GamesCrafters 2007

Using Text Resource Files

Tutorial Documentation

Documentation Team

Revision Summary			
Author	Date	Version	Comments
Daniel Wei, Yiding Jia	2007.11.27	0.1	Created
Daniel Wei	2007.12.05	0.2	Revision 1.0

Table of Contents:

1. Overview

Text Resource Files are an extension to the current Gamesman core that allows games to be updated with much fluidity and internationalized for usability. The goal of text resource files (thereafter called resource files) is to offer a centralized location with a documented format for keeping all strings used by Gamesman.

The major current problem that text resource files tackle is the fact that there are currently many instances of strings for each module of Gamesman. A particular string can have instances in: the gamescrafter's website, c code, and tcl code. This can cause pains when needing to update string values. By consolidating all strings to a centralized location with different modules calling the same resource file, one would only have to modify one string instead of many instances.

An added benefit is the ability to internationalize the game system. By writing another resource file in another language, the game system can quickly switch to the new language resource file.

Games currently implemented using this include:
Tic-Tac-Toe (w/ English, Spanish, and Piglatin)

We envision this extension to be implemented in all game modules, all menus, and the gamesman core/extensions.

2. Goals and Non-Goals of this Document

This documentation target developers who are seeking to add text resource files to anywhere that prints out strings. This documentation will focus on the interface with game modules since it is the primary site of strings. Developers looking to adding text resource files to other sections of the game system can still use this documentation to adapt text resource files to his/her specific section.

- High level system overview of Text Resource Files.
- Familiarity with the API between Text Resource Files and the game module.
- Reference to an example implementation of resource functionalities.

Readers of this documentation are expected to have familiarity with C, the high-level Gamesman system architecture, and the API between Gamesman core and modules. In particular this documentation will not

- Offer tutorial on C.
- Offer tutorial on Gamesman architecture or other extensions to Gamesman.
- Offer tutorial on how to implement other parts of the game modules.
- Detail how Text Resource Files are implemented.

It is not necessary to have an in-depth, below-the-API understanding of Text Resource Files to implement Tier functionalities in game modules.

All other issues not listed as goals would be considered non-goals.

3. Inclusion of Strings in Text Resource Files

A few points determine whether or not to include a string in the resource files. Not all strings are automatically included in a resource file. This is to prevent a cluttering of resource files and to keep files usable.

Strings to include in resource files have one or more of the following characteristics:

- Have instances in many locations (ie. Description of how to win a game)
- Need to be modified, updated (ie. Game rules)
- Crucial to the understanding and interaction with the game system (ie. How to input moves)

Strings to leave out of resource files have one or more of the following characteristics:

- Debugging statements/menus
- Draw the board/Formatting strings

In general, strings that allow for customization of the game will be included in the resource files. Strings that do not have any of the characteristics warranting inclusion should not be included.

4. API Between Tier Gamesman and Game Modules

API Overview

There are no new functions that need to be implemented in the game modules.

Modifications to the existing InitializeGame() function and locations where strings are called are necessary.

Creation of separate xml resource files is necessary for the Text Resource File system to fetch data. They should be located in gamesman/bin/xmlfiles/. Each game xml file should be named according to the following convention:

[language]/[gamename].xml

where [gamename] is replaced with the name of the game (ie. Mttt) and [language] is replaced with the two letter abbreviation of the language. A table for the two letter abbreviation can be found at http://www.loc.gov/standards/iso639-2/php/code_list.php.

A separate xml for game specific menu strings are located in **[textgui].xml**.

A last xml file for rule of the game (toMove, toWin) are located in **[rules].xml**.

The text resource file system will try to locate a given xml file when directed. If no file is found (ie. bad address) a warning will appear stating so. The game will continue to run, however. Strings not previously set will display as (null).

When a string key is not found, a warning will appear detailing the invalid string key. The string will be displayed as (null).

The resource file system also allows for loading of multiple text resource files. The system will try to locate strings starting with the last or most recent text resource file loaded. Thus, string values can be overwritten by loading a new text resource file.

