CDOCS (Comprueba Documentos) — Documentación.

CDOCS (Comprueba Documentos) es una aplicación experimental, divulgativa, de escritorio, gratuita y de Código Abierto. Para **Windows**, **Linux** y **Mac OS X**; con el motor de ejecución **NW.js** (antes conocido como **node-webkit**).

Mediante el cálculo del dígito o dígitos de control, pretende comprobar la consistencia o veracidad de los números de algunos documentos de identificación personal, fiscal y laboral, de la Administración española. También la de algunos códigos de identificación financiera, en el ámbito internacional.

> Los resultados ofrecidos por la aplicación son sólo orientativos, deben considerarse de forma prudente. La Administración o las entidades pueden modificar la estructura o los algoritmos de comprobación de sus códigos de identificación.

Herramientas.

Para el desarrollo de aplicaciones con tecnologías Web y **NW.js** es necesario disponer de:

- **NW.js** — Permite llamar a todos los módulos **Node.js** directamente desde el **DOM**. Conviene trabajar con versiones estables (0.13 o superior) y con el *sabor* **SDK** durante la programación, para disponer de la ventana "inspector" de **Chromium** (útil para el control de errores y auditoría de rendimiento) y distribuir con el *sabor* **NORMAL**, más ligero.
- **Node.js®** — Un entorno de ejecución para JavaScript construido con el motor de **JavaScript V8 de Chrome**. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos. El ecosistema de paquetes de Node.js, **npm**, es el más grande en librerías de código abierto del mundo.

Programación.

CDOCS está programada con lenguajes estándar del desarrollo Web: **HTML5**, **CSS3**, **JAVASCRIPT** de propósito general y específico para la **API's** de **NW.js** (**versión 0.13 ó superior**), **Node.js** y **Chrome/Chromium**. También **Markdown** y **JSON** (archivo "manifiesto"). Los archivos enumerados a continuación se encuentran en el **directorio de trabajo** de la aplicación que, **por convención**, tendrá el nombre exclusivo de **package.nw** en los sistemas **Windows** y distribuciones **Linux**. En el caso de **Mac OS X** este directorio tendrá el nombre exclusivo de **app.nw**.

Manifiesto:

package.json — Cada paquete de aplicación **NW.js** debe contener un archivo de **manifiesto** denominado (necesariamente) **package.json**, indicará a **NW.js** cómo abrir la aplicación y controlar cómo se comporta en el navegador integrado. Contiene una serie de campos, requeridos, recomendados y opcionales, estructurados siguiendo las reglas del formato **JSON**.

Hoja de Estilos en Cascada (Cascading Style Sheets):

index.css — Contiene el código **CSS** (v3). Conviene definir algunas propiedades con la sintaxis exclusiva para **WebKit** (motor de *renderizado* de Chrome) como proveedor, para conseguir los efectos deseados. El archivo **index_min.css** con el mismo código que el anterior aunque comprimido y optimizado con **YUI Compressor**.

Markdown (lenguaje de marcado ligero):

`documentation.md` — Es este mismo documento, escrito en lenguaje **Markdown** (MD), contiene la descripción, propósito, estructura y funcionalidad de esta aplicación. En el directorio **/pdf** encuentras el archivo **documentation.pdf** que también es este mismo documento pero con formato **PDF**.

HTML5:

index.html — Es el documento **HTML** (v5) que consigue de la **ventana principal** de la aplicación. Hay otros archivos escritos en este lenguaje. Los contendidos en la carpeta **/forms** son sólo fragmentos de código que ocuparán su lugar en el **DOM** de la **ventana principal** de forma interactiva, mediante llamadas locales **AJAX**. La carpeta **/pages** almacena páginas **HTML5** completas (incluyen su correspondiente código CSS y particular JAVASCRIPT); sirven, como contenedores, para visualizar las ventanas secundarias.

JAVASCRIPT:

Código de terceros ...

