly.

Form Data

In binding, the form data is assinged to the component through the methods to which you bind textfield, button and so on. You don't forget to define accessor methods.

Elements

Element is a framework which shows Ruby's objects/methods as HTML. To use elements is the core of CGIKit development.

Because component is also one type of elements, components can be nested. As a result of it, the web page is formed by some components and elements. In this case, the top-level component usually determines the layout of the web page, other components and elements gives the top-level component their result of HTML conversion.

Attribute

Element has attributes which determine its behavior. By binding these attributes to Ruby's objects and methods, the objects are embeded in the outputed HTML. Because Ruby objects are accessed through Ruby's method, all the subjects to bind are ckd's literal or Ruby's methods.

List of Elements

Currently, there are 19 types of elements. CKString is used the most frequently. This element simply shows the result of its binding. Other than CKString, there are varied elements. For example, CKConditional controls the display of its content. CKRepetition shows its content repeatedly by iterating `"list"` attribute. Here, we introduce some elements briefly. The detail is available as "Dynamic Elements".

Table 2-2: General

Element	Description
CKString	Shows the result of binding simply.
CKHyperlink	Links to other component or normal URL.

Table 2-2: General

Element	Description
CKImage	Shows an image in resource directory.

Table 2-3: Flow Control

Element	Description
CKConditional	Decides whether the element shows its content by the result of binding.
CKRepetition	Repeats its content.

Table 2-4: Form

Element	Description
CKForm	Shows a HTML form. The form data is assigned to a component through bound methods.
CKTextField	Shows a textfield.
CKRadioButton	Shows a radiobutton.
CKCheckbox	Shows a checkbox.
CKPopUpButton	Shows a pop-up button.
CKText	Shows a textarea.
CKBrowser	Shows a list where you can select multiple items.
CKFileUpload	Shows a file-upload field.
CKSubmitButton	Shows a submit-button.
CKResetButton	Shows a reset-button.

Table 2-5: Coordination of Component

Element	Description
CKFrame	Sets components in frame.
CKComponent	Locates elements or components in itself.
CKContent	Shows the grandparent’s content.
CKGenericElement	Generates a generic HTML tag.

Process of CGIKit

Startup

The startup of CGIKit is carried out by startup-program as cgi program. The startup-program is different from components. Normally, it is not neccessary to require components directly in startup-program.

The startup-program does three things.

1. Creates a CKApplication object.
2. Sets parameters of the CKApplication object.
3. Calls CKApplication#run.

Creates a CKApplication object

CKApplication is the central class in CGIKit. This class has the parameters, for example, CGI program's path, MainPage and component path. You can create a CKApplication object by calling CKApplication.new simply.

List 2-7: Creating a CKApplication object

```
app = CKApplication.new
```

Sets parameters of the CKApplication object

CKApplication has many attributes. Here, two of them are introduced. The detail is explained in CKApplication's RDoc document.

Table 2-6: Parameters of CKApplication objects

Parameter	Description
`element_id`	Component name which CGIKit shows.
`main`	Component name which CGIKit shows if target is not set. The default value is "`MainPage`".

Calls CKApplication#run

Finally, you call CKApplication#run. By this method, CKApplication loads a component and initializes it.

When CKApplication#run is called, CKApplication decides what component is shown. CKApplication has two ways to decide the name of the top-level component to be shown. One way is CKApplication#element_id and another is query data. If CKApplication#element_id is set as CKElementID object in startup-program, CKApplication loads the component whose name is CKApplication#element_id. If CKApplication#element_id is not set, CKApplication tries to decide the component's name from query data. For example, when a client accesses "`http://localhost/hello.cgi?element_id=FooBar`" CKApplication loads `FooBar` component. If both of ways fails, CKApplication load the component specified by CKApplication#main.

HelloWorld.cgi

This is one of the simplest startup-program.

List 2-8: HelloWorld (HelloWorld.cgi)

```
#!/usr/local/bin/ruby

require 'cgikit'

app = CKApplication.new
app.run
```

After startup

Ordinally, you don't have to know the detail after startup. But, if you want to know about the process after startup, see the source and document of CKApplication. The document is provided in RDoc document.

Here, the process after startup is explained succinctly.

1. When CKApplication#run is called, the CKApplication object creates a CKAdapter object, which is an interface between CGIKit and a web server.
2. The CKApplication object gets a request object from the CKAdapter object. Then, as explained above, the CKApplication object determines the component to be shown from the parameters of request object.
3. The CKApplication object loads the specified component and converts it to HTML.
4. The HTML is added to a response object created by the CKApplication object. The CKApplication object sends the response object to the CKAdapter object.
5. The CKAdapter object receives the response object and shows it to the browser.

