

1 Target consumers of this standard

The target consumer of this standard is a desktop application. Not all desktop services nor a Internet or enterprise service but a desktop application.

Desktop applications like browsers, office suites, E-mail clients and instant messengers that need to store personal preferences of the user.

This standard is about handling personal preferences that influence the behaviour of desktop applications.

2 Technology dependencies of this standard

For a desktop developer, the only dependency for using this standard is D-BUS and a service implementation that will implement it.

This standard doesn't add any other dependency. An implementation that respects this standard, can be used by any consumer of this standard.

3 Serve and consume using D-BUS

This standard defines a consumer versus service protocol. A server (the service) delivers the information and that performs requests coming from a consumer (a client).

The client consumes the service. D-BUS is the transport layer technology between client and service.

The desktop application plays the role of the consumer. An implementation of this standard plays the role of the service.

4 Overview of the supported features

This standard supports a few features. This is an overview of these features:

- The following basic types: int, string, Boolean, double
- Non homogeneous or mixed lists with instances using the basic types and again lists (recursive types with a depth limit of 32 levels of recursion)
- A standardised root namespace specification
- A standard way of defining the path to a setting (defining the key)
- The possibility to define combined types or structures using non homogeneous lists
- Support for type enforcement for a specific setting using schemas
- Support for a default value for a specific setting using schemas
- Support for a localised long and short description for a specific setting using schemas
- Usage of the D-BUS inter process communication system
- Notification when specified keys change

5 Schemas

This standard defines the schema DTD

5.1 The purpose

A schema has the purpose of making the system type safe, providing the description and default value of a key.

5.2 File naming conventions

Schemas are installed by applications and or packages. Each such application should install one ".schemas" file which contains all the new schemas the application will use.

The filename is formed by replacing all slash characters with underscores.

The filename is appended with ".schemas."

The directory where the schemas files are located is "\$XDG_DATA_DIRS/configuration"

Some examples:

- \$XDG_DATA_DIRS/configuration/org_gnome_nautilus.schemas for keys in /org/gnome/nautilus/
- \$XDG_DATA_DIRS/configuration/org_kde_konqueror.schemas for keys in /org/kde/konqueror/

5.3 Definition of a key

A key is the full path to one configuration setting. The full path can be constructed by appending an "item" with "the application name" with the "root namespace" of the organisation who uses the configuration setting.

- The path must begin with an ASCII '/' (integer 47) character, and must consist of elements separated by slash characters.
- The path may be of any length.
- Each element must only contain the ASCII characters "[A-Z][a-z][0-9].[-]"
- No element may be the empty string.
- Multiple '/' characters cannot occur in sequence.
- A trailing '/' character is not allowed unless the path is the root path (a single '/' character).

5.4 Root namespace conventions

The prefix of a unique namespace is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981.

Subsequent components of the namespace name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.

For the subsequent components, all characters with exception of the slash and the space are allowed.

The components are separated with a slash character. The root namespace always ends with a final slash character.

Some examples of root namespaces:

- `"/org/gnome/"`
- `"/org/kde/"`
- `"/org/xfce/"`
- `"/com/sun/java/"`
- `"/com/apple/quicktime/"`

Note that a root namespace is not yet a complete key. You need to append a preference to it to form a full key.

5.5 Preference naming conventions

A "preference" is one setting stored somewhere inside your root namespace. Once inside your root namespace you are free to choose your own naming scheme for the preferences and subsequent components.

All characters with exception of the slash and the space are allowed for preferences. Subsequent components are separated with a slash character.

You cannot share the name of a subsequent component with the name of a preference.

The following examples are to be interpreted as suggestions:

- `"/org/gnome/nautilus/always_use_browser"` as a boolean
- `"/org/kde/konqueror/position"` as a rect
- `"/com/sun/java/classpath"` as a string
- `"/com/apple/quicktime/default_player"` as a string
- `"/com/cheeseonline/products"` as a list

5.6 Shareable items

A committee decides about the naming of shareable items.

As an application developer you can get in touch with the members of this committee to register your shareable items.

For example:

- `"/networking/web/proxy_url"` as a string
- `"/networking/web/proxy_port"` as an integer
- `"/networking/email/smtp_host"` as a string
- `"/networking/email/smtp_port"` as an url

5.7 Overview of the DTD

TODO: Convert this to an XSD

The root of the document contains node nodes

```
<!ELEMENT node (node*|schema*|choice*)>
```

You can define enums (unfinished) <!ELEMENT enum (choice*)>

```
<!ATTLIST enum name CDATA #REQUIRED>
```

```
<!ELEMENT choice (description*)>
```

```
<!ATTLIST choice value CDATA #REQUIRED>
```

Each node represents a part of a key. For example if the key to a preference is "/one/two/three" then you'd have a node "two" in a node "one" and a schema for "three" (see below).

```
<!ELEMENT node (node*|schema*)>
```

```
<!ATTLIST node name CDATA #REQUIRED>
```

For each key the application uses, you create a schema node. A schema node describes the type, default and all the translations of the description. It optionally also contains a minimum and a maximum of the value.

```
<!ELEMENT schema (type,min?,max?,default,description*)>
```