El directorio **/node_modules** contiene dependencias opcionales que incrementan la funcionalidad o facilitan el uso de los módulos nativos de **Node.js**. Los principales son:

- **marked** — Un veloz analizador y compilador de **Markdown**. Utilizado para traducir código **MD** a **HTML** o viceversa.
- **unzip** — Descomprime archivos comprimidos con formato **.zip** en plataformas **Windows**, **Mac OS X** y distribuciones **Linux**.
- **fs-extra** — Añade métodos para facilitar el uso del sistema de archivos que no se incluyen en el módulo **fs** nativo. Es un módulo popular, de hecho puede reemplazar completamente a su ascendente, **fs**.
- **walk-sync** — De forma asíncrona y recursiva obtiene una matriz con todos los elementos (propiedades) contenidos en un directorio determinado.

En esta aplicación estos módulos se han instalado a **nivel local**, para conseguir una instalación de este tipo: Crea en tu proyecto una carpeta con el nombre **node_modules**, si aún careces de ella. Abre una terminal y situándote en el directorio de trabajo (en el que reside el archivo "manifiesto"), descarga e instala el módulo elegido con **npm** (Package manager). La instrucción en la línea de comandos guardará la sintaxis siguiente:

$ `npm install --save` *nombre-del-módulo-elegido*

La opción **--save** indica a **npm** que incluya el paquete dentro de la sección de dependencias de tu archivo "manifiesto" (**package.json**) automáticamente, lo que te ahorrará un paso adicional.

> Estos y otros módulos pueden estar programados para soportar una versión específica de **Node.js**; si ésta es inferior a la integrada en la versión de **NW.js** utilizada, la aplicación puede fallar. Conviene actualizar estos módulos a su última versión (estable) publicada.

En el directorio **/js** encontramos:

- Los archivos esenciales de la librería **jsPDF** — Genera documentos **PDF** : **jspdf.js**, **addimage.js**, **split_text_to_size.js** y **standard_fonts_metrics.js**. El primero de los archivos contiene el código principal de la librería, el segundo (*plugin* opcional) permite la inclusión de imágenes estáticas **.jpg** codificadas en *formato* **base64**; el tercero y el cuarto (*plugins* opcionales) facilitan el escalado del documento y la representación de símbolos de longitud variable **UTF-8**, necesarios para la correcta escritura de caracteres en español, entre otros. El código de estos archivos (depurado y minimizado mediante la herramienta **Closure Compiler**) está integrado en: **jspdf_and_some_plugins_min.js**.
- El guión **iban.js** — Código listo para verificar la exactitud del número **IBAN** de una cuenta bancaria de cualquier país adherido a este sistema.

Código del autor ...

En el directorio **/js** :

- **jspdfgenerador.js** — Con el método que, con dependencia de la librería **jsPDF**, consigue archivos **PDF** de contenido variable, representados en una ventana secundaria mediante el visor de documentos de esta clase que integra **NW.js**, (Chromium PDF Viewer).
- **index_auxiliar.js** — Contiene funciones de propósito general adicionales al guión principal: **index.js**, escritas aquí por motivos de comodidad y legibilidad del código.
- **index_principal.js** — El guión (script) principal con código comentado. Objetos **Javascript** estándar de propósito general y también exclusivo de la **API** de **Node.js** y de **NW.js**. Es dependiente de todos los otros archivos **Javascript** mencionados anteriormente. Características y referencias:
- **index_min.js** — Integra el código de los *scripts* anterioriores en uno sólo archivo, comprimido y optimizado con: **Closure Compiler**.
- **tabla_entidades_financieras.js** — Con la función que escribe los datos de la tabla "**entidades**" en la base "**bancos**" creada en la zona persistente **WebSQL** de la aplicación.

Estos datos tienen su origen en la información periódica que facilita el **Registro de Entidades del BE** sobre las entidades financieras con establecimiento en España y son tratados para su distribución en programas como este con la herramienta, **DB Browser for SQLite**.

