

# 共享的咪咪信息数据库

## X 桌面组

### 托马斯·伦纳德

<[tall97\\_at\\_users.sf.net](mailto:tall97_at_users.sf.net)>

---

## 目录

### [介绍](#)

[版本](#)

[这个规格是什么？](#)

[本规范中使用的语言](#)

### [统一系统](#)

[目录布局](#)

[源 XML 文件](#)

[介质/子类型.xml文件](#)

[格洛布文件](#)

[神奇的文件](#)

[XM 语言空间文件](#)

[图标文件](#)

[树魔术文件](#)

[哑剧缓存文件](#)

[使用扩展属性存储 MIME 类型](#)

[子类化](#)

[建议的检查顺序](#)

[非常规文件](#)

[卷的内容类型](#)

[URI 方案处理程序](#)

[安全隐患](#)

[用户修改](#)

### [贡献](#)

### [引用](#)

## 介绍

### 版本

这是共享 MIME 信息数据库规范的 0.21 版，最后更新于 2018 年 10 月 2 日。

### 这个规格是什么？

许多程序和桌面使用 MIME 系统 [MIME] 来表示文件类型。通常，有必要为文件确定正确的 MIME 类型。这通常是通过检查文件的名称或内容，并在数据库中查找正确的 MIME 类型来完成的。

存储有关每种类型的信息（如其文本说明）或可用于查看或编辑该类型的文件的应用程序列表也很有用。

对于互操作性，不同的程序使用相同的数据库非常有用，这样不同的程序就可以就文件类型达成一致，并且不会重复信息。对于应用程序作者来说，只需在一个位置安装新信息也很有帮助。

该规范试图统一目前由 GNOME[GNOME]、KDE[KDE] 和 ROX[ROX] 使用的 MIME 数据库系统，并为未来的可扩展性提供空间。

MIME 数据库不存储用户首选项（如用户用于处理特定类型文件的首选应用程序）。它可用于存储静态信息，例如可以使用特定应用程序查看特定类型的文件。

## 本规范中使用的语言

本文档中的关键词“必须”、“不得”、“必需”、“应”、“不应”、“应”、“不应”、“推荐”、“可能”和“可选”应按 RFC 2119[RFC-2119] 中所述进行解释。

## 统一系统

在讨论 GNOME、KDE 和 ROX 之前使用的系统时（参见“历史和相关系统”文档），很明显，数据库之间的差异只是它们分开的结果，而不是由于开发人员之间的任何根本分歧。每个人都渴望看到它们合并。

该规范建议：

- 应用程序安装新的 MIME 相关信息的标准方法。
- 获取文件的 MIME 类型的标准方法。
- 获取有关 MIME 类型的信息的标准方法。
- 所有文件的标准位置以及解决冲突的方法。

此外，现有的数据库已合并到一个包 [共享MIME] 中。

## 目录布局

对于 MIME 数据库的存储方式，有两个重要要求：

- 应用程序必须能够在安装时以任何方式扩展数据库，以添加用于确定类型的新规则和有关特定类型的新信息。
- 必须能够在 /usr、/usr/本地和用户的主目录中安装应用程序（以正常的 Unix 方式），并使用 MIME 信息。

此规范使用 XDG 基本目录规范 [BaseDir] 来定义存储数据库的前缀。在本文档的其余部分中，带有前缀的路径指示应从 中每个目录的子目录加载文件。<MIME>mimeXDG\_DATA\_HOMEXDG\_DATA\_DIRS

例如，使用默认路径时，“加载所有<MIME>/文本/html.xml文件”意味着加载、和（如果存在，并按此顺序）。在目录中找到的信息将添加到以前目录中找到的信息中，除非使用“全局删除”或“魔术删除”来覆盖 mimetype 定义的部分。

分。/usr/share/mime/text/html.xml/usr/local/share/mime/text/html.xml~/.local/share/mime/text/html.xml

每个希望参与 MIME 数据库的应用程序都将以应用程序命名的单个 XML 文件安装到三个目录之一中（具体取决于用户请求安装应用程序的位置）。安装、卸载或修改此文件后，应用程序必须运行 **更新-mime-数据库** 命令，该命令由 freedesktop.org 共享数据库 [\[SharedMIME\]](#) 提供。

