Free Download Manager 6 Plugins Architecture (API version 9)

Table of Contents

1. Purpose

Plugins are intended to extend the ability of FDM to download useful content from a URL provided by the user.

There are two possible ways to achieve this:

1. By the use of a plugin user will be able to download a resource (e.g. media) which otherwise cannot be downloaded by the user using the standard means of a web browser.

2. Improve the user experience, i.e. simplify the way of downloading things. This is still under consideration.

The current API allows to only extend the ability of FDM to download media content.

2. Basic principles

Plugin should be written using JavaScript. Also, FDM does support launching Python scripts, by passing arbitrary command line arguments to a script and getting console output from it.

Plugin consists of manifest file, icon file and javascript files with the code.

3. Manifest file

Manifest file is a json file. Its file name must be manifest.json. Here is an example:

```
{
  "uuid": "fdm-supremecourt-gov",
  "author": "FreeDownloadManager.ORG",
  "name": "Supreme Court Argument Audio",
  "description": "Provides support for downloading argument audio from supremecourt.gov site.",
  "version": "1.0.0",
  "icon": "icon.png",
  "mediaParser": true,
  "mediaListParser": true,
  "scripts": [
    "msparser.js",
    "msbatchparser.js"
  ],
  "updateUrl": "https://files2.freedownloadmanager.org/updates/plugins/fdm-supremecourt-gov-update.json",
  "minApiVersion": 1,
  "minFeaturesLevel": 1,
}
```

3.1. Manifest json format

- **uuid** – a unique identifier of the plugin. This identifier must be unique across all the plugins.
- **author** – this plugin author's name.
- **name** – plugin's name.
- **description** – short description of the plugin.
- **version** – string with version. It's also used when checking for plugin's updates.
- **icon** – name of a file with icon. SVG, PNG, etc. formats are supported.
- **mediaListParser** – boolean value indicating if the plugin can be used to download playlists.
- **mediaParser** - boolean value indicating if the plugin can be used to download single media content.
- **scripts** – array of strings with the names of javascript files the plugin consists of.
- **updateUrl** – URL of json file with the information about the latest version of this plugin. Only HTTPS URLs are accepted. Its format is described below.
- **dependencies** – information about additional components with plugin depends on.
- **permissions** – list of strings indicating the special permissions this plugin requires for its work.
- **minApiVersion** – the minimum API version plugin does support.
- **minFeaturesLevel** – the minimum set of the required app features the plugin requires.
- **targetApiVersion** – API version for which the add-on was developed and tested. Defaults to 5 if not specified.
- **allowedPaths** – list of additional directories on the disk the add-on requires read-only access to.

3.1.1. Dependencies section

The are two components supported for now: Python and Deno.

Here is an example of this section in manifest.json file:

```
"dependencies": {
   "Python": {"minVersion": "3.0"}
  }
```

minVersion parameter is optional. Any Python:

```
"dependencies": {
   "Python": {}
}
```

Whenever the user tries to install a plugin with such dependency FDM will offer the user to automatically install Python (if it’s not installed yet). For now, FDM can only automatically install Python 3.8.10 (features level 1) and Python 3.10.11 (features level 2).

Deno support requires features level 3.

3.1.2. Permissions section

The are two existing special permissions: "launchPython" and "launchDeno". These permissions allows a plugin to launch Python and Deno accordingly. The user is asked to confirm these permissions whenever he tries to install the plugin. An example:

"**permissions**": [
  "launchPython"
]

3.1.3. Allowed paths section

Add-ons’ Python code is running in a sandbox environment. In case an add-on needs an access to external directories, it must list them in the manifest.

- Windows platform. Each path listed must start from environment variable. E.g. “%LOCALAPPDATA%\somepath”.
- Unix platform. Each path listed must start from “~/” string. E.g. “~/.share/somepath”.

An example:

“**allowedPaths**”: [

  “%LOCALAPPDATA%\\somepath”,

  “~/.share/myaddon”

]

Add-on must be signed for the **allowedPaths** to take effect. You can override this by using developer options. Open the **Add-ons** page in FDM, click on **Add-ons** header 10 times. Developer options should appear then.

3.1.4. Json file with the information about updates

Important notice. Since FDM 6.24, an add-on must be signed for the auto-update feature to work.

FDM provides an automatic mechanism to update add-ons. For this, add-on must provide updateUrl in its manifest. This URL must point to a json file with the information about new version. This json must has the following members:

- **version**. The latest version available.
- **distributiveUrl**. URL to add-on file. Must be an HTTPS URL.

If the current add-on's version is lower than the one's specified in this json, FDM will download the file specified in distributiveUrl and install it. In case the add-on requires additional dependencies / permissions – FDM will warn user before installing the add-on.

