Configuration Files Details

- Configuration Files
  - Configuration Files Location
    - Missing Subdirectories?
    - Starting Fresh
    - Undoing Portable
  - Editing Configuration Files
    - Configuration File Encoding
  - Configuration Files during Upgrades
- Specifics on Configuration Files
  - The context menu: `contextMenu.xml`
    - Grouping items into sub-menus
    - Overriding a menu item name
    - Context menu syntax summary
  - The context menu: `tabContextMenu.xml`
  - Keyboard shortcuts: `shortcuts.xml`
    - Virtual Key Number
    - `<Macros>`
      - Attributes for the `<Macro>` node
      - Attributes for the `<Action>` tag
    - `<UserDefinedCommands>`
      - Variables for Run Commands
    - v8.5.3 `shortcuts.xml` updates
  - User Interface settings: `config.xml`
  - Keyword lists: `langs.xml`
  - Highlighting schemes: `stylers.xml`
  - `Function List`
    - Function List Definitions

Upgrading old Function List entries
Toolbar Customization
Toolbar Icon Visibility: `toolbarButtonsConf.xml`
Toolbar Icon Customization
Set of Icons
Other Configuration Files
Logfiles
Validating Config-File XML

Configuration Files

Notepad++ offers a comprehensive user interface to review or change most of its settings. However, there are some special cases where it is worthwhile to edit the configuration files directly, including:

- Customizing the Context Menu
- Editing previously-recorded macros, or crafting new macros manually
- Adding keywords to a language, because the new language version isn't matched yet

Please note that Notepad++ writes the configuration files when it exits, which is why the Editing Configuration Files section below says that Notepad++ may overwrite your changes. But this also means that, when you make a change using Notepad++ menus and dialogs (like changing a preference, or saving a macro), your change will not be written to the configuration file until Notepad++ exits. So if you open up the configuration file after you have changed the preference but before you have exited Notepad++, you will *not* see that change reflected in the file yet. Do not be surprised by this.

Configuration Files Location

In a standard installation, the per-user configuration files go in `%AppData%\Notepad++\` . (For a refresher course on `%AppData%` , see the Community Forum FAQ's `%AppData%` entry.) The machine-wide configuration files go in the executable directory (so usually `c:\program files\notepad++\`). When Notepad++ is looking for shortcuts, stylers and themes, function list definitions, and user defined language definitions (and usually the default session file, though that has exceptions enumerated below), it will try to read from the per-user directory; if the right file isn't found there, then it will check machine-wide configuration files.

In a portable installation, the configuration files all go in the same directory as the `notepad++.exe` executable (or the appropriate subdirectory), and there is no per-user configuration location. An installation is treated as "portable" if the zero-byte `doLocalConf.xml` file is present alongside the `notepad++.exe` executable (see also Undoing Portable, below). The **?** menu's **Debug Info** will show `Local Conf mode: ON` for a portable version.

If you enable the Cloud settings, some per-user configuration files will go in the defined directory, including `contextMenu.xml`, `shortcuts.xml`, `userDefineLang.xml`, `langs.xml`, `stylers.xml`, and `config.xml`; the `themes\` subfolder will go in the cloud directory, and will start empty (when the cloud directory is first populated); the `userDefineLang\` subfolder can be placed there as well, though it won't be created by default when the Cloud settings folder is first populated. It should be emphasized that `session.xml` does *not* work in the cloud folder, so if you have the cloud settings, `session.xml` will still be found in either the `%AppData%` hierarchy for a normal installation, or in the installed directory for a portable installation.

There is a command-line option `-settingsDir` which will set a specific directory for the per-user configuration-files location. This folder allows all the configuration files metioned for cloud directory in the paragraph above; but, unlike the cloud directory, `session.xml` *does* work in the `-settingsDir` directory (so if you want to have all your per-user configuration, including your session information, in your cloud folder, it might be wise to not use the Cloud settings option, and instead point `-settingsDir` to the cloud folder).