Dynamic Elements

Optional HTML attributes and other attribute

All elements can have optional HTML attributes `other` attribute (elements without displaying HTML tags like CKString and CKRepetition, etc. do nothing). If you set attributes of HTML as key and value for elements, the HTML attributes are added for output HTML tag.

`other` attribute adds setted string to HTML tag. The attribute is used to set HTML attribute without value.

List 3-1: Setting optional HTML attributes and “other” attribute

```
Link : CKHyperlink {
  href  = “http://www.foobar.com/“
  key   = “value”
  other = “anykeywords“
}

# <a href=”http://www.foobar.com/” key=”value” anykeywords></a>
```

Validating input

You can validate input with using `validate` and `pass` attributes for CKTextField and CKText. If the input pass validation, variable for `pass` attribute is true, or `false`.

Example: validating mail addresses

For example, we validate a text field to input mail addresses. We need to write code for checking whether the mail addresses exist or not, but we can check format of them with validating input.

In this case, we check the addresses include at mark (@). If the addresses don t include at mark, `pass_mail` variable is `true`.

List 3-2: Validating mail addresses

```
Mail : CKTextField {
  value    = mail
  validate = "mail =~ /[^@]+@(.+)/"
  pass     = pass_mail
}
```

This example is attached to archive as Registration application.

Format for rules

Examples of format are the following.

List 3-3: Examples of format

```
name == ‘MyName’
(title =~ /R/) and (title.size > 10)
not (count < 20)
```

Format for rules is attribute operator value (Attribute is accessor method or instance variable defined in component class). You can join rules with `and`/`or`, use `not` to deny rules. If you use the operators, enclose rules with parenthesis.

Converting data types

Last rule In the above example, set value as number. Form data is setted as string for the valule, but the data is converted temporarily when validating. For example, the following validates input as number whether greater than or equal to 100 and less than equal to 500.

List 3-4: Validating input as number

```
Number : CKTextField {
  value    = number
  validate = “(number >= 100) and (number <= 500)“
  pass     = pass_number
}
```

Operators

Operators can be included in rules are the following.

Table 3-1: Operators in rules

Operator	Description
`==`	Both sides are equal.
`!=`	Both sides are not equal.
`>`	Left is greater than right.
`<`	Left is less then right.

Table 3-1: Operators in rules

Operator	Description
`>=`	Left is greater than or equal to right.
`<=`	Left is less than or equal to right.
`=~`	Pattern matching.

General elements

CKString

CKString shows the result of binding as string.

Required attribute: `value`

Table 3-2: Attributes of CKString

Attribute	Type	Description
`value`	`String`	Text to be displayed.
`escape`	`true/false`	Escapes HTML control characters in `value` if the `escape` is true. The default value is true.
`empty`	`String`	Text that is substituted for `value` when the `value` is nil.

CKHyperlink

CKHyperlink generates a hypertext link.

Required attributes: `action`, `href` or `page`

Table 3-3: Attributes of CKHyperlink

Attribute	Type	Description
`action`	`CKComponent`	Method to be invoked when the link is clicked.
`enabled`	`true/false`	Generates a non-active link if the value is true.
`href`	`String`	You specify the URL to other web page directly. This attribute prevails over `action` or `page` attribute.

Table 3-3: Attributes of CKHyperlink

Attribute	Type	Description
`page`	`String`	Name of component to display when the link is clicked.
`string`	`String`	Text of the link. If the body of CKHyperlink tag is not empty, the body is displayed. For example, if the template includes `<cgikit name=link>foo</cgikit>`, this element shows "foo" as the link.
`target`	`String`	target attribute of HTML's`<a>` tag.
`secure`	`true/false`	Appends "`https://`" to the URL if the value is true. The default value is false.
`query`	`Hash`	Hash as the query string. The value of `query` attribute is converted to string. Then, the string is added to append the URL.

CKImage

Creates an image tag.

Required attributes: `file`, `src`, or `data`

Table 3-4: Attributes of CKImage

Attribute	Type	Description
`alt`	`String`	Alternative text to the picture.
`border`	`Integer`	Size of image border.
`width`	`Integer`	Width of image.
`height`	`Integer`	Height of image.
`file`	`String`	Name of an image file in web server resource directory.
`src`	`String`	You specify an image file directly. This value prevails over `file` attribute.
`data`	`CKByteData`	A CKByteData object to display as image. If you create the object without resource manager, You must use this with `mime` attribute.
`mime`	`String`	MIME type for a resource of `data` attribute.

