txtEngine Documentation

Toby Herbert, Tataihono Nikora, Michael Abrams, James Boocock

Language Specification

Introduction

Utilising the advantages of XML we are able to suitably appropriate the framework to interpret instruction directly into object-orientated items. There are several objects that are critical to the framework design. These are World, Area, AreaCommand, Item, ItemCommand, Combine, and StateDescriptor. These are discussed in depth below.

DTD

There is a DTD in the games dtd folder. Please read it for further clarification on XML formatting.

XML Version Tag

Specifies the type of encoding used in the game file.

```
<?xml version ="1.0"?>
```

World

The World element contains everything in the game.

Attributes:

initialarea - the start area for the game
language - the language spec used
author - who wrote the game

Example:

```
<world initialarea="start_location" language="txtEngine" author="Reginald">
        ……
</world>
```

Area

Area elements are the places the player can ‘move’ to in the game. They contain the Items initially.

Attributes:

id - identifier for the area
initialdescription - the id of the state descriptor to use when the area is first visited.
status - used to indicate whether area is a ‘win’ or ‘die’ area

locked - true or false indicating whether or not the area is locked.

Example:

```
<area id="start_location" initialdescription="maze_start" status=”win” locked=”false”>
        This is a magic room.
</area>
```

Area Command

Area commands define actions that can be done inside a certain area but does not involve an item. eg) go north

Attributes:

name - the name of the areacommand
area - the area that this command takes you to
depends - specifies whether you need a certain item in order to use this command
synonyms - alternative names for the area command
status - changes the status of the area to this
locked - true locks the area, false does not

Example:

```
<areacommand name="east" area="furnace_room" depends="lamp" synonyms=”up” >
        The smell of coal fills your nostrils and you feel like you are walking into the sun.
</areacommand>
```

Item

Things throughout the world which the player can interact with. If collectable, the item can be moved to the players inventory.

Attributes:

id - a name that uniquely identifies the item
initialdescription - the description of an item when first
collectable - true or false indicating whether the item can be picked up or not
synonyms - alternative names to refer to the item by
depends - what getting the item depends on
locked - true if item is locked, or false if it is not
container - true if item is a container for other items, otherwise false

Example:

```
<item id="lamp" initialdescription="onpodium" collectable ="true"
synonyms="torch,light,flashlight">
 ……..
```

```
</item>
```

Item Command

The itemcommand element is placed inside an item element and defines the interactions that the player can have with the item.

Attributes:

name - a name that uniquely identifies the item command
statedescription - the state of the item changes to this when the command is executed
changecollectable - true if this command changes the collectable tag of the item when called. false if it does not
collectabledependent - true if this command can only be called when the item is collecatble
areachange - the area the item changes to when this command is executed
synonyms - alternative names to refer to the item command
status - change the status of the item
unlock - if this command unlocks the item

Example:

```
<itemcommand name="get" synonyms="fetch,grab,pickup" statedescription="onplayer"
changecollectable="true" collectabledependent="true" areachange="inventory">
        You pick up the lamp the light is warm and comforting.
</itemcommand>
```

Combine

Use this tag inside of **one** item tag of the two items that you want to combine.
For a user to combine two items they must use the combine keyword followed by the two items they wish to combine. For example: combine peas carrots.
Conjunctions can be used in the input but they must be put in the ignorewords.txt file so that they can be removed from the input. This way the input ‘combine the bullets with the gun’ will still work if the words ‘the’ and ‘with’ are in the ignore list.

Attributes:

id - differentiates between combines.
first_id - the item id of the item the combine tag is inside.
second_id - the item id of a second item to be combined.

Example:

```
<combine id="playdoh" first_id=”hairy_mince” second_id=”rainbow_marshmellow”>
        <item id="wikid_playdoh" ……..>
        ……..
        </item>
</combine>
```

StateDescriptor

The statedescriptor element is used inside Items and Areas to describe the Items and Areas in different states during the game.

