# Icon Theme Specification

Alexander Larsson Frans Englich ⋮ 22-27 minutes

## Overview

An icon theme is a set of icons that share a common look and feel. The user can then select the icon theme that they want to use, and all apps use icons from the theme. The initial user of icon themes is the icon field of the desktop file specification, but in the future it can have other uses (such as mimetype icons).

From a programmer perspective an icon theme is just a mapping. Given a set of directories to look for icons in and a theme name it maps from icon name and nominal icon size to an icon filename.

## Definitions

Icon Theme

> An icon theme is a named set of icons. It is used to map from an iconname and size to a file. Themes may inherit from other themes as a way to extend them.

Icon file

> An icon file is an image that can be loaded and used as an icon. The supported image file formats are PNG, XPM and SVG. PNG is the recommended bitmap format, and SVG is for vectorized icons. XPM is supported due to backwards compability reasons, and it is not recommended that new themes use XPM files. Support for SVGs is optional.

Base Directory

> Icons and themes are searched for in a set of directories, called base directories. The themes are stored in subdirectories of the base directories.

Icon scale

> On very high density (high dpi) screens the UI is often scaled to avoid the UI being so small it is hard to see. In order to support this icons can have a target scale, describing what scale factor they are designed for.
>
> For instance, an icon with a directory size of 48 but scale 2x would be 96x96 pixels, but designed to have the same level of detail as a 48x48 icon at scale 1x. This can be used on high density displays where a 48x48 icon would be too small (or ugly upscaled) and a normal 96x96 icon would have a lot of detail that is hard to see.

## Directory Layout

Icons and themes are looked for in a set of directories. By default, apps should look in $HOME/.icons (for backwards compatibility), in $XDG_DATA_DIRS/icons and in /usr/share/pixmaps (in that order). Applications may further add their own icon directories to this list, and users may extend or change the list (in application/desktop specific ways).In each of these directories themes are stored as subdirectories. A theme can be spread across several base directories by having subdirectories of the same name. This way users can extend and override system themes.

In order to have a place for third party applications to install their icons there should always exist a theme called "hicolor" [1]. The data for the hicolor theme is available for download at: http://www.freedesktop.org/software/icon-theme/. Implementations are required to look in the "hicolor" theme if an icon was not found in the current theme.

Each theme is stored as subdirectories of the base directories. The internal name of the theme is the name of the subdirectory, although the user-visible name as specified by the theme may be different. Hence, theme names are case sensitive, and are limited to ASCII characters. Theme names may also not contain comma or space.

In at least one of the theme directories there must be a file called index.theme that describes the theme. The first index.theme found while searching the base directories in order is used. This file describes the general attributes of the theme.

In the theme directory are also a set of subdirectories containing image files. Each directory contains icons designed for a certain nominal icon size and scale, as described by the index.theme file. The subdirectories are allowed to be several levels deep, e.g. the subdirectory "48x48/apps" in the theme "hicolor" would end up at $basedir/hicolor/48x48/apps.

The image files must be one of the types: PNG, XPM, or SVG, and the extension must be ".png", ".xpm", or ".svg" (lower case). The support for SVG files is optional. Implementations that do not support SVGs should just ignore any ".svg" files. In addition to this there may be an additional file with extra icon-data for each file. It should have the same basename as the image file, with the extension ".icon". e.g. if the icon file is called "mime_source_c.png" the corresponding file would be named "mime_source_c.icon".

## File Formats

Both the icon theme description file and the icon data files are ini-style text files, as described in the desktop file specification. They don't have any encoding field. Instead, they must always be stored in UTF-8 encoding.

The index.theme file must start with a section called *Icon Theme*, with contents according to table 1 below. All lists are comma-separated.

**Table 1. Standard Keys**

| Key | Description | Value Type | Required |
|---|---|---|---|
| Name | short name of the icon theme, used in e.g. lists when selecting themes. | localestring | YES |
| Comment | longer string describing the theme | localestring | YES |
| Inherits | The name of the theme that this theme inherits from. If an icon name is not found in the current theme, it is searched for in the inherited theme (and recursively in all the inherited themes).<br>If no theme is specified implementations are required to add the "hicolor" theme to the inheritance tree. An implementation may optionally add other default themes in between the last specified theme and the hicolor theme. | strings | NO |
| Directories | list of subdirectories for this theme. For every subdirectory there must be a section in the index.theme file describing that directory. | strings | YES |
| ScaledDirectories | Additional list of subdirectories for this theme, in addition to the ones in Directories. These directories should only be read by implementations supporting scaled directories and was added to keep compatibility with old implementations that don't support these. | strings | NO |
| Hidden | Whether to hide the theme in a theme selection user interface. This is used for things such as fallback-themes that are not supposed to be visible to the user. | boolean | NO |
| Example | The name of an icon that should be used as an example of how this theme looks. | string | NO |

Each directory specified in the Directory key has a corresponding section with the same name as the directory. The contents of this section is listed in table 2 below.