Flow control

CKConditional

Controls generating HTML.

Required attribute: `condition`

Table 3-5: Attributes of CKConditional

Attribute	Type	Description
`condition`	`true/false`	If the value is true and `negate` is false, the body of the CKCoditional tag is displayed.
`negate`	`true/false`	Inverts the meaning of the `condition`.

Table 3-6: Contorol Table

condition	negate	Result
true	false	show
false	false	not show
true	true	not show
false	true	show

CKRepetition

A CKRepeition object repeats its contents.

Required attributes: `list` and `item`, or `count`

Table 3-7: Attributes of CKRepetition

Attribute	Type	Description
`count`	`Integer`	CKRepeition repeats its contents this number of times.
`list`	`Enumerable`	Array which is iterated through.
`item`	`-`	Current item when the list is iterated through.
`index`	`Integer`	Index of the current item.

Form

CKForm

Creates a fill-in form. Dynamic Elements of form, for example, CKBrowser, CKCheckbox, CKRadioButton, CKPopUpButton, CKText, CKTextField, CKSubmitButton and CKResetButton, are used within CKForm or HTML form. Required attributes are none.

To upload files with form, set enctype attribute to multipart/form-data or fileupload attribute to true. Even if you set multipart form, form data is processed as String excepted data setted content type. The data is used by CKFileUpload as CKByteData.

Table 3-8: Attributes of CKForm

Attribute	Type	Description
`method`	`String`	Method to send form data. You can use POST, GET or HEAD as the value.
`enctype`	`String`	Method to encode form data. Set "`multipart/form-data`" to this attribute when using CKFileUpload.
`href`	`String`	URL to which the browser directs.
`target`	`String`	Frame in a frameset that receive the page.
`query`	`Hash`	Hash as the query string. The value of `query` attribute is converted to string. Then, the string is added to append the URL.
`fileupload`	`true/false`	If you set this attribute to true, `enctype` attribute is setted to `"multipart/form-data"`. You can use this instead of `enctype` when using CKFileUpload.

CKTextField

Creates a text input field. This element must be used within CKForm or HTML form.

Required attribute: `value`

Table 3-9: Attributes of CKTextField

Attribute	Type	Description
`type`	`String`	Type of the text field. `text` is for a normal text input field, `password` is for a password field and `hidden` is for a hidden field.
`value`	`String`	Value of the text field. If you set an accessor method to this element, the form data is set to a component automatically by the method.
`size`	`Integer`	Size of the text field.
`maxlength`	`Integer`	Max length of data for the text field.
`validate`	`String`	Format string to validate input value.
`pass`	`true/false`	If validating is passed, the value is true.
`enabled`	`true/false`	If the value is false, the element appears but is not active.

CKText

Creates a text area. This element must be used within CKForm or HTML form.

Required attribute: `value`

Table 3-10: Attributes of CKText

Attribute	Type	Description
`value`	`String`	Value of the text area.
`columns`	`Integer`	Column size.
`rows`	`Integer`	Row size.
`validate`	`String`	Format string to validate input value.
`pass`	`true/false`	If validating is passed, the value is true.
`enabled`	`true/false`	If the value is false, the element appears but is not active.

CKCheckbox

Creates a checkbox. This element must be used within CKForm or HTML form.

Required attributes: `selection` and `value`, or `checked`

Table 3-11: Attributes of CKCheckbox

Attribute	Type	Description
`checked`	`true/false`	If neither `value` nor `selection` attribute is nil and the value of `selection` is equal to that of `value`, the check box is checked.
`value`	`String`	When the check box is checked, the value of `value` attribute is set to the component by the method specified by `selection` attribute.
`selection`	`Array`	Object that the user chose from the check box.
`enabled`	`true/false`	If the value is false, the element appears but is not active.

You use this element in two ways. One it the way to use `checked` attribute. The other is the way to use both `selection` and `value` attributes.

checked attribute

If you use checkboxes with `checked` attribute, the checkboxes are controlled with on/off.

List 3-5: Template

```
<cgikit name=Form>
<cgikit name=Checkbox1>One</cgikit>
<cgikit name=Checkbox2>Two</cgikit>
<cgikit name=Checkbox3>Three</cgikit>
<cgikit name=Submit/>
</cgikit>
```

List 3-6: Binding

```
Form : CKForm {
}

Checkbox1 : CKCheckbox {
  checked = checkedOne;
}

Checkbox2 : CKCheckbox {
  checked = checkedTwo;
}

Checkbox3 : CKCheckbox {
  checked = checkedThree;
}
```

```
Submit : CKSubmitButton {
}
```

List 3-7: Code

```
class Checkbox < CKComponent
  attr_accessor :checkedOne, :checkedTwo, :checkedThree
end
```

The variables are substituted true when the checkboxes clicked.

value and selection attributes

Another usage of CKCheckbox is combination with `selection` and `value` attributes. `selection` attribute is substituted value of `value` attribute when a checkbox clicked. Checkboxes turn on if values of `selection` and `value` attributes are equal.

