Qt Documentation

QFileInfo Class

The QFileInfo class provides system-independent file information. More...

Header:	#include <QFileInfo>
qmake:	QT += core

Note: All functions in this class are reentrant.

- List of all members, including inherited members
- Obsolete members

Public Functions

	QFileInfo()
	QFileInfo(const QString & *file*)
	QFileInfo(const QFile & *file*)
	QFileInfo(const QDir & *dir*, const QString & *file*)
	QFileInfo(const QFileInfo & *fileinfo*)
	~QFileInfo()
QDir	**absoluteDir**() const
QString	**absoluteFilePath**() const
QString	**absolutePath**() const
QString	**baseName**() const
QString	**bundleName**() const
bool	**caching**() const
QString	**canonicalFilePath**() const
QString	**canonicalPath**() const

QString	**completeBaseName**() const
QString	**completeSuffix**() const
QDateTime	**created**() const
QDir	**dir**() const
bool	**exists**() const
QString	**fileName**() const
QString	**filePath**() const
QString	**group**() const
uint	**groupId**() const
bool	**isAbsolute**() const
bool	**isBundle**() const
bool	**isDir**() const
bool	**isExecutable**() const
bool	**isFile**() const
bool	**isHidden**() const
bool	**isNativePath**() const
bool	**isReadable**() const
bool	**isRelative**() const
bool	**isRoot**() const
bool	**isSymLink**() const
bool	**isWritable**() const
QDateTime	**lastModified**() const
QDateTime	**lastRead**() const
bool	**makeAbsolute**()
QString	**owner**() const
uint	**ownerId**() const
QString	**path**() const
bool	**permission**(QFile::Permissions *permissions*) const
QFile::Permissions	**permissions**() const
void	**refresh**()
void	**setCaching**(bool *enable*)
void	**setFile**(const QString & *file*)
void	**setFile**(const QFile & *file*)
void	**setFile**(const QDir & *dir*, const QString & *file*)
qint64	**size**() const
QString	**suffix**() const
void	**swap**(QFileInfo & *other*)
QString	**symLinkTarget**() const
bool	**operator!=**(const QFileInfo & *fileinfo*) const
QFileInfo &	**operator=**(const QFileInfo & *fileinfo*)
QFileInfo &	**operator=**(QFileInfo && *other*)
bool	**operator==**(const QFileInfo & *fileinfo*) const

Static Public Members

bool	**exists**(const QString & *file*)

Related Non-Members

typedef	**QFileInfoList**

Detailed Description

The QFileInfo class provides system-independent file information.

QFileInfo provides information about a file's name and position (path) in the file system, its access rights and whether it is a directory or symbolic link, etc. The file's size and last modified/read times are also available. QFileInfo can also be used to obtain information about a Qt resource.

A QFileInfo can point to a file with either a relative or an absolute file path. Absolute file paths begin with the directory separator "/" (or with a drive specification on Windows). Relative file names begin with a directory name or a file name and specify a path relative to the current working directory. An example of an absolute path is the string "/tmp/quartz". A relative path might look like "src/fatlib". You can use the function isRelative() to check whether a QFileInfo is using a relative or an absolute file path. You can call the function makeAbsolute() to convert a relative QFileInfo's path to an absolute path.

The file that the QFileInfo works on is set in the constructor or later with setFile(). Use exists() to see if the file exists and size() to get its size.

The file's type is obtained with isFile(), isDir() and isSymLink(). The symLinkTarget() function provides the name of the file the symlink points to.

