TIAMAT

OPEN SOURCE
ROLE-PLAYING GAME FRAMEWORK

SPECIFICATIONS

Table of Contents

Introduction

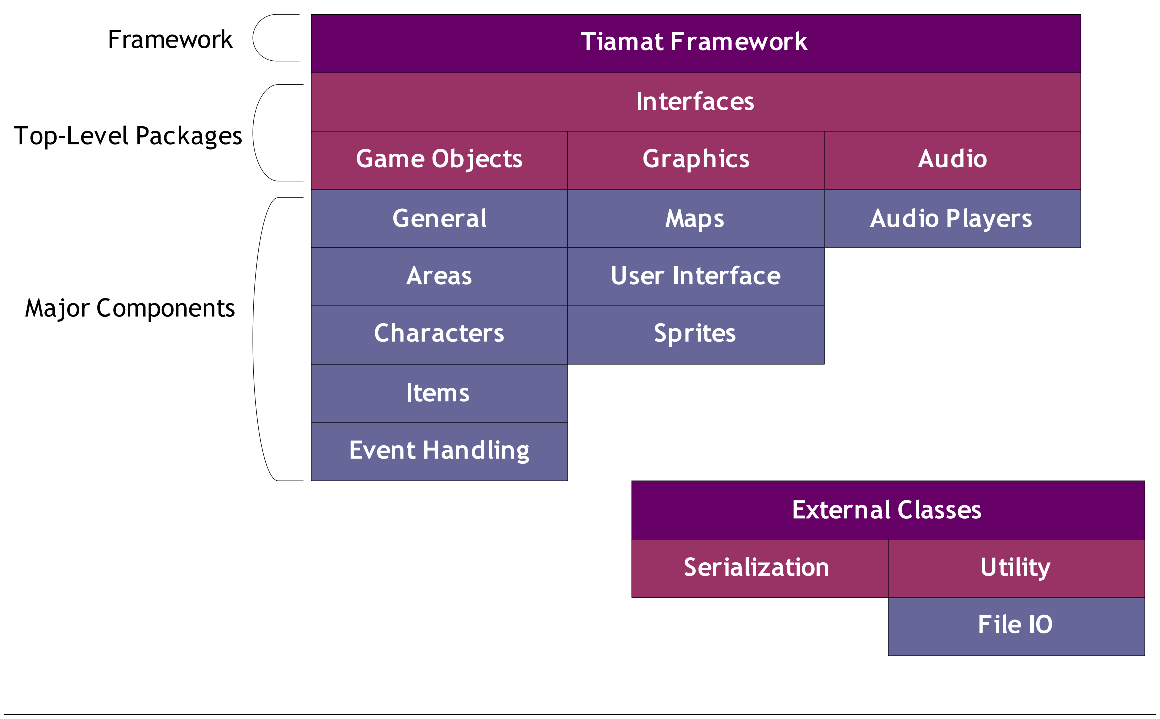
Overview

So what's the goal of this whole mess? Tiamat is designed to provide a framework for developing role-playing games. This is not the same thing as a "game engine", I don't know much about mechanics but I do know how to write libraries of highly reusable code. Tiamat is intended to provide all the basic elements of a role-playing game such as maps, characters, music, and to a limited degree graphics. I say "limited degree" for graphics because I'm not a particularly good graphics programmer and because graphic technologies change at a faster pace than I care to keep track of. The remaining elements of a role-playing game are relatively static. Tiamat should provide the building blocks of an RPG without forcing a developer to follow a specific set of rules.

So what are some of the design objectives of Tiamat? In no particular order.. All game objects (i.e. characters & maps) should be able to be stored in some platform neutral format. Basically, every game object should be able to be saved in a data file that has a clearly defined specification. Why? So that editors for these objects can easily be made and data is interchangeable between programs (even ones that don't use Tiamat).

What is the overall approach for designing the Tiamat Framework? Tiamat is being designed through an iterative process that resembles an agile or extreme programming methodology. The main principal is to break down the concept of an RPG framework into many small, manageable, and independent components.

The first step in this approach is to identify the major building-blocks an RPG framework should contain. These would be things like characters, maps, items, background music and sprites. The current version of Tiamat has identified and grouped these major components as diagrammed below:



So what games inspired this framework? This framework was designed to create games that resemble Phantasy Star I-IV. However, there are certain elements of the Phantasy Star series that I don't care for (such as random battles) that will not appear in Tiamat. Numerous other console role-playing/adventure games played a role in inspiring this framework. In no particular order they are: Lunar series, Chrono Cross, Legend of Heroes: Dragon Slayer, Ys I&II, Grandia II, Super Mario RPG, and Shadowrun.

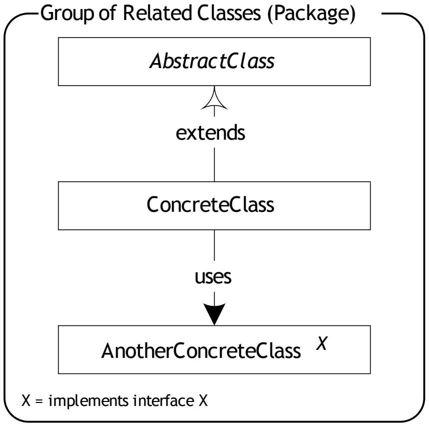
Framework Specifications Guide

Key

Don’t you hate when you’re reading a diagram and it doesn’t tell you what any of the symbols mean? I sure do. These specifications use a syntax that is remotely similar to UML. However, I don’t want to read the giant UML specifications so this won’t conform to them. Anyway, here’s what stuff means:

Class Hierarchy Diagrams

The purpose of this diagram is to show the relationship between classes. There’s no class design here, just a visual representation of how classes relate to each other.

 Group of Related Classes (Package) AbstractClass extends ConcreteClass uses AnotherConcreteClass X X = implements interface X	➢ Classes that are related are grouped in a box together. Classes may be tightly related if one inherits from or uses the other. They may be loosely related if they serve a common function (i.e. graphic classes would all be grouped together). ➢ Classes with names in italics are abstract. ➢ Open-headed arrows signify an inheritance relationship. ➢ Closed-headed arrows signify a usage or contains relationship. ➢ Arrows are captioned with the specific relationship. ➢ A letter in superscript means the class supports an interface. A key to the interfaces will appear on the bottom of a diagram.

Class Diagrams

Defines the specification of a class. Defines all properties, methods, and constructors for a class. Also contains some notes about implementation. It would be nice to port Tiamat to multiple platforms so there needs to be notes about implementation in the class definitions.

SomeClass **private properties** int whatever **protected properties** float[] arrayOfFloats	➢ The name of the class is at the top. If the name is in *italic* it is an abstract class. ➢ From here there are four groupings of class attributes: ➢ Properties: member variables, can be a simple data type or another class.

public constants int CONSTANT=0 **constructors** SomeClass(void) SomeClass(int someParameter) **public properties** string status readonly boolean isAlive **public methods** void doSomething(int parameter)	➢ Methods: invoke some function of the class. ➢ Constants: constant variables or enumerations. ➢ Constructor: create a new instance of the class, always public. ➢ The scope for these can be: ➢ Public: accessible to external classes, public in inherited classes. ➢ Private: not accessible to external classes, not accessible to inherited classes. ➢ Protected: not accessible to external classes, accessible & protected to inherited classes. ➢ Readonly: property that can only be read by an external class ➢ Data types: ➢ int: integer i.e. 4. ➢ float: decimal i.e. 3.14159. ➢ char: single character. ➢ string: array of characters. ➢ boolean: true or false. ➢ <class name>: another class. ➢ void: nothing. ➢ []: array

Sample Code

Code samples will be given for classes whenever possible. The purpose is to illustrate how a class would be used in an object-oriented programming language such as Java or C# .NET. Code samples will look remarkably like this:

```
//Hello World
public static void main(string[] arguments){
        WriteLine("Hello World!");
        Exit();
}
```

Packages

A *package* is a logical grouping of classes that share a common function. The Tiamat RPG Framework has three packages so far:

- Game Objects: This is the root package of the framework. It is meant to contain the logical elements required to construct an RPG. Classes at this root include things like characters, items, attributes, and areas.
- Audio: This package contains the classes required to play audio.
- Graphics: This package contains the classes needed to visually represent logical

entities such as maps and sprites.

There are also a series of external classes and packages defined that are used by members of the Tiamat framework. They are not part of the Tiamat framework because they serve functions for other applications in addition to Tiamat.

External Classes and Interfaces

Serialization Package

Serialization is a mechanism for saving an object's state to a file or other physical representation. Most programming languages support serialization in some form. The purpose of this package is to provide a standard method for serializing objects regardless of platform. The underlying implementations will depend heavily on the serialization support provided in the target languages.

Storable

Indicates an object can be serialized. All storable objects must implement the method onDeserialize(...). This method should be called by a formatter after it has loaded an object. This method allows the object to populate any transient members it has.

The absoluteDir parameter is used to tell the object what the current working directory is since any file paths it stores should be relative. Not all storable objects will necessarily need this information. Any storable object that contains paths to files will have a transient absoluteDir parameter. For example, a good way to store a path in an XML file would be:

```
<mapPath>.\maps\mymap.map.xml</mapPath>
```

and a bad way would be:

```
<mapPath>c:\some-dir-on-my-machine\maps\mymap.map.xml</mapPath>
```

In the first case, the full path to the XML file can be set at runtime by resolving the working directory and the relative path. In the second case, the path is hard-coded to a specific location which may not actually exist.

interface Storable

public methods

void onDeserialize(Formatter formatter,string absoluteDir) //event that should be fired after an object has been de-serialized, allows it to populate transient properties and de-serialize any storable members it contains

It would be desirable if all Storable objects had a platform independent schema associated with them.

Formatter

A Formatter is used to write a Storable object to a physical representation. After loading a storable object, the formatter should invoke the onDeserialize(...) method.

interface Formatter

public methods

void saveTo(Storable object,string outputPath) throws StorageException //save the object to the output path

Storable loadFrom(string sourcePath) throws StorageException //load an object from the specified source path

StorageException

Thrown if an error occurs while trying to save or load a serialized object.

StorageException

XmlFormatter

Stores and retrieves objects from an XML file.

XmlFormatter implements Formatter

Utility Classes

There are several classes used by the Tiamat framework that serve specific utility functions.

PathResolver

Utility class to resolve absolute and relative paths.

PathResolver

static public methods

string getRelativePath(string absolutePath1,string absolutePath2) //resolve the relative path from absolutePath1 to absolutePath2

string getAbsolutePath(string absolutePath,string relativePath) //resolve the absolute path from absolutePath to relativePath

ImageLoader

Utility class to load an image from a file into memory.

ImageLoader

static public methods

Image loadmage(string imagePath) //load the image stored at imagePath into memory

Common Interfaces

This section describes interfaces that can be implemented by any class within the Tiamat RPG Framework.