API de NW.js:

- **nw** — Este es un objeto global (no necesita de la función, *require*). Contiene toda la interfaz de programación de aplicaciones (**API**) de **NW.js**. El objeto **nw** es el legado o heredero del módulo *nw-gui* utilizado en las versiones 0.12 y posteriores de **NW.js** que necesitaba de la función *require* para cargar en la aplicación la *interfaz gráfica del usuario* (`var nw = require("nw-gui")`).
- **App** — La aplicación utiliza algunos métodos de este objeto para: Leer las etiquetas del **manifiesto**, registrar "**atajos de teclado**", borrar el almacenamiento caché del navegador integrado y salir de la aplicación.
- **Menu** y **MenuItem** — La aplicación dispone de un menú nativo emergente en la ventana principal para todas las plataformas.
- **Shell** — La aplicación usa algunos métodos de este objeto para abrir un archivo, un directorio (en estos casos hay que señalar la **ruta absoluta** al elemento tratado) local, o una página Web remota en el navegador del equipo, designado por defecto.
- **Shortcut** — (Atajos de teclado). Pulsando la combinación de teclas **Ctr+P** se abrirá, cuando la función este disponible, una ventana secundaria para visualizar el contenido **PDF** relacionado con el cálculo de verificación realizado.
- **Tray** — En **Windows** , muestra un icono en la bandeja del sistema con un menú contextual con un primer ítem que restaura, en su caso, el estado de la ventana principal y otro que sirve para cerrar la aplicación.
- **Window** — Además de la ventana principal, la aplicación puede abrir ventanas secundarias. La comunicación entre la ventana principal y las secundarias se consigue enviando los argumentos necesarios en la llamada (método GET). También se ha programado la captura de los **eventos** que suceden automáticamente o por la intervención del usuario. Por ejemplo: `nw.Window.get().on('loaded', function(){...});` para la ventana principal, se utiliza en la puesta en marcha de la aplicación y sustituye a `window.onload=function(){...};`, método tradicional (evento que se produce cuando el documento se ha cargado por completo, en memoria).
  - `nw.Window.get().capturePage(callback [, config ])` — (**Capturas de pantalla**). Desde el menú nativo de la aplicación, en algunas situaciones, se puede generar una imagen de la ventana principal y su contenido. Estas imágenes se guardan en la carpeta, **cdocs-screenshots** que reside en el directorio de entorno del usuario. En la sección **Acerca de ...**, se detalla la ruta

hasta el directorio de almacenamiento de las imágenes, conviene limpiar manualmente su contenido si es abundante para ganar espacio en el disco.

- `nw.Window.get().showDevTools()` — (**Herramienta de desarrollo/Depuración**). La **ventana inspector** (DevTools) sólo es funcional en las versiones **NW.js** con *sabor* **SDK**. Además de con el método anterior (en programación), también puedes abrir esta ventana mediante el "atajo de teclado" [**F12**], en **Windows** o **Linux**. Y en **Mac OS X**, con la combinación de teclas: [⌘+⌥+**i**].

En el caso de "bloqueo" de la aplicación, puedes cerrarla desde la **ventana inspector**, si está operativa. Pulsa en la pestaña **Console**, en la línea inferior (de comandos) escribe: `nw.App.quit()` y pulsa, ↵(Enter).

API Chrome:

- **Notificaciones** (enriquecidas) — Notificaciones en la bandeja del sistema del usuario. (**chrome.notifications** API).
- **<webview>** — Esta etiqueta permite la carga en directo a través de la red de contenido externo en una ventana de la aplicación. Es diferente del conocido **iframe**, la **webview** se ejecuta en un proceso distinto. No tiene los mismos permisos que la aplicación y todas las interacciones entre la aplicación y el contenido incrustado son asíncronas. Esto permite mantiener la aplicación a salvo del contenido incrustado, además se puede controlar la apariencia y reaccionar ante eventos que ocurren en su ámbito. El documento **contenedor_webview.html** dispone de una etiqueta de esta clase dispuesta para visualizar el código fuente de esta aplicación expuesto en **GitHub**.