Search order: If the `-settingsDir` option is set, that configuration file directory will take priority over any other configuration file directory. If the Cloud directory setting is defined and enabled, that will take priority over the portable or standard configuration file directory. If `doLocalConf.xml` is present, the portable configuration file location will take priority over the `%AppData%\Notepad++\` directory. If none of the other configuration file directories are active, then the standard configuration file directory is `%AppData%\Notepad++\`. If it cannot find a configuration file in any of the per-user locations, it will use the version from the same directory as the executable.

Missing Subdirectories?

Reading the above section, one may go looking for the user `functionList\` or `themes\` folders in `%AppData%\Notepad++\` (or your Cloud folder, or wherever `-settingsDir` option points) after an initial installation, and be surprised those directories aren't there: the installer doesn't create those empty directories for you. If you want to add a custom function list definition or theme to your user-config folder in the `%AppData%\Notepad++` (or equivalent) hierarchy, you may have to create the appropriate subdirectory first, then put your custom function list or theme definition in the newly-created folder.

Starting Fresh

If you want to reset your Notepad++ back to "factory condition", and are using the normal installation, you can exit all instances of Notepad++ then delete (or rename) the `%AppData%\Notepad++` folder (or your Cloud folder, or wherever `-settingsDir` option points). The next time you run Notepad++, it will create `%AppData%\Notepad++` (or Cloud or `-settingsDir` location) and populate it with a minimal set of config files.

If you just want to start fresh on a particular config file: Notepad++ knows how to recreate the necessary files if it doesn't find them. For the ones that have an `<XXX>.model.xml` file in the Notepad++ application directory, all you have to do is delete your per-user `<XXX>.xml` (from `%AppData%\Notepad++\` or the Cloud or `-settingsDir` location) and restart Notepad++, and Notepad++ will copy the `<XXX>.model.xml` version into the right place for you. If the `config.xml` or your session or shortcuts or contextMenu config files are intentionally deleted (or otherwise missing) from the config-file folder, Notepad++ can recreate those as necessary as well on its next start (it is programmed on how to create them from scratch, rather than using a `.model.` file – though Notepad++ *will* use `config.model.xml` instead of its internal defaults, if it exists).

Undoing Portable

Since `doLocalConfig.xml` being in the same directory as `notepad++.exe` will cause Notepad++ to use the config files in that directory ("portable mode"), deleting `doLocalConfig.xml` will *stop* Notepad++ from using the config files in the executable directory, and it will go back to looking for config files in the `%AppData%` hierarchy. (As with any change to an external config file in Notepad++, for it to take effect, you have to exit Notepad++ then restart the application for it to run in the new mode.)

When you delete `doLocalConfig.xml`, it will no longer look for `config.xml`, `session.xml`, and other such files in the application directory. If you want to clean up the application-directory version of unnecessary config files, those two are safe to delete. Also, `stylers.xml` and `langs.xml` are safe to delete from the application folder after leaving local-config mode; however, **make sure** you **keep** the `stylers.model.xml` and `langs.model.xml`, because those are the model files that are used to create the files in `%AppData%`. It is recommended that you not delete `shortcuts.xml` or `contextMenu.xml` from the application directory, because those are used as the fallback/default version of those configurations if there is no version found in `%AppData%`, so it's best to always have a copy in the application directory. Please note: it is never *required* that you delete any of the files mentioned in this paragraph (except `doLocalConfig.xml`) to leave local-configuration / portable mode; when in normal mode, Notepad++ will use the copies in the `%AppData%` hierarchy and ignore the ones in the application directory; this section is just here to make it more clear what it's safe to delete if you're trying to save a handful of bytes after leaving portable mode.

Editing Configuration Files

ALWAYS BACKUP THE FILE TO BE EDITED. If you make a mistake, Notepad++ may erase the whole contents and replace it with useless defaults. This is probably the worst that can happen, but it does happen.

If changes are made in the Notepad++ UI to settings which are stored in configuration files, those will be written to disk when you exit Notepad++ *after* any file saves you do. Thus, if you are going to edit a Notepad++ configuration file *with* Notepad++ (and why would you want to edit it with anything else?), you will need to be careful. The safest sequence when editing a configuration file:

1. Close *all* active instances of Notepad++
2. Open *one* instance of Notepad++
3. Edit the configuration file
4. Save
5. Exit Notepad++
6. Reload Notepad++
7. The changes will now be in effect.