In case there are several versions of the add-on with different supported API versions, **versions** array should be used instead. An example:

```
{
  "versions": [
      {
          "version": "...",
          "distributiveUrl": "…",
          "minApiVersion": 1,
          "minFeaturesLevel": 1
      },
      {
          "version": "...",
          "distributiveUrl": "…",
          "minApiVersion": 2,
          "minFeaturesLevel": 2
      },
      ...
  ]
}
```

FDM will pick up the most suitable version then.

4. Javascript code

4.1. Single media parser

This corresponds to mediaParser property in manifest json. If mediaParser is set to true, the plugins's javascript code must define a global variable named "**msParser**". This global variable must be an object with the following members:

- **isSupportedSource: function(url)**. This function should return *true* if plugin supports downloading from the specified URL.
- **supportedSourceCheckPriority: function()**. It's possible that several plugins could support downloading from the same URL. In this case a plugin with the greatest priority value will be chosen. Returns integer value from 0 to 2^{31}-1 (0x7FFFFFFF).
- **isPossiblySupportedSource: function(obj)**. In some cases it can be hard (or even impossible) to implement *isSupportedSource* function. This function is designed to be used in such a case instead. If the user creates an HTTP(S) download, and there were no plugins returned *true* from *isSupportedSource* call, then this function is called for every plugin. In case a plugin returns *true* from it, FDM will try to use this plugin to download from this URL. In case the plugin fails, FDM will try next plugin and so on. The *obj* variable is an object with the following members:

    - url – URL entered by the user.
    - contentType – content type, as reported by a server.
    - resourceSize – content length, as reported by a server.
- **parse: function (obj)**. This function is called when a plugin is chosen to download from a URL provided by the user. The *obj* variable is an object with the following members:
    - url – URL entered by the user.
    - cookies – array of cookies (since API version 3). Each array's member is an object with the following members: domain, isSecure, expirationDate, name, value.
    - browser – string representing the source web browser's name (can be empty). Since API version 3. Loot at *qtJsSystem.defaultWebBrowser* description for the list of possible names.
    - userAgent – string representing the source web browser's default user agent (can be empty). Since API version 3.
    - requestId – integer identifying this request. It's used in some other APIs.
    - interactive – boolean value indicating if this request is performing in foreground (i.e. the user is waiting for it to finish).

      This function must return a Promise object. On a success, this Promise must be resolved to an object with the parsed data. Its format will be described below. In case of a failure, this Promise must be rejected with an object with the following members:
    - **error**. String. It will be shown to the user as is.
    - **isParseError**. Boolean value, indicated if this error is caused by parsing algorithms of the plugin (true) or something else (false).
- **minIntevalBetweenQueryInfoDownloads: function()**. This function is used to define a minimum required interval (in milliseconds) between *parse* function calls. Return 0 to specify no interval. A typical use of this function is to reduce the load of the remote server.
- **overrideUrlPolicy: function(url).** Downloads from some specific URLs can be prohibited in FDM due to various reason. Plugin can define this function and return true for such URLs in order to lift such restriction.

Please take a look at msparser.js file from fdm-supremecourt-gov add-on for an example. You can find it in *examples* folder.

4.2. Playlist parser

This corresponds to mediaListParser property in manifest json. If mediaListParser is set to true, the plugins's javascript code must define a global variable named "**msBatchVideoParser**" (it's called so

due to historical reasons; it's not video-only really). This global variable must be an object with the same members as for a single media parser.

Please take a look at msbatchparser.js file from fdm-supremecourt-gov add-on for an example. You can find it in *examples* folder.

4.3. Parse result

It's an object describing a single media to download or a playlist.

4.3.1. Single media

It's an object. All members are optional if not explicitly stated otherwise.

- **id**. A string identifying media.
- **title**. Mandatory member. Media's title. It will be shown to the user.
- **webpage_url**. URL of the web page the media content resides on.
- **upload_date**. String. This must be an either ISO 8601 date or RFC 2822 date. Look here for more details.
- **duration**. In seconds.
- **formats**. Mandatory member. An array of objects describing available formats.
- **subtitles**. An object describing available subtitles.
- **thumbnails**. An array of objects describing media's thumbnails.

4.3.1.1. "Formats"

It's an array of objects with the following members (all members are optional if not explicitly stated otherwise):

- **url**. Mandatory member. URL to video/audio file.
- **manifestUrl**. HLS stream's ("m3u8_native" protocol) manifest file URL.
- **filesize** – size of the file.
- **protocol**. Mandatory member. Must be "http", "https", "m3u8_native" or "http_dash_segments".
- **ext** – file extension (e.g. "mp4", "m4a", "mp3" and so on).
- **video_ext**. Mandatory member in case of a video file. File's extension.
- **audio_ext**. Mandatory member in case of an audio file. File's extension.
- **vcodec**. Name of the codec used to encode video stream. Can be a name, "unknown" or "none".
- **acodec**. Name of the codec used to encode audio stream. Can be a name, "unknown" or "none".