  El uso de **<webview>** es opcional (Menú -> Ajustes). En algunas plataformas este procedimiento bloquea la aplicación.

HTML5 especificaciones:

- **localStorage** — (Almacenamiento local). La aplicación utiliza los métodos `localStorage.setItem("nombreClave")` y `localStorage.getItem("nombreClave")` para guardar el valor de algunas variables de manera persistente para, por ejemplo, comprobar la nueva instalación de una actualización de la aplicación.
- **WebSQL** — La aplicación maneja una **base datos** en la que almacena información sobre las entidades financieras con establecimiento en España. Es conveniente señalar que todos los métodos relacionados con este objeto, (**bd**), son **asíncronos**.

  El **W3C** declaró **WebSQL obsoleto** por la dificultad de crear un estándar adecuado. Actualmente no forma parte de las especificaciones **HTML5**; no obstante **NW.js** mediante su navegador integrado, ofrece aún soporte para esta

tecnología. En sucesivas actualizaciones de la aplicación trataré de implementar **IndexedDB** u otra técnica similar, alternativa.

- **Multimedia** — La aplicación incluye dos pequeños archivos de **sonido** con formato comprimido **ogg**, utilizados para advertir las notificaciones o el proceso de captura de pantalla.
- **<dialog>** — Este elemento representa una caja de diálogo u otro componente interactivo, como inspector o ventana. (**Especificaciones del elemento HTML <dialog>**). Los documentos **index.html** y **contenedor_webview.html** disponen de esta etiqueta para anunciar, en el primer caso, que quedan ventanas abiertas y advertir, en el segundo, de posibles errores en la carga de contenido **webview**.

Node.js API:

- **Sistema de archivos** — La aplicación utiliza los métodos (síncoronos y asíncronos) del módulo **fs-extra** (que incluye el nativo **fs**) para manejar archivos en el área local, con sus operaciones de lectura, escritura, desplazamiento y borrado. El módulo **walk-sync** (secuencial) permite además obtener una matriz con los elementos contenidos en un directorio junto con las propiedades de cada uno de sus elementos.
- **Sistema operativo** - El módulo **os** proporciona una serie de métodos de utilidad relacionados con el sistema operativo. Se puede acceder a él mediante la llamada: `require("os")`.
- **Conectividad**:
- **dns** — Con este módulo `require("dns")` y su método `dns.lookup()` comprobamos si el equipo está conectado a Internet. En realidad examinamos sólo la disponibilidad de un determinado dominio Web.
- **http** — La interfaz HTTP están diseñada para admitir muchas características de este protocolo de red.
- **https** — El protocolo HTTP sobre TLS / SSL. En Node.js se implementa como un módulo independiente. `require("https")`, utilizado en la aplicación, con el método `https.get()` para la transferencia de descarga de un archivo residente en un servidor remoto (actualizaciones).
- **Procesos**:
- **process** — Este es un objeto **global** (no necesita de la llamada de carga `require()`). Proporciona información y control sobre el proceso **Node.js** actual. La aplicación recoge algunos métodos de este objeto para, por ejemplo, obtener la ruta local del directorio de datos de la aplicación, del directorio de trabajo, del entorno del usuario, y del archivo ejecutable principal (del paquete **NW.js**). Por ejemplo con `process.cwd()` obtenemos la ruta relativa hasta el directorio de trabajo de la aplicación.