Drawable

Any object that can be drawn on a screen should implement the Drawable interface.

interface Drawable

public methods

void draw() //draw the entire object

void draw(int x,int y) //draw the entire object at location specified by (x,y)

void draw(int x1,int y1,int x2,int y2) //draw the object within the boundaries specified by (x1,y1) and (x2,y2)

Game Objects Package

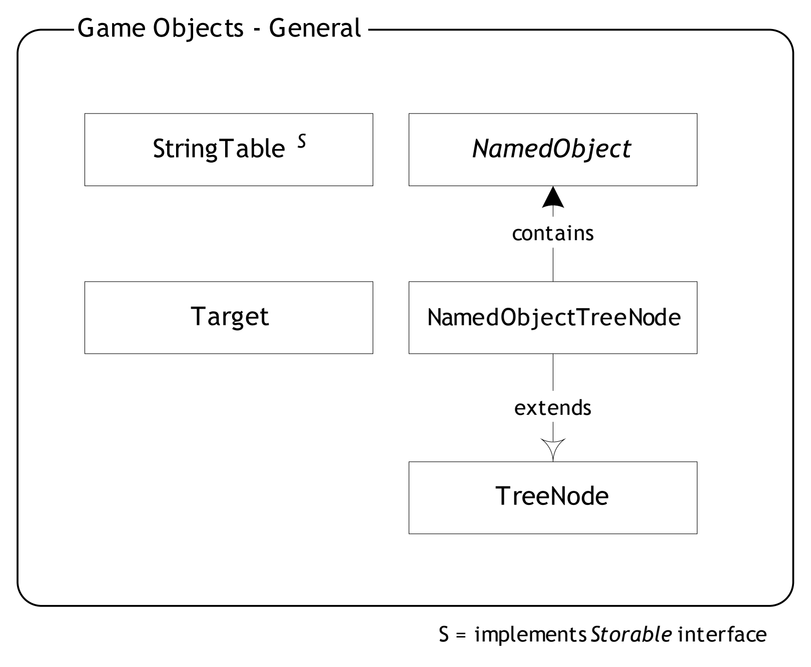
This is the root package for Tiamat. The package contains the core elements needed to build an RPG or adventure game. Although some objects may reference items in the Graphics package, everything in the root package should exist independent of a user interface. In theory, these objects should be usable in a text-only game.

Class Hierarchy

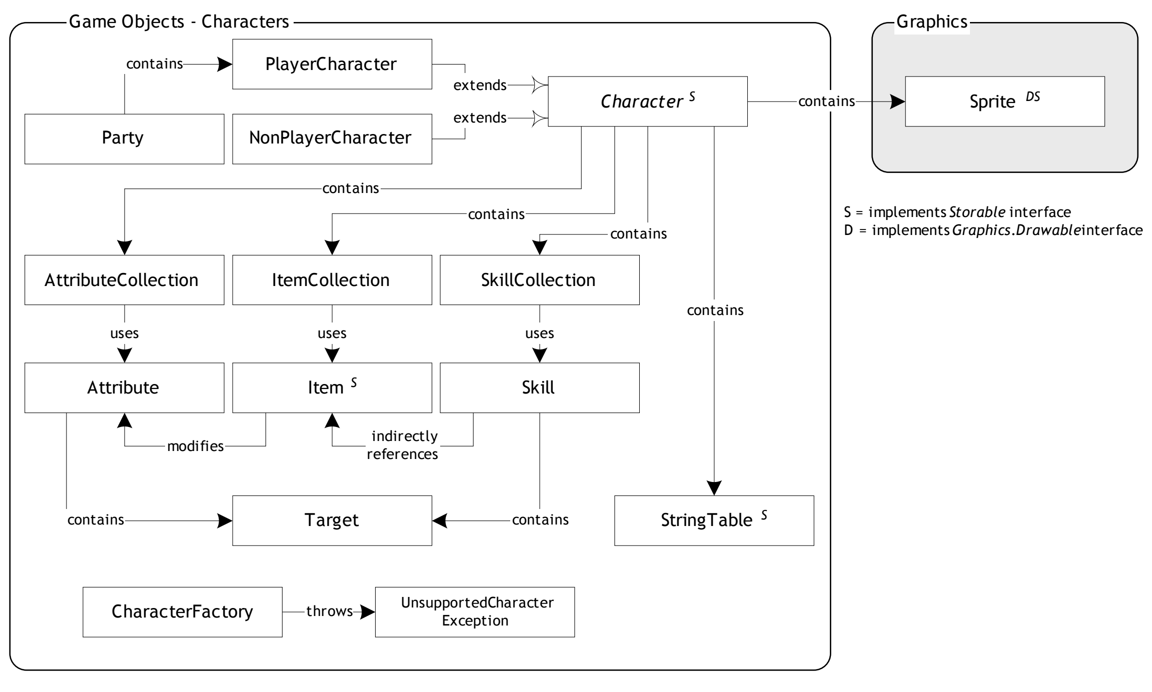
Classes within the root package can be logically grouped by function as follows:

Game Objects		
General	**Characters**	**Items**
StringTable ^{S}	*Character*	Item
TreeNode	NonPlayerCharacter	ItemCollection
NamedObject	PlayerCharacter	**Event Handling**
NamedObjectTreeNode	CharacterFactory	Event
Target	UnsupportedCharacter Exception	GameDataManager ^{S}
BaseGameState ^{S}	Party	GameScript ^{S}
BaseAdventureGame	Skill	ScriptTableKey
Areas	SkillCollection	ScriptTableValue
Area ^{S}	Attribute	
Location ^{S}	AttributeCollection	
LocationExit		
LocationObjectData		

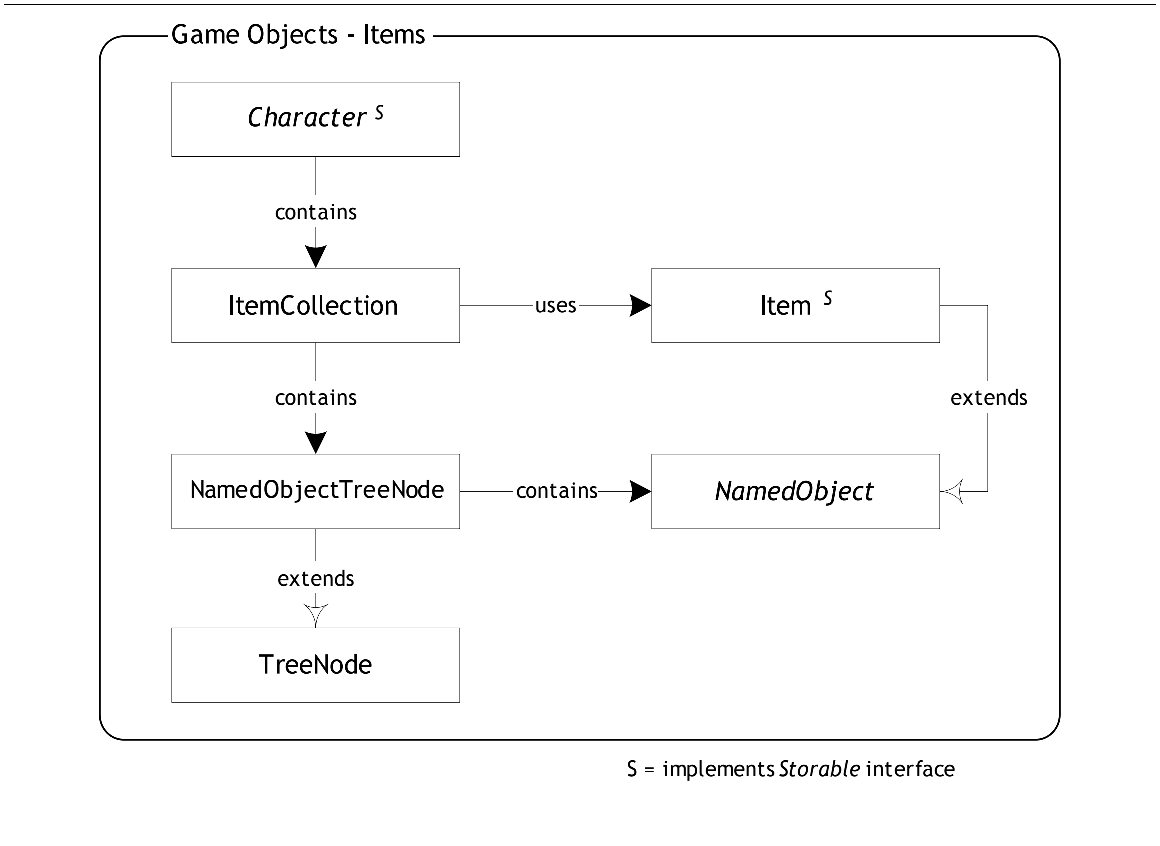
General classes are ones that support commonly used functionality across multiple classes.



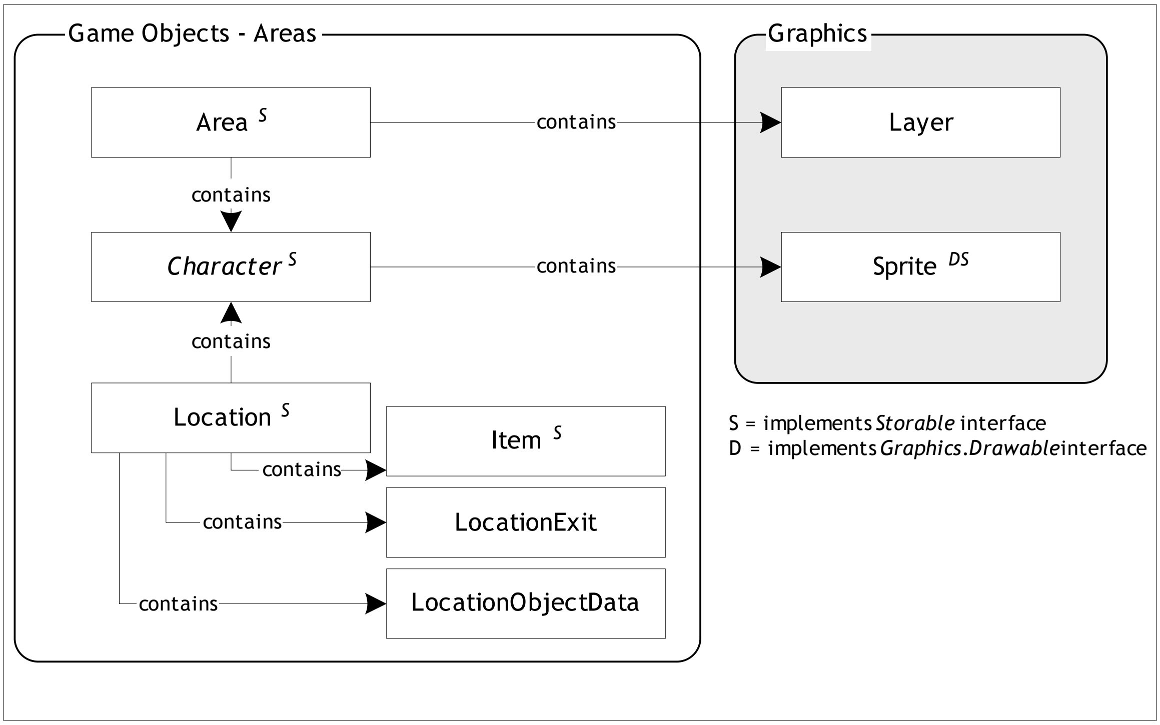
The Character and related classes are used to represent characters along with their skills and attributes.