Text Resource File Format

Format of the Text Resource File follows loosely with XML styling. The most basic file contains information in the following format:

- <resource> tag (first tier of file format)
- <string> tag with string key.

```
<resource>
      <string key="stringtag">the string here</string>
</resource>
```

Furthermore, groups of strings (of the same nature/function) can be grouped together for added readability. Groups are implemented under the <resource> tag using:

- <group> tag with string name.

```
<resource>
      <group name="groupName">
            <string key="str1tag">str1</string>
            <string key=" str2tag">str2</string>
            <string key=" str3tag">str3</string>
      </group>
</resource>
```

Strings can contain subsets of other strings. Substrings are basically strings called from another string. Substrings do not have to be below the string that called them. Substrings are implemented inside the string argument using:

- <subst> tag with string name.

```
<resource>
      <string key="str1tag">str1<subst key="substrtag"/> </string>
      <string key="substrtag">substr</string>
</resource>
```

An added functionality is the inclusion of <tab/>. Inclusion of this anywhere in the string will cause a tab to be displayed.

Please look the example text resource files for the specific strings, groups, and overall format of a resource file for a game module. See section 5: Example Implementation.

Strings must adhere to requirements set by the libxml library. Specifically, certain characters reserved by the libxml library must be replaced by special entity references. The table below details five common characters. For more information please see http://www.w3schools.com/xml/xml_cdata.asp.

Entity Reference	Character	Description
&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Figure 1. Common character replacements

String Key Naming Convention

Strings keys have the following conventions:

- Keys that start with "ui" are text user interface related.
- Keys that start with "gui" are graphic user interface related.
- Keys that start with "k" are game specific.
- Keys that end with "F" are format templates for printf(). (see example)

```
<string key="percentF">%d%%</string>
<!--For example, this will result in "50%"-->
```

How to Name New Keys

Strings keys should be named according to the following rules, always following the above naming convention (see section above):

- If a string is contained in a variable, use the variable name for the key.
- If a string is not contained in a variable, choose a short but descriptive name. Some suggestions include the function name or the first few characters of the string.
- String keys should NEVER conflict.

New Data Types

None.

Global Variables

Game modules need to be aware of a new global variable:

```
extern STRING gLanguage;
extern STRING kShortName;
```

The game system sets gLanguage at the beginning of the game based on command line parameter.

The game module sets kShortName statically to be name of the game module (ie. mttt)

Essential Functions to Call

There are two public functions that are used to interface with the text resource system.

`int loadResource(char* filename)`
Input: String of filename
Output: Int (1 if successful, 0 otherwise)
Description: Used to load external resource files to the problem. Filenames are assumed to be under the gamesman/bin directory.

`char* getResourceString(char* key)`
Input: String of key of string to be looked up
Output: String (NULL is string is not found)
Description: Used to retrieve string given a known key. The user does not need to free string. CAUTION: Pointer will be invalidated if load resource is called.

Essential Modifications – InitializeGame()

There are generally two types of strings in a game module that need to be modified. One category is global strings that are interfaced by the gamesman core (ie kGameName). The other category is local strings.

For global strings, do the following in `initializeGame()`:

- Do not modify the default value of the strings. This is to allow for the system to still display meaningful strings in the rare case that text resource files are not found or are missing.
- Load all text files needed (Remember newer files have higher precedence than older files)
- Set all global strings necessary using the `getResourceString(STRING)` function.
    - Some strings that must be included: kAuthorName, kGameName, kHelpGraphicInterface, kHelpTextInterface, kHelpOnYourTurn, kHelpStandardObjective, kHelpReverseObjective, kHelpTieOccursWhen, kHelpExample, gBlankOXString

For local strings, do the following anywhere needed:

- Replace all instances of `printf("str")` with instances of `printf("%s", getResourceString("strkey"))`.

5. Example Implementation

An example implementation is found in gamesman/src/mttt.c.

The xml files associated with mttt.c are located in the gamesman/bin/xmlfiles directory. An example implementation of the text resource file format for mttt.c can be found in under the xmlfiles directory.

This module implements Tic-Tac-Toe with Text Resource Files. It is also completely compatible with the original Gamesman.

6. Running Text Resource Files

Setting Up Cygwin

Text Resource Files should run automatically. For international fonts to run in cygwin, a recommended add-on program is Ponderosa which can be downloaded at http://en.poderosa.org/. Ponderosa is an interface used with cygwin that allows for different character encodings such as utf-8.

Choosing a Language

To change language, start gamesman with the flag ––lang [language] (ex. ./mttt ––lang en)