On Unix (including OS X and iOS), the symlink has the same size() has the file it points to, because Unix handles symlinks transparently; similarly, opening a symlink using QFile effectively opens the link's target. For example:

```
#ifdef Q_OS_UNIX

QFileInfo info1("/home/bob/bin/untabify");
info1.isSymLink();          // returns true
info1.absoluteFilePath();   // returns "/home/bob/bin/untabify"
info1.size();               // returns 56201
info1.symLinkTarget();      // returns "/opt/pretty++/bin/untabify"

QFileInfo info2(info1.symLinkTarget());
info2.isSymLink();          // returns false
info2.absoluteFilePath();   // returns "/opt/pretty++/bin/untabify"
info2.size();               // returns 56201

#endif
```

On Windows, symlinks (shortcuts) are `.lnk` files. The reported size() is that of the symlink (not the link's target), and opening a symlink using QFile opens the `.lnk` file. For example:

```
#ifdef Q_OS_WIN

QFileInfo info1("C:\\Documents and Settings\\Bob\\untabify.lnk");
info1.isSymLink();           // returns true
info1.absoluteFilePath();    // returns "C:/Documents and Settings/Bob/untabify.lnk"
info1.size();                // returns 743
info1.symLinkTarget();       // returns "C:/Pretty++/untabify"

QFileInfo info2(info1.symLinkTarget());
info2.isSymLink();           // returns false
info2.absoluteFilePath();    // returns "C:/Pretty++/untabify"
info2.size();                // returns 63942

#endif
```

Elements of the file's name can be extracted with path() and fileName(). The fileName()'s parts can be extracted with baseName(), suffix() or completeSuffix(). QFileInfo objects to directories created by Qt classes will not have a trailing file separator. If you wish to use trailing separators in your own file info objects, just append one to the file name given to the constructors or setFile().

The file's dates are returned by created(), lastModified() and lastRead(). Information about the file's access permissions is obtained with isReadable(), isWritable() and isExecutable(). The file's ownership is available from owner(), ownerId(), group() and groupId(). You can examine a file's permissions and ownership in a single statement using the permission() function.

Note: On NTFS file systems, ownership and permissions checking is disabled by default for performance reasons. To enable it, include the following line:

```
extern Q_CORE_EXPORT int qt_ntfs_permission_lookup;
```

Permission checking is then turned on and off by incrementing and decrementing `qt_ntfs_permission_lookup` by 1.

```
qt_ntfs_permission_lookup++; // turn checking on
qt_ntfs_permission_lookup--; // turn it off again
```

Performance Issues

Some of QFileInfo's functions query the file system, but for performance reasons, some functions only operate on the file name itself. For example: To return the absolute path of a relative file name, absolutePath() has to query the file system. The path() function, however, can work on the file name directly, and so it is faster.

Note: To speed up performance, QFileInfo caches information about the file.

To speed up performance, QFileInfo caches information about the file. Because files can be changed by other users or programs, or even by other parts of the same program, there is a function that refreshes the file information: refresh(). If you want to switch off a QFileInfo's caching and force it to access the file system every time you request information from it call setCaching(false).

See also QDir and QFile.

Member Function Documentation

QFileInfo::QFileInfo()

Constructs an empty QFileInfo object.

Note that an empty QFileInfo object contain no file reference.

See also setFile().

QFileInfo::QFileInfo(const QString & *file*)

Constructs a new QFileInfo that gives information about the given file. The *file* can also include an absolute or relative path.

See also setFile(), isRelative(), QDir::setCurrent(), and QDir::isRelativePath().

QFileInfo::QFileInfo(const QFile & *file*)

Constructs a new QFileInfo that gives information about file *file*.

If the *file* has a relative path, the QFileInfo will also have a relative path.

See also isRelative().

QFileInfo::QFileInfo(const QDir & *dir*, const QString & *file*)

Constructs a new QFileInfo that gives information about the given *file* in the directory *dir*.

If *dir* has a relative path, the QFileInfo will also have a relative path.

If *file* is an absolute path, then the directory specified by *dir* will be disregarded.

See also isRelative().

QFileInfo::QFileInfo(const QFileInfo & *fileinfo*)

Constructs a new QFileInfo that is a copy of the given *fileinfo*.

QFileInfo::~QFileInfo()

Destroys the QFileInfo and frees its resources.

QDir QFileInfo::absoluteDir() const

Returns the file's absolute path as a QDir object.

See also dir(), filePath(), fileName(), and isRelative().

QString QFileInfo::absoluteFilePath() const

Returns an absolute path including the file name.

