

Bonsai Project Technical Documentation

Andrew Benedict <dev@andybenedict.com>

December 29, 2014

Contents

1	Database	7
1.1	Content Tables	7
1.1.1	Content Registry Table	7
1.1.2	Content Table	8
1.1.3	Locale Table	8
1.1.4	Content Type Table	9
1.1.5	Content Category Table	9
1.2	Node Tables	9
2	Primary Components	11
2.1	Bonsai Tree	11
2.2	Renderer	11
2.2.1	Pre-processors	11
2.2.2	Templates	13
2.3	Modules	13
2.3.1	Registry	13
2.3.2	Vocab	13
2.3.3	Callback	13
2.3.4	Tools	13
2.4	Content Mapper	13
2.4.1	Converters	13
2.5	Exceptions	13
2.5.1	Bonsai Strict Exception	13
3	Installation	15
3.1	Database Initialization Script	15
4	Configuration	17
5	Common Usage Examples	19
5.1	Dynamic Node Examples	19
5.2	Vocab Examples	19

Introduction

The Bonsai Project is an API for managing content, data structures, and their corresponding templates.

Aside from a small number of syntactic rules and interfaces required to facilitate it's basic operation, all Bonsai core components can be extended, overridden, or reconfigured to conform to the user's specific needs. The Bonsai development team strives to make as few assumptions as possible to maximize the control the user has over data structures, processors, and templates.

As a result of this control, the Bonsai project is not intended to be a content management system akin to the mainstream all-in-one solutions. Rather, it is intended to be used in custom development projects to provide an abstraction layer for managing templates and static content with built in support for mapping dynamic content into the generated output.

Features

- Simple tree structure
- Recursive creation and rendering
- Easy-to-customize templates
- Optional preprocessing of fields in templates
- Automated resolution of localized content
- Field mapping of dynamic content
- Dynamic fields have optional callback for conversion or formatting.

Chapter 1

Database

The Bonsai Project uses a handful of database tables to store information, they can be divided into two groups:

- Content
- Node

1.1 Content Tables

1.1.1 Content Registry Table

The primary table in the content part of the database is the content registry.

Figure 1.1: Content Registry Table

contentRegistry			
Name	DataType	Keys	Notes
id	Int	PK	
reference	Varchar	Unique	
contentTypeID	Int	Foreign: contentType.id	
dataFormat	Varchar		
contentCategoryID	Int	Foreign: contentCategory.id	
startDate	Timestamp		
endDate	Timestamp		
active	Boolean		

Two fields are present for searching purposes, contentType being the primary and contentCategory being the secondary. For example, Bonsai natively has two contentTypes, BonsaiNode and Vocab. BonsaiNodes are complex datatypes intended to be paired with a template in the render tree. On the other hand, are plain text and useful for things like labels, or mapping the same term into multiple locations. See the Vocab

Examples section of the Common Usage Examples chapter for further information.

There are two methods provided for limiting the display of content.

The startDate and endDate fields allow scheduling of content either as part of the node tree or using dynamic nodes. (See the Dynamic Node Examples in the Common Usage Examples chapter for further information.)

Setting active to false merely makes the content unavailable. This allows the data to be preserved for future use or alteration without making available to the content system.

1.1.2 Content Table

As the name implies, the content table stores the actual content associated with items listed in the contentRegistry.

Figure 1.2: Content Table

content			
Name	DataType	Keys	Notes
id	Int	PK	
contentRegistryID	Int	Foreign: contentRegistry.id Unique w/ localeID	
localeID	Int	Foreign: locale.id Unique w/ contentRegistryID	
content	CLOB		
			JSON

The content field generally contains a JSON object for most contentType, the format of which should conform to the dataFormat listed in the contentRegistry table. Vocab being an obvious exception.

The content table shares a many-to-one relationship with the contentRegistry table, allowing for one entry per locale. When a locale is set in the Bonsai Registry module, Bonsai will automatically return either the content for that locale, or content from the default locale if localized content is not specified.