- **child_process** (proceso hijo) — `require("child_process").exec` y `require("child_process").spawn`. El módulo **child_process** proporciona la capacidad de generar procesos secundarios de naturaleza asíncrona, sin bloqueos. El método **exec** se utiliza aquí para ejecutar los comandos: `cat /etc/issue` para obtener el nombre de la **distribución Linux** del usuario, `uname -m`, para obtener la arquitectura del sistema operativo (en **Linux** y **Mac OS X**) y `wmic OS get OSArchitecture`, con idéntico propósito en **Windows**. El método **spawn** consigue la realización de un guión ejecutable (**batch** en Windows o **bash** en Linux y Mac OS X) en la **línea de comandos**.

Sistema de actualizaciones de la aplicación CDOCS (NW) *(resumen)*:

Por convención, el **paquete de actualizaciones de la aplicación** contiene exclusivamente los archivos (fuente) correspondientes a ésta, válidos para cualquier plataforma y cualquier arquitectura. Es decir, no incluye los archivos del entorno **NW.js** en el que se ejecutará el programa. El paquete incluye además el directorio **cdocs-nw-updater** con los archivos (ejecutables en línea de comandos - batch y bash -) que permiten los últimos pasos de instalación de una actualización. El paquete es un único archivo, contenedor, comprimido **zip** nombrado de la siguiente manera:

***nombreDeLaAplicación-númeroDeVersión*-update.zip**. Por ejemplo, **cdocs-nw-0.1.0-update.zip** (primera versión - lanzamiento Beta - de la aplicación).

Los paquetes residen en un sitio Web remoto con un árbol de directorios organizado por versiones. La etiqueta **"updates_repository"** en el archivo **package.json** (manifiesto) muestra la ruta de origen hasta este repositorio.

1. La función **actualizaciones.comprobar**, realiza una llamada **AJAX** (POST) hasta un archivo remoto **index.php** que reside en el origen del árbol de directorios que contienen los paquetes de actualización. Si la llamada prospera, la aplicación recibe como respuesta un objeto **JSON** que contiene la información necesaria para conocer si hay alguna actualización disponible para su descarga e instalación posterior.

2. Si hay alguna actualización disponible y el usuario opta por su instalación. La función **actualizaciones.preparar** comprueba los requisitos mínimos que la aplicación debe cumplir para que este sistema funcione: La existencia de un **directorio de trabajo** que, entre otros, contiene el archivo manifiesto, con el nombre y situación convenidos para cada plataforma. Si esta primera comprobación es positiva, la aplicación crea un **directorio temporal** que después almacenará el paquete (comprimido "zip") descargado.

3. La función **actualizaciones.descargar** utiliza el método **http.get** de **Node.js** para obtener el paquete de la actualización en la memoria y después en el directorio temporal creado al efecto, visualizando en la ventana el progreso de la operación mediante una barra dinámica. Finalizada la descarga, verificará la consistencia del nuevo paquete.

4. Mediante el método **unzip.Extract** (del módulo de terceros, "unzip"), desembalamos el paquete comprimido (zip) en el mismo directorio temporal en el que se ha descargado.

5. La función **actualizaciones.copiarArchivosNuevos** copiará los archivos desempaquetados en el directorio temporal mencionado, en una carpeta **provisional** situada en el **directorio raíz** de la aplicación con el nombre de **package.nw.new** (en **Windows** y **Linux**) y el de **app.nw.new**, en el caso de **Mac OS X**.

El directorio provisional **package.nw.new** o **app.nw.new**, además de todos los archivos (fuente) de la nueva versión de la aplicación, contiene una carpeta con el nombre **cdocs-nw-updater** (instalador) que será desplazada (mediante la función **actualizaciones.moverInstalador**) hasta el directorio raíz de la aplicación.

1. Por último, la función **actualizaciones.configurar** elimina el directorio temporal creado para la descarga y desempaquetado y genera un proceso secundario para ejecutar, en modo silencioso, desde la **línea de comandos de la terminal del sistema** el archivo de instalación adecuado a la plataforma del usuario, residente en **cdocs-nw-updater** (instalador) que, con la aplicación cerrada, procura las siguientes operaciones finales:

- Cambia el nonmbre del primitivo directorio (de trabajo) **package.nw** (en **Windows** y **Linux**) o **app.nw**, en **Mac OS X**, a **package.nw.old** o **app.nw.old**, en el segundo caso.

