Preferences Registry Format 1.0, Application Programming Interface, Specification and Implementation Notes

A. Sosnin

11th May 2004

Contents

1 Preamble

Abstract

Preferences Registry Format (is short: "PRF") is an XML-based simple configuration file format meant to be used in software that needs a simple, yet powerful, platform-independent way of storing configuration data. It is based on the simple name-value principle commonly used in many software configuration formats.

This format is meant to be generic and multi-purpose, but it is not designed to be universal. If you need a more complex, or a more specialised way of storing application configuration data, it is suggested to refer to a more comprehensive preferences file format named "CC/PP" which stands for Composite Capabilities/Preference Profiles [5]. The Preferences Registry Format is a major simplification of the ideas put into the CC/PP Resource Description Framework [6] based file format.

This document describes the language-independent and architecture-neutral Application Programming Interface (herein: API) for randomly accessing the PRF 1.0 format data from applications. Additionally, it gives some notes on implementing libraries using this API.

Note, that the described API is restricted to language-independent features to guarantee, that conforming API implementations contain all the described features. It might be useful or even necessary to add language-specific extensions to the described here for a library to be more useful.

1.1 Status of this document

This document is a request for comments (RFC). It is a work in progress, but is already considered stable enough. Comments about any part of this document are still welcome.

The design solutions, expressed in this document, are not yet intended for production use.

This document is subject to the copyright notice below.

1.2 Copyright notice

2 PRF 1.0 API Specification

2.1 Design goals

This API's design is aiming to make the PRF 1.0 file usage easy for application developers, thus the following goals apply:

- incapsulate configuration data structures into a common "container" idiom and define interfaces, common to containers and familiar to all (or most) programmers.
- perform "in-line" checks for format consistency, after each structure change.

2.2 Overview and structure

The PRF 1.0 API consists of the following interfaces:

Containers: `Config` — main interface for operating on PRF 1.0 files.

`Profile` — a single configuration profile.

`Component` (alias: `Context`) — a single configuration component/context.

Elements: `Option` — a single configuration option with a name and a text value associated with it. (Sec. 2.7).

Base interfaces: `Container` — defines all generic container functions to be derived in containers. (Sec. 2.8).

`ContainerItem` — an interface for Container items. (Sec. 2.9).

Additional interfaces: `IDContainer` — defines functions to access PRF data directly via IDs. Optional. (Sec. 2.10).

`ConfigFactory` — Factory interface. Defines functions to create `Config` (Sec. 2.4) instances using different ways (by providing a source file name, a URL, an open I/O stream etc. (Sec. 2.3).

2.3 ConfigFactory

Factory interface to construct `Config` instances from different data sources.

`Config loadFromFile(...)` – Load the PRF data from a file with a given name and construct a `Config` instance. Arguments: `String fname` — file name.

`const Config loadFromURL(...)` – *Read-only*, *Optional*. Load the PRF data from a source via URL. Arguments: `String url` — URL string.

`Config loadFromIOstream(...)` – Load the PRF data from a file with a given name and construct a `Config` instance. Arguments: `iostream iost` — an open in read-write mode I/O stream.

`Config loadFromString(...)` – Load the PRF data from a file with a given name and construct a `Config` instance. Arguments: `String data` — a string, containing the PRF 1.0 formatted data.

`Config createNew(...)` – create a new minimal configuration file from scratch. This file will contain one `profile` element with *id* attribute set to "p1", one `component` element with *id* attribute set to "c1" and one empty option with a "name" attribute set to "o1".

2.4 Config

This interface inherits the Container interface and defines createItem method:

`void <<create>>(...)` — Constructor, initialises the element and puts it in the DOM document tree. Arg uments:`DOMElement xelem` – a corresponding XML element's DOM represendation.

`iterator createItem(...)` — Creates a new empty item given the data. Arguments: `String id` — the ID of the new item.

2.5 Profile

This interface defines a set of functions to manipulate on the configuration file's *profile* and its elements — `Component` (Sec. 2.6) instances. Classes that implement this interface must also implement the `Container` and the `ContainerItem` interfaces. Functions defined here:

`void <<create>>(...)` — Constructor, initialises the element and puts it in the DOM document tree. Arg uments:`DOMElement xelem` – a corresponding XML element's DOM represendation.

`void setId(...)` — sets the ID of the `profile` XML element. Arguments: `String id` — the new ID.

`String getId(...)` — gets the ID of the `profile` XML element. Arguments: none.

`iterator createItem(...)` — Creates a new empty item given the data. Arguments: `String id` — the ID of the new item.

2.6 Component

Alias: `Context`. This interface defines a set of functions to manipulate on the configuration file's *component* and its elements — `Option` (Sec. 2.7) instances. Classes that implement this interface must also implement the `Container` and the `ContainerItem` interfaces. Functions defined here:

`void <<create>>(...)` — Constructor, initialises the element and puts it in the DOM document tree. Arg uments:`DOMElement xelem` – a corresponding XML element's DOM represendation.

`void setId(...)` — sets the ID of the `profile` XML element. Arguments: `String id` — the new ID.

`String getId(...)` — gets the ID of the `profile` XML element. Arguments: none.