**Table 2. Per-Directory Keys**

| Key | Description | Value Type | Required | Type |
|---|---|---|---|---|
| Size | Nominal (unscaled) size of the icons in this directory. | integer | YES | |
| Scale | Target scale of of the icons in this directory. Defaults to the value 1 if not present. Any directory with a scale other than 1 should be listed in the ScaledDirectories list rather than Directories for backwards compatibility. | integer | NO | |
| Context | The context the icon is normally used in. This is in detail discussed in the section called “Context”. | string | NO | |
| Type | The type of icon sizes for the icons in this directory. Valid types are Fixed, Scalable and Threshold. The type decides what other keys in the section are used. If not specified, the default is Threshold. | string | NO | |
| MaxSize | Specifies the maximum (unscaled) size that the icons in this directory can be scaled to. Defaults to the value of Size if not present. | integer | NO | Scalable |
| MinSize | Specifies the minimum (unscaled) size that the icons in this directory can be scaled to. Defaults to the value of Size if not present. | integer | NO | Scalable |
| Threshold | The icons in this directory can be used if the size differ at most this much from the desired (unscaled) size. Defaults to 2 if not present. | integer | NO | Threshold |

In addition to these groups you may add extra groups to the index.theme file in order to extend it. These extensions must begin with "X-", and can be used to add desktop specific information to the theme file. Example group names would be "X-KDE Icon Theme" or "X-Gnome Icon Theme".

The optional filename.icon file contains a group called "Icon Data", with the content listed in table 3.

**Table 3. Icon Data Keys**

| Key | Description | Value Type | Required |
|---|---|---|---|
| DisplayName | A translated UTF8 string that can be used instead of the icon name when the icon is listed in e.g. a user interface. | localestring | NO |
| EmbeddedTextRectangle | If this exists, it specifies the four corners of a rectangle where the program displaying the icon can embed text. This is normally used by e.g. file managers that want to display a preview of text file contents in the icon. The corners are specified by a list of four values: x0,y0,x1,y1. The values are pixel coordinates from the top left corner of the | integers | NO |

| Key | Description | Value Type | Required |
|---|---|---|---|
| | icon, except for SVG files, where they are specified in a 1000x1000 coordinate space that is scaled to the final rendered size of the icon. | | |
| AttachPoints | A list of points, separated by "\|" that may be used as anchor points for emblems/overlays. The points are pixel coordinates from the top left corner of the icon, except for SVG files, where they are specified in a 1000x1000 coordinate space that is scaled to the final rendered size of the icon. | points | NO |

Extensions to the filename.icon file are allowed, but the keys must be begin with "X-" to avoid collisions with future standardized extensions to this format.

### Context

The `Context` allows the designer to group icons on a conceptual level. It doesn't act as a namespace in the file system, such that icons can have identical names, but allows implementations to categorize and sort by it, for example.

These are the available contexts:

- **Actions.** Icons representing actions which the user initiates, such as Save As.
- **Devices.** Icons representing real world devices, such as printers and mice. It's not for file system nodes such as character or block devices.
- **FileSystems.** Icons for objects which are represented as part of the file system. This is for example, the local network, “Home”, and “Desktop” folders.
- **MimeTypes.** Icons representing MIME types.

## Icon Lookup

The icon lookup mechanism has two global settings, the list of base directories and the internal name of the current theme. Given these we need to specify how to look up an icon file from the icon name, the nominal size and the scale.

The lookup is done first in the current theme, and then recursively in each of the current theme's parents, and finally in the default theme called "hicolor" (implementations may add more default themes before "hicolor", but "hicolor" must be last). As soon as there is an icon of any size that matches in a theme, the search is stopped. Even if there may be an icon with a size closer to the correct one in an inherited theme, we don't want to use it. Doing so may generate an inconsistant change in an icon when you change icon sizes (e.g. zoom in).

The lookup inside a theme is done in three phases. First all the directories are scanned for an exact match, e.g. one where the allowed size of the icon files match what was looked up. Then all the directories are scanned for any icon that matches the name. If that fails we finally fall back on unthemed icons. If we fail to find any icon at all it is up to the application to pick a good fallback, as the correct choice depends on the context.