List 3-8: Template

```
<cgikit name=Form>
<cgikit name=Checkbox1>One</cgikit>
<cgikit name=Checkbox2>Two</cgikit>
<cgikit name=Checkbox3>Three</cgikit>
<cgikit name=Submit></cgikit>
</cgikit>
```

List 3-9: Binding

```
Form : CKForm {
}

Checkbox1 : CKCheckbox {
  value = "One";
  selection = checkedOne;
}

Checkbox2 : CKCheckbox {
  value = "Two";
  selection = checkedTwo;
}

Checkbox3 : CKCheckbox {
  value = "Three";
  selection = checkedThree;
}

Submit : CKSubmitButton {
}
```

List 3-10: Code

```
class Checkbox < CKComponent
  attr_accessor :checkedOne, :checkedTwo, :checkedThree
end
```

CKRadioButton

Creates a radio button. This element must be used within CKForm or HTML form.

Required attributes: `selection` and `value`, or `checked`

Table 3-12: Attributes of CKRadioButton

Attribute	Type	Description
`name`	`String`	Name that identifies the radio button's group.
`checked`	`true/false`	If neither `value` nor `selection` attribute is nil and the value of `selection` is equal to that of `value`, the check box is checked.
`value`	`String`	When the check box is checked, the value of `value` attribute is set to the component by the method specified by `selection` attribute.
`selection`	`String`	Object that the user chose from the check box.
`enabled`	`true/false`	If the value is false, the element appears but is not active.

This is a sample with `checked` attribute.

List 3-11: Template

```
<cgikit name=Form>
<cgikit name=Radio1>One</cgikit>
<cgikit name=Radio2>Two</cgikit>
<cgikit name=Radio3>Three</cgikit>
<cgikit name=Submit></cgikit>
</cgikit>
```

List 3-12: Binding

```
Form : CKForm {
}

Radio1 : CKCheckbox {
  name = "radio";
  checked = checkedOneTwoThree;
}

Radio2 : CKCheckbox {
  name = "radio";
  checked = checkedOneTwoThree;
}
```

```
Radio3 : CKCheckbox {
  name = "radio";
  checked = checkedOneTwoThree;
}

Submit : CKSubmitButton {
}
```

List 3-13: Code

```
class Checkbox < CKComponent
Å@Å@attr_accessor :checkedOneTwoThree
end
```

CKPopUpButton

Creates a pop-up menu. This element must be used within CKForm or HTML form.

Required attribute: `list`

Table 3-13: Attributes of CKPopUpButton

Attribute	Type	Description
`escape`	`true/false`	Escapes HTML control characters in the items of the list if the `escape` is true. The default value is true.
`list`	`Enumerable`	Array which is iterated through.
`default`	`String`	The first item if no item is selected.
`selected`	`String`	Item that are chosen from the selection list.
`values`	`Array`	Array which is value for each `value` attributes of `<option>` elements.
`enabled`	`true/false`	If the value is false, the element appears but is not active.

CKBrowser

Creates a list whose multiple items can be selected. This element must be used within CKForm or HTML form.

Required attribute: `list`

Table 3-14: Attributes of CKBrowser

Attribute	Type	Description
`escape`	`true/false`	Escapes HTML control characters in the items of the list if the `escape` is true. The default value is true.
`list`	`Enumerable`	Array which is iterated through.
`selected`	`Array`	Items which are chosen from the list.
`values`	`Array`	Array which is value for each `value` attributes of `<option>` elements.
`multiple`	`true/false`	Multiple items of the list can be selected if the value is true.
`size`	`Integer`	Size of item in appearance.
`enabled`	`true/false`	If the value is false, the element appears but is not active.

CKSubmitButton

Creates a submit button. This element must be used within HTML form. Required attributes are none.

Table 3-15: Attributes of CKSubmitButton

Attribute	Type	Description
`action`	`CKComponent`	Method to invoke when the button is clicked.
`value`	`String`	Title of the button.
`enabled`	`true/false`	If the value is false, the element appears but is not active. In addition, it doesn't send the form data although the button is clicked.

CKResetButton

Creates a reset button. This element must be used within CKForm or HTML form. Required attributes are none.