<MIME>/packages/

**更新-mime-数据库** 被传递包含子目录的目录，该子目录被修改为其唯一参数。它扫描子目录中的所有 XML 文件，合并其中的信息，并创建许多输出文件。mimepackagespackages

如果来自这些文件的信息发生冲突，则来自列表下方目录的信息优先。任何命名的文件都优先于同一目录中的所有其他文件。这可以通过允许用户编辑数据库的工具来使用，以确保用户的更改生效。Override.xmlpackages

由**更新哑剧数据库**创建的文件是：

- <MIME>/globs （包含从名称到 MIME 类型的映射）[已弃用于球体2]
- <MIME>/globs2 （包含从名称到 MIME 类型和球体权重的映射）
- <MIME>/magic (contains a mapping from file contents to MIME types)
- <MIME>/subclasses (contains a mapping from MIME types to types they inherit from)
- <MIME>/aliases (contains a mapping from aliases to MIME types)
- <MIME>/icons (contains a mapping from MIME types to icons)
- <MIME>/generic-icons (contains a mapping from MIME types to generic icons)
- <MIME>/XMLnamespaces (contains a mapping from XML (namespaceURI, localName) pairs to MIME types)
- <MIME>/MEDIA/SUBTYPE.xml (one file for each MIME type, giving details about the type, including comment, icon and generic-icon)
- <MIME>/mime.cache (contains the same information as the , , , , , and files, in a binary, mmappable format) globs2magicssubclassesaliasesiconsgeneric-iconsXMLnamespaces

The format of these generated files and the source files in are explained in the following sections. This step serves several purposes. First, it allows applications to quickly get the data they need without parsing all the source XML files (the base package alone is over 700K). Second, it allows the database to be used for other purposes (such as creating the file if desired). Third, it allows validation to be performed on the input data, and removes the need for other applications to carefully check the input for errors themselves.

packages/etc/mime.types

## The source XML files

Each application provides only a single XML source file, which is installed in the directory as described above. This file is an XML file whose document element is named `mime-info` and whose namespace URI is <http://www.freedesktop.org/standards/shared-mime-info>. All elements described in this specification MUST have this namespace too. packages

The document element may contain zero or more `mime-type` child nodes, in any order, each describing a single MIME type. Each element has a `type` attribute giving the MIME type that

it describes.

Each `mime-type` node may contain any combination of the following elements, and in any order:

- `glob` elements have a `pattern` attribute. Any file whose name matches this pattern will be given this MIME type (subject to conflicting rules in other files, of course). There is also an optional `weight` attribute which is used when resolving conflicts with other `glob` matches. The default weight value is 50, and the maximum is 100.

KDE's `glob` system replaces GNOME's and ROX's `ext/regex` fields, since it is trivial to detect a pattern in the form `'*.ext'` and store it in an extension hash table internally. The full power of regular expressions was not being used by either desktop, and `glob` patterns are more suitable for filename matching anyway.

The first `glob` element represents the "main" extension for the file type. While this doesn't affect the `mimetype` matching algorithm, this information can be useful when a single main extension is needed for a `mimetype`, for instance so that applications can choose an appropriate extension when saving a file.

- A `glob-deleteall` element, which indicates that patterns from previously parsed directories must be discarded. The patterns defined in this file (if any) are used instead.
- `magic` elements contain a list of `match` elements, any of which may match, and an optional `priority` attribute for all of the contained rules. Low numbers should be used for more generic types (such as 'gzip compressed data') and higher values for specific subtypes (such as a word processor format that happens to use gzip to compress the file). The default priority value is 50, and the maximum is 100.

Each `match` element has a number of attributes:

Attribute	Required?	Value
<code>type</code>	Yes	<code>string</code> , <code>host16</code> , <code>host32</code> , <code>big16</code> , <code>big32</code> , <code>little16</code> , <code>little32</code> or <code>byte</code> .
<code>offset</code>	Yes	The byte offset(s) in the file to check. This may be a single number or a range in the form <code>`start:end'</code> , indicating that all offsets in the range should be checked. The range is inclusive.
<code>value</code>	Yes	The value to compare the file contents with, in the format indicated by the <code>type</code> attribute. The <code>string</code> type supports the C character escapes ( <code>\0</code> , <code>\t</code> , <code>\n</code> , <code>\r</code> , <code>\xAB</code> for hex, <code>\777</code> for octal).
<code>mask</code>	No	The number to AND the value in the file with before comparing it to <code>`value'</code> . Masks for numerical types can be any number, while masks for strings must be in base 16, and start with <code>0x</code> .