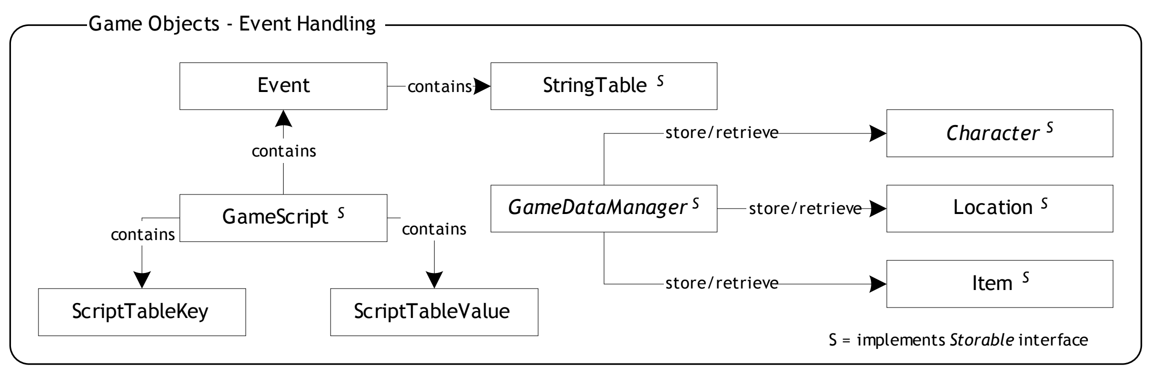
The Item and related classes are used to represent and store items.



Area classes deal with how a location is logically represented.

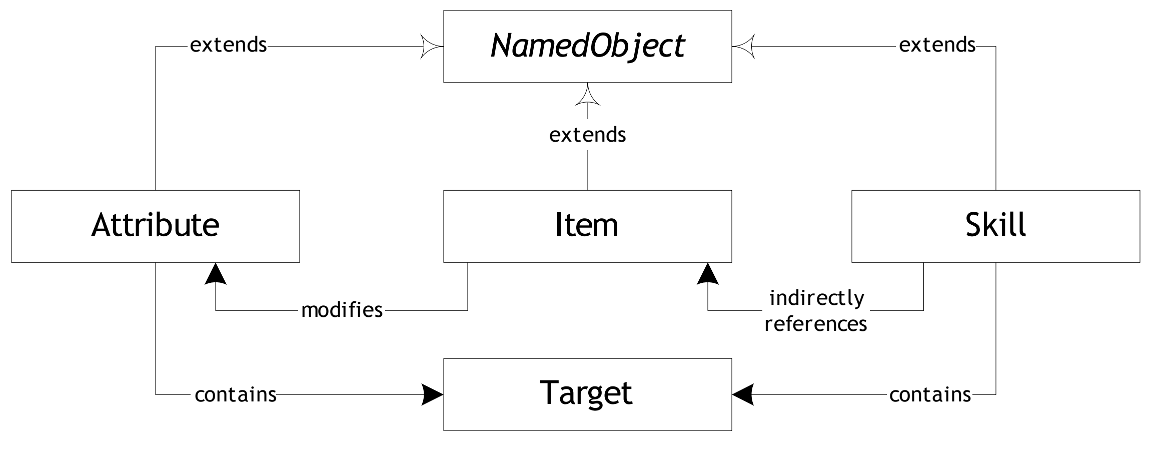


Event Handling classes support game events, data, and scripts.



NamedObject

So how is all this supposed to work? NamedObject is the base class for Attributes, Items, and Skills.



A NamedObject is simply something with a name and a description. Items, Skills, and Attributes all needed a name and a description. Having them inherit from a common base allows them to be stored in a common structure (NamedAttributeTreeNode). This all starts to come together with TreeNodes and collections which are a bit further down.

- An Attribute is used to store a value for a character. Examples include level, hit points, magic points, experience, strength, or poison resistance.
- An Item is something a character possesses. It item modifies an attribute, some can be equipped, some disappear after being used. Items include weapons, healing items, or even spells. The Target collection of the Item describes which

attributes are modified.

- A Skill represents a character's proficiency with a specific Item or class of Items. The Target collection of the Skill describes which Item the skill effects.

abstract NamedObject

constructors

NamedObject(string name,string description)

public properties

string name

string description

Item

Item extends NamedObject implements Storable

public constants

bool DEFAULT_EQUIPPED=false

bool DEFAULT_EQUIPABLE=false

bool DEFAULT_DISAPPEAR_ON_USE=false

public properties

string absoluteDir //full path to directory where images are stored, should be set at runtime

string fullImagePath //path to image for this item, changing this should re-load the actual image, relative to absoluteDir

string thumbailImagePath //path to thumbnail image for this item, what would appear in a menu or dialog, changing this should reload the actual image, relative to absoluteDir

readonly Image fullImage //full-size image for this item, change this by setting fullImagePath

readonly Image thumbailImage //thumbnail image for this item, what would appear in a menu or dialog, change this by setting thumbnailImagePath

Target[] targets //which attributes does this item modify

bool equipped //whether or not this item is currently equipped

bool equipable //whether or not this item can be equipped

bool disappearOnUse //whether or not this item disappears after being used

constructors

Item(string name,string description)

Item(string name,string description,Target[] targets)

Item(string name,string description,Target[] targets,bool equipable)

Item(string name,string description,Target[] targets,bool equipable,bool disappearOnUse)

Item(string name,string description,Target[] targets,bool equipped,bool equipable,bool disappearOnUse)

Item(string name,string description,string fullImagePath,string thumbnailImagePath,bool equipped,bool equipable,bool disappearOnUse)

Item(string name,string description,string fullImagePath,string thumbnailImagePath)

Item(string name,string description,string fullImagePath,string thumbnailImagePath,Target[] targets)

Item(string name,string description,string fullImagePath,string thumbnailImagePath,Target[] targets,bool equipable)

Item(string name,string description,string fullImagePath,string thumbnailImagePath,Target[] targets,bool equipable,bool disappearOnUse)

public methods

int getTargetCount(void) //how many targets for this item

void addTarget(Target target) //add a Target to the collection of Targets

private methods

Image loadImage(string imagePath) //convenience method to load images

Skill

Skill extends NamedObject

constructors

Skill(string name,string description) //blank skill with no targets

Skill(string name,string description,Target[] targets)

public properties

Target[] targets //what item is modified by this skill

public methods

int getTargetCount(void) //how many targets for this skill

void addTarget(Target target) //add a Target to the collection of Targets

Attribute

Attribute extends NamedObject

constructors

Attribute(string name,string description,long baseValue)

Attribute(string name,string description,long baseValue,long currentValue)

public properties

long baseValue //the base (initial) value for this attribute

long currentValue //the current value for this attribute

Target

Target

public constants

enum AffectedEntities{ //who this target effects

USER //the character that used the item

PARTY_SINGLE //a single member of the character’s party

PARTY_ALL //all members of the character’s party

ENEMY_SINGLE //a single enemy

ENEMY_ALL //all enemies

EVERYONE //everyone is effected }

AffectedEntities DEFAULT_AFFECTED_ENTITY=AffectedEntities.USER

constructors

Target(string targetPath,long value) //uses DEFAULT_AFFECTED_ENTITY

Target(string targetPath,long value,AffectedEntity affectedEntity)

public properties

string targetPath //describes the path of the item or attribute being modified

long value //how much to modify the target, used differently for items & skills

AffectedEntity affectedEntity //who this target effects

TreeNode

The purpose of TreeNodes are to store objects in a heirarchical structure. A TreeNode stores an object and a name for that object. Objects can be accessed through a fully qualified path. Setting the parent of TreeNode updates its path. The NamedObjectTreeNode extends the abstract TreeNode.

TreeNode

private members

string pathDelimiter //character used to separate names in path

public constants

string DEFAULT_PATH_DELIMITER="."

constructors

TreeNode(string name)

TreeNode(string name,string pathDelimiter)

TreeNode(string name,Object object)

TreeNode(string name,string pathDelimiter,Object object)

TreeNode(string name,Object object,TreeNode parent)

TreeNode(string name,string pathDelimiter,Object object,TreeNode parent)

public properties

TreeNode parent //each node can have at most one parent

TreeNode[] children //child nodes

Object object //the object this node is storing

string name //name of this node

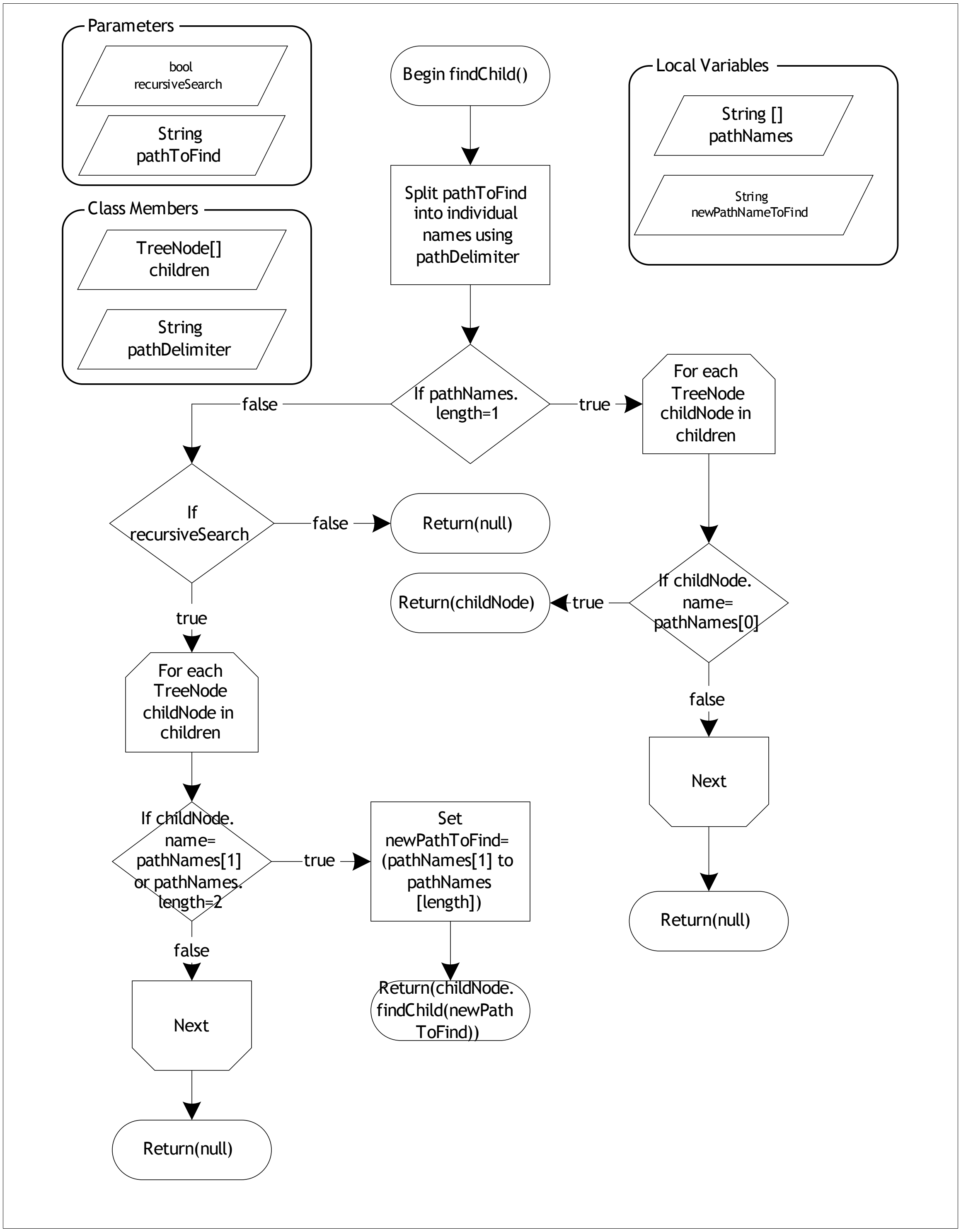
protected methods

void addChild(TreeNode childNode) //should be called by another node when it sets this node as the parent