Table 3-16: Attributes of CKResetButton

Attribute	Type	Description
`value`	`String`	Title of the button.

CKFileUpload

CKFileUpload generates an input form to upload files. To use this, set `enctype` attribute of CKForm to "`multipart/form-data`".

Required attributes: `data` and `file`

Table 3-17: Attributes of CKFileUpload

Attribute	Type	Description
`data`	`CKByteData`	Variable of the attribute is set the uploaded file as a CKByteData object.
`file`	`String`	Path of the uploaded file.
`enabled`	`true/false`	If the value is false, the element appears but is not active.

Reusable Components

CKFrame

CKFrame generates frame tag in HTML.

Required attributes: `page` or `src` or `value`

Table 3-18: Attributes of CKFrame

Attribute	Type	Description
`name`	`String`	`name` attribute of HTML’s`<frame>` tag.
`page`	`String`	Name of component that supplies the content for the frame.
`src`	`String`	You specify the URL or file for the frame.
`value`	`CKComponent`	Method that supplies the content. The parent of this element must have the specified method.

To use frames, ready components that include framesets and CKFrame elements for frame components.

List 3-14: Template

```
<frameset cols="200,*">
<cgikit name=Index></cgikit>
```

```
<cgikit name=Contents></cgikit>
</frameset>
```

List 3-15: Binding

```
Index : CKFrame {
  name = "Index";
  page = "IndexPage";
}

Contents : CKFrame {
  name = "Contents";
  page = "IntroductionPage";
}
```

CKComponent

You can use one component in another component like elements. It means that it is possible to nest components with specifying in binding files.

List 3-16: Nesting MainPage component

```
OtherComponent : MainPage {}
```

Components has no attributes like elements. Instead of this, components' instance variables are as attributes.

List 3-17: Code (MainPage)

```
class MainPage < CKComponent
  attr_accessor :title
end
```

List 3-18: Binding (parent's component for the MainPage)

```
OtherComponent : MainPage {
  title = "Example for CKComponent";
}
```

MainPage component's`title` attribute is substituted "Example for CKCompo-nent".

CKPartsMaker

You can write a component which converts itself to a part of a web page. In some cases, a web page is composed of these components. These components are called parts component. Parts componente is recommended to include CKPartsMaker. A component which includes CKPartsMaker isn't displayed even if its name is set to the CKApplication#target.

A name of parts component is recommended to have "Parts" or Component at the end of the name to distinguish it from page component.

Table 3-19: Object attributes of CKPartsMaker module

Object attribute	Description
`substitute_page`	When CGIKit recieves requests to show component parts, CGIKit shows the page specified by this attribute. A main page of an application is displayed when the value is not defined.

CKContent

CKContent is used in nested components. This element tag in the template is replaced with a part of the template of its grandparent component. CKContent has no attributes.

List 3-19: Template (parent's component)

```
<cgikit name=OtherComponent>Content of parent</cgikit>
```

List 3-20: Binding (parent's component)

```
OtherComponent : MainPage {}
```

List 3-21: Template (nested component)

```
<b><cgikit name=Content></cgikit></b>
```

List 3-22: Binding (nested component)

```
Content : CKContent {}
```

List 3-23: Result

```
<b>Content of parent</b>
```

CKGenericElement

CKGenericElement generates generic HTML tags.

Required attribute: `tag`

Table 3-20: Attributes of CKGenericElement

Attribute	Type	Description
`tag`	`String`	Name of the HTML tag. If the attribute is nil, body enclosed by the element or `string` attribute are displayed.
`enabled`	`true/false`	Enables or disables the tag. If the attribute is false, body enclosed by the element or `string` attribute are displayed.
`string`	`String`	String to display if body enclosed by the element isn't exist.
`option`	`String`	String to append for the open tag. For example, `checked` or `selected`.
`form_value`	`String`	If the element is form, the attribute is setted form datas as a string.
`form_values`	`Array`	If the element is form, the attribute is setted form datas as an array.
`invoke_action`	`CKComponent`	If the element is executable (hyperlink, button, etc.), the method is called when clicked.

You can define other voluntary attributes. The attributes is appended to the tag in format as "`attribute=value`".

Cookie

CKCookie is a class for cookie. To send cookies to a browser needs to create cookie objects and set them to a response object. Instead of creating cookie objects, you can also get cookie objects from a request object.

Class CKCookie

CKCookie objects have a pair of a cookie name and value. If you make the objects have multiple values for one name, you must write code by yourself.