The `config.xml` file may be overwritten by Notepad++ on exit, even if you follow this procedure, so that sequence won't reliably work for `config.xml`. To edit `config.xml`, close all instances of Notepad++; edit `config.xml` in some "other" editor and save; reload Notepad++ and the changes should take effect. (For the "other" editor, you *could* use something like

Windows' builtin notepad.exe. But it would be better if you had another portable Notepad++ somewhere on your machine, and use that portable Notepad++ to edit your main Notepad++'s `config.xml`, thus never having to use a non-Notepad++ editor: so close your main Notepad++, run the portable Notepad++ and open your main `config.xml`, edit and save, exit the portable Notepad++, then re-run the main Notepad++, and everything should be updated.)

Configuration File Encoding

The Notepad++ XML configuration files are expected to be in UTF-8 encoding (with no BOM), and so should use the prolog of `<?xml version="1.0" encoding="UTF-8" ?>`. If you use the BOM character at the beginning of a UTF-8 encoded XML file, or if you use another encoding like UTF-16, Notepad++ will not be able to read that configuration file.

Configuration Files during Upgrades

When you use the installer to upgrade your existing copy of Notepad++ (either manually, or through the **?**-menu's **Upgrade Notepad++**, or through Notepad++'s auto-update feature), the installer will avoid overwriting configuration files that you have customized – this is to prevent you losing your preferred settings and customizations.

However, this means that occasionally, updated functionList parsing files, or theme definitions, or new keywords in `langs.xml`, or new default contextMenu entries, or new shortcut entries might not be added to your local copy of Notepad++. So if you upgrade and keep your existing configuration, it is a good idea to occasionally compare your configuration files to the "standard/default" configuration files: some of the config files come with `*.model.xml` versions in the installation directory, which show you the default settings for those files; for other configuration files, you could download a portable zipfile of the same version you are using, and compare the config files from the portable to your installed; and you can always look in the source code repository for the various configuration files. (You can use two Views to look at the files side-by-side, and the synchronized scrolling feature can help keep the copies aligned; plugins such as the Compare plugin are designed to show differences between files as well.)

For the `config.xml` (which contains the settings from the GUI's **Settings > Preferences** dialog), if Notepad++ has a setting that it doesn't find defined in your current `config.xml`, it will apply the default value compiled in the program, and the next time it writes `config.xml` it will write that default value in `config.xml`. Once it gets written, future changes to that default value will not override the value already in your file, even when you upgrade Notepad++. `config.xml` will only get a default value for a setting in three situations:

1. On a new installation (running the installer when there isn't a previous Notepad++, or when there isn't a `config.xml` file and/or settings directory for the active user).
2. When you run Notepad++, and there is no `config.xml` (either because there isn't a `config.xml` file and/or settings file for the current user, or the `config.xml` has been deleted), so Notepad++ writes a complete `config.xml` with all default values.
3. When your `config.xml` is missing the attribute/value pair for a given setting.

Specifics on Configuration Files

The context menu: `contextMenu.xml`

The context menu does not have a GUI-based editor; you just need to edit the file. As a result, the **Settings > Edit Popup Context Menu** entry exists to make it easy for you to access this config file.

All menu commands can be added to the Context Menu, including plugin commands:

- To add a built-in command, you may provide the main menu name (as it appears in the main menu bar) as the value of the MenuEntryName attribute and the command's item name (as it appears in the menu) as the value of the MenuItemName attribute. The MenuEntryName attribute must reference an entry on the main menu bar and must be an ancestor of the MenuItemName attribute, regardless of its depth.
    - For example, `<Item MenuEntryName="Edit" MenuItemName="Cut"/>` will add a context-menu entry that calls the **Edit** menu's **Cut** entry.
- Alternately, to add a built-in command, you may instead provide the menu command ID via the `id` attribute. The command ID values for a given menu can be found in your localization file (like english.xml), which will map the localized text you see in the menu to the command ID, or in menuCmdID.h.