1.1.3 Locale Table

The locale table stores the basic information about each locale you intend to use for your content. The first key (0) is installed by the Database Initialization Script and is 'no-ne' and is the catch-all default locale, this is used if you don't set a locale on your project. It is recommended that you use one of the standard locale formats for this code to promote easier use.

The title and sort fields are not specifically used in the Bonsai Core, though they are used by some plugins. They are also useful if you want to dynamically populate your locale switcher.

Figure 1.3: Locale Table

locale			
Name	DataType	Keys	Notes
id	Int	PK	
title	Varchar		
code	Varchar	Unique	
sort	Varchar		

1.1.4 Content Type Table

Content Types are used for broad typing, by default two content types are added by the Database Initialization Script:

1. BonsaiNode
2. Vocab

Content types allow easy sorting and handling of like data types. See the Dynamic Node Examples in the Common Usage Examples chapter for further information.

Figure 1.4: Content Type Table

contentType			
Name	DataType	Keys	Notes
id	Int	PK	
name	Varchar	Unique	

1.1.5 Content Category Table

Content Categories work much like Content Types, except that they have no bearing on any of the Bonsai Core behaviors. (It is possible that some plugins may utilize this functionality.) But their inclusion is mainly to allow additional subdivision to the user when needed.

Figure 1.5: Content Category Table

contentCategory			
Name	DataType	Keys	Notes
id	Int	PK	
name	Varchar	Unique	

1.2 Node Tables

The node tables organize the content into a structure that can be retrieved and manipulated.

Chapter 2

Primary Components

2.1 Bonsai Tree

The primary usage of the Bonsai Project is the Bonsai Tree. The tree is a very light-weight tool for compiling your templates and data into a rendered content tree.

The tree is made up of nodes and leafs, both of which are stored in the nodes table. The only difference between leafs and nodes in the structure is that instead of having children, a leaf references the ID of an item in the content registry.

When you instantiate a branch object, its constructor automatically traverses its children recursively and loads the entire tree. Likewise, when making a content call, it will also recursively traverse the tree and return a compiled DOM.

For more information consult the Tree Class Diagram on page 12.

2.2 Renderer

The renderer class handles the conversion of the user-defined data structures into HTML code by way of a template.

Under normal usage circumstances, the renderer won't normally be called by itself, instead it will be called automatically by either a branch or a leaf in the Bonsai Tree.

For a usage example see the Example Render Call on page 12.

The render can be extended into a plug-in's namespace to allow access to private templates in plug-in. To access those templates, pass the renderer into the Branch or Leaf on instantiation, if none is provided the core renderer will be instantiated when needed.

2.2.1 Pre-processors

Pre-processors are scripts than can be called on data to handle conversions of data specific to the templates.

An example of a preprocessor is the included AutoWrap class. When called it detects any lines in the copy that are not already wrapped in a tag and automatically wraps each in the tag specified in the arguments.