The absolute path name consists of the full path and the file name. On Unix this will always begin with the root, '/', directory. On Windows this will always begin 'D:/' where D is a drive letter, except for network shares that are not mapped to a drive letter, in which case the path will begin '//sharename/'. QFileInfo will uppercase drive letters. Note that QDir does not do this. The code snippet below shows this.

```
    QFileInfo fi("c:/temp/foo"); => fi.absoluteFilePath() => "C:/temp/foo"
```

This function returns the same as filePath(), unless isRelative() is true. In contrast to canonicalFilePath(), symbolic links or redundant "." or ".." elements are not necessarily removed.

Warning: If filePath() is empty the behavior of this function is undefined.

See also filePath(), canonicalFilePath(), and isRelative().

QString QFileInfo::absolutePath() const

Returns a file's path absolute path. This doesn't include the file name.

On Unix the absolute path will always begin with the root, '/', directory. On Windows this will always begin 'D:/' where D is a drive letter, except for network shares that are not mapped to a drive letter, in which case the path will begin '//sharename/'.

In contrast to canonicalPath() symbolic links or redundant "." or ".." elements are not necessarily removed.

Warning: If filePath() is empty the behavior of this function is undefined.

See also absoluteFilePath(), path(), canonicalPath(), fileName(), and isRelative().

QString QFileInfo::baseName() const

Returns the base name of the file without the path.

The base name consists of all characters in the file up to (but not including) the *first* '.' character.

Example:

```
QFileInfo fi("/tmp/archive.tar.gz");
QString base = fi.baseName();  // base = "archive"
```

The base name of a file is computed equally on all platforms, independent of file naming conventions (e.g., ".bashrc" on Unix has an empty base name, and the suffix is "bashrc").

See also fileName(), suffix(), completeSuffix(), and completeBaseName().

QString QFileInfo::bundleName() const

Returns the name of the bundle.

On OS X and iOS this returns the proper localized name for a bundle if the path isBundle(). On all other platforms an empty QString is returned.

Example:

```
QFileInfo fi("/Applications/Safari.app");
QString bundle = fi.bundleName();                // name = "Safari"
```

This function was introduced in Qt 4.3.

See also isBundle(), filePath(), baseName(), and suffix().

bool QFileInfo::caching() const

Returns `true` if caching is enabled; otherwise returns `false`.

See also setCaching() and refresh().

QString QFileInfo::canonicalFilePath() const

Returns the canonical path including the file name, i.e. an absolute path without symbolic links or redundant "." or ".." elements.

If the file does not exist, canonicalFilePath() returns an empty string.

See also filePath(), absoluteFilePath(), and dir().

QString QFileInfo::canonicalPath() const

Returns the file's path canonical path (excluding the file name), i.e. an absolute path without symbolic links or redundant "." or ".." elements.

If the file does not exist, canonicalPath() returns an empty string.

See also path() and absolutePath().

QString QFileInfo::completeBaseName() const

Returns the complete base name of the file without the path.

The complete base name consists of all characters in the file up to (but not including) the *last* '.' character.

Example:

```
QFileInfo fi("/tmp/archive.tar.gz");
QString base = fi.completeBaseName();  // base = "archive.tar"
```

See also fileName(), suffix(), completeSuffix(), and baseName().

QString QFileInfo::completeSuffix() const

Returns the complete suffix of the file.

The complete suffix consists of all characters in the file after (but not including) the first '.'.

Example:

```
QFileInfo fi("/tmp/archive.tar.gz");
QString ext = fi.completeSuffix();  // ext = "tar.gz"
```

See also fileName(), suffix(), baseName(), and completeBaseName().

QDateTime QFileInfo::created() const

Returns the date and time when the file was created.

On most Unix systems, this function returns the time of the last status change. A status change occurs when the file is created, but it also occurs whenever the user writes or sets inode information (for example, changing the file permissions).

If neither creation time nor "last status change" time are not available, returns the same as lastModified().

See also lastModified() and lastRead().

QDir QFileInfo::dir() const

Returns the path of the object's parent directory as a QDir object.

Note: The QDir returned always corresponds to the object's parent directory, even if the QFileInfo represents a directory.