    - For example, `<Item id="42001"/>` will add a context-menu entry that calls the **Edit > Cut** action by its command ID.
    - **NOTE**: `<Item id="0"/>` has a special meaning: it acts as a horizonal separator line – that's how you can get a line between groups in the context menu.
- To add a plugin command, you need to provide the plugin's menu item name (as it appears in the Plugins menu) as the value of the PluginEntryName attribute and the command's menu item name (as it appears in the plugin's sub-menu) as the value of the PluginCommandItemName attribute. (*Note*: Context menu entries will only work for entries immediately under the plugin name; entries deeper than that, like **Plugins > Super Plugin Name > Submenu Here > I Want This Command**, will not work.)
    - For example: `<Item PluginEntryName="MIME Tools" PluginCommandItemName="Base64 Encode" />` will add a context-menu entry that calls **Plugins > MIME Tools > Base64 Encode**

Note that the menu names and menu item names (whether built-in or plugin names) that you use in the should be in English, not in a translated language. The Shortcut Mapper can help you find the English name of plugin commands; simply switch to English localization for the raw name of built-in commands; or you can look at the english.xml that shipped with your distribution.

By default, each `<Item>` will be rendered in the top level of the context menu with a localized name matching the normal menu entry, unless overridden by the attributes described next.

Grouping items into sub-menus

Each `<Item>` will go in the first level of the context menu. This can be overridden by adding a `FolderName="name_of_submenu"` attribute to consecutive items, so that they will be grouped into a sub-menu with that name. Specifying "" is the same as leaving the FolderName attribute out. Note that sub-menus do not nest - you cannot add a sub-menu to a sub-menu. Non-Latin characters are supported.

Overriding a menu item name

Each `<Item>` will use the same text as the main menu entry uses, as defined by your current localization. This can be overridden by adding an `ItemNameAs="new_name_for_the_item"` attribute, so that the new name will be displayed instead of the standard one. This is useful when the name is lengthy, as it makes the Context Menu unwieldy otherwise. Non-Latin characters are supported.

Context menu syntax summary

In brief, the structure of the context menu file is as follows

- Top Level Node: `<NotepadPlus>` (no attributes)
- Intermediate Node: `<ScintillaContextMenu>` (no attributes)
- Entry Nodes: `<Item ...>`
    - `MenuEntryName` attribute: Used for accessing Notepad++ menu items by name. The value should be name of the menu.
    - `MenuItemName` attribute: Used for accessing Notepad++ menu items by name. The value should be the name of the menu item inside the given menu.
    - `id` attribute: Used for accessing Notepad++ commands by CommandID. The value should be the CommandID number, as seen in english.xml and similar translation files.
        - `<Item id="0" />` will make a separator line in the menu.
    - `ItemNameAs` attribute: Used for for overriding a menu item name: using your own "display name" for a given entry in your context menu, instead of using the one inherited from the main Notepad++ menu system.
    - `PluginEntryName` attribute: Used for accessing Plugin commands. The value should be the name of the plugin as seen in the Plugins menu.
    - `PluginCommandItemName` attribute: Used for accessing Plugin commands. The value should be the name of the menu entry inside that Plugin's menu. (*Note*: Context menu entries will only work for entries immediately under the plugin name; entries deeper than that, like **Plugins > Super Plugin Name > Submenu Here > I Want This Command**, will not work.)
    - `FolderName` attribute: Used for grouping items into sub-menus in the context menu.
    - `TranslateID` attribute: Used for auto-translating your sub-menu names. If the attribute's value matches one of the nodes in the `<MiscStrings>` section of english.xml or similar translation files, the menu will use your active translation's value for the displayed FolderName.
- As with all XML files, using `<!--` and `-->` around the code will "comment it out". So you can use text comments or comment out XML syntax that you want Notepad++ to ignore. To "uncomment" the XML, remove the `<!--` and `-->` surrounding that piece of code.

The context menu: `tabContextMenu.xml`

Starting in v8.4.8, the tab-bar context menu (the popup menu that you see when you right-click on the title of the tab in the tab bar) is user-configurable using the `tabContextMenu.xml` config file.

The format is the same as the `contextMenu.xml` described above, except the “Intermediate Node” is `<TabContextMenu>` instead of `<ScintillaContextMenu>`.

You may have gotten a copy of `tabContextMenu_example.xml` when you installed, or you can download a copy from the Notepad++ source here (possibly needed if you have a portable or minimal copy of Notepad++, or if the file didn’t get