Each element corresponds to one line of `file(1)`'s file. They can be nested in the same way to provide the equivalent of continuation lines. That is, `<a><b/><c/></a>` means 'a and (b or c)'. `magic.mime`

- A `magic-deleteall` element, which indicates that magic matches from previously parsed directories must be discarded. The magic defined in this file (if any) is used

instead.

- `alias` elements indicate that the type is also sometimes known by another name, given by the `type` attribute. For example, `audio/midi` has an alias of `audio/x-midi`. Note that there should not be a `mime-type` element defining each alias; a single element defines the canonical name for the type and lists all its aliases.
- `sub-class-of` elements indicate that any data of this type is also some other type, given by the `type` attribute. See [the section called "Subclassing"](#).
- `comment` elements give a human-readable textual description of the MIME type, usually composed of an acronym of the file name extension and a short description, like "ODS spreadsheet". There may be many of these elements with different `xml:lang` attributes to provide the text in multiple languages.
- `acronym` elements give experienced users a terse idea of the document contents. for example "ODS", "GEDCOM", "JPEG" and "XML". There may be many of these elements with different `xml:lang` attributes to provide the text in multiple languages, although these should only be used if absolutely necessary.
- `expanded-acronym` elements are the expanded versions of the acronym elements, for example "OpenDocument Spreadsheet", "GEnealogical Data COMMunication", and "eXtensible Markup Language". The purpose of these elements is to provide users a way to look up information on various MIME types or file formats in third-party resources. There may be many of these elements with different `xml:lang` attributes to provide the text in multiple languages, although these should only be used if absolutely necessary.
- `icon` elements specify the icon to be used for this particular mime-type, given by the `name` attribute. Generally the icon used for a mimetype is created based on the mime-type by mapping "/" characters to "-", but users can override this by using the `icon` element to customize the icon for a particular mimetype. This element is not used in the system database, but only used in the user overridden database. Only one `icon` element is allowed.
- `generic-icon` elements specify the icon to use as a generic icon for this particular mime-type, given by the `name` attribute. This is used if there is no specific icon (see `icon` for how these are found). These are used for categories of similar types (like spreadsheets or archives) that can use a common icon. The Icon Naming Specification lists a set of such icon names. If this element is not specified then the mimetype is used to generate the generic icon by using the top-level media type (e.g. "video" in "video/ogg") and appending "-x-generic" (i.e. "video-x-generic" in the previous example). Only one `generic-icon` element is allowed.
- `root-XML` elements have `namespaceURI` and `localName` attributes. If a file is identified as being an XML file, these rules allow a more specific MIME type to be chosen based on the namespace and localname of the document element.

If `localName` is present but empty then the document element may have any name, but the namespace must still match.

- `treemagic` elements contain a list of `treematch` elements, any of which may match, and an optional `priority` attribute for all of the contained rules. The default priority value is 50, and the maximum is 100.

Each `treematch` element has a number of attributes:

Attribute	Required?	Value
path	Yes	A path that must be present on the mounted volume/filesystem. The path is interpreted as a relative path starting at the root of the tested volume/filesystem
type	No	The type of path. Possible values: <code>file</code> , <code>directory</code> , <code>link</code>
match-case	No	Whether path should be matched case-sensitively. Possible values: <code>true</code> , <code>false</code>
executable	No	Whether the file must be executable. Possible values: <code>true</code> , <code>false</code>
non-empty	No	Whether the directory must be non-empty. Possible values: <code>true</code> , <code>false</code>
mimetype	No	The mimetype for the file at path

`treematch` elements can be nested, meaning that both the outer and the inner `treematch` must be satisfied for a "match".

Applications may also define their own elements, provided they are namespaced to prevent collisions. Unknown elements are copied directly to the output XML files like `comment` elements. A typical use for this would be to indicate the default handler application for a particular desktop ("Galeon is the GNOME default text/html browser"). Note that this doesn't indicate the user's preferred application, only the (fixed) default.