The schema node has the optional attribute "item". It's not used for schemas in lists but it is used for all other schemas. The value is the name of the preference. In the above example, this value would be "three".

```
<!ATTLIST schema item CDATA #IMPLIED>
```

Each schema node has a type element.

```
<!ELEMENT type EMPTY>
```

Such a type element has a required "dbus" and an optional "name" attribute. The dbus attribute holds the type in D-BUS signature format.

```
<!ATTLIST type dbus CDATA #REQUIRED>
```

The "name" attribute is for high level use. The application developer (or high level library writer) can use this field freely to provide type safety for composed special types. Types like "rect" or "color".

```
<!ATTLIST type name CDATA #IMPLIED>
```

Both "min" and "max" are optional nodes in the schema node. For integer types they will enforce a minimum and a maximum on the value. They can also be freely used, by the application developer or high level library writer, for composed types.

```
<!ELEMENT min (#PCDATA)>
```

```
<!ELEMENT max (#PCDATA)>
```

The "default" node can contain or a as a string serialized default value (for simple non-list types) or a list of schemas (for a size limited list type with different types per instance in the list) or a single schema (for a non limited list type where each instance in the list uses the same type).

```
<!ELEMENT default (#PCDATA|schema)*>
```

The description field contains one translated node per supported language.

```
<!ELEMENT description (translated*)>
```

The character data of a translated node is the long description of the key translated in the language as defined by the "xml:lang" attribute (see below).

```
<!ELEMENT translated (#PCDATA)>
```

The "xml:lang" attribute holds the language of this translated description.

```
<!ATTLIST translated xml:lang CDATA #REQUIRED>
```

The "short" attribute holds the short description translated in the language as defined by the "xml:lang" attribute.

```
<!ATTLIST translated short CDATA #REQUIRED>
```

5.8 schema.dtd

```
<!-- DTD for fd.o configuration schema data -->
<!-- (C) 2005-08-29 Philip Van Hoof; -->
<!--
This document is in the public domain. Permission to use,
copy, modify, and distribute this document for any purpose
and without fee is hereby granted, without any conditions
or restrictions. This document is provided "as is" without
express or implied warranty.
-->
<!ELEMENT schemas (node*)>

<!ELEMENT node (node*|schema*|choice*)>
  <!ATTLIST node name CDATA #REQUIRED>

<!ELEMENT enum (choice*)>
  <!ATTLIST enum name CDATA #REQUIRED>
<!ELEMENT choice (description*)>
  <!ATTLIST choice value CDATA #REQUIRED>

<!ELEMENT schema (type,min?,max?,default,description*)>
  <!ATTLIST schema prefname CDATA #IMPLIED>

<!ELEMENT type EMPTY>
  <!ATTLIST type dbus CDATA #REQUIRED>
  <!ATTLIST type name CDATA #IMPLIED>

<!ELEMENT min (#PCDATA)>
<!ELEMENT max (#PCDATA)>

<!ELEMENT default (#PCDATA|schema)*>

<!ELEMENT description (translated*)>
<!ELEMENT translated (#PCDATA)>
  <!ATTLIST translated xml:lang CDATA #REQUIRED>
  <!ATTLIST translated short CDATA #REQUIRED>
```