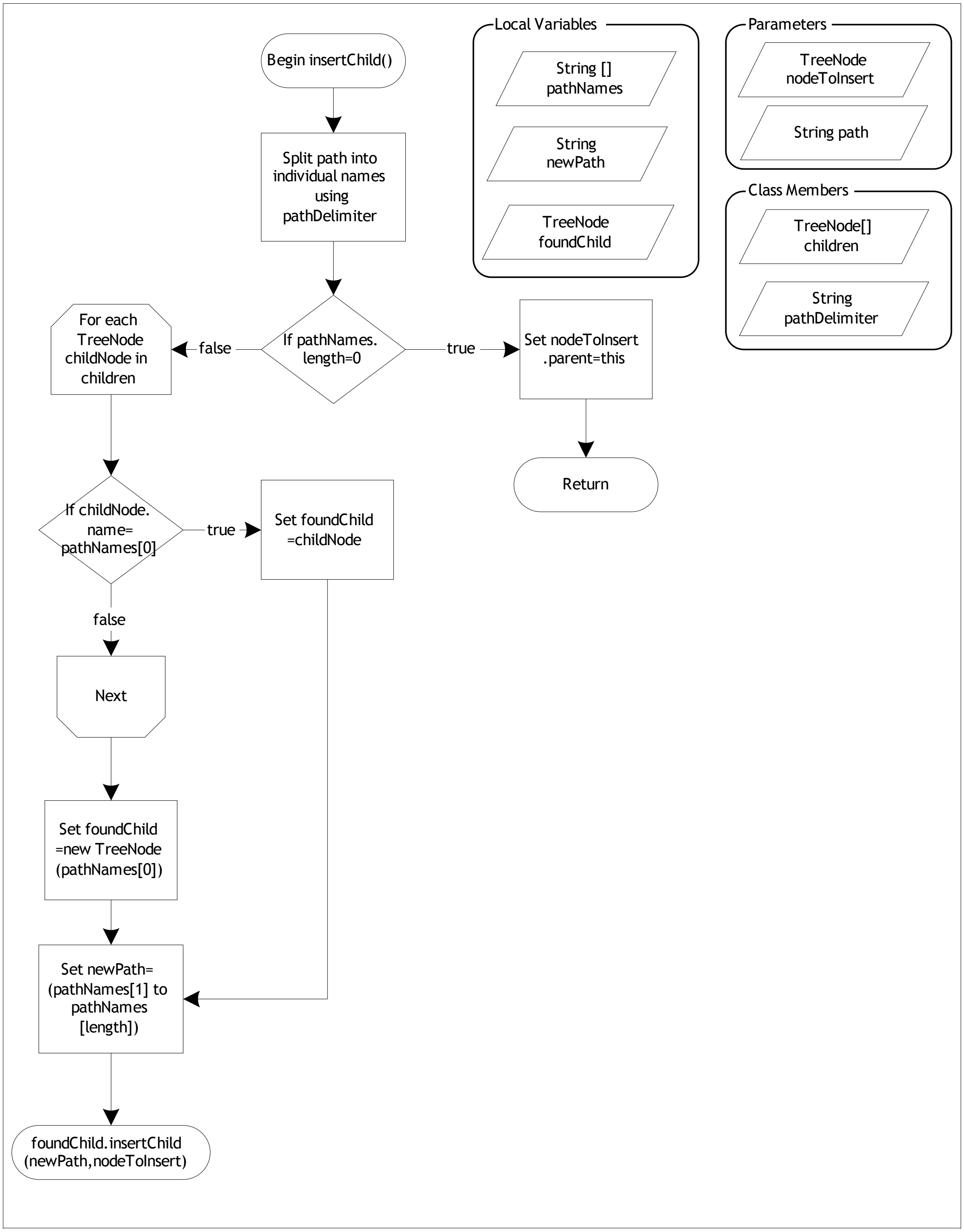
public methods

```
string getPath() //return the full path to this node, if this is not a root node, dynamically generate the path by recursively calling parent.getPath()

bool isRoot() //true if parent is null

bool isLeaf() //true if no children

bool removeChildNode(string path,bool recursive) //return success

TreeNode getChildNode(string path,bool recursive) //return the child node with that name if it exists, else null

int getChildNodeCount(bool recursive) //number of children for this node

int getChildLeafCount(bool recursive) //number of leaves for this node

TreeNode[] getChildLeaves(bool recursive) //get all leaves (terminal nodes) for this node

void insertChildNode(string path,TreeNode childNode) //recursively inserts the node
```

Logic to find a child node, used for getChildNode() and removeChildNode()


Parameters
bool
recursiveSearch
String
pathToFind
Class Members
TreeNode[]
children
String
pathDelimiter
Begin findChild()
Split pathToFind into individual names using pathDelimiter
Local Variables
String []
pathNames
String
newPathNameToFind
If pathNames. length=1
false
true
For each TreeNode childNode in children
If recursiveSearch
false
Return(null)
If childNode. name= pathNames[0]
true
Return(childNode)
true
For each TreeNode childNode in children
false
Next
If childNode. name= pathNames[1] or pathNames. length=2
true
Set newPathToFind= (pathNames[1] to pathNames [length])
Return(null)
false
Next
Return(childNode. findChild(newPath ToFind))
Return(null)

Logic to insert a child node, insertChildNode()


Begin insertChild()
Split path into individual names using pathDelimiter
If pathNames. length=0
true
Set nodeToInsert .parent=this
Return
false
For each TreeNode childNode in children
If childNode. name= pathNames[0]
true
Set foundChild =childNode
false
Next
Set foundChild =new TreeNode (pathNames[0])
Set newPath= (pathNames[1] to pathNames [length])
foundChild.insertChild (newPath,nodeToInsert)
Local Variables
String [] pathNames
String newPath
TreeNode foundChild
Parameters
TreeNode nodeToInsert
String path
Class Members
TreeNode[] children
String pathDelimiter

NamedObjectTreeNode

NamedObjectTreeNode extends TreeNode

constructors

NamedObjectTreeNode(NamedObject namedObject)

NamedObjectTreeNode(NamedObject namedObject,string pathDelimiter)

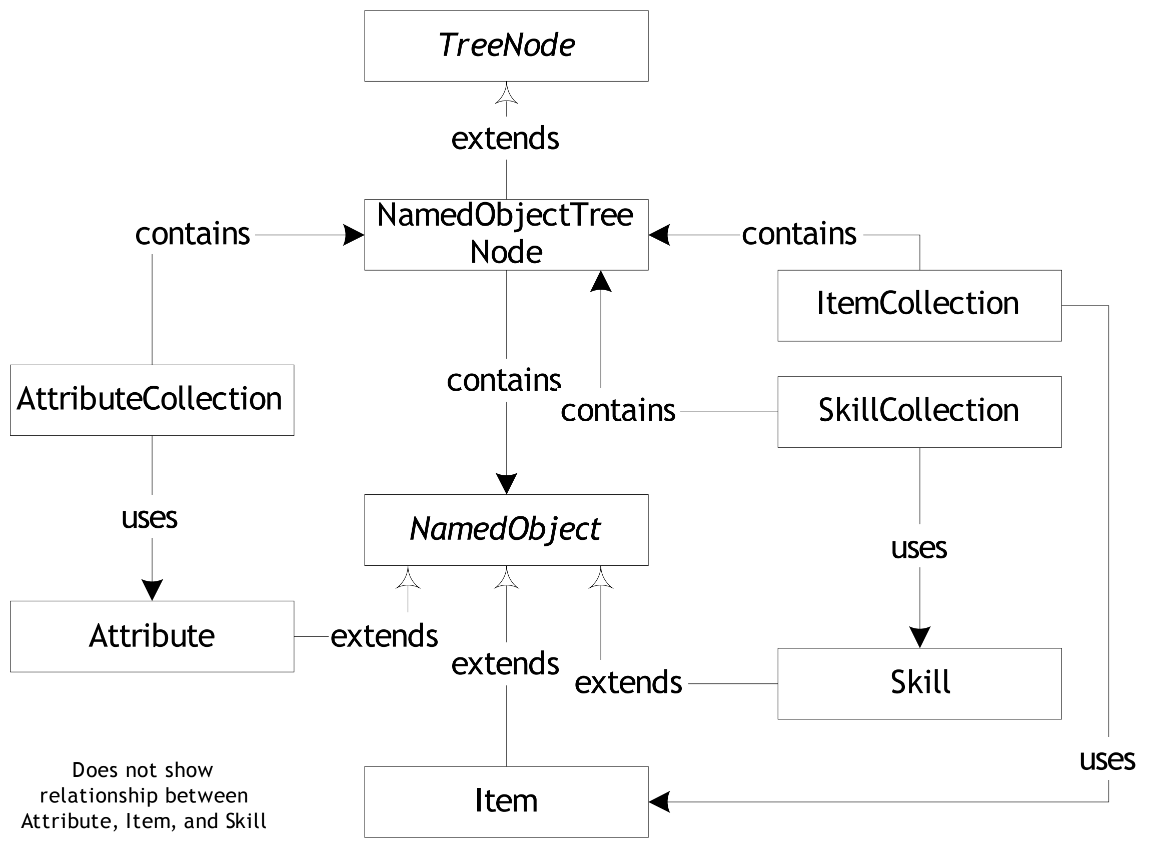
NamedObjectTreeNode(NamedObject namedObject,NamedObjectTreeNode parent)

NamedObjectTreeNode(NamedObject namedObject,string pathDelimiter,NamedObjectTreeNode parent)

protected properties

string name //override to return object.name

This next diagram is not as straightforward as I'd like, but I'll try to explain what's going on:



A NamedObjectTreeNode extends the abstract TreeNode. The only new functionality it offers is to override the name method to return the name of the NamedObject it's storing. The three collections (AttributeCollection, ItemCollection, and SkillCollection) each contain a NamedObjectTreeNode that serves as the root for the collection. Each collection offers additional convenience functions to better use the items they store. Most of the functionality the collections need is found in the abstract TreeNode object.