- Cambia el nombre del nuevo directorio provisional de trabajo, **package.nw.new** (en **Windows** y **Linux**) o **app.nw.new**, en **Mac OS X**, a **package.nw** o **app.nw**, en el segundo caso.

- Lanza la aplicación, que notificará la nueva versión con un saludo, en su primera puesta en marcha.

Problemas conocidos :

Si por un error de proceso o por carecer de los permisos suficientes, la operación de renombrar el directorio de trabajo primitivo (añadiendo el sufijo **.old**) fracasa, la actualización no se llevará a cabo y se interrumpirá. La aplicación funcionará pero con la versión inicial.

Si (cosas que no deberían de pasar, pero que pasan), por error singular, la primera operación tuvo éxito y esta segunda, renombrar el directorio de trabajo provisional (eliminando el sufijo **.new**) fracasa: La actualización se interrumpirá y la aplicación quedará inservible. Para arreglar la avería (de forma manual), entra en el directorio raíz de la aplicación. Elimina el directorio **package.nw.new** o **app.nw.new** y renombrar **package.nw.old** o **app.nw.old** como **package.nw** y **app.nw**, en el segundo supuesto. Reinicia la aplicación, no se habrá actualizado, pero funcionará.

Si durante el proceso de actualización, la aplicación se cierra y después de un tiempo prudente (seis o siete segundos) no reinicia (sin mensajes de error en pantalla). Intenta la

puesta en marcha de forma manual.

Contenido del instalador (**cdocs-nw-updater**):

El directorio **cdocs-nw-updater** no forma parte de la aplicación. Se obtiene dentro de los paquetes de actualización y en esta operación se mueve desde el directorio de trabajo provisional hasta el directorio raíz de la aplicación. Contiene los siguientes archivos:

En **Windows** (directorio **windows**): Un archivo de *procesamiento por lotes*, de nombre: **cdocs-nw-updater-windows-launcher.bat** que, mediante un proceso hijo se desarrolla en la **línea de comandos** de la ***Shell*** (interfaz gráfica de usuario principal del sistema operativo). Si la secuencia de comandos programada en este archivo "batch" termina sin errores, lanzará la ejecución de un segundo archivo: **cdocs-nw-updater-windows.vbs** (archivo de secuencia de comandos de **VBScript**), encargado de realizar la tarea de renombar los archivos provisionales de actualización, notificar los errores, si suceden y de reiniciar la aplicación (actualizada), si todo fue bien.

En **Mac OSX** y **Linux** (directorio **unix**): El archivo **cdocs-nw-updater-unix.sh** (**Shell Bourne**) de procesamiento de comandos que, mediante un proceso hijo generado por la aplicación se ejecuta para realizar la tarea de renombar los archivos provisionales de actualización, advertir sobre los errores, si aparece alguno y de reiniciar la aplicación (actualizada), si todo ha funcionado correctamente.

Empaquetado y distribución.

- Desde este enlace: **https://github.com/fooghub/Cdocs-NW** accedes al código fuente de la aplicación en **GitHub**. Pulsa en el botón (desplegable) "**Clone or download**" y después en "**Download ZIP**" para obtener el archivo (comprimido ZIP) **Cdocs-NW-master.zip**. Descomprime el archivo descargado. Comprueba que hay dos directorios o carpetas: **cdocs-nw-proyect** y **cdocs-nw-tools**.

cdocs-nw-proyect contiene **todos** los archivos (fuente) de la aplicación.

- **Descarga aquí el paquete NW.js** adecuado para el sistema y la arquitectura de tu equipo.

  Recuerda, la aplicación Cdocs-NW está diseñada para rodar con las versiones 0.13 ó superior de NW.js.