For each of the following, dir() returns a QDir for `"~/examples/191697"`.

```
    QFileInfo fileInfo1("~/examples/191697/.");
    QFileInfo fileInfo2("~/examples/191697/..");
    QFileInfo fileInfo3("~/examples/191697/main.cpp");
```

For each of the following, dir() returns a QDir for `"."`.

```
    QFileInfo fileInfo4(".");
    QFileInfo fileInfo5("..");
    QFileInfo fileInfo6("main.cpp");
```

See also absolutePath(), filePath(), fileName(), isRelative(), and absoluteDir().

bool QFileInfo::exists() const

Returns `true` if the file exists; otherwise returns `false`.

Note: If the file is a symlink that points to a non-existing file, false is returned.

bool QFileInfo::exists(const QString & *file*) `[static]`

Returns `true` if the *file* exists; otherwise returns `false`.

Note: If *file* is a symlink that points to a non-existing file, false is returned.

Note: Using this function is faster than using `QFileInfo(file).exists()` for file system access.

This function was introduced in Qt 5.2.

QString QFileInfo::fileName() const

Returns the name of the file, excluding the path.

Example:

```
QFileInfo fi("/tmp/archive.tar.gz");
QString name = fi.fileName();                // name = "archive.tar.gz"
```

Note that, if this QFileInfo object is given a path ending in a slash, the name of the file is considered empty.

See also isRelative(), filePath(), baseName(), and suffix().

QString QFileInfo::filePath() const

Returns the file name, including the path (which may be absolute or relative).

See also absoluteFilePath(), canonicalFilePath(), and isRelative().

QString QFileInfo::group() const

Returns the group of the file. On Windows, on systems where files do not have groups, or if an error occurs, an empty string is returned.

This function can be time consuming under Unix (in the order of milliseconds).

See also groupId(), owner(), and ownerId().

uint QFileInfo::groupId() const

Returns the id of the group the file belongs to.

On Windows and on systems where files do not have groups this function always returns (uint) -2.

See also group(), owner(), and ownerId().

bool QFileInfo::isAbsolute() const

Returns `true` if the file path name is absolute, otherwise returns false if the path is relative.

See also isRelative().

bool QFileInfo::isBundle() const

Returns `true` if this object points to a bundle or to a symbolic link to a bundle on OS X and iOS; otherwise returns `false`.

This function was introduced in Qt 4.3.

See also isDir(), isSymLink(), and isFile().

bool QFileInfo::isDir() const

Returns `true` if this object points to a directory or to a symbolic link to a directory; otherwise returns `false`.

See also isFile(), isSymLink(), and isBundle().

bool QFileInfo::isExecutable() const

Returns `true` if the file is executable; otherwise returns `false`.

See also isReadable(), isWritable(), and permission().

bool QFileInfo::isFile() const

Returns `true` if this object points to a file or to a symbolic link to a file. Returns `false` if the object points to something which isn't a file, such as a directory.

See also isDir(), isSymLink(), and isBundle().

bool QFileInfo::isHidden() const

Returns `true` if this is a `hidden' file; otherwise returns `false`.

Note: This function returns `true` for the special entries "." and ".." on Unix, even though QDir::entryList threats them as shown.

bool QFileInfo::isNativePath() const

Returns `true` if the file path can be used directly with native APIs. Returns `false` if the file is otherwise supported by a virtual file system inside Qt, such as the Qt Resource System.

Note: Native paths may still require conversion of path separators and character encoding, depending on platform and input requirements of the native API.

This function was introduced in Qt 5.0.

See also QDir::toNativeSeparators(), QFile::encodeName(), filePath(), absoluteFilePath(), and canonicalFilePath().

bool QFileInfo::isReadable() const

Returns `true` if the user can read the file; otherwise returns `false`.

Note: If the NTFS permissions check has not been enabled, the result on Windows will merely reflect whether the file exists.

See also isWritable(), isExecutable(), and permission().

bool QFileInfo::isRelative() const

Returns `true` if the file path name is relative, otherwise returns false if the path is absolute (e.g. under Unix a path is absolute if it begins with a "/").

