The Problem

1. Remote files, such as JPEG images, are always opened in web browser, despite the fact that dedicated image viewers such as EOG are perfectly capable of handing http:// and other downloads.
2. On encountering a file unsupported by web browser, it prompt the user if they want to view *or* save the file. The user has to choose between seeing the file just once and not seeing the file right away but being able to view it later. Moreover, they have to choose one of the options before they even know what the file contains.

Proposed Solution #1

Design

Web browser (and desktop handler of read-only URLs such as http://, which is usually set to the web browser) checks mimetype of every file it's requested to display; if there's an application other than web browser associated with this mimetype and it's known to support this design, the file is opened in the "viewer" application instead of the web browser. Otherwise the current behavior is retained.

The application that receives the file detects that it's a remote file being viewed, and displays a "Save" button in the toolbar. Alternatively, it automatically saves the file to a special area of ~/Downloads that's cleaned up as the disk runs out of space, similarly to logrotate.

Implementation

Detecting MIME type

We have three ways of detecting the mime type of a file: by filename (read: extension), by protocol data (e.g. HTTP header), and by content (using shared-mime-info database). There are occasional collisions in detection by file name (is .otf an OpenDocument template or a font file?) and the mimetype in HTTP header may be reported incorrectly, so if mimetypes detected in those two ways don't match, the web browser / url handler downloads the beginning of the file that contains magic data and makes a more educated guess about the mimetype.

Marking apps that support this design

Marking apps that support this design should be done in their respective .desktop file, e.g. "X-elementary-Download-Support=True". The app is not required to actually have networking support; apps with %F in Exec are fine too, as long as they have a way to tell "downloaded" files from local ones (e.g. they can have a separate .desktop with NoDisplay=True, the download support key and an extra command-line argument in Exec that

tells them that the file is downloaded).

Corner Cases / Discussion Items

How do we know if the “viewer” app supports a specific protocol used in the URL? Specifying the list of protocols was considered several times on XDG list but never accepted (although KDE uses X-KDE-Protocols extension that does exactly that).

What happens if a file is passed from a “downloader” app to another app? If the user edits the file in a non-downloader app that’s not aware of the fact that the file it’s given is temporary, the edits will be lost. Shall we make the downloaded files read-only to force saving edited versions elsewhere? Shall passing files to non-downloader apps be allowed at all?

What happens if a downloader app without URL support tries to pass the file to an app with URL support? The app without URL support doesn’t know the original URL, and app with networking support doesn’t have a way to explicitly specify downloading mode for a local file.

Proposed Solution #2

Design

Web browser (and desktop handler of read-only URLs such as http://, which is usually set to the web browser) downloads every file it’s requested to display and then checks its mimetype. If there’s an application other than web browser associated with this mimetype and it’s known to support this design, the file written by the web browser to a user-specific temporary directory (or to a special area of ~/Downloads that’s cleaned up as the disk runs out of space, similarly to logrotate) and is opened in the “viewer” application instead of web browser. Otherwise the current behavior is retained.

The application that receives the file is told by the web browser that it’s a remote or temporary file being viewed, and displays a “Save” button in the toolbar. It’s also supplied with the original file URL, if requested.

Implementation

Detecting MIME type

We have three ways of detecting the mime type of a file: by filename (read: extension), by protocol data (e.g. HTTP header), and by content (using shared-mime-info database). In this case all three can be used.

Marking apps that support this design + passing data to them

The format of passing data to the apps is described in an Exec-like key; it has everything Exec has and two additional keys: `%o` for the original file URI and `%O` for a list of original URLs in case of passing multiple files. If the original URL is not known, `%o` or the respective item in the

`%O` list is an empty string.
For example, if there's an image viewer with the following Exec field in its .desktop file:
`Exec=image-viewer %F`
The proposed TempFileExec field could look like:
`TempFileExec=image-viewer --temporary-file %F`
Or, if the image viewer can use the original URLs somehow (e.g. update the file or tell Zeitgeist where the file came from), the TempFileExec= could look like this:
`TempFileExec=image-viewer --temporary-file %F --sources=%O`
Applications that have this key in .desktop file are assumed to support this design.

Corner Cases / Discussion Items

originally brought up at http://shnatsel.blogspot.com/2012/08/idea-dump-todo-list.html
blueprint: https://blueprints.launchpad.net/elementaryos/+spec/better-remote-file-handling