AttributeCollection

AttributeCollection
private members
NamedObjectTreeNode rootNode
private constants
string ROOT_NAME=”ATTRIBUTES”
string ROOT_DESCRIPTION=”Root node for AttributeCollection”
constructors
AttributeCollection()
AttributeCollection(string pathDelimiter)
AttributeCollection(Attribute[] attributes)
AttributeCollection(Attribute[] attributes,string pathDelimiter)
public methods
bool removeAttribute(string path) //return success
Attribute getAttribute(string path) //return the Attribute with that name if it exists, otherwise null
void setAttribute(string path,Attribute attribute) //sets the Attribute with the given path, adds it if it doesn’t exist
int getCount() //returns the total number of Attributes in the collection
Attribute[] getAll() //return all Attributes in the collection

ItemCollection

ItemCollection
private members
NamedObjectTreeNode rootNode
private constants
string ROOT_NAME=”ITEMS”
string ROOT_DESCRIPTION=”Root node for ItemCollection”
constructors
ItemCollection()
ItemCollection(string pathDelimiter)
ItemCollection(Item[] items)
ItemCollection(Item[] items,string pathDelimiter)
public methods
bool removeItem(string path) //return success
Item getItem(string path) //return the Item with that name if it exists, otherwise null

void setItem(string path,Item item) //sets the Item with the given path, adds it if it doesn't exist

int getCount() //returns the total number of Items in the collection

int getCount(string path) //returns the total number of Items starting from the path

ItemCollection getModifiersFor(string targetPath) //return an array of all equipped Items that have a target with a targetName equal to targetPath

int getNetModification(string targetPath) //use getModifiersFor() to get all the modifiers for a target, return the sum of their values

Item[] getAll()// return all Items in the collection

SkillCollection

SkillCollection

private members

NamedObjectTreeNode rootNode

private constants

string ROOT_NAME="SKILLS"

string ROOT_DESCRIPTION="Root node for SkillCollection"

constructors

SkillCollection()

SkillCollection(string pathDelimiter)

SkillCollection(Skill[] skills)

SkillCollection(Skill[] skills,string pathDelimiter)

public methods

bool removeSkill(string path) //return success

Skill getSkill(string path) //return the Skill with that name if it exists, otherwise null

void setSkill(string path,Skill skill) //sets the Skill with the given path, adds it if it doesn't exist

int getCount() //returns the total number of Skills in the collection

int getCount(string path) //returns the total number of Skills starting from the path

SkillCollection getModifiersFor(string targetPath) //return an array of all Skills that have a target with a targetPath equal to targetPath

int getNetModification(string targetPath) //use getModifiersFor() to get all the modifiers for a target, return the sum of their values

void adjustAllSkills(int percentage) //adjusts the values of all Skills in the collection by the given percentage, useful for updating all Skills uniformly for gaining/losing a level

void adjustSkills(string rootPath,int percentage) //adjusts the values of all Skills under the rootPath by the given percentage, useful for updating all Skills uniformly for gaining/losing a level

Skill[] getAll() //return all Skills in the collection

StringTable

A StringTable stores key-value pairs of strings. StringTable Implements *Storable* so it can be used to store simple configuration settings.

StringTable implements Storable

private members

HashTable table //underlying implementation

constructors

StringTable()

public methods

void set(string key,string value) //sets an entry in the table

string get(string key,string defaultValue) //returns the entry with the specified key, defaultValue if not found

string[2][] getAll() //return all key-value pairs in the collection, [0][n]=key for item n, [1][n]=value for item n

Character

abstract Character implements Storable

public properties

AttributeCollection attributes //attributes such as strength, hit points, or experience points

ItemCollection items //items the character is carrying

SkillCollection skills //skills the character possesses

StringTable profile //string values for a character such as names, class, or race

string spritePath //path to the public sprite's underlying file

readonly Sprite sprite //the physical sprite for the character, change by setting spritePath

string absoluteDir //full path to directory where images are stored, should be set at runtime

string fullImagePath //path to image for this character, relative to absoluteDir

string thumbailImagePath //path to thumbnail image for this character, what would appear in a menu or dialog, relative to absoluteDir

readonly Image fullImage //full-size image for this item, change this by setting fullImagePath

readonly Image thumbailImage //thumbnail image for this item, what would appear in a menu or dialog, change this by setting thumbnailImagePath

private methods

Image loadImage(string filePath) //convenience method to load images

constructors

Character() //creates an empty Character

NonPlayerCharacter

This represents a Character that the player does not have any control over. This would include townspeople or monsters.

NonPlayerCharacter extends Character

PlayerCharacter

This represents a Character that the player can control.

PlayerCharacter extends Character

CharacterFactory

CharacterFactory is an abstract static class that contains a single public method load(string path). This method is called with the path to a character file. The CharacterFactory creates an instance of the appropriate character, loads the character, and returns a reference to the loaded character. If no character can be found an UnsupportedCharacterException is thrown.

abstract CharacterFactory

public static methods

Character load(string path,Formatter formatter) //determines the correct Character for the specified file and loads it uses the specified Formatter, throws exception if a suitable Character can not be found

UnsupportedCharacterException

This should extend the exception class for the platform it's running on. No special behavior is needed.

UnsupportedCharacterException

Party

Simply put, a Party is a collection of PlayerCharacters. The first character in the Party (zero element) is considered the "main character". If this character is moved the others will follow. The order of the elements in the collection is therefore important and could have specific implications in a game.

Party

public properties

PlayerCharacter[] members //the members of this Party

public methods

int getMemberCount() //return the number of characters in this Party

PlayerCharacter getMember(int index) //get the character at index, null if index<0 || index>=count

void add(PlayerCharacter character) //adds a character to the end of the Party

void addAt(PlayerCharacter character,int index) //adds a character at the specified index and shifts all character past index down a position, adds to the end of the list if index>=count, adds to the start of the list if index<=0

Character remove(int index) //removes and returns the character at the specified index, moves members down as needed, returns null if index<0 || index>=count

void swapMembers(int index1,int index2) //swaps the players at the two indexes

constructors

Party() //creates an empty Party

Party(PlayerCharacter character) //creates a Party with a single member

Party(PlayerCharacter[] members) //creates a Party with the specified members

Area

Area implements Storable

public properties

Layer[] layers //the layers for this area

readonly Character[] characters //the characters in this area, change by setting character paths

string absoluteDir //full path to directory where characters are stored, should be set at runtime

string[] characterPaths //the paths to saved character files, relative to absoluteDir

public methods

Character getCharacter(int index) //get the character at the specified index

Layer getLayer(int index) //get the layer at the specified index

constructors

Area() //creates an empty Area

Area(Layer[] layers,String[] characterPaths) //create a layer with the given Layers and Characters

Event

An Event is something that occurs in the game, initiated by either the player or the game itself. The Event contains an action and a set of parameters for that action.

Event

public properties

string action //the action to perform

StringTable parameters //the parameters (arguments) for the event

constructors

Event(string action)

Event(string action,StringTable parameters)

Some example usages of Event are:

```
void menuMove.Click(){
        //move the main character
        Event e=new Event("move");
        e.getParameters().set("old-area",this.txtCurrentArea.getName());
```

```
                e.getParameters().set("new-area",this.txtNewArea.getName());
                this.processEvent(e);
        }
        [...]
        void receiveEvent(Event e){
                String actionText=new String(event.getAction());
                if(actionText.equals("look")){
                        String target=event.getParameters().get("target","null");
                        System.out.println("You are looking at a "+target);
                }
                [...]
        }
```

GameScript

The GameScript is used to store and process the Events that occur in a game.

GameScript implements Storable

public properties

readonly HashTable scriptTable //contains key-value pairings of game events

public methods

Event[] processEvent(Event event) //processes an event, returns any Events that are triggered by the initial Event

Event[] processEvent(Event event,string[] conditions) //processes an Event, returns any Events that are triggered by theinitial event

void add(Event event,string[] conditions,Event[] returnEvents) //add a new Event to the table

constructors

GameScript()

The internal scriptTable member is a HashTable that uses the ScriptTableKey and ScriptTableValue to store Events. An Event initiated by the player, or by the game itself, is used to construct the ScriptTableKey. A set of optional *conditions* can also be passed. Conditions can be used as a means of state management in a game if applicable. If the ScriptTableKey finds a matching ScriptTableValue then the Events stored in the ScriptTableValue are returned.

ScriptTableKey

The ScriptTableKey is used as the key value for the GameScript's internal script table.

ScriptTableKey

public properties

Event event //The Event for the key

string[] conditions //The conditions for the key

constructors

ScriptTableKey(Event event,string[] conditions)

ScriptTableValue

The ScriptTableValue is what is stored in the GameScript's internal script table. It simply contains a list of Events to process based on the original Event being processed.

ScriptTableValue
public properties
Event[] events //The Event stored in this value.
constructors
ScriptTableValue()
ScriptTableValue(Event[] events)

Location

The Location class is used to represent a place that can be visited. It contains collections of data used to track exits, items, and characters in the Location. Concrete Items and Characters are not stored in the Location, GameDataManager is used to create instances.

See *Framework Implementations: Adventure Style Game* for more information on the usage of this class and how GameDataManager interacts with it.

Location implements Storable
public properties
LocationExit[] exits //the exits from the Location
LocationObjectData[] characters //the Characters in the Location
LocationObjectData[] items //the Items in the location
string absoluteDir //full path to directory where characters are stored, should be set at runtime
string bgMusicPath //relative path to the background music
string bgImagePath //relative path to the background image
string overlayImagePath //relative path to the overlay image
public methods
void addCharacter(string name) //adds a Character with default settings
void addCharacter(LocationObjectData locationData) //adds a Character
void removeCharacter(string name) //removes the Character with the specified name
void addItem(string name) //adds an Item with default settings
void addItem(LocationObjectData locationData) //adds an Item
void removeItem(string name) //removes the Item with the specified name
constructors
Location()

LocationObjectData

This class is used to specify where an object resides in a Location. This class does not store any actual objects, just the name and coordinates of where one resides.

```
LocationObjectData
public constants
enum XAlignment{
  FIXED //fixed position
  CENTER //centered on x-axis
  LEFT //align on left side
  RIGHT //alight on right side
}
XAlignment DEFAULT_X_ALIGN=XAlignment.FIXED
enum YAlignment{
  FIXED //fixed position
  CENTER //centered on y-axis
  TOP //align on top
  BOTTOM //alight on bottom
}
YAlignment DEFAULT_Y_ALIGN=YAlignment.FIXED
boolean DEFAULT_VISIBLE=true
public properties
XAlignment xalign //x alignment of the object
YAlignment yalign //y alignment of the object
int x //actual x location of the object
int y //actual y location of the object
string name //the name of the object
boolean visible //whether the object is visible
constructors
LocationObjectData(string name)
LocationObjectData(string name,boolean visible)
LocationObjectData(string name,XAlignment xalign,YAlignment yalign)
LocationObjectData(string name,XAlignment xalign,YAlignment yalign,int x,int y)
LocationObjectData(string name,boolean visible,XAlignment xalign,YAlignment yalign)
LocationObjectData(string name,boolean visible,XAlignment xalign,YAlignment yalign,int x,int y )
```

LocationExit

This class is used to define an exit from a location.

LocationExit
public properties
String direction //the direction of the exit
String name //the name of the location the exit leads to
constructors
LocationExit(string direction,string name)

GameDataManager

GameDataManager stores paths to all data used by a game. Objects are dynamically loaded and saved as needed.

See *Framework Implementations: Adventure Style Game* for more information on the usage of this class it.