See also isAbsolute().

bool QFileInfo::isRoot() const

Returns `true` if the object points to a directory or to a symbolic link to a directory, and that directory is the root directory; otherwise returns `false`.

bool QFileInfo::isSymLink() const

Returns `true` if this object points to a symbolic link (or to a shortcut on Windows); otherwise returns `false`.

On Unix (including OS X and iOS), opening a symlink effectively opens the link's target. On Windows, it opens the `.lnk` file itself.

Example:

```
QFileInfo info(fileName);
if (info.isSymLink())
    fileName = info.symLinkTarget();
```

Note: If the symlink points to a non existing file, exists() returns false.

See also isFile(), isDir(), and symLinkTarget().

bool QFileInfo::isWritable() const

Returns `true` if the user can write to the file; otherwise returns `false`.

Note: If the NTFS permissions check has not been enabled, the result on Windows will merely reflect whether the file is marked as Read Only.

See also isReadable(), isExecutable(), and permission().

QDateTime QFileInfo::lastModified() const

Returns the date and time when the file was last modified.

See also created() and lastRead().

QDateTime QFileInfo::lastRead() const

Returns the date and time when the file was last read (accessed).

On platforms where this information is not available, returns the same as lastModified().

See also created() and lastModified().

bool QFileInfo::makeAbsolute()

Converts the file's path to an absolute path if it is not already in that form. Returns `true` to indicate that the path was converted; otherwise returns `false` to indicate that the path was already absolute.

See also filePath() and isRelative().

QString QFileInfo::owner() const

Returns the owner of the file. On systems where files do not have owners, or if an error occurs, an empty string is returned.

This function can be time consuming under Unix (in the order of milliseconds). On Windows, it will return an empty string unless the NTFS permissions check has been enabled.

See also ownerId(), group(), and groupId().

uint QFileInfo::ownerId() const

Returns the id of the owner of the file.

On Windows and on systems where files do not have owners this function returns ((uint) -2).

See also owner(), group(), and groupId().

QString QFileInfo::path() const

Returns the file's path. This doesn't include the file name.

Note that, if this QFileInfo object is given a path ending in a slash, the name of the file is considered empty and this function will return the entire path.

See also filePath(), absolutePath(), canonicalPath(), dir(), fileName(), and isRelative().

bool QFileInfo::permission(QFile::Permissions *permissions*) const

Tests for file permissions. The *permissions* argument can be several flags of type QFile::Permissions OR-ed together to

Tests for file permissions. The *permissions* argument can be several flags of type QFile::Permissions OR-ed together to check for permission combinations.

On systems where files do not have permissions this function always returns `true`.

Note: The result might be inaccurate on Windows if the NTFS permissions check has not been enabled.

Example:

```
QFileInfo fi("/tmp/archive.tar.gz");
if (fi.permission(QFile::WriteUser | QFile::ReadGroup))
    qWarning("I can change the file; my group can read the file");
if (fi.permission(QFile::WriteGroup | QFile::WriteOther))
    qWarning("The group or others can change the file");
```

See also isReadable(), isWritable(), and isExecutable().

QFile::Permissions QFileInfo::permissions() const

Returns the complete OR-ed together combination of QFile::Permissions for the file.

Note: The result might be inaccurate on Windows if the NTFS permissions check has not been enabled.

void QFileInfo::refresh()

Refreshes the information about the file, i.e. reads in information from the file system the next time a cached property is fetched.

Note: On Windows CE, there might be a delay for the file system driver to detect changes on the file.

void QFileInfo::setCaching(bool *enable*)

If *enable* is true, enables caching of file information. If *enable* is false caching is disabled.

When caching is enabled, QFileInfo reads the file information from the file system the first time it's needed, but generally not later.

Caching is enabled by default.

See also refresh() and caching().

void QFileInfo::setFile(const QString & *file*)

Sets the file that the QFileInfo provides information about to *file*.

The *file* can also include an absolute or relative file path. Absolute paths begin with the directory separator (e.g. "/" under Unix) or a drive specification (under Windows). Relative file names begin with a directory name or a file name and specify a path relative to the current directory.