The exact algorithm (in pseudocode) for looking up an icon in a theme (if the implementation supports SVG) is:

```
FindIcon(icon, size, scale) {
  filename = FindIconHelper(icon, size, scale, user selected theme);
  if filename != none
    return filename

  filename = FindIconHelper(icon, size, scale, "hicolor");
  if filename != none
    return filename

  return LookupFallbackIcon (icon)
}
FindIconHelper(icon, size, scale, theme) {
  filename = LookupIcon (icon, size, scale, theme)
  if filename != none
    return filename

  if theme has parents
    parents = theme.parents

  for parent in parents {
    filename = FindIconHelper (icon, size, scale, parent)
    if filename != none
      return filename
  }
  return none
}
```

With the following helper functions:

```
LookupIcon (iconname, size, scale, theme) {
  for each subdir in $(theme subdir list) {
    for each directory in $(basename list) {
      for extension in ("png", "svg", "xpm") {
        if DirectoryMatchesSize(subdir, size, scale) {
          filename = directory/$(themename)/subdir/iconname.extension
          if exist filename
            return filename
        }
      }
    }
  }
  minimal_size = MAXINT
  for each subdir in $(theme subdir list) {
    for each directory in $(basename list) {
      for extension in ("png", "svg", "xpm") {
        filename = directory/$(themename)/subdir/iconname.extension
        if exist filename and DirectorySizeDistance(subdir, size, scale) < minimal_size {
           closest_filename = filename
           minimal_size = DirectorySizeDistance(subdir, size, scale)
        }
      }
    }
  }
  if closest_filename set
     return closest_filename
  return none
}

LookupFallbackIcon (iconname) {
  for each directory in $(basename list) {
    for extension in ("png", "svg", "xpm") {
      if exists directory/iconname.extension
        return directory/iconname.extension
    }
  }
  return none
}

DirectoryMatchesSize(subdir, iconsize, iconscale) {
  read Type and size data from subdir
  if Scale != iconscale
     return False;
  if Type is Fixed
    return Size == iconsize
  if Type is Scaled
    return MinSize <= iconsize <= MaxSize
  if Type is Threshold
    return Size - Threshold <= iconsize <= Size + Threshold
}

DirectorySizeDistance(subdir, iconsize, iconscale) {
  read Type and size data from subdir
  if Type is Fixed
    return abs(Size*Scale - iconsize*iconscale)
  if Type is Scaled
    if iconsize*iconscale < MinSize*Scale
        return MinSize*Scale - iconsize*iconscale
    if iconsize*iconscale > MaxSize*Scale
        return iconsize*iconscale - MaxSize*Scale
    return 0
  if Type is Threshold
    if iconsize*iconscale < (Size - Threshold)*Scale
        return MinSize*Scale - iconsize*iconscale
```

```
    if iconsize*iconsize > (Size + Threshold)*Scale
        return iconsize*iconsize - MaxSize*Scale
    return 0
}
```

In some cases you don't always want to fall back to an icon in an inherited theme. For instance, sometimes you look for a set of icons, prefering any of them before using an icon from an inherited theme. To support such operations implementations can contain a function that finds the first of a list of icon names in the inheritance hierarchy. I.E. It would look something like this:

```
FindBestIcon(iconList, size, scale) {
  filename = FindBestIconHelper(iconList, size, scale, user selected theme);
  if filename != none
    return filename

  filename = FindBestIconHelper(iconList, size, scale, "hicolor");
  if filename != none
    return filename

  for icon in iconList {
    filename = LookupFallbackIcon (icon)
    if filename != none
      return filename
  }
  return none;
}
FindBestIconHelper(iconList, size, scale, theme) {
  for icon in iconList {
    filename = LookupIcon (icon, size, theme)
    if filename != none
      return filename
  }

  if theme has parents
    parents = theme.parents

  for parent in parents {
    filename = FindBestIconHelper (iconList, size, scale, parent)
    if filename != none
      return filename
  }
  return none
}
```

This can be very useful for example when handling mimetype icons, where there are more and less "specific" versions of icons.

## Example

Here is an example index.theme file:

```
[Icon Theme]
Name=Birch
Name[sv]=Björk
Comment=Icon theme with a wooden look
Comment[sv]=Träinspirerat ikontema
Inherits=wood,default
Directories=48x48/apps,48x48@2/apps48x48/mimetypes,32x32/apps,32x32@2/apps,scalable/apps,scalable/mimetypes

[scalable/apps]
Size=48
Type=Scalable
MinSize=1
MaxSize=256
Context=Applications

[scalable/mimetypes]
Size=48
```

```
Type=Scalable
MinSize=1
MaxSize=256
Context=MimeTypes

[32x32/apps]
Size=32
Type=Fixed
Context=Applications

[32x32@2/apps]
Size=32
Scale=2
Type=Fixed
Context=Applications

[48x48/apps]
Size=48
Type=Fixed
Context=Applications

[48x48@2/apps]
Size=48
Scale=2
Type=Fixed
Context=Applications

[48x48/mimetypes]
Size=48
Type=Fixed
Context=MimeTypes
```