Para el desarrollo es aconsejable obtener el *sabor* **SDK** del paquete **NW.js** que, entre otros, dispone de la posibilidad de visualizar y auditar el rendimiento de la aplicación con la ventana **inspector** de **Chrome**. Para la distribución final es mejor utlizar el *sabor* **normal** del paquete **NW.js** elegido, más liviano.

- **Instalación (manual) en Windows, Linux y Mac OSX**:

- **En común**: Construye una copia del paquete **NW.js** descargado desde el sitio oficial, (por ejemplo: **nwjs-sdk-v0.14.7-*sistema-arquitectura***). Cambia el nombre de esta copia a **cdocs-nw-app**. En **Windows** y **Linux** , **cdocs-nw-app** será la carpeta raíz de la aplicación, sin embargo en **Mac OS X** llamaremos carpeta raíz de la aplicación al

directorio ejecutable (paquete) **nwjs.app**. Pulsando con el botón derecho del ratón sobre **nwjs.app** se abrirá un menú contextual con algunas opciones, elige "**Mostrar contenido del paquete**" para navegar por su interior, trabajaremos en el directorio: **cdocs-nw-app/nwjs.app/Contents/Resources/** donde residirá nuestro directorio de trabajo **app.nw**.

- **Pasos específicos para Windows**:
  - Cambia el nombre de la carpeta del proyecto, **cdocs-nw-proyect** a **package.nw** (por convención no puede tener otro nombre).
  - Copia/pega **package.nw** dentro de **cdocs-nw-app**.
  - Renombar el archivo **nw.exe** a **CDOCS.exe**.
  - Con la herramienta *freeware* : **Resource HackerTM** puedes sustituir el icono asociado al archivo ejecutable. En la carpeta **cdocs-nw-tools** descargada junto con todo el código fuente de la aplicación hay un icono **logo.ico** que puedes utilizar para esta operación.
  - Sólo queda hacer doble clic sobre **CDOCS.exe** y, con algo de suerte, la apliación empezará a funcionar.
- **Pasos específicos para Linux**:

  Las aplicaciones **NW.js**, no necesitan ninguna dependencia especial en distribuciones **Linux**, si bien en este contexto, conviene disponer de algún gestor de mensajes gráficos previamente instalado, como **Zenity** (escritorios **GNOME**) o **Kdialog** (escritorios **KDE**). Y del emulador de terminal estándar **Xterm** (instalado en la mayoría de distribuciones **Linux**).
  - Cambia el nombre de la carpeta del proyecto, **cdocs-nw-proyect** a **package.nw** (por convención no puede tener otro nombre).
  - Copia/pega **package.nw** dentro de **cdocs-nw-app**.
  - Copia los archivos **cdocs-nw-tools/logo.png** y **cdocs-nw-tools/CDOCS.desktop**, dentro de la carpeta raíz de la aplicación (de nombre, **cdocs-nw-app**, si has seguido los pasos anteriores).
  - Sitúate sobre el archivo **CDOCS.desktop** (lanzador) y activa el diálogo contextual pulsando el botón derecho del ratón (propiepades) para asociarle el icono elegido **logo.png** y asignar la ruta (comando) adecuada hasta el archivo ejecutable principal de la aplicación, que se llama **nw** (bilblioteca compartida).
  - Ahora puedes hacer doble clic sobre el archivo **CDOCS** y, con algo más de suerte, la aplicación arrancará.
- **Pasos específicos para Mac OS X**:
  - Cambia el nombre de la carpeta del proyecto, **cdocs-nw-proyect** a **app.nw** (por

convención no puede tener otro nombre).