Example:

```
QString absolute = "/local/bin";
```

```
  QString absolute = "/local/bin";
  QString relative = "local/bin";
  QFileInfo absFile(absolute);
  QFileInfo relFile(relative);

  QDir::setCurrent(QDir::rootPath());
  // absFile and relFile now point to the same file

  QDir::setCurrent("/tmp");
  // absFile now points to "/local/bin",
  // while relFile points to "/tmp/local/bin"
```

See also isFile(), isRelative(), QDir::setCurrent(), and QDir::isRelativePath().

void QFileInfo::setFile(const QFile & *file*)

This is an overloaded function.

Sets the file that the QFileInfo provides information about to *file*.

If *file* includes a relative path, the QFileInfo will also have a relative path.

See also isRelative().

void QFileInfo::setFile(const QDir & *dir*, const QString & *file*)

This is an overloaded function.

Sets the file that the QFileInfo provides information about to *file* in directory *dir*.

If *file* includes a relative path, the QFileInfo will also have a relative path.

See also isRelative().

qint64 QFileInfo::size() const

Returns the file size in bytes. If the file does not exist or cannot be fetched, 0 is returned.

See also exists().

QString QFileInfo::suffix() const

Returns the suffix of the file.

The suffix consists of all characters in the file after (but not including) the last '.'.

Example:

```
  QFileInfo fi("/tmp/archive.tar.gz");
  QString ext = fi.suffix();  // ext = "gz"
```

The suffix of a file is computed equally on all platforms, independent of file naming conventions (e.g., ".bashrc" on Unix has an empty base name, and the suffix is "bashrc").

See also fileName(), completeSuffix(), baseName(), and completeBaseName().

void QFileInfo::swap(QFileInfo & *other*)

Swaps this file info with *other*. This function is very fast and never fails.

This function was introduced in Qt 5.0.

QString QFileInfo::symLinkTarget() const

Returns the absolute path to the file or directory a symlink (or shortcut on Windows) points to, or a an empty string if the object isn't a symbolic link.

This name may not represent an existing file; it is only a string. QFileInfo::exists() returns `true` if the symlink points to an existing file.

This function was introduced in Qt 4.2.

See also exists(), isSymLink(), isDir(), and isFile().

bool QFileInfo::operator!=(const QFileInfo & *fileinfo*) const

Returns `true` if this QFileInfo object refers to a different file than the one specified by *fileinfo*; otherwise returns `false`.

See also operator==().

QFileInfo & QFileInfo::operator=(const QFileInfo & *fileinfo*)

Makes a copy of the given *fileinfo* and assigns it to this QFileInfo.

QFileInfo & QFileInfo::operator=(QFileInfo && *other*)

Move-assigns *other* to this QFileInfo instance.

This function was introduced in Qt 5.2.

bool QFileInfo::operator==(const QFileInfo & *fileinfo*) const

Returns `true` if this QFileInfo object refers to a file in the same location as *fileinfo*; otherwise returns `false`.

Note that the result of comparing two empty QFileInfo objects, containing no file references (file paths that do not exist or are empty), is undefined.

Warning: This will not compare two different symbolic links pointing to the same file.

Warning: Long and short file names that refer to the same file on Windows are treated as if they referred to different files.

See also operator!=().

Related Non-Members

typedef QFileInfoList

Synonym for QList<QFileInfo>.

Download

Start for Free
Qt for Application Development
Qt for Device Creation
Qt Open Source
Terms & Conditions
Licensing FAQ

Product

Qt in Use
Qt for Application Development
Qt for Device Creation
Commercial Features
Qt Creator IDE
Qt Quick

Services

Technology Evaluation
Proof of Concept
Design & Implementation
Productization
Qt Training
Partner Network

Developers

Documentation
Examples & Tutorials
Development Tools
Wiki
Forums
Contribute to Qt

About us

Training & Events
Resource Center
News
Careers
Locations
Contact Us

Like 14k

Follow 17K followers

Qt Merchandise Sign In Feedback © 2015 The Qt Company