The corresponding directory tree in the /usr/share/icons directory could look like this:

```
birch/index.theme
birch/scalable/apps/mozilla.svg
birch/scalable/mimetypes/mime_text_plain.svg
birch/scalable/mimetypes/mime_text_plain.icon
birch/48x48/apps/mozilla.png
birch/48x48@2/apps/mozilla.png
birch/32x32/apps/mozilla.png
birch/32x32@2/apps/mozilla.png
birch/48x48/mimetypes/mime_text_plain.png
birch/48x48/mimetypes/mime_text_plain.icon
```

Where birch/scalable/mimetypes/mime_text_plain.icon contains:

```
[Icon Data]
DisplayName=Mime text/plain
EmbeddedTextRectangle=100,100,900,900
AttachPoints=200,200|800,200|500,500|200,800|800,800
```

And birch/48x48/mimetypes/mime_text_plain.icon contains:

```
[Icon Data]
DisplayName=Mime text/plain
EmbeddedTextRectangle=8,8,40,40
AttachPoints=20,20|40,40|50,10|10,50
```

In this example a lookup of "mozilla" would get the prerendered 48x48 and 32x32 icons before the SVG icons due to the order of Directories.

## Installing Application Icons

So, you're an application author, and want to install application icons so that they work in the KDE and Gnome menus. Minimally you should install a 48x48 icon in the hicolor theme. This means installing a PNG file in $prefix/share/icons/hicolor/48x48/apps. Optionally you can install icons in

different sizes. For example, installing a svg icon in $prefix/share/icons/hicolor/scalable/apps means most desktops will have one icon that works for all sizes. You might even want to install icons with a look that matches other well known themes so your application will fit in with some specific desktop environment.

It is recommended that the icons installed in the hicolor theme look neutral, since it is a fallback theme that will be used in combination with some very different looking themes. But if you don't have any neutral icon, please install whatever icon you have in the hicolor theme so that all applications get at least some icon in all themes.

## Implementation Notes

The algorithm as described in this document works by always looking up filenames in directories (a stat in unix terminology). A good implementation is expected to read the directories once, and do all lookups in memory using that information.

This caching can make it impossible for users to add icons without having to restart applications. In order to handle this, any implementation that does caching is required to look at the mtime of the toplevel icon directories when doing a cache lookup, unless it already did so less than 5 seconds ago. This means that any icon editor or theme installation program need only to change the mtime of the the toplevel directory where it changed the theme to make sure that the new icons will eventually get used.

## Background

The icon theme specification is based on the original KDE icon theme system designed by Antonio Larossa, Geert Janssen and Torsten Rahn. The common specification mostly adds support for .icon files, renames the icon theme description files and removes a few references to kde in them.

## A. Change history

**Version 0.13, July 2 2013, Alexander Larsson.**

- Add icon Scales.

**Version 0.12, 24 December 2006, Octavio Alvarez.**

- Fixed "hicolor" lookup in the pseudocode, so it works with multiple parents.

**Version 0.11, 7 February 2006, Alexander Larsson.**

- Fixed icon lookup clarification to work with multiple inheritance.

**Version 0.10, 7 February 2006, Alexander Larsson.**

- Clarify that icon lookup looks in all parent themes before falling back to nonthemed icons.
- Added lookup function that takes a list of icon names (FindBestIcon)

**Version 0.9, 4 April 2005, Alexander Larsson.**

- Cleanups and fixes to language from Rodney Dawes and Frans Englich.
- Added section describing Contexts in more details (by Frans Englich).

**Version 0.8, 5 February 2004, Alexander Larsson.**

- Fix language problems as pointed out by Rodney Dawes and Michael Terry.
- Added background section.

**Version 0.7, 13 September 2003, Heinrich Wendel.**

- Converted to basedir spec.
- Changed type of MaxSize, MinSize and Threshold to integer.
- Removed typo in code example.
- Corrected path to default-icon-theme.

**Version 0.6, 2 December 2002, Alexander Larsson.**

- Added Hidden key.
- Removed multiple inheritance.
- Renamed the default theme hicolor.
- Added the application icon install section.
- Fixed some xml issues.

**Version 0.5, 18 September 2002, Alexander Larsson.**

- Added DisplayName to icon data.
- Fixed up example svg icon data.

- Fixed some spelling and grammar errors.

**Version 0.4, 16 May 2002, Alexander Larsson.**

- Fixed some spelling and grammar errors.

**Version 0.3, 14 May 2002, Alexander Larsson.**

- Made support for SVGs optional.
- Added a default fallback theme.
- Changed the example directory layout a bit to match the default theme.

**Version 0.2, 29 April 2002, Alexander Larsson.**

- Changed search order to png, svg, xpm.
- Added comment to say that xpm is supported for backwards compat and not recommended in new themes.
- Default Type for a directory is now Threshold
- Added implementation notes section.
- Added Example key.

**Version 0.1, 22 April 2002, Alexander Larsson.**

- Created initial draft.