GameDataManager implements Storable
private properties
StringTable characterPaths //table containing names and paths to saved character files, relative to absoluteDir
StringTable itemPaths //table containing names and paths to saved item files, relative to absoluteDir
StringTable locationPaths //table containing names and paths to saved location files, relative to absoluteDir
StringTable sequencePaths //table containing names and paths to saved sequence files, relative to absoluteDir
public properties
string absoluteDir //full path to directory where objects are stored, should be set at runtime
public methods
void addCharacter(string name,string absoluteCharacterPath) //add a Character to the game
Character getCharacter(string name,Formatter formatter) //retrieve a Character by name
void saveCharacter(string name,Character character,Formatter formatter) //save a Character
void addItem(string name,string absoluteItemPath) //add an Item to the game
Item getItem(string name,Formatter formatter) //retrieve an Item by name
void saveItem(string name,Item item,Formatter formatter) //save an Item
void addLocation(string name,string absoluteLocationPath) //add a Location to the game
Location getLocation(string name,Formatter formatter) //retrieve a Location by name
void saveLocation(string name,Location location,Formatter formatter) //save a Location
void addSequence(string name,string absoluteLocationPath) //add a Sequence to the game
Sequence getSequence(string name,Formatter formatter) //retrieve a Sequence by name
void saveSequence(string name,Sequence location,Formatter formatter) //save a Sequence
constructors

GameDataManager() //default constructor

BaseGameState

BaseGameState is an abstract class that contains the minimal members and functions needed to track a game state. It conditions a string array to store *conditions,* or events that have happened in the game.

See *Framework Implementations: Adventure Style Game* for more information on the usage of this class it.

abstract BaseGameState implements Storable

private properties

string[] conditions //used to represent events that have occurred in the game

public properties

string absoluteDir

public methods

string[] getCondition() //returns all conditions

void addCondition(string condition) //adds a new condition

boolean hasCondition(string condition) //tests if the game state contains a specific condition

constructors

BaseGameState() //default constructor

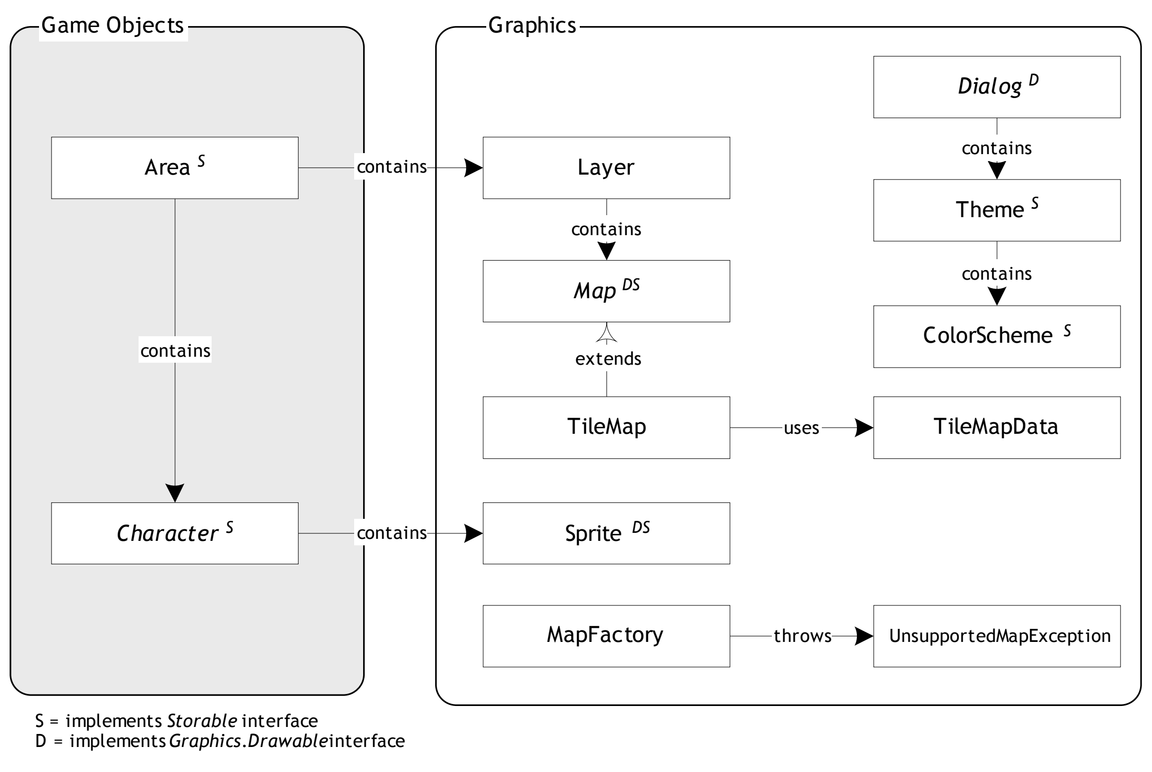
BaseGameState(string[] conditions)

Graphics Package

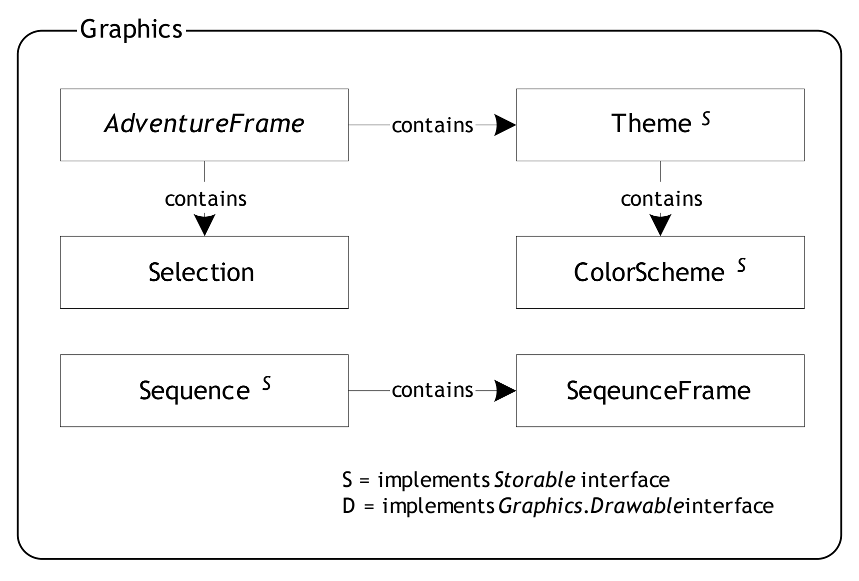
The graphics package contains the classes needed to visually represent objects. Although the classes in this package were designed with RPGs in mind, they could be used for many other types of games or applications.

Class Hierarchy

The classes in the graphics package can be logically separated into two groups. The first being items that are drawn onto a canvas such as a Map or Sprite:



The second category would be items rendered as a frame or other standalone user interface:



The Dialog class is grouped with the first set because it is drawn on a canvas and not displayed as an independent frame.

Sprite

Sprite implements Storable, Drawable

public constants

enum directions{ UP, DOWN, LEFT, RIGHT }

int MAX_DIRECTION //top index of directions

bool DEFAULT_MOVABLE=true

constructors

Sprite(int direction,int x,int y,string baseImagePath,string [][] imagePaths,Canvas canvas) //uses importImageData

public properties

int direction //must be one of directions defined above

int stepIndex //which frame of animation the sprite is in

int x //x-coordinate of sprite

int y //y-coordinate of sprite

Image[][] images //# of directions x # of steps in each movement animation, [n][0]=still image for direction n

string absoluteDir //full path to directory where images are stored, should be set at runtime

readonly string[][] imagePaths //location of images, relative to baseImagePath, needed to save & load Sprite, relative to absoluteDir

Sprite follower //reference to another sprite that follows this sprite

bool movable //whether or not this sprite can be moved, true by default

Canvas canvas //where to draw this sprite boolean isMoving //whether this Sprite is currently moving

public methods

boolean save(string filePath) //save this sprite to a file, return success

boolean load(string filePath) //load this sprite from a file, return success

boolean importImagedata(string baseImagePath,string[][] imagePaths) //import image data, return success

void move(directions direction) //moves the sprite in the specified direction

Image getCurrentImage(void) //get the image representing the sprite in its current direction and stepIndex

private methods

boolean loadImages() //loads images (filePaths[][]) into memory (images[][]), return success

Layer

Layer

public constants

enum LayerTypes{

MAP //a layer with a map associated to it

SPRITE //a layer where sprites move

DIALOG //a layer where dialogs are displayed }

LayerTypes DEFAULT_LAYER_TYPE=LayerTypes.SPRITE

public properties

LayerType type

readonly Map map //map for this layer, should be null unless type==MAP, change by setting the map path

string absoluteDir //full path to directory where the map is stored, should be set at runtime

string mapPath //path to the file where the map is saved, relative to absoluteDir

constructors

Layer() //create a default layer

Layer(LayerTypes type) //create a layer of the specified type

Layer(string mapPath) //create a map layer with the map saved at mapPath, use MapFactory to load correct map type

Map

abstract Map implements Storable, Drawable

public properties

int height

int width

string name

Canvas canvas //where to draw stuff

MapFactory

MapFactory is an abstract static class that contains a single public method load(string path). This method is called with the path to a map file. The MapFactory creates an instance of the appropriate map, loads the map, and returns a reference to the loaded map. If no Map can be found an UnsupportedMapException is thrown.

abstract MapFactory
public static methods
Map load(string path) //determines the correct Map for the specified file and loads it, throws exception if a suitable Map can not be found

UnsupportedMapException

This should extend the exception class for the platform it's running on. No special behavior is needed.

UnsupportedMapException

TileMap

TileMap extends Map
constructors
TileMap(int width,int height,Canvas canvas) //assumes import methods will be used
TileMap(string filePath,Canvas canvas) //calls loadMap
private properties
string[] tilePaths //needed to save map
public properties
Image[] tileSet
TileMapData[][] mapData
string absoluteDir //full path to directory where images are stored, should be set at runtime
string[] tilePaths//paths to where images are stored, relative to absoluteDir
public methods
boolean importTileSet(string baseImagePath,string[] tilePaths)
boolean importTileData(int[][] tileData)
boolean importTypeData(int[][] typeData
boolean importObjectData(Object[][] objectData)
void setObjectAt(int x,int y,Object object) //set the object at (x,y)
Object getObjectAt(int x,int y) //get the object at (x,y)
private methods
boolean loadImages() //loads images (tilePaths[]) into memory (tileSet[]), return success

TileMapData

TileMapData

public constants

```
enum TileTypes{
  NORMAL //tile you walk over with no effect on anything
  SOLID //tile that can not be walked over
  EXIT //exits the map
  EVENT //has some event associated to it }
TileTypes DEFAULT_TYPE=TileTypes.NORMAL
```

constructors

```
TileMapData(int tile) //create TileMapData with default settings
TileMapData(int tile,TileTypes type) //create TileMapData with default settings
TileMapData(int tile,TileTypes type,Object object)
```

public properties

```
int tile //index of tile in array or list of tiles
TileTypes type //one of TileTypes
Object object //an object that is standing on this tile
```

ColorScheme

ColorScheme implements Storable

public constants

```
Color DEFAULT_FORECOLOR=#666699
Color DEFAULT_BACKCOLOR=#c6c6c6
Color DEFAULT_BORDERCOLOR=DEFAULT_FORECOLOR
Color DEFAULT_BORDERINSETCOLOR=DEFAULT_BACKCOLOR
```

public properties

```
Color foreColor //the foreground color
Color backColor //the background color
Color borderColor //the color of the border
Color borderInsetColor //the color of the border inset
```

constructors

```
ColorScheme() //use default settings for everything
ColorScheme(Color foreColor,Color backColor) //borderColor=foreColor, borderInsetColor=backColor
ColorScheme(Color foreColor,Color backColor,Color borderColor,Color borderInsetColor)
```

Theme

A Theme is used to define the style of a dialog or other top-level UI container.

