

# Simple Object Markup Language

## Specification v1.0

### Changes

Ver.	Editors	Date	Changes
1.0	Timothy Sassone	March 2, 2012	Initial Revision

### Abstract

SOML (Simple Object Markup Language) is designed to serve as a markup language which provides a simpler and more readable syntax than XML, while still providing more powerful features such as tag-nesting which are lacking from INI-style formats. SOML also aims to be simple to parse, allowing software based on various systems, languages and design methodologies to use it effectively.

# Simple Object Markup Language

## Specification v1.0

### Table of Contents

[Changes](#)

[Abstract](#)

[Table of Contents](#)

[Introduction](#)

[Justification](#)

[Goals](#)

[Terminology](#)

[Logical Structure](#)

[Tags](#)

[Structuring](#)

[Comments](#)

[Parsing SOML Documents](#)

# Simple Object Markup Language

## Specification v1.0

### **Introduction**

Simple Object Markup Language (SOML), describes a class of data objects called SOML documents and describes the process of parsing them. SOML documents are made up of units of data called tags, which contain strings of characters separated by a colon. The markup encodes the parent/child structuring of the tags, as well as the names of the tags and the data contained within them.

### **Justification**

The idea for SOML arose from the need for a simple alternative to XML for storing basic, object-oriented data for a rogue-like engine I was developing. XML was functional, but complicated to parse, difficult to read, and unnecessarily complex and powerful for my purposes. The INI format worked for simple things, but lacked some of the features I needed (namely, nested tags). SOML was designed to fill that gap between the powerful XML, and the incredibly simple INI.

### **Goals**

1. SOML should be simple to parse.
2. SOML should provide support for tags with multiple arguments/pieces of data.
3. SOML should provide functionality for child/parent relationships between tags. (single parent, multiple children)
4. SOML documents should be human legible.
5. Markup in SOML documents should be minimal.

### **Terminology**

- Tag - A unit of data within an SOML file.
- Parent/Child - A SOML document represents a tree-like structure, where certain tags are under/a part of others. In this relationships, the containing tag is called the parent, and any tags contained within it are called its children.
- Arguments - Arguments are the individual pieces of data contained within a tag.

# Simple Object Markup Language

## Specification v1.0

### Logical Structure

Each SOML document contains an arbitrary number of tags containing one or more pieces of data. The tags are structured in a tree-like format supporting a single-parent-multiple-child relationship between tags. These relationships are designated via white-space, and the format allows for plentiful and verbose comments without adding unnecessary markup.

### Tags

Tags are formed by a starting bracket, a closing bracket, and one or more arguments between them, separated by colons. The first argument is referred to as the tag's "name" and the others are referred to as its "data."

#### Examples:

```
[TagName:Data1:Data2]
[Tag2:15]
[simpleTag]
```

The first example defines a tag with the name "TagName", and with two pieces of data: "Data1" and "Data2." The second example shows a tag with the name "Tag2" and one piece of data: "15." The last shows a tag with no data, and the name "simpleTag."

### Structuring

Tags are stored in a tree-like structure allowing for a single-parent-multiple-child relationship. These are designated using an arbitrary number of tabs preceding a given tag. If a tag has zero tabs, it is considered a "root" tag. Otherwise, a tag is considered to be the child of the most recent tag with one fewer tabs than itself.

#### Example:

```
[Tag1]
    [Tag2]
    [Tag3]
        [Tag4]
    [Tag5]
```

In this example, Tag1 is a root tag, while Tags 2, 3 and 5 are its children, and tag 4 is a child of tag 3.

### Comments

Comments in a SOML document are any text which does not meet the conditions for a tag. This includes any line which starts with text instead of a tab or left-square-bracket, as well as any lines which contain a tag followed by text. A comment cannot precede a tag on a line,

# Simple Object Markup Language

## Specification v1.0

however, and must be placed on the line before it.

### Examples:

This is a valid comment.

[tag1] This is a valid comment.

Blah! [Tag2]

The above is an invalid comment, and causes the tag after it not to be parsed.

# Simple Object Markup Language

## Specification v1.0

### Parsing SOML Documents

Parsing a SOML document is fairly simple. Each line is read by the program one at a time. If the line fits the regex expression:

```
(\t*)\[([^\]\]\t\n]*)\]
```

then it is a legal tag, and can be parsed into an appropriate data-structure. This is done by counting the tabs, then removing them, stripping the brackets, and separating the arguments at each colon. The first argument is used as the tag's name, any other arguments (if present) are interpreted as its data.

While running, a parser keeps track of both the current stack/path of tags (based upon indentation), generally as a stack-style data-structure. When a new tag is read in, the number of tabs preceding it is considered, and one of three actions is taken:

- if there are zero tabs, clear the stack, add this tag as a root and push it to the stack.
- if there are fewer tabs than elements on the stack, back down the stack until we reach the appropriate tag (if necessary), add this one as a child, then push this tag to the stack.
- if there are an equal number of tabs as the previous tag, add this tag as a child of the tag at the top of the stack, then push it to the stack.

Parsers have been written in both C# and Python, and can be found on the [Google Code](#) page. These parsers are both provided as-is, and without any guarantee or warranty. Credit is appreciated, but not necessary. Both the SOML format and the C# and Python parsers are licensed under the [GPL v3](#).