`iterator createItem(...)` — Creates a new empty item given the data. In Component interface this method has another counterpart, that enables creation of Option instances given its name and value. Arguments: `String id` — the ID of the new item; `String value` — optional; value of the new Option.

2.7 Option

This interface provides functions to manipulate *option* XML elements, to access their names and values. Classes that implement this interface must also implement the `ContainerItem` interface. This interface provides the following functions:

`void <<create>>(...)` — Constructor, initialises the element and puts it in the DOM document tree. Arg uments:`DOMElement xelem` – a corresponding XML element's DOM represendation.

`String assign(...)` — Assign a value to the subsequent option. Arguments: `String value` — assigned value.

`String value(...)` — Get the value of the current option. Arguments: none.

`String name(...)` — Get the name of the current option. Arguments: none.

`String key(...)` — *optional*, same as above: get the name of the current option. Arguments: none.

`void setId(...)` — sets the name attribute of the `profile` XML element. Arguments: `String id` — the new attribute value.

`String getId(...)` — gets the name attribute value of the `profile` XML element. Arguments: none.

2.8 Container

A container is an object that *contains* a set of elements. Thus, container interfaces should provide a minimal set of methods for manipulating these elements: at least adding, deleting, getting the values of and modifying the values of them. The `Container` base interface defines the following methods:

`void deleteItem(...)` — delete an item from the sequence. Arguments: `String idkey` – a string with the element's ID.

`element_type getItem(...)` — get an item by reference. Arguments: `String idkey` – a string with the element's ID. *Aliases*: `=operator[]` – a function, that enables the interface to emulate standard container syntax for getting an element's value, e.g. `def __getitem__(self, key)` in Python and `element_type &operator[](const key_type key)` in C++.

`void setItem(...)` — set an item's value. Arguments: `String idkey` – a string with the element's ID, `element_type value` – an element instance. *Aliases*: `operator[]=` – a function, that enables the interface to emulate standard container syntax for setting an element's value, e.g. `def __setitem__(self, key, value)` in Python and `element_type &operator[](const key_type key)` in C++.

`void addItem(...)` — append an element to the set. Arguments: `element_type elem` – an element instance to append.

`void addItems(...)` — append a set of elements to the container. Arguments: `container_type elems` – a standard container of element instances.

`void createItem(...)` — Create an empty element. Arguments: `key_type id` – new item's ID.

`unsigned int size(...)` — return the size of the set. Arguments: none.

`Node getNode(...)` — return the DOM Node element, to which the current object is assigned. Arguments: none.

2.9 ContainerItem

ContainerItem defines functions common to elements of "containers", e.g. Profile, Component, Option instances. Contains abstract functions getId(), setId() and getNode(), and doesn't define any instance variables.

`void setId(...)` — sets the ID of the `profile` XML element. Arguments: `String id` — the new ID.

`String getId(...)` — gets the ID of the `profile` XML element. Arguments: none.

`Node getNode(...)` — return the DOM Node element, to which the current object is assigned. Arguments: none.

2.10 IDContainer

This interface provides functions to access elements of PRF file directly via IDs. It provides the following functions:

`Profile getProfile(...)` – get (construct) a `Profile` instance by its ID. Arguments: `String id` — id of the element.

`void setProfile(...)` – set a `Profile` instance by its ID to a given value. Arguments: `String id` — id of the element; `Profile p` — new `Profile` instance to assign.

`Component getComponent(...)` – get (construct) a `Component` instance by its ID. Arguments: `String id` — id of the element.

`void setComponent(...)` – get (construct) a `Component` instance by its ID. Arguments: `String id` — id of the element; `Component c` — new `Component` instance to assign.

3 Implementation notes

This part is informative

3.1 Dependencies

It is recommended for a library or application that uses the following API to *randomly* (in read and write mode) access the PRF 1.0 file to be based on a World Wide Web Consortium's [3] Document Object Model [4] implementation. The current API is best suited for random access using an underlying DOM library.

Informative: (DOM Implementations reference)

3.2 Document Type Definition

The following code is a non-normative DTD, useful for Preferences Registry Format files validation:

```
<!ELEMENT prf (profile*)>

<!ATTLIST profile
xmlns CDATA #IMPLIED >

<!ELEMENT profile (component*)>

<!ATTLIST profile
id ID #REQUIRED
name CDATA #IMPLIED >

<!ELEMENT component (option*)>

<!ATTLIST component
id ID #REQUIRED
name CDATA #IMPLIED >

<!ELEMENT option (#PCDATA)*>

<!ATTLIST option
name NMTOKEN #REQUIRED >
```

References

4 Normative references

[1] PRF 1.0 Specification, *http://tomato.dyn.ee/?id=prf1*

[2] Extensible Markup Language 1.0, Second Edition, W3C Recommendation 2 October 2000, *http://www.w3.org/TR/REC-xml*

5 Informative references

[3] World Wide Web Consortium web-site: *http://www.w3.org/*

[4] Document Object Model web-site: *http://www.w3.org/DOM/*

[5] CC/PP (Composite Capabilities/Preference Profiles): Structure and Vocabularies 1.0, W3C Proposed Recommendation 15 October 2003, *http://www.w3c.org/TR/2003/PR-CCPP-struct-vocab-20031015/*

[6] Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, *http://www.w3.org/TR/REC-rdf-syntax/*