Table 4-1: Object attributes

Attributes	Description
`name`	Name of the cookie.
`value`	Value of the cookie.
`path`	Restricts the cookie in the site.
`domain`	Domain that can receive the cookie.
`expires`	Expiry date. You set Time object to the cookie object. The value is formatted when the cookie is returned.
`secure`	Decides whether the cookie is encrypted or not.

Controlling cookie objects

Creating cookies

Give arguments of initialize() a name or a pair of name/value. The value of cookie is omittable.

List 4-1: Creating cookies

```
cookie = CKCookie.new( name, value )
```

Getting cookies from a request object

CKRequest has some methods for getting cookies. The methods are cookie(key), cookies, cookie_value(key), cookie_values(key). You can get CKRequest objects by CKApplication#request.

Table 4-2: Getting cookies methods of CKRequest

Method	Description
`cookie(key)`	Returns a CKCookie object whose `key` is the same as the argument.
`cookies`	Returns an array of CKCookie objects.
`cookie_value(key)`	Returns the value of CKCookie object whose `key` is the the same as the argument.
`cookie_values(key)`	If the argument is nil(by default, argument is nil.), this method returns an array which has all the values of cookies. Otherwise, it returns an array which has the values of cookies specified by the argument.

Setting cookies to a response object

CKResponse has methods for setting cookies. These methods are defined in CKMessage, the superclass of CKResponse. Use add_cookie(cookie) and remove_cookie(cookie).

List 4-2: Adding a cookie

```
cookie = CKCookie( 'name' )
application.response.add_cookie( cookie )
```

Removing cookies from browser

Send cookies with the same name to browser. If you set past expiration time for the cookie when of that, browser removes the cookie completely.

List 4-3: Removing cookies from browser

```
cookie = CKCookie "name"
cookie.expire = Time.new - 60
response.add_cookie cookie
```

Removing cookies from a response object

CKResponse#remove_cookie removes cookies in CKResponse object.

List 4-4: Removing cookies from a response object

```
application.response.remove_cookie( 'name' )
```

Session Management

CKApplication and CKSession classes are for session management. CKSession objects have a hash of arbitary objects and information about browser name, IP address, etc. However, you can't set objects that can't be marshal(IO, Proc, etc.) to the session with default database manager CKSessionStore::FileStore.

Automatic session management

Sessions can also be managed automatically. CGIKit reads and saves sessions in automatic session management. If you use automatic session management, sessions are always created when accsessed CGIKit applications.

Set true for CKApplication#manage_session to use automatic session management (the default value is false).

Basic control

Getting sessions

Session objects are get with CKApplication#session. The method returns a new session objects if session don't exist.

List 5-1: Getting a session

```
session = application.session
p session #-> <CKSession:0x....>
```

Getting and setting session data

You can get and set session data with the same interface as hash.

List 5-2: Getting and settiong session data

```
session['key']      =   'value'
session['array']    =   [1,2,3,4,5]
p session['key']   #-> 'value'
p session['array'] #-> [1,2,3,4,5]
```

Saving sessions

Call `CKApplication#save_session` method to save sessions. However, it is unnecessary on automatic session management.

Closing or clearing sessions

Note the ways of deleting sessions are different in manual session management and automatic one. Call `CKApplication#clear_session` in manual, `CKSession#clear` in automatic.

`CKSession#clear` flags for deleting sessions. A session data is deleted if you call the method, however the session is not deleted. The session is deleted completely when saving sessions by automatic session management.

Saving session IDs

Session IDs are saved in URLs or cookies that the same expiration time with the session. You can set expires of the cookies for `session_cookie_expires`.

Set up methods of saving session IDs with the following attributes.

Table 5-1: Methods for saving session IDs (CKApplication class)

Attribute	Default	Description
`store_in_url`	`true`	Stores session IDs in URLs.
`store_in_cookie`	`true`	Stores session IDs in cookies.
`session_cookie_expires`	`604800` (a week)	Expiry date of cookie for session. If you set the value to nil, session cookies will be invalid when closing browser.

Authorization

Session expiration time

Session has expiration time. Exception SessionTimeoutError is raised when accessed with expired session.

Session expiration time is specified with seconds in &quot;timeout&quot; attribute of CKApplication. Timeout is time progressed with seconds than &quot;timeout&quot; after last accessed time for the session. You make sessions postpone indefinitely by setting 0 for &quot;timeout&quot; attribute. Sessions that session IDs don’t exist is also timeout.

Browsers and IP addresses

Sessions can authorize by browsers and IP addresses. Exception SessionAuthorizationError is raised when accessed with browser or IP address that are different from ones when a session created.

Set up methods of authorization with the following attributes. If the attributes is true, the mechanism is enabled.