Attributes:

id - the unique id of the statedescriptor

Example:

```
<statedescriptor id="dropped">
    A glowing lamp lying amongst the rubble.
</statedescriptor>
```

Simplified Overview of a Game:

```
<world>
    <area>
        <statedescriptor>
        </statedescriptor>
        <areacommand>
        </areacommand>
        <item>
            <statedescriptor>
            </statedescriptor>
            <itemcommand>
            </itemcommand>
            <combine>
                <item>
                </item>
            </combine>
        </item>
    </area>
</world>
```

Code Documentation

Inside the documentation folder there is Doxygen documentation in webpage format and also

PDF format.

Customisable Files

Constants.h - Certain error messages are defined here and can be customised to suit your implementation.

ignorewords.txt - Add words to this file on their own line. At run time the words are loaded from this file into a vector so that they can be removed from user input. This feature is intended to allow more natural input. For example, if a user inputs “put the lamp in the box” if ignorewords.txt includes the words “the” and “in”, they can be removed from the string and passed as a valid three word command “put lamp box”.

Installing the Game

Any standard C++ compiler can be used to compile txtEngine. The framework uses the C++ Standard library so is portable across platforms.

Alternatively for:

Linux

A Makefile is included which builds the txtEngine configuration for Linux operating systems. GNU Make and G++ Compiler are needed to use this feature.
Simply navigate to the txtengine folder and run ‘make’.

Playing the Game

New Game

To start a new game, from the build directory type:

```
./txtgame [name of game file]
```

New to the txtEngine?

How about playing the tutorial game to get used to different commands and interactions.

Loading a Game from a Saved File

To load a previously saved game:

```
./txtgame [name of game file] [name of save file]
```

Saving a Game to a File

When playing a game, simply type the command ‘save’ and press enter.
You will be prompted to enter a file name.

This file is now saved. To load a previously saved file see Loading a game.

Developing your own games

Please refer to the Language Specification part of this document to get help making your own game. A tutorial game has been created [input/tutorial_game.xml] for use as a guide and template if you wish.

Inbuilt Commands

inventory - Lists the items in the players inventory in the order they were picked up.
Synonyms for inventory are: **bag** or **i**
help - Displays a help string which is defined in the Constants.h file
save - Brings up prompts to save the current game
quit - Exits the game
look - Look around the area

Movement Commands

go north - also: **n**, **north** - Move the character to the north.
go south - also: **s**, **south** - Move the character to the south.
go east - also: **e**, **east** - Move the character to the east.
go west - also: **w**, **west** - Move the character to the west.

txtEngine External API

The textEngine API is a unique addition to the txtEngine family. The API is especially designed to externalize the GUI/CLI mechanisms so that custom implementations can be created. An example program is provided showing how one can fully utilize the API.

To call the API provide the following parameters to the executable:
.txtEngine -exec [game-file] [save-file] [command]

The executable will provide output to stdout in the following format:
The output is in provided in JSON format.

```
{
   "response": "string",
   "inventory": ["string", "string", …],
   "areaname": "string",
   "areadescription": "string"
}
```

To use the example application, place a compiled copy of the app in the root directory of the web folder. The "webserver" directory requires complete placement into a server environment,

for this requirement an apache server with php5 support will suffice.

NOTE: The server only works where support for:
- shellexec command under PHP is available
- save folder is read/writeable
- games folder is readable

A precompiled binary for OSX is provided in the root directory of the web server for convenience purposes.

Instructions for OSX:
- Copy webserver folder to ~/Sites
- Enable "Web Sharing" under System Preferences->Sharing
- Open web browser to "http://localhost/webserver"

Current Limitations in the txtEngine Implementation

- Duplicate synonyms across area commands, item commands, and items are not checked for and will be implemented in the next version of the game. For now, care must be taken when id attributes are given to elements to ensure there are no duplicates.

Future Features and To Do

- A graphical user interface for writing games.