Here is an example source file, named : `diff.xml`

```
<?xml version="1.0"?>
<mime-info xmlns='http://www.freedesktop.org/standards/shared-mime-info'>
  <mime-type type="text/x-diff">
    <comment>Differences between files</comment>
    <comment xml:lang="af">verskille tussen lêers</comment>
    ...
    <magic priority="50">
      <match type="string" offset="0" value="diff\t"/>
      <match type="string" offset="0" value="***\t"/>
      <match type="string" offset="0" value="Common subdirectories: "/>
    </magic>
    <glob pattern="*.diff"/>
    <glob pattern="*.patch"/>
  </mime-type>
</mime-info>
```

In practice, common types such as `text/x-diff` are provided by the `freedesktop.org` shared database. Also, only new information needs to be provided, since this information will be merged with other information about the same type.

### The MEDIA/SUBTYPE.xml files

These files have a `mime-type` element as the root node. The format is as described above. They are created by merging all the `mime-type` elements from the source files and creating

one output file per MIME type. Each file may contain information from multiple source files. The `magic`, `glob` and `root-XML` elements will have been removed.

The example source file given above would (on its own) create an output file called containing the following: `<MIME>/text/x-diff.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<mime-type xmlns="http://www.freedesktop.org/standards/shared-mime-info" type="text/x-diff">
<!--Created automatically by update-mime-database. DO NOT EDIT!-->
  <comment>Differences between files</comment>
  <comment xml:lang="af">verskille tussen lêers</comment>
  ...
</mime-type>
```

## The glob files

The `globs2` file is a simple list of lines containing weight, MIME type and pattern, separated by a colon. The lines are ordered by glob weight. For example:

```
# This file was automatically generated by the
# update-mime-database command. DO NOT EDIT!
...
55:text/x-diff:*.patch
50:text/x-diff:*.diff
50:text/x-c++src:*.C:cs
...
```

The `glob` file is a simple list of lines containing a MIME type and pattern, separated by a colon. It is deprecated in favour of the `globs2` file which also lists the weight of the glob rule. The lines are ordered by glob weight. For example:

```
# This file was automatically generated by the
# update-mime-database command. DO NOT EDIT!
...
text/x-diff:*.patch
text/x-diff:*.diff
...
```

Applications **MUST** match globs case-insensitively, except when the case-sensitive attribute is set to true. This is so that e.g. will be seen as a C++ file, but will still use the `*.gif` pattern. `main.CIMAGE.GIF`

If several patterns of the same weight match then the longest pattern **SHOULD** be used. In particular, files with multiple extensions (such as `data.tar.gz`) **MUST** match the longest sequence of extensions (eg `*.tar.gz` in preference to `*.gz`). Literal patterns (eg, 'Makefile') must be matched before all others. It is suggested that patterns beginning with `*.` and containing no other special characters (`*?[]`) should be placed in a hash table for efficient lookup, since this covers the majority of the patterns. Thus, patterns of this form should be matched before other wildcarded patterns. `Data.tar.gz`

If a matching pattern is provided by two or more MIME types, applications **SHOULD** not rely on one of them. They are instead supposed to use magic data (see below) to detect the actual MIME type. This is for instance required to deal with container formats like Ogg or AVI, that map various video and/or audio-encoded data to one extension.

There may be several rules mapping to the same type. They should all be merged. If the same pattern is defined twice, then they **MUST** be ordered by the directory the rule came

from, as described above.

The `glob-deleteall` element, which means that implementations SHOULD discard information from previous directories, is written out into the `globs2` file using `__NOGLOBS__` as the pattern. For instance:

```
0:text/x-diff:__NOGLOBS__
50:text/x-diff:*.diff
...
```

In the above example, the mimetype `text/x-diff` is redefined (for instance in a user's `~/.local/share/mime`) to only be associated with the pattern `*.diff`, so the other patterns like `*.patch` were removed. The weight in front of the `__NOGLOBS__` line is ignored. In a given `globs2` file, the `__NOGLOBS__` line for a given mimetype is always written out before any other globs for this mimetype.

Lines beginning with ``#'` are comments and should be ignored. Everything from the ``:'` character to the newline is part of the pattern; spaces should not be stripped. The file is in the UTF-8 encoding. The format of the glob pattern is as for `fnmatch(3)`. The format does not allow a pattern to contain a literal newline character, but this is not expected to be a problem.

Common types (such as MS Word Documents) will be provided in the X Desktop Group's package, which MUST be required by all applications using this specification. Since each application will then only be providing information about its own types, conflicts should be rare.

The fourth field ("`cs`" in the first `globs2` example) contains a list of comma-separated flags. The flags currently defined are: `cs` (for case-sensitive). Implementations should ignore unknown flags.

Implementations should also ignore further fields, so that the syntax of the `globs2` file can be extended in the future. Example: `"50:text/x-c++src:*.C:cs,newflag:newfeature:somethingelse"` should currently be parsed as `"50:text/x-c++src:*.C:cs"`.

## The magic files

The magic data is stored in a binary format for ease of parsing. The old magic database had complex escaping rules; these are now handled by **update-mime-database**.

The file starts with the magic string `"MIME-Magic\0\n"`. There is no version number in the file. Incompatible changes will be handled by creating both the current ``magic'` file and a newer ``magic2'` in the new format. Where possible, compatible changes only will be made. All numbers are big-endian, so need to be byte-swapped on little-endian machines.

The rest of the file is made up of a sequence of small sections. Each section is introduced by giving the priority and type in brackets, followed by a newline character. Higher priority entries come first. Example:

```
[50:text/x-diff]\n
```

Each line in the section takes the form:



```
[ indent ] ">" start-offset "=" value
[ "&" mask ] [ "~" word-size ] [ "+" range-length ] "\n"
```

Part	Example	Meaning
indent	1	The nesting depth of the rule, corresponding to the number of '>' characters in the traditional file format.
">" start-offset	>4	The offset into the file to look for a match.
"=" value	=\0x0\0x2\0x55\0x40	Two bytes giving the (big-endian) length of the value, followed by the value itself.
"&" mask	&\0xff\0xf0	The mask, which (if present) is exactly the same length as the value.
"~" word-size	~2	On little-endian machines, the size of each group to byte-swap.
"+" range-length	+8	The length of the region in the file to check.

Note that the value, value length and mask are all binary, whereas everything else is textual. Each of the elements begins with a single character to identify it, except for the indent level.

The word size is used for byte-swapping. Little-endian systems should reverse the order of groups of bytes in the value and mask if this is greater than one. This only affects 'host' matches ('big32' entries still have a word size of 1, for example, because no swapping is necessary, whereas 'host32' has a word size of 4).

The indent, range-length, word-size and mask components are optional. If missing, indent defaults to 0, range-length to 1, the word-size to 1, and the mask to all 'one' bits.

Indent corresponds to the nesting depth of the rule. Top-level rules have an indent of zero. The parent of an entry is the preceding entry with an indent one less than the entry.

If an unknown character is found where a newline is expected then the whole line should be ignored (there will be no binary data after the new character, so the next line starts after the next "\n" character). This is for future extensions.

The text/x-diff above example would (on its own) create this magic file:

```
00000000 4d 49 4d 45 2d 4d 61 67 69 63 00 0a 5b 35 30 3a |MIME-Magic.. [50:|
00000010 74 65 78 74 2f 78 2d 64 69 66 66 5d 0a 3e 30 3d |text/x-diff].>0=|
00000020 00 05 64 69 66 66 09 0a 3e 30 3d 00 04 2a 2a 2a |..diff..>0=..***|
00000030 09 0a 3e 30 3d 00 17 43 6f 6d 6d 6f 6e 20 73 75 |..>0=..Common su|
00000040 62 64 69 72 65 63 74 6f 72 69 65 73 3a 20 0a |bdirectories: .|
```

The magic-deleteall attribute, which means that implementations SHOULD discard information from previous directories, is written out into the magic file using \_\_NOMAGIC\_\_ as the value:

```
>0=__NOMAGIC__\n
```

This can be followed by other magic rules for the mimetype.

## The XMLnamespaces files

Each file is a list of lines in the form: XMLnamespaces

namespaceURI " " localName " " MIME-Type "\n"

For example:

http://www.w3.org/1999/xhtml html application/xhtml+xml

The lines are sorted (using strcmp in the C locale) and there are no lines with the same namespaceURI and localName in one file. If the localName was empty then there will be two spaces following the namespaceURI.

## The icon files

The and files are list of lines in the form: iconsgeneric-icons

MIME-Type ":" icon-name "\n"

For example:

application/msword:x-office-document

## The treemagic files

The tree magic data is stored in a file with a format that is very similar to the magic file format.

The file starts with the magic string "MIME-TreeMagic\0\n". There is no version number in the file. Incompatible changes will be handled by creating both the current `treemagic' and a newer `treemagic2' in the new format. Where possible, changes will be made in a compatible fashion.

The rest of the file is made up of a sequence of small sections. Each section is introduced by giving the priority and type in brackets, followed by a newline character. Higher priority entries come first. Example:

[50:x-content/image-dcf]\n

Each line in the section takes the form:

[ indent ] ">" "\" path "\" "=" type [ ",", option ]\* "\n"

Part	Meaning
indent	The nesting depth of the rule.
path	The path to match.
type	The required file type, one of "file", "directory", "link" or "any"
option	Optional for the optional attributes of <code>treematch</code> elements. Possible values are "executable", "match-case", "non-empty", or a MIME type

## The mime.cache files

The files contain the same information as the `mime.types` and `mime.types.gz` files, in a binary, mmappable format:  
`mime.cacheglobs2magicsubclassesaliasesXMLnamespaces`

Header:		
2	CARD16	MAJOR_VERSION 1
2	CARD16	MINOR_VERSION 2
4	CARD32	ALIAS_LIST_OFFSET
4	CARD32	PARENT_LIST_OFFSET
4	CARD32	LITERAL_LIST_OFFSET
4	CARD32	REVERSE_SUFFIX_TREE_OFFSET
4	CARD32	GLOB_LIST_OFFSET
4	CARD32	MAGIC_LIST_OFFSET
4	CARD32	NAMESPACE_LIST_OFFSET
4	CARD32	ICONS_LIST_OFFSET
4	CARD32	GENERIC_ICONS_LIST_OFFSET
AliasList:		
4	CARD32	N_ALIASES
8*N_ALIASES	AliasListEntry	
AliasListEntry:		
4	CARD32	ALIAS_OFFSET
4	CARD32	MIME_TYPE_OFFSET
ParentList:		
4	CARD32	N_ENTRIES
8*N_ENTRIES	ParentListEntry	
ParentListEntry:		
4	CARD32	MIME_TYPE_OFFSET
4	CARD32	PARENTS_OFFSET
Parents:		
4	CARD32	N_PARENTS
4*N_PARENTS	CARD32	MIME_TYPE_OFFSET
LiteralList:		
4	CARD32	N_LITERALS
12*N_LITERALS	LiteralEntry	
LiteralEntry:		
4	CARD32	LITERAL_OFFSET
4	CARD32	MIME_TYPE_OFFSET
4	CARD32	WEIGHT in lower 8 bits
		FLAGS in rest:
		0x100 = case-sensitive
GlobList:		
4	CARD32	N_GLOBS
12*N_GLOBS	GlobEntry	
GlobEntry:		
4	CARD32	GLOB_OFFSET
4	CARD32	MIME_TYPE_OFFSET
4	CARD32	WEIGHT in lower 8 bits
		FLAGS in rest:
		0x100 = case-sensitive
ReverseSuffixTree:		
4	CARD32	N_ROOTS
4	CARD32	FIRST_ROOT_OFFSET
ReverseSuffixTreeNode:		
4	CARD32	CHARACTER
4	CARD32	N_CHILDREN
4	CARD32	FIRST_CHILD_OFFSET

4	CARD32	0
4	CARD32	MIME_TYPE_OFFSET
4	CARD32	WEIGHT in lower 8 bits
		FLAGS in rest:
		0x100 = case-sensitive
MagicList:		
4	CARD32	N_MATCHES
4	CARD32	MAX_EXTENT
4	CARD32	FIRST_MATCH_OFFSET
Match:		
4	CARD32	PRIORITY
4	CARD32	MIME_TYPE_OFFSET
4	CARD32	N_MATCHLETS
4	CARD32	FIRST_MATCHLET_OFFSET
Matchlet:		
4	CARD32	RANGE_START
4	CARD32	RANGE_LENGTH
4	CARD32	WORD_SIZE
4	CARD32	VALUE_LENGTH
4	CARD32	VALUE_OFFSET
4	CARD32	MASK_OFFSET (0 if no mask)
4	CARD32	N_CHILDREN
4	CARD32	FIRST_CHILD_OFFSET
NamespaceList:		
4	CARD32	N_NAMESPACES
12*N_NAMESPACES	NamespaceEntry	
NamespaceEntry:		
4	CARD32	NAMESPACE_URI_OFFSET
4	CARD32	LOCAL_NAME_OFFSET
4	CARD32	MIME_TYPE_OFFSET
GenericIconsList:		
IconsList:		
4	CARD32	N_ICONS
8*N_ICONS	IconListEntry	
IconListEntry:		
4	CARD32	MIME_TYPE_OFFSET
4	CARD32	ICON_NAME_OFFSET

Lists in the file are sorted, to enable binary searching. The list of aliases is sorted by alias, the list of literal globs is sorted by the literal. The SuffixTreeNode siblings are sorted by character. The list of namespaces is sorted by namespace uri. The list of icons is sorted by mimetype.

Mimetypes are stored in the suffix tree by appending suffix tree leaf nodes with '\0' as character. These nodes appear at the beginning of the list of children.

All offsets are in bytes from the beginning of the file.

Strings are zero-terminated.

All numbers are in network (big-endian) order. This is necessary because the data will be stored in arch-independent directories like or even in user's home directories. /usr/share/mime