Table 5-2: Methods for authorizing sessions (CKApplication class)

Attribute	Default	Description
`auth_by_user_agent`	`false`	Authorizes by browser.
`auth_by_remote_addr`	`false`	Authorizes by IP address.

Handling session errors

To process for handling session errors, override `CKApplication#handle_session_error` and return a component to display. The hook method is called when errors for timeout or authorization are raised.

List 5-3: Overriding CKApplication#handle_error

```
class CKApplication
  def handle_error( error )
    if error.class == CKSession::SessionTimeoutError then
      # ... code for timeout
    elsif error.class == CKSession::SessionAuthorizationError then
      # ... code for authorizaion error
    end

    error_page       = page @error_page
    error_page.error = error
    error_page.debug = @debug
    error_page
  end
end
```

Database manager

Database manager objects, such as CKSessionStore::FileStore, save sessions. The objects has these 3 methods, implement the methods if you develop or customize database manager class.

Table 5-3: Methods of database manager

Method	Description
`save`	Saves the session.
`clear`	Clear the session.
`restore`	Returns session restored from the saved.

Other notes

Permission error

Permission error can be raised when you create sessions. The reason is that you don’t have permission to write a session file or create a temporary directory on a directory for saving session. If the error is raised, change permission the directory.

Deleting session files

Session files for saving sessions that the same number of session IDs are generated. The session files are deleted when raising timeout or deleting sessions by calling clear methods, however you have to delete the files manually.

Deploying Applications

How to deploy CGIKit applications is same as generic CGI applications. An example in this case is Examples application attached CGIKit. Server configuration for the examples is the following.

Table 6-1: Server configuration

Preference	Description
Host name	`localhost`
Document root	`/var/www/htdocs`
CGI directory	`/var/www/cgi-bin`

Installing applications

Copy applications to CGI directory and change permission of startup scripts to executable.

List 6-1: Changing permission after copying Examples

```
[localhost:samples] user% cp -R Examples /var/www/cgi-bin
[localhost:samples] user% cd /var/www/cgi-bin/Examples
[localhost:/var/www/cgi-bin/Examples] user% chmod 755 Examples.cgi
```

URLs for the applications

URLs for installed applications are path of startup scripts. For example, a URL for Examples the above is `http://localhost/cgi-bin/Examples/Examples.cgi`.

Managing resources

Resources directories manage resource files like image files or preference files. The directories are resources directory and web server resources directory setted for `resources` and `web_server_resources` attributes of CKApplication.

Resources and web server resources

Resources are files not sent to browsers, web server resources are files to send to browsers. Locate web server resources directory in document root of web server.

Accessing to resources

Use CKResourceManage to access to resources. You can get CKResourceManager object with CKApplication#resource_manager.

Main methods of CKResourceManager are `url()` and `path()`. url() returns URLs for resources, path() does absolute file paths. url() works only for resources in web server resources directory. You can t get URLs for resources in resources directory.

Table 6-2: Methods of CKResourceManager

Methods	Description
`url(name)`	Returns the public URL for the specified resource when it is under the web server resources directory. Otherwise returns nil.
`path(name)`	Returns the file path of the specified resource.
`bytedata(name)`	Returns a CKByteData object for the specified resource.
`content_type(path)`	Finds the content type for extension of the specified path. If the path don't have extension, returns nil.

An example: displaying an image file

CKImage can display resource files as image. Do setting the following to display an image file (`cgikit.png`) in ImagePage of Examples.

1. move `resources` directory to path enables displaying images.
2. set path of the resource directory for `resource` attribute of CKAppication.
3. set the image file name for file attribute of CKImage.

List 6-2: move `resources` directory

```
[localhost:/var/www/cgi-bin/Examples] user% mv resources ../../
htdocs
```

List 6-3: set path of `resources` directory (`Example.cgi`)

```
app = CKApplication.new
app.web_server_resources = '../../htdocs/resources'
app.run
```

List 6-4: set an image file name of `file` attribute (`ImagePage.ckd`)

```
FileInResource : CKImage {
  alt  = "File in resource direcory";
  file = "cgikit.png";
}
```

This is an example to display static an image file. CKImage can display dinamically using with `data` attribute.

Debugging

Running on command-line

Applications run on offline-mode when you run them on command-line. Input form data for the applications in name=value format and push Ctrl-D to run.

List 6-5: Running on offline-mode

```
[localhost:/cgi-bin/Examples] user% ./Examples.cgi
(offline mode: enter name=value pairs on standard input)
# Ctrl-D
Content-Type: text/html

<html>
<head>
<title>Examples</title>
</head>
<frameset cols="200,*">
    <frame name="Index" src="?element_id=IndexPage">
    <frame name="Contents" src="?element_id=IntroductionPage">
    <noframes>
        <body>
            Use other browser.
        </body>
    </noframes>
</frameset>
```

Checking attributes of elements

If you set true to `check_attributes` attribute of CKApplication, CGIKit checks attributes of elements on runtime. It raises errors if nonexistent attributes are setted or required attributes aren t setted.

Logging

CKLog is a simple logging class with 5 debug levels, It writes log messages higher than setted level. The debug level is DEBUG < INFO < WARN < ERROR < FATAL .

Table 6-3: Logging methods

Method	Description
`debug(message)`	Write `message` on DEBUG level.
`info(message)`	Write `message` on INFO level.
`warn(message)`	Write `message` on WARN level.
`error(message)`	Write `message` on ERROR level.
`fatal(message)`	Write `message` on FATAL level.

Options

Logging options are the following. Use `log_options` attribute of CKApplication to initialize CKLog objects instead of setting each options to do directly.

Table 6-4: Options

Option	Description
`level`	Debug level.
`name`	Program name.
`out`	Output. By default is standard error.
`file`	File name to output logs. Set this or `out` option.
`max_file_size`	Max file size (this enables if you set file to output). If size of the file is over this size, `FileSizeError` is raised.

List 6-6: Setting logging options

```
options = {‘level’         => CKLog::DEBUG,
           ‘name’          => ‘CGIKit Application’,
           ‘file’          => ‘log.txt’,
           ‘max_file_size’ => 1000000}

app = CKApplication.new
app.log_options = options
app.run
```

List 6-7: Writing a log message

```
class MainPage < CKComponent
  def logging
    log = CKLog.new(application.log_options)
    log.debug ‘log message’
  end
end
```

Performance tuning applications

mod_ruby

mod_ruby is a module to embed Ruby interpreter in Apache web server. Some processes are needed to use CGIKit with it.

Using with mod_ruby is experimental.

Saving name space of components

If you don t save name space of components in mod_ruby, CGIKit applications can affect each other. Then, create subclasses of CKApplication to save the name space.

Create a new file except a startup script and define a subclass of CKApplication in thi file (loaded in the startup script). Change the subclass name along with the application.

List 6-8: Defining a subclass of CKApplication in "application.rb" file

```
class Application < CKApplication
end
```

Next, define each components inside the subclass.

List 6-9: Defining Application::MainPage component

```
class Application
    class MainPage < CKComponent
        ...
    end
end
```

You can use subclasses of CKApplication but for mod_ruby. If you collect methods related whole of an application into the subclass, you use effectively it because CKApplication objects are shared in each components.

Changing an adapter to CKAdapter::ModRuby

CGIKit communicates with browsers using **adapters**. By default, CGI and mod_ruby adapters are selected automatically. If you use customized adapter or adapters don t be selected, specify an adapter for `interface` attribute of CKApplication.

List 6-10: Changing an adapter to mod_ruby

```
#!/usr/local/bin/ruby

require 'cgikit'
require 'application'

app = Application.new
app.interface = CKAdapter::ModRuby
app.run
```

WEBrick

WEBrick is a toolkit to builled web sesrver. To work CGIKit application with WEBrick, create an instance of the application and mount it as servlet.

Handlers for CGIKit are 3 types.

List 6-11: Handlers for CGIKit

Handler	Description
`WEBrick::CGIKitServlet::PathHandler`	Handler that receives component path in the second argument.
`WEBrick::CGIKitServlet::HashHandler`	Handler that receives a hash for accessors of CKApplication in the second argument.
`WEBrick::CGIKitServlet::ApplicationHandler`	Handler that receives a CKApplication object in the second argument.

以下は `ApplicationHandler` を使った起動スクリプトです（付属サンプルの HelloWorld に添付してあります)。このスクリプトは、コンポーネントのパスとポート番号を指定して起動します。

A startup script using with `ApplicationHandler` is the following (attached in HelloWorld an example application). Specify component path and port number to run.

```
% webrick-app.rb '.' 8080
```

List 6-12: Using WEBrick with ApplicationHandler (webrick-app.rb)

```
# webrick-app.rb [component_path [port]]
require 'webrick'
require 'cgikit'

path = ARGV.shift || Dir.pwd
port = (ARGV.shift || 8080).to_i

app = CKApplication.new
app.component_path = path

server = WEBrick::HTTPServer.new({:Port => port})
server.mount('/', WEBrick::CGIKitServlet::ApplicationHandler, app)

trap("INT"){ server.shutdown }
server.start
```