5.9 sample_namespace_sample_application.schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE schema PUBLIC "-//freedesktop//DTD configuration schema 1.0//EN"
"http://www.freedesktop.org/standards/configuration/1.0/schema.dtd">
<schemas><!-- Simple types -->
<node name="sample_namespace"><node name="sample_application"><node name="prefs">
  <schema prefname="my_string">
    <type dbus="s" name="string"/><default>Default string</default></schema>
  <schema prefname="my_integer">
    <type dbus="i" name="integer"/><default>20</default></schema>
  <schema prefname="my_double">
    <type dbus="d" name="double"/><default>20.99</default></schema>
  <schema prefname="my_boolean">
    <type dbus="b" name="boolean"/><default>0</default></schema>
<!-- Simple lists -->
  <schema prefname="my_boolean list">
    <type dbus="ab" name="boolean"/>
    <default>
      <schema><type dbus="b" name="boolean list"/><default>0</default></schema>
    </default>
  </schema>
  <schema prefname="my_string list">
    <type dbus="as" name="string list"/>
    <default>
      <schema><type dbus="s" name="string"/><default>Default string</default></schema>
    </default>
  </schema>
  <schema prefname="my_integer list">
    <type dbus="ai" name="integer list"/>
    <default>
      <schema><type dbus="i" name="integer"/><default>10</default></schema>
    </default>
  </schema>
  <schema prefname="my_double list">
    <type dbus="ad" name="double list"/>
    <default>
      <schema><type dbus="d" name="double"/><default>10.99</default></schema>
    </default>
  </schema>
  <!-- Samples of possible complex types -->
  <!-- Font -->
  <schema prefname="my_font">
    <type dbus="si" name="font"/>
    <default>
      <schema><type dbus="s" name="font_name"/><default>Arial</default></schema>
      <schema><type dbus="i" name="font_size"/>
        <min>8</min><max>32</max><default>12</default></schema>
    </default>
  </schema>
  <!-- A rectangle -->
  <schema prefname="my_rect">
    <type dbus="iiii" name="rect"/>
    <default>
      <schema><type dbus="i" name="rect_x1"/>
        <min>0</min><max>100</max><default>1</default></schema>
      <schema><type dbus="i" name="rect_y1"/>
        <min>0</min><max>100</max><default>10</default></schema>
      <schema><type dbus="i" name="rect_x2"/>
        <min>0</min><max>100</max><default>10</default></schema>
      <schema><type dbus="i" name="rect_y2"/>
        <min>0</min><max>100</max><default>1</default></schema>
    </default>
  </schema>
  <!-- A color -->
  <schema prefname="my_color">
    <type dbus="iii" name="color"/>
    <default>
      <schema><type dbus="i" name="color_R"/>
        <min>0</min><max>255</max><default>110</default></schema>
      <schema><type dbus="i" name="color_G"/>
        <min>0</min><max>255</max><default>120</default></schema>
      <schema><type dbus="i" name="color_B"/>
        <min>0</min><max>255</max><default>130</default></schema>
    </default>
  </schema></node></node></node>
</schemas>
```

6 The protocol

6.1 The errors

An error reply has the message type `ERROR` (decimal 3) as specified in the D-BUS specification. The first argument of such a reply is the error message. Possible error messages are (with a label):

ERROR_NAME	First argument
<code>NOSUCHKEYERROR</code>	"No such key error"
<code>KEYISREADONLYERROR</code>	"Key is read only error"
<code>INVALIDVALUEERROR</code>	"Key is not compliant with the schema"
<code>UNKNOWNERROR</code>	"Unknown error"

6.2 The types

value VARIANT

D-BUS signature: `v`

can be:

D-BUS Type	description	or
<code>STRING</code>	the string data	<code>or</code>
<code>INT64</code>	the integer data	<code>or</code>
<code>BOOLEAN</code>	the Boolean data	<code>or</code>
<code>DOUBLE</code>	the double data	<code>or</code>
<code>ARRAY of value VARIANT</code>	the list data	

The METAINFO DICT

D-BUS signature: `a{sv}`

D-BUS Type	description
<code>STRING</code>	name of the information
<code>VARIANT</code>	the information

metainfo VARIANT

D-BUS signature: `v`

can be:

D-BUS Type	description	or
<code>STRING</code>	the string data	i.e. a description <code>or</code>
<code>INT64</code>	the integer data	i.e. a UNIX timestamp

6.3 Special types

The value DICT for GetValues

D-BUS signature: `a{sv}`

D-BUS Type	description
<code>STRING</code>	the key
<code>value VARIANT</code>	the value

The data VARIANT for KeyChanged

D-BUS signature: v
can be:

D-BUS Type	description	or
BOOLEAN	removed	if removed and always true
value VARIANT	the newvalue	if not removed

6.4 Getting values

Gets the value of a key. In case no value is available, the default from the schema of the key is returned.

In case there's also no schema an NOSUCHKEYERROR error is returned.

org.freedesktop.configuration.GetValues

org.freedesktop.configuration.GetValue

As methods:

DICT	GetValues	(STRING root)
value VARIANT	GetValue	(STRING key)

6.5 Setting values

This action sets (overwrites or creates) the value of a key.

In case the key is read-only, a KEYISREADONLYERROR error is returned.

This action can return a INVALIDVALUEERROR in case the value being set isn't compliant with the schema for the key.

SetValue	(STRING key, value VARIANT)
SetValues	(STRING root, DICT data)

6.6 Getting metainfo

Gets the metainfo of a key. In case no metainfo is available, the default from the schema of the key is returned.

In case there's also no schema an NOSUCHKEYERROR error is returned.

org.freedesktop.configuration.GetMetaInfo

As a method:

metainfo DICT	GetMetaInfo	(STRING key)
---------------	-------------	--------------

6.7 Setting metainfo

This action sets (overwrites or creates) the metainfo of a key.

In case the key is read-only, a KEYISREADONLYERROR error is returned.

org.freedesktop.configuration.SetMetaInfo

As a method:

SetMetaInfo	(STRING key, metainfo DICT metainfo)
-------------	--------------------------------------

6.8 Subscribing to notification of changes

A client uses the signal arguments to get informed about specific KeyChanged keys.

```
"type='signal', "  
"interface='org.freedesktop.configuration', "  
"member='KeyChanged', arg0='/the/key'"
```

6.9 Receiving notification of changes

This signal happens (on the client) if the client is subscribed to a key that was changed.

org.freedesktop.configuration.KeyChanged This is a signal:

```
KeyChanged          (STRING key, VARIANT data, UINT32 next)
```

Signal arguments:

```
arg0                (STRING key)
```

6.10 Removing

This action removes all keys starting from the root.

org.freedesktop.configuration.RemoveKeys

As a method:

```
RemoveKeys          (STRING root)
```