Cache files have to be written atomically - write to a temporary name, then move over the old file - so that clients that have the old cache file open and mmap'ed won't get corrupt data.

## Storing the MIME type using Extended Attributes

An implementation MAY also get a file's MIME type from the `user.mime_type` extended attribute. The type given here should normally be used in preference to any guessed type, since the user is able to set it explicitly. Applications MAY choose to set the type when saving files. Since many applications and filesystems do not support extended attributes, implementations MUST NOT rely on this method being available.

## Subclassing

A type is a subclass of another type if any instance of the first type is also an instance of the second. For example, all `image/svg+xml` files are also `application/xml`, `text/plain` and `application/octet-stream` files. Subclassing is about the format, rather than the category of the data (for example, there is no 'generic spreadsheet' class that all spreadsheets inherit from).

Some subclass rules are implicit:

- All `text/*` types are subclasses of `text/plain`.
- All streamable types (ie, everything except the `inode/*` types) are subclasses of `application/octet-stream`.

In addition to these rules, explicit subclass information may be given using the `sub-class-of` element.

Note that some file formats are also compressed files (`application/x-jar` files are also `application/zip` files). However, this is different to a case such as a compressed postscript file, which is not a valid postscript file itself (so `application/x-gzpostscript` does not inherit from `application/postscript`, because an application that can handle the latter may not cope with the former).

Some types may or may not be instances of other types. For example, a spreadsheet file may be compressed or not. It is a valid spreadsheet file either way, but only inherits from `application/gzip` in one case. This information cannot be represented statically; instead an application interested in this information should run all of the magic rules, and use the list of types returned as the subclasses.

## Recommended checking order

Because different applications have different requirements, they may choose to use the various methods provided by this specification in any order. However, the RECOMMENDED order to perform the checks is:

- If a MIME type is provided explicitly (eg, by a `ContentType` HTTP header, a MIME email attachment, an extended attribute or some other means) then that should be used instead of guessing.
- Otherwise, start by doing a glob match of the filename. Keep only globs with the biggest weight. If the patterns are different, keep only globs with the longest pattern,

as previously discussed. If after this, there is one or more matching glob, and all the matching globs result in the same mimetype, use that mimetype as the result.

- If the glob matching fails or results in multiple conflicting mimetypes, read the contents of the file and do magic sniffing on it. If no magic rule matches the data (or if the content is not available), use the default type of application/octet-stream for binary data, or text/plain for textual data. If there was no glob match, use the magic match as the result.

Note: Checking the first 128 bytes of the file for ASCII control characters is a good way to guess whether a file is binary or text, but note that files with high-bit-set characters should still be treated as text since these can appear in UTF-8 text, unlike control characters.

- If any of the mimetypes resulting from a glob match is equal to or a subclass of the result from the magic sniffing, use this as the result. This allows us for example to distinguish text files called "foo.doc" from MS-Word files with the same name, as the magic match for the MS-Word file would be application/x-ole-storage which the MS-Word type inherits.
- Otherwise use the result of the glob match that has the highest weight.

There are several reasons for checking the glob patterns before the magic. First of all doing magic sniffing is very expensive as reading the contents of the files causes a lot of seeks, which is very expensive. Secondly, some applications don't check the magic at all (sometimes the content is not available or too slow to read), and this makes it more likely that both will get the same type.

Also, users can easily understand why calling their text file makes the system think it's an MP3, whereas they have trouble understanding why their computer thinks is a PostScript file. If the system guesses wrongly, the user can often rename the file to fix the problem.

README. mp3README. txt