- **container**. Mandatory member in case of DASH format. Can be something like “mp4_dash”, “m4a_dash”, “webm_dash” etc.
- **fps**. In case of a video – its FPS.
- **height** – video’s height.
- **width** – video’s width.
- **quality** – an arbitrary integer value (the higher the better).
- **tbr** – total bitrate (audio + video).
- **abr** – audio stream bitrate.
- **preference** – an arbitrary integer value indicating the preference of this format over the others for the user (the higher the better).
- **languagePreference** – an arbitrary integer value indicating the preference of this audio stream’s language over the others for the user (the higher the better).
- **language** – audio stream’s language code (e.g. “en” for English).
- **httpHeaders** – an object with custom HTTP headers to pass to the server. Example: *{"Accept":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language":"en-us,en;q=0.5"}*.
- **cookies** – string with cookies separated by semicolon. Example: name1=value1; Domain=.domain.com; Path=/; Expires=1778142081; name2=value2; Domain=.domain.com; Path=/; Secure; name3="value3=="; Domain=.domain.com; Path=/; Secure; Expires=1762590081.
- **fragment_base_url** – string with base URL for array of fragments (if any).
- **fragments** – array of objects, each of which represents a media stream fragment’s description.

4.3.1.1.1 “Fragments”

Media can be represented using an array of media segments. FDM merges all of them into a single media file.
Each object of this array consists of the following members:

- **path** – relative path to the segment’s file on a web server (base URL is provided in *fragment_base_url***)**.

4.3.1.2. “Subtitles”

It’s an object. Its each member must has name equal to a language code (e.g. “en”, “en_US”, …) and must be an array of objects with the following members (i.e. one array per one language code):

- **name**. Mandatory member. Language name.
- **url**. Mandatory member. URL of the file with subtitles.

- **ext**. Mandatory member. File's extension.
- **protocol**. Optional. Requires API 8+. Can be set to "m3u8_native" in case *url* points to M3U8 manifest.

Example:

{
"en": [{name: "English", url: "someurl", ext: "vtt"}],
"uk": [{name: "Ukrainian", url: "someotherurl", ext: "vtt"}],
"ru": [{name: "Russian", url: "someotherurl", ext: "vtt"}]
}

4.3.1.3. "Thumbnails"

It's an array of objects with the following members (all members are optional if not explicitly stated otherwise):

- **url**. Mandatory member. URL of the thumbnail image.
- **height** – image's height.
- **width** – image's width.
- **preference**. An arbitrary integer value indicating the preference of this format over others for the user (the higher the better).

4.3.2. Playlist

It's an object. All members are optional if not explicitly stated otherwise.

- **_type**. Mandatory member. Must be "playlist".
- **id**. A string identifying playlist.
- **title**. Mandatory member. Media's title. It will be shown to the user.
- **webpage_url**. URL of the web page the playlist content resides on.
- **thumbnails**. An array of objects describing playlist's thumbnails.
- **entries**. Mandatory member. An array of objects describing each media the playlist consists of.

4.3.2.1. "Entries"

It's an array of objects with the following members:

- **_type**. Mandatory member. Must be "url".
- **url**. Mandatory member. Media's URL. This URL will be passed then to a single media's parser.
- **title**. Mandatory member.

- **duration**. In seconds.

4.4. Available JavaScript API

FDM is not a web browser. It's using Qt framework's JavaScript engine. So there is not much available to use. For example, XMLHttpRequest is not available. Instead, we provide our own APIs. If you find them inconvenient or miss something you need – please let us know.

4.4.1. Console

We do have implementation of this object provided by Qt framework.

The list of functions added is as follows:

console.assert()

console.debug()

console.exception()

console.info()

console.log() (equivalent to console.debug())

console.error()

console.time()

console.timeEnd()

console.trace()

console.count()

console.warn()

print() (equivalent to console.debug())

4.4.2. setTimeout and clearTimeout

Are implemented by FDM. If you find any bugs, please report them.

4.4.3. downloadUrlAsUtf8Text

This function accepting the following arguments:

- **url**. Request URL.
- **cookies**. String. Can be empty. List of cookies, delimited by '\n' character. E.g. "name1=value1\nname2=value2".
- **headers**. Array of objects. Can be empty. Each object must has the following members:
  - key. HTTP header name.

    - Value. HTTP header value.
- **postData**. Can be empty. Either an UTF8 encoded string or an ArrayBuffer.