- **Renombrar nwjs.app** a **CDOCS.app**.
- Copia/pega **app.nw** dentro de **cdocs-nw-app/CDOCS.app/Contents/Resources/**.
- Cambia las imágenes **cdocs-nw-app/CDOCS.app/Contents/Resources/app.icns** y **cdocs-nw-app/CDOCS.app/Contents/Resources/document.icns** por las del mismo nombre, que encontrarás en **cdocs-nw-tools**.
- Edita el archivo: **cdocs-nw-app/CDOCS.app/Contents/Info.plist** (El archivo **Info.plist** se utiliza en **Mac OS X** para mostrar la información de la aplicación y en el **Mac App Store** con fines de identificación) y realiza los siguientes cambios (en el contenido de la etiqueta *string*).

Original:	**Modificaciones:**
...<key>CFBundleDisplayName</key><string>nwjs</string> ...	...<key>CFBundleDisplayName</key><string>**CDOCS**</string> ...
...<key>CFBundleName</key><string>Chromium</string> ...	...<key>CFBundleName</key><string>**CDOCS**</string> ...
...<key>CFBundleShortVersionString</key><string>50.0.2661.102</string> ...	...<key>CFBundleShortVersionString</key><string>**0.1.0**</string> ...
...<key>CFBundleVersion</key><string>2661.102</string> ...	...<key>CFBundleVersion</key><string>**0.1.0**</string> ...

- Para establecer el nombre correcto de la aplicación en la barra única de menús (superior) de **Mac OS X**. Edita también el archivo: **cdocs-nw-app/CDOCS.app/Contents/Resources/en.lproj/InfoPlist.strings** y realiza los siguientes cambios.

Original:	**Modificado:**
...CFBundleDisplayName = "nwjs";...	...CFBundleDisplayName = "**CDOCS**";...

Ahora sólo queda hacer doble clic sobre el *directorio ejecutable* (paquete), con su nuevo nombre: **CDOCS.app** y la aplicación arrancará. Con mucha, mucha suerte, funcionará sin errores.

> Las aplicaciones **NW.js** en **Mac OS X** deben de estar firmadas, en el caso contrario no arrancará si **Gatekeeper** está activado. Para obtener más información, consulta **Support for Mac App Store** en la documentación de **NW.js**.

- **Instalación utomática de aplicaciones NW.js, con la herramienta Web2Executable**: Descarga e instala en tu equipo **Web2Executable**. Lanza la herramienta y abre la carpeta que contiene los archivos fuente de la aplicación descargada desde **GitHub** que será: **cdocs-nw-proyect** (**File/Open Proyect/cdocs-nw-proyect**). Comprueba que quedan alimentados los campos esenciales de la primera "**App Settings**" y segunda pestaña "**Window Settings**". Pulsa sobre la pestaña "**Exports Settings**" y elige las plataformas que quieres utilizar. Ahora, sobre la pestaña siguiente: "**Compression Settings**" y (para cumplir las covenciones estipuladas en este proyecto), "**Compression Level**" debe estar situado en **cero**; la casilla "**Uncompressed Folder**"

debe estar **marcada**. Pulsa ahora sobre la pestaña "**Download Settings**" y elige la versión de **NW.js** que quieres utilizar con la aplicación (**NW.js version : 0.14.7**, en mi caso - puedes obtener el *sabor* **SDK** o el *sabor* **normal** -). Por último pulsa sobre el botón "**Export**" y **Web2Executable** empezará a trabajar.

Finalizado el proceso, dentro de la carpeta del proyecto, **cdocs-nw-proyect** aparecerá una nueva, **output** que contiene todas las distribuciones elegidas.

Hay otras herramientas para automatizar el empaquetado y la distribución de aplicaciones **NW.js**. He elegido **Web2Executable** porque se adapta perfectamente a las convenciones de este proyecto. Con **Web2Executable** puedes implementar paquetes de distribución para **Windows** y **Linux** con arquitecturas de 32 y 64 bit, (en el caso de **Mac OS X** sólo podemos optar por la arquitectura de 64 bit).

Licencia.

Foog.Software — Marzo 2017 — **www.foog.es**