## Non-regular files

Sometimes it is useful to assign MIME types to other objects in the filesystem, such as directories, sockets and device files. This could be useful when looking up an icon for a type, or for providing a textual description of one of these objects. The media type 'inode' is provided for this purpose, with the following types corresponding to the standard types of object found in a Unix filesystem:

inode/blockdevice  
inode/chardevice  
inode/directory  
inode/fifo  
inode/mount-point  
inode/socket  
inode/symlink

An inode/mount-point is a subclass of inode/directory. It can be useful when adding extra actions for these directories, such as 'mount' or 'eject'. Mounted directories can be detected by comparing the 'st\_dev' of a directory with that of its parent. If they differ, they are from different devices and the directory is a mount point.

## Content types for volumes

Traditional MIME types apply to individual files or bytestreams. It is often useful to apply the same methodologies when classifying the content of mountable volumes or filesystems. The x-content type has been introduced for this purpose. Typical examples are x-content/audio-dvd, x-content/blank-cd or x-content/image-dcf.

Matching of content types works with `treemagic` elements, which are analogous to the `magic` elements used for MIME type matching. Instead of looking for byte sequences in files, `treemagic` elements allow to look for files with certain names, permissions or mime types in a directory hierarchy.

## URI scheme handlers

URI scheme handling (such as a movie player handling `mms://` URIs, or a Podcast program handling `feed://` URIs) are handled through applications handling the `x-scheme-handler/foo` mime-type, where `foo` is the URI scheme in question.

This scheme allows URI scheme handling to enjoy the same benefits as mime-type handlers, such as the ability to change the default handler, the cross-desktop support, and easier application launching.

Note that this virtual mime-type is not for listing URI schemes that an application can load files from. For example, a movie player would not list `x-scheme-handler/http` in its supported mime-types, but it would list `x-scheme-handler/rtsp` if it supported playing back from RTSP locations.

## Security implications

The system described in this document is intended to allow different programs to see the same file as having the same type. This is to help interoperability. The type determined in this way is only a guess, and an application **MUST NOT** trust a file based simply on its MIME type. For example, a downloader should not pass a file directly to a launcher application without confirmation simply because the type looks 'harmless' (eg, `text/plain`).

Do not rely on two applications getting the same type for the same file, even if they both use this system. The spec allows some leeway in implementation, and in any case the programs may be following different versions of the spec.

## User modification

The MIME database is **NOT** intended to store user preferences. Users should never edit the database. If they wish to make corrections or provide MIME entries for software that doesn't provide these itself, they should do so by means of the `Override.xml` mentioned in [the section called "Directory layout"](#). Information such as "text/html files need to be opened with Mozilla" should **NOT** go in the database.

# Contributors

Thomas Leonard <[tall97@users.sf.net](mailto:tall97@users.sf.net)>

David Faure <[faure@kde.org](mailto:faure@kde.org)>

Alex Larsson <[alexl@redhat.com](mailto:alexl@redhat.com)>

Seth Nickell <[snickell@stanford.edu](mailto:snickell@stanford.edu)>

Keith Packard <[keithp@keithp.com](mailto:keithp@keithp.com)>

Filip Van Raemdonck <[mechanix@debian.org](mailto:mechanix@debian.org)>

Christos Zoulas <[christos@zoulas.com](mailto:christos@zoulas.com)>

Matthias Clasen <[mclasen@redhat.com](mailto:mclasen@redhat.com)>

Bastien Nocera <[hadess@hadess.net](mailto:hadess@hadess.net)>

# References

[GNOME] *The GNOME desktop*, <http://www.gnome.org>

[KDE] *The KDE desktop*, <http://www.kde.org>

[ROX] *The ROX desktop*, <http://rox.sourceforge.net>

[DesktopEntries] *Desktop Entry Specification*,  
<http://www.freedesktop.org/standards/desktop-entry-spec.html>

[SharedMIME] *Shared MIME-info Database*, <http://www.freedesktop.org/standards/shared-mime-info.html>

[RFC-2119] *Key words for use in RFCs to Indicate Requirement Levels*,  
<http://www.ietf.org/rfc/rfc2119.txt?number=2119>

[BaseDir] *XDG Base Directory Specification*  
<http://www.freedesktop.org/standards/basedir/draft/basedir-spec/basedir-spec.html>

[ACAP] *ACAP Media Type Dataset Class* <http://ftp.ietf.org/internet-drafts/draft-ietf-acap-mediatype-01.txt>