

## **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 if they store personal configuration data sometimes also called the preferences of the user.

This standard is about 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 the D-BUS standard and the 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 client versus service protocol. The service delivers the information and that performs requests coming from the client.

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

The desktop application is the client. The implementation of this standard is the service.

## 4 Schemas

This standard defines the schema DTD

### 4.1 The purpose

A schema has the purpose of making the system type safe. Schemas also provide the description and default value of a key.

### 4.2 File naming conventions

Schemas are installed by applications and or packages. Each such application should install one schemas file which contains all the schemas the application uses and that aren't shared with existing applications.

The filename is formed by replacing the slash characters with underscores. Between the root namespace and the application name there's two underscores. The root namespace is a namespace that is known to be unique for your organisation. See below for more information.

The filename is appended with .schemas.

The directory where the schemas files are located is "PREFIX/share/configuration" on a UNIX system and "Drive://Program Files/Common Files/configuration" on a Windows system

Some examples:

- /usr/share/configuration/org\_gnome\_\_nautilus.schemas for keys in /org/gnome/nautilus/
- /usr/share/configuration/org\_kde\_\_konqueror.schemas for keys in /org/kde/konqueror/

### 4.3 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 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).

### 4.4 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 are allowed.

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

Some examples:

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

#### **4.5 Item naming conventions**

A item is one setting stored somewhere inside your root namespace. Once inside your root namespace you are free to choose your own naming scheme.

All characters with exception of the slash are allowed.

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

#### **4.6 Shareable items**

A committee decides about the naming of shareable items.

As a application developer you can to 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

## 4.7 Overview of the DTD

The root of the document contains schema nodes

```
<!ELEMENT schemas (schema*)>
```

For each key the application uses, you create a schema node. A schema node contains the nodes type, default and at least one description node. It optionally also contains a min and a max node.

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

The schema node has two optional attributes:

- `root=/org/namespace` holds the root namespace of your organisation
- `item=preference key` holds the key to one setting

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

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

Each schema node has a type element.

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

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

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

The real attribute is for high level use. The application developer (or high level library writer) can use this field freely to define composed types

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

Both min and max are optional nodes in the schema node. For integer types they'll enforce a minimum and a maximum on the value. They can also be freely used for composed types. `<!ELEMENT min (#PCDATA)>`

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

The default node can contain or 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 arg node per supported language.

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

The character data of an arg node is the long description of the key translated in the language as defined by the language attribute (see below).

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

The language attribute holds the language of this translated description.

```
<!ATTLIST arg language CDATA #REQUIRED>
```

The short attribute holds the short description translated in the language as defined by the language attribute. `<!ATTLIST arg short CDATA #REQUIRED>`

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

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

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

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

<!ELEMENT description (arg*)>
<!ELEMENT arg (#PCDATA)>
  <!ATTLIST arg language CDATA #REQUIRED>
  <!ATTLIST arg short CDATA #REQUIRED>
```

## 4.9 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 -->
<schema root="/sample_namespace/sample_application/" item="prefs/my_string">
<type dbus="s" real="string"/><default>Default string</default></schema>
<schema root="/sample_namespace/sample_application/" item="prefs/my_integer">
<type dbus="i" real="integer"/><default>20</default></schema>
<schema root="/sample_namespace/sample_application/" item="prefs/my_double">
<type dbus="d" real="double"/><default>20.99</default></schema>
<schema root="/sample_namespace/sample_application/" item="prefs/my_boolean">
<type dbus="b" real="boolean"/><default>0</default></schema>
<!-- Simple lists -->
<schema root="/sample_namespace/sample_application/" item="prefs/my_boolean list">
<type dbus="ab" real="boolean list"/>
<default>
<schema><type dbus="b" real="boolean list"/><default>0</default></schema>
</default>
</schema>
<schema root="/sample_namespace/sample_application/" item="prefs/my_string list">
<type dbus="as" real="string list"/>
<default>
<schema><type dbus="s" real="string"/><default>Default string</default></schema>
</default>
</schema>
<schema root="/sample_namespace/sample_application/" item="prefs/my_integer list">
<type dbus="ai" real="integer list"/>
<default>
<schema><type dbus="i" real="integer"/><default>10</default></schema>
</default>
</schema>
<schema root="/sample_namespace/sample_application/" item="prefs/my_double list">
<type dbus="ad" real="double list"/>
<default>
<schema><type dbus="d" real="double"/><default>10.99</default></schema>
</default>
</schema>
<!-- Samples of possible complex types -->
<!-- Font -->
<schema root="/sample_namespace/sample_application/" item="prefs/my_font">
<type dbus="si" real="font"/>
<default>
<schema><type dbus="s" real="font_name"/><default>Arial</default></schema>
<schema><type dbus="i" real="font_size"/>
<min>8</min><max>32</max><default>12</default></schema>
</default>
</schema>
<!-- A rectangle -->
<schema root="/sample_namespace/sample_application/" item="prefs/my_rect">
<type dbus="iiii" real="rect"/>
<default>
<schema><type dbus="i" real="rect_x1"/>
<min>0</min><max>100</max><default>1</default></schema>
<schema><type dbus="i" real="rect_y1"/>
<min>0</min><max>100</max><default>10</default></schema>
<schema><type dbus="i" real="rect_x2"/>
<min>0</min><max>100</max><default>10</default></schema>
<schema><type dbus="i" real="rect_y2"/>
<min>0</min><max>100</max><default>1</default></schema>
</default>
</schema>
<!-- A color -->
<schema root="/sample_namespace/sample_application/" item="prefs/my_color">
<type dbus="iii" real="color"/>
<default>
<schema><type dbus="i" real="color_R"/>
<min>0</min><max>255</max><default>110</default></schema>
<schema><type dbus="i" real="color_G"/>
<min>0</min><max>255</max><default>120</default></schema>
<schema><type dbus="i" real="color_B"/>
<min>0</min><max>255</max><default>130</default></schema>
</default>
</schema>
</schemas>
```

## 5 The protocol

### 5.1 The errors

An error reply has the messagetype `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):

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

### 5.2 The types

#### value `VARIANT`

D-BUS signature: `v`

can be:

<b>D-BUS Type</b>	<b>description</b>	<b>or</b>
<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</code> of value <code>VARIANT</code>	the list data	

#### The `METAINFO DICT`

D-BUS signature: `a{sv}`

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

#### metainfo `VARIANT`

D-BUS signature: `v`

can be:

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

### 5.3 Special types

#### The value `DICT` for `GetValues`

D-BUS signature: `a{sv}`

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

## 5.4 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:

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

## 5.5 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 an INVALIDVALUEERROR in case the value being set isn't compliant with the schema for the key.

SetValue	(STRING key, value VARIANT)
----------	-----------------------------

## 5.6 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 DICTIONARY	GetMetaInfo	(STRING key)
---------------------	-------------	--------------

## 5.7 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)

## 5.8 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'"
```

## 5.9 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)

## 5.10 Removing

This action removes all keys starting from the root.

### org.freedesktop.configuration.RemoveKeys

As a method:

RemoveKeys (STRING root)