It contains a ColorScheme to define the colors and attributes to define how specific UI elements should look.

Theme implements Storable

public properties

ColorScheme colorScheme //the color scheme for this theme

string absoluteDir //full path to directory where images are stored, should be set at runtime

readonly Image radioButtonImage //image to display on a radio button when it's not selected

string radioButtonImagePath //path to the file containing the image, relative to absoluteDir

readonly Image radioButtonSelectedImage //image to display on a radio button when it's selected

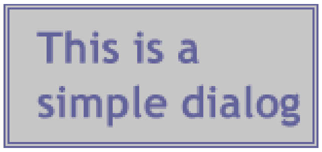
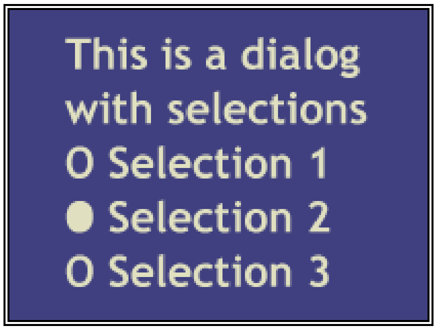
string radioButtonSelectedImagePath //path to the file containing the image, relative to absoluteDir

constructors

Theme() //create an empty theme

Dialog

A Dialog is used to display a message or present a list of options. A Dialog should be modal in that the game should effectively be paused while one is displayed.

 This is a simple dialog	 This is a dialog with selections ○ Selection 1 ● Selection 2 ○ Selection 3
This is an example of a Dialog in the default style.	This is a Dialog that presents the user three selections.

Dialog implements Drawable

private members

int textIndex //index of the text to display

int x,y,x1,y1 //used to save the position of the dialog

public properties

string[] text //the text to display, each array element represents one "screen" of text, using forward() and back() methods scroll through the text

string[] selections //if this dialog allows a user to select an option after all the text has been displayed the selections are stored here

Theme style //the style for this dialog

Canvas canvas //where to draw stuff

constructors

Dialog(string[] text,Canvas canvas,Theme style)

Dialog(string[] text,int x,int y,int x1,int y1,Canvas canvas,Theme style)

Dialog(string[] text,string[] selections,Canvas canvas,Theme style)

Dialog(string[] text,string[] selections,int x,int y,int x1,int y1,Canvas canvas,Theme style)

public methods

bool forward() //scrolls the text ahead one "screen", also causes redraw, returns false if on the last page of text

bool back() //scrolls the text behind one "screen", also causes redraw, returns false if on the first page of text

void hide() //hides the dialog

void onCancel() //called when the dialog is closed

void onSelection(int index) //invoked when the player has confirmed a selection in the dialog, index is the index of the selection made (relative to the selections array)

Selection

A Selection is used to represent a menu option. It contains a caption and an associated Event.

Selection

public properties

readonly string caption //the caption for the Selection

readonly Event event //the Event associated with the Selection

constructors

public Selection(string caption,Event event)

An example usage of Selection is:

```
private void loadDefaultMenu(){
        Selection[] selections=new Selection[8];
        Event event=new Event("look");
        selections[0]=new Selection("Look",event);
        event=new Event("move");
        selections[1]=new Selection("Move",event);
        [...]
        event=new Event("help");
        selections[7]=new Selection("Help",event);
        this.frame.setSelections(selections);
}
```

Sequence

A Sequence represents a non-interactive event. A dialog, game introduction, or ending credits would all be uses of a Sequence. It contains a collection of SequenceFrames and Events to execute upon completion of the Sequence.

Sequence implements Storable

public properties

SequenceFrame[] frames //the underlying frames

Event[] exitEvents //the Events to execute after the Sequence is complete

constructors

Sequence(SequenceFrame[] frames,Event[] exitEvents)

public methods

int getFrameCount //return the number of frames

void addFrame(SequenceFrame frame) //adds a new frame to the end of the Sequence

void removeFrame(int index) //removes the frame at the specified index

SequenceFrame getFrame(int index) //returns the frame at the specified index

int getExitEventCount //return the number of exit Events

void addExitEvent(Event event) //adds a new exit Event to the end of the Sequence

void removeExitEvent(int index) //removes the exit Event at the specified index

Event getExitEvent(int index) //returns the exit Event at the specified index

SequenceFrame

A SequenceFrame represents a single frame in a Sequence.

SequenceFrame

public properties

string text //the text to draw

string absoluteDir //full path to directory where images and music are stored, should be set at runtime

string bgMusicPath //the path to the background music to play, relative to absoluteDir

string bgImagePath //the path to the image to draw, relative to absoluteDir

readonly image bgImage //the image to draw

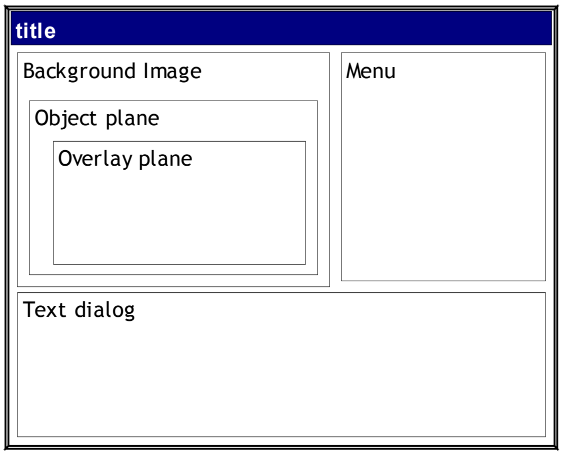
constructors

SequenceFrame()

SequenceFrame(string Text,string absoluteDir,string bgMusicPath,string bgImagePath)

AdventureFrame

An AdventureFrame provides the basic user interface for an adventure style dialog. The components of the frame are:



See *Framework Implementations: Adventure Style Game* for more information on the usage of this class.

abstract AdventureFrame

public properties

string title //the title of the frame

string bgImagePath //the full path to the background image

ImageData[] objects //objects to be drawn in the object plane

ImageData[] overlays //objects to be drawn in the overlay plane

Theme theme //the Theme to apply to the frame

Font font //the Font to use for text

Selection selections //the selections that appear in the menu

string text //the text drawn in the dialog region

public methods

abstract void postEvent(Event event) //sends an Event back to the implementing class

abstract void onFrameClose() //notifies the implementing class that the frame has been closed

public void setBgImagePath(string bgImagePath,boolean redraw) //sets background image, default set method calls this method passing true for redraw parameter

public void setObjects(ImageData[] objects,boolean redraw) //sets object images, default set method calls this method passing true for redraw parameter

public void setOverlays(ImageData[] overlays,boolean redraw) //sets overlay images, default set method calls this method passing true for redraw parameter

public setVisible(boolean visible) //show or hide the frame

public void redrawImages() //force redraw of images

constructors

AdventureFrame(String title,Theme theme,Font font)

ImageData

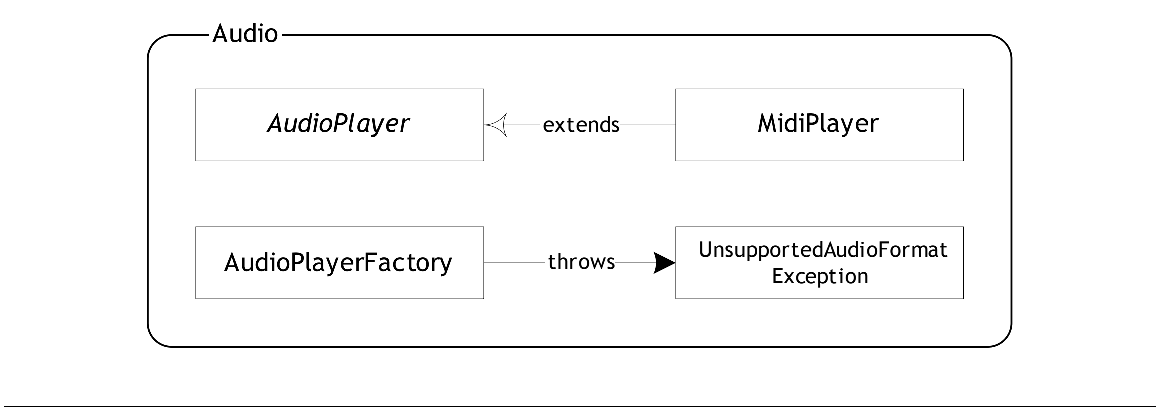
ImageData is used by AdventureFrame to specify the location of an image on a particular plane.

ImageData
public properties
int x //x location of the image
int y //y location of the image
string imagePath //full path to the image file
constructors
ImageData(string imagePath,int x,int y)

Audio Package

The audio package contains the basic classes needed to play audio. The purpose is to simplify development by providing components that hide the underlying audio implementation.

Class Hierarchy



AudioPlayer

abstract AudioPlayer

protected properties

int loop //number of times to loop

string[] trackList //list of tracks to play

protected int trackIndex //index of the track currently being played

public constants

int INFINITE_LOOP=-1

public methods

void play(string path) //no loop

void play(string path,int loop) // <0=infinite loop

void play(string[] paths) //no loop

void play(string[] paths,int loop) //loop back to 1st track after playing all

void stop() //stop playing

MidiPlayer

MidiPlayer extends AudioPlayer

AudioPlayerFactory

AudioPlayerFactory is an abstract static class that contains a single public method getAudioPlayerFor(string path). This method is called with the path to an audio file. The AudioPlayerFactory returns the correct AudioPlayer for the given file. If no AudioPlayer can be found an UnsupportedAudioFormatException is thrown.

abstract AudioPlayerFactory
public static methods
AudioPlayer getAudioPlayerFor(string path) //determines the correct AudioPlayer for the specified file throws exception if a suitable AudioPlayer can not be found

Below is an example of using an AudioPlayerFactory to obtain a player for an audio file:

```
....
String someAudioFile=”c:\someAudioFile.mid”;
try{
	AudioPlayer player=AudioPlayerFactory.getAudioPlayerFor(someAudioFile);
	player.play(someAudioFile,AudioPlayer.INFINITE_LOOP);
} catch(UnsupportedAudioFormatException uafx){
	//error handling
}
```

UnsupportedAudioFormatException

This should extend the exception class for the platform it's running on. No special behavior is needed.

UnsupportedAudioFormatException

[note: preliminary section for Tiamat 0.3.x]

Framework Applications

Alright, these are some nifty classes and all but how do they fit together to create a working game? That's a fine question, after all the goal of Tiamat is to provide the foundation for developing an RPG. This section details how the Tiamat RPG Framework can be used to to build a functional game.

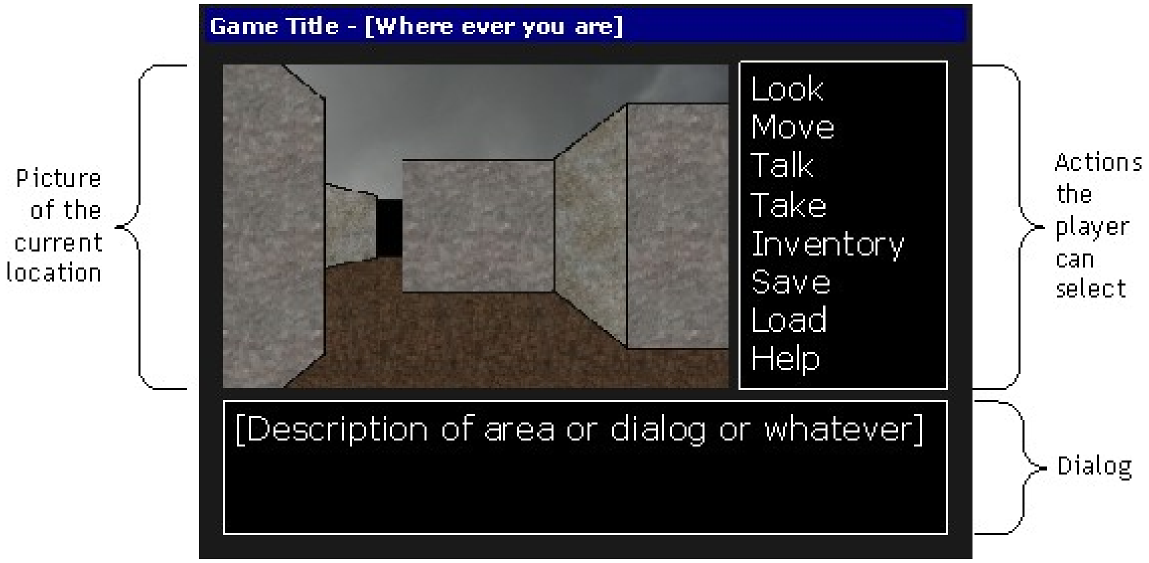
Adventure Style Game

Overview

An "Adventure Style" game (for lack of a better term) is one where the player travels between locations and executes commands though a dynamic menu. Dialogs from most console RPGs use a similar system.

The interface for the game contains three sections (also illustrated below):

- A panel showing an image of the current location and any characters or items there.
- A panel listing the actions the player can select.
- A panel containing a description of the area or the text corresponding to the action that was selected.


Game Title - [Where ever you are]
Look
Move
Talk
Take
Inventory
Save
Load
Help
[Description of area or dialog or whatever]
Picture of the current location
Actions the player can select
Dialog

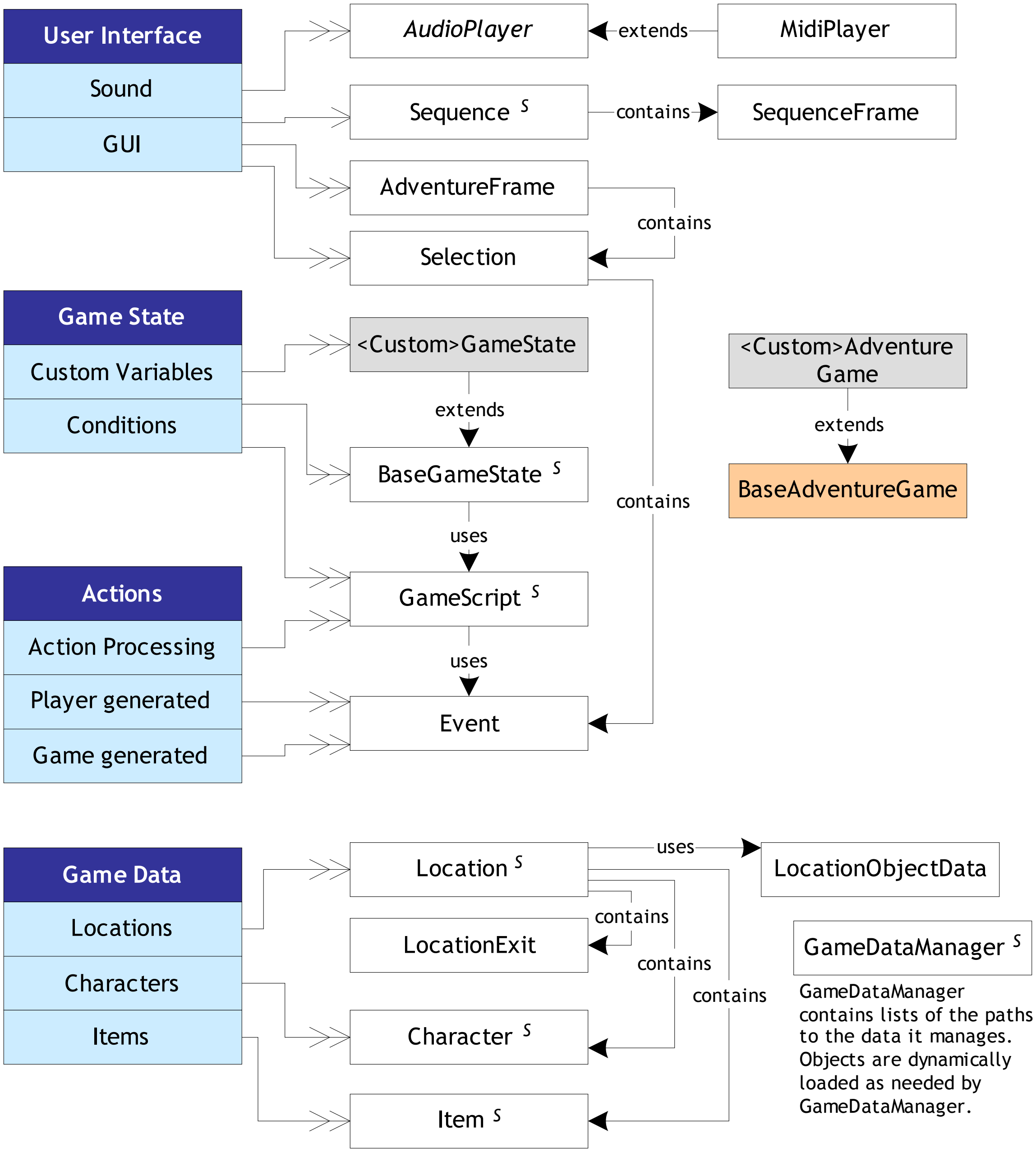
This style can be used as a standalone game or for dialogs/interactions within a separate game. For example, when the player enters a shop the perspective could switch to this style.

Components

To facilitate this style of game, the following major components are required:

- **User interface**: The user interface needs to render the three panels described above. Additionally, it needs to be able to play background music.
- **Game state**: The game needs to manage its internal state. It needs to track events that have occurred (conditions) and any custom variables.
- **Action/event handling**: The game needs to be able to handle events, both user initiated and those scripted in the game.
- **Game data**: The game data contains all the places the player can go, and all the objects in the game (characters and items).

Mapping these general requirements to framework components yields:



GameDataManager contains lists of the paths to the data it manages. Objects are dynamically loaded as needed by GameDataManager.

[to do: update when design/implementation are complete]

Example

[to do: add example]

Implementation Notes

External Classes

Several external classes are assumed to exist for Tiamat. Although one of the goals of Tiamat is to not rely on external classes, there are cases where it is not reasonable to recreate a class that already exists. Below is a list of external classes and what they map to in a specific language implementation:

Class	*Purpose*	*Java Equivalent*
Canvas	Area to draw images	javax.awt.Component
Image	Display image (gif,jpg) on Canvas	java.awt.image.BufferedImage
Color	Logical representation of a color	java.awt.Color
HashTable	Stores key-value pairs, used as the underlying implementation of StringTable	java.util.Properties (in StringTable class)java.util.Hashtable (everywhere else)
Font	Font to use on user interface	java.awt.Font

Java

- In Java there are no such thing as "properties". It's possible to expose members publicly but ill-advised. Instead, get/set methods are used to access public properties.
- Java does not have "readonly" data types. Readonly properties are represented by having a public get method and a private set method (or none at all).
- Enumerations didn't exist in Java when the Tiamat implementation started. In some classes public static members are used instead. These should be converted to enums at some future point.
- Typed ArrayLists are used instead of arrays.
- A few public methods were added to some objects that are not defined in framework specification. This was done to accommodate serialization. Images were marked *transient* so they would not be stored in a file. This reduces the size of serialized objects and to makes it easier to change images. The specific changes are:
    - Item: added public bufferImages() method to be called after de-serializing an Item (or Character) from a file.
    - Character: added public bufferImages() method to be called after de-serializing a Character from a file. This file also calls bufferImages() for its

    sprite and item collection.
  - Sprite: added public bufferImages() method to be called after de-serializing a Sprite (or Character) from a file. Sprite also has a private loadImages() method that's used by the constructor and importImageData() methods.
  - TileMap: added public bufferImages() method to be called after de-serializing a TileMap from a file. TileMap also has a private loadImages() method that's used by the constructor and import*xxx*() methods.
  - Area: added public bufferCharacters() method to be called after de-serializing an Area from a file.
- ScriptTableKey: added overrides of Object.equals(Object) and hashCode() so it could be used a key on a HashTable.
- Some UI components contain methods not present in the specifications because they are implementation specific:
  - AdventureFrame:
    - Extends JFrame so any public attrbiutes of JFrame are also exposed
    - Added several methods to handle UI events
    - Title property and setVisible method not implemented because they're already contained in the parent JFrame class
    - Get/Set methods are present for the text property but no private variable is needed
    - Added a limit on number the number of selections that can be drawn, needed to manage UI

Desired Implementations

- C# .NET would be a logical platform to implement Tiamat in. The syntax is very similar to Java and it includes properties.
- Mobile Frameworks such as .NET Compact Framework or MIDP.

Legal Notes

Although Tiamat was inspired by several games, this document reflects the original work of the author. To the best of the author's knowledge, these specifications do not infringe on any copyright or patent.

The cover page of this document uses the Alan Den font © 1997 UnAuthorized Type (http://uatype.faithweb.com/)

Phantasy Star, Phantasy Star II, Phantasy Star III, and Phantasy Star IV are copyrights of Sega (http://www.sega.com/).
Lunar the Silver Star, and Lunar Eternal Blue are copyrights of Working Designs (http://www.workingdesigns.com)
Chrono Cross is a copyright of Square (http://www.squaresoft.com/)
Legend of Heroes: Dragon Slayer and Ys I&II are copyrights of Nihon Falcom (http://www.falcom.com/)
Grandia II is a copyright of UBI Soft (http://www.ubisoft.com/), Game Arts (http://www.gamearts.co.jp/)
Super Mario RPG is a copyright of Nintendo (http://www.nintendo.com/)
Shadowrun is a copyright of FASA (http://www.fasastudio.com/)
Java is a registered trademark of Sun Microsystems (http://www.sun.com/)
C# .NET is a copyright of Microsoft (http://www.microsoft.com/)

This document was created using OpenOffice (http://www.openoffice.org/) copyright Sun Microsystems (http://www.sun.com/)

GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgments", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was

based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgments" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgments", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgments", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the
> Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.