Returns Promise object. It's resolved to an object with the following members:

- **url**. Request URL.
- **body**. String with the downloaded content.
- **cookies**. HTTP cookies for this request.

It's rejected with an object with the following members:

- error. Error string.
- isParseError. A boolean value indicating if the failure is caused by the parser. It will be set to true in case it's not a network-related error.

Please take a look at msparser.js file from fdm-supremecourt-gov add-on for an example of using this function. You can find it in *examples* folder.

4.4.4. launchPythonScript

Add-on must has *"launchPython"* permission to use this function. It will get access denied error otherwise.

This function accepting the following arguments:

- **requestId**. Value, passed to parse function, or 0. It's used to automatically terminate Python's process when the user stops download.
- **interactive**. Value, passed to parse function. In case it's true, this Python process is launched immediately ignoring all queued processes. This should be "true" only for a download the user is creating currently, so he will not have to wait.
- **script**. Path to python's file to execute. This must be a relative path to plugin's root directory.
- **args**. Some arbitrary args (array) to pass.

It's resolved to an object with the following members:

- **exitCode**: Python's process exit code.
- **output**: Python's process console output. API version 6 and above: contains std output only. API version 5 and below: contains both std output and std error (merged).
- **errorOutput**: Python's process std error output. Requires API version 6.

In case of a failure it's rejected with an object with the following members:

- **error**. Error string.

4.4.5. Proxy support

It’s possible for a plugin to query proxy settings. For this, **qtJsNetworkProxyMgr**.**proxyForUrl**(url) should be used. As a result, plugin gets a full URL to a proxy, which includes authorization information.

4.4.6. Temporary files support

Requires API version 3. It’s possible for a plugin to create a temporary file. For this, **qtJsTools**.**createTmpFile**(desiredName) should be used. As a result, plugin gets an object with these methods:

- **path**. Returns full path to the file.
- **writeText(text)**. Writes the specified text to file (the previous contents are overwritten).
- **writeBinary(data)**. Writes the specified binary data to file (the previous contents are overwritten).

The created temporary file gets deleted automatically once the object is destroyed.

4.4.7. qtJsSystem

Requires API version 3. It’s used to query the current system’s information. Contains the following properties:

- **defaultUserAgent**. String with system’s default user agent string, or system’s default web browser’s user agent string. Can be empty.
- **defaultWebBrowser**. System’s default web browser’s name. Can be empty. Currently supported browsers are: firefox, chrome, edge, brave, opera, safari, vivaldi.

4.4.8. App

Requires API version 5. It’s used to query various information regarding app and its settings. Fow now it contains the only property:

- **pluginsAllowWbCookies**. A boolean. In case it’s set to false, a plugin MUST NOT use any cookies it can possibly extract from any web browser installed on the user’s machine.

4.4.9. Other

There are also the following objects in the global namespace:

- **qtJsPluginApiVersion**. Integet with the current API version.
- **qtJsPluginRootPath**. String with the full path to the current plugin. Requires API version 6.

5. Signing a plugin

FDM requires an add-on to be signed for some features to work.

In order to sign your add-on, you'll have to do the following steps:

1) We require your add-on to be hosted on GitHub. So you'll have to upload it there first.

2) Write us a request to support@freedownloadmanager.org with the following information:

- A link to GitHub repository with the add-on (for example, https://github.com/FreeDownloadManagerTeam/fdm-addons-docs);
- Optional. This repository's commit hash you would like to sign (for example, ddd5159f63280172d93d16072a09ea3be48bb8c0). By default, we sign the latest commit in your repository.

3) In the output, you'll get **signature.dat** file from us. You'll need to put it inside the root directory of your add-on.

6. Building a plugin

FDM expects a plugin to be in a form of a single file with .fda extension.

This .fda file is just a ZIP archive of your add-on.

So your steps to get a production ready add-on would include the following:

- Upload your add-on to GitHub
- Request a signature from us and get signature.dat file.
- Put this signature.dat file in your add-on's root directory (you don't have to upload it to GitHub) and archive this directory to get the resulting .zip file.
- Rename your .zip file to .fda.

7. Feedback

This is just an experimental API. We do not know exactly if it can be useful and demanded. Please tell us what you think. Do not hesitate to report us bugs / suggestions on improvement this.

We prefer you to use our forum for this.

8. API versions history

8.1. API version 7

A media can be represented using fragments array.

8.2. API version 8

Protocol field has been added to the subtitles node in order to support downloading subtitles via M3U8 protocol.

8.3. API version 9

Allowed paths field has been added to manifest.

8.4. Features level 3

Deno support is added. A plugin can specify "deno" in its dependencies for FDM to install Deno 2.6.3. A plugin can specify "launchDeno" in its permissions to be able to launch Deno.