Figure 2.1: Tree Class Diagram

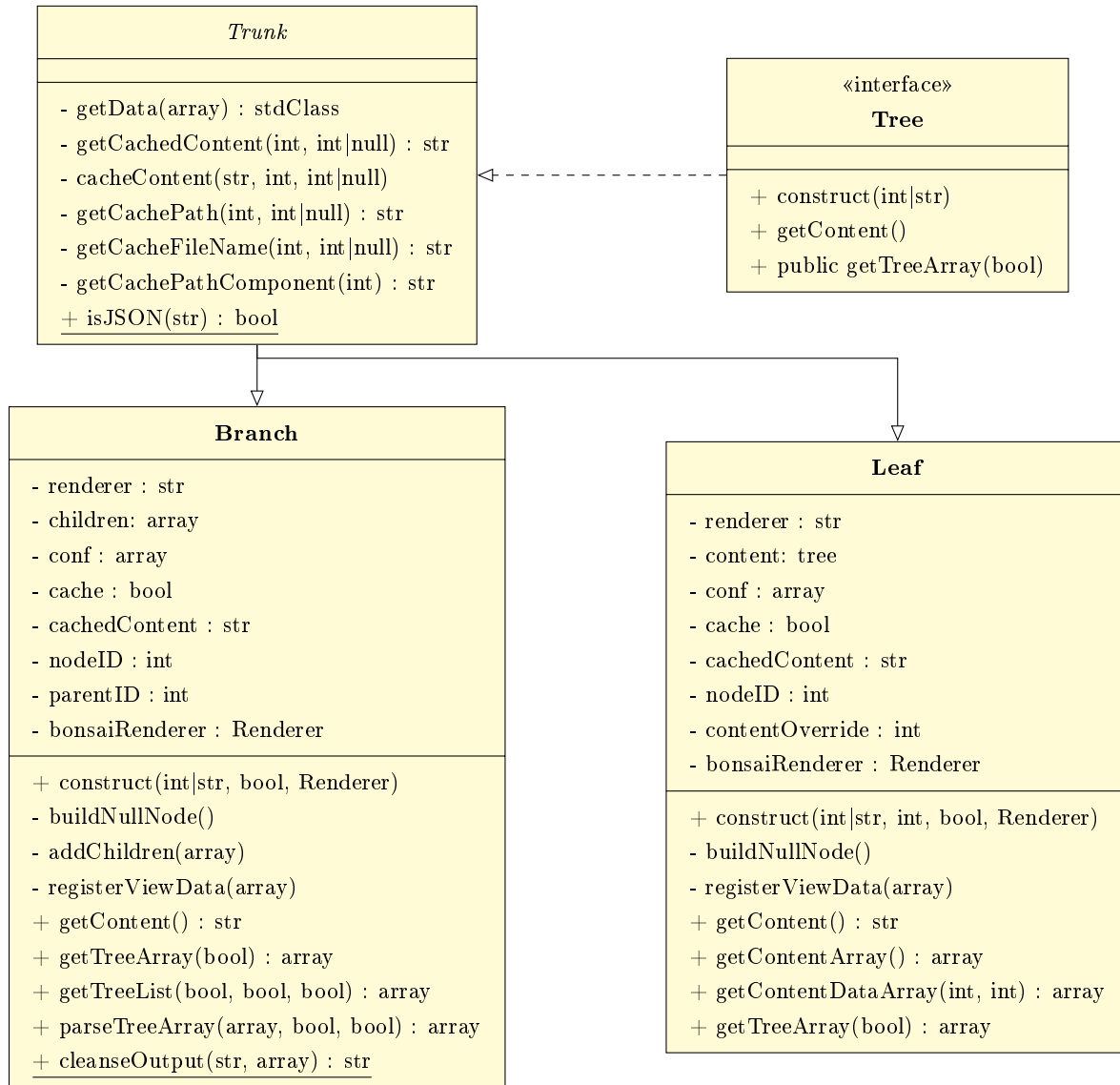


Figure 2.2: Example Render Call

```

$renderer = new \Bonsai\Render\Renderer(); //Instantiate the object
$dataObject = jsonDecode($dataString); //Decode the data json into a stdClass object
$contentObject = jsonDecode($contentString); //Decode the content json into a stdClass object
$template = 'templatename'; //Set the file name of the render template

//Call renderContent to build the output
$output = $renderer->renderContent($template, $contentObject, $dataObject);

```

Pre-processors are resolved in the following order:

1. Class found in the user-specified namespace
2. Class found in the plugins' PreProcess namespaces in the order they are listed in the config file
3. Class found in the Bonsai Core's PreProcess namespace

It will use the first valid preprocessor it discovers that implements the `\Bonsai\Renderer\PreProcess\PreProcess` interface.

By default, strict mode is turned on and if the pre-processor is not found or if it does not implement the correct interface it will throw a Bonsai Strict Exception. If strict mode has been turned off, it will continue checking if it encounters an invalid pre-processor and if no pre-processor is found, pre-processing will be skipped.

2.2.2 Templates

2.3 Modules

2.3.1 Registry

2.3.2 Vocab

2.3.3 Callback

2.3.4 Tools

2.4 Content Mapper

2.4.1 Converters

2.5 Exceptions

2.5.1 Bonsai Strict Exception

Chapter 3

Installation

3.1 Database Initialization Script

Chapter 4

Configuration

Chapter 5

Common Usage Examples

5.1 Dynamic Node Examples

5.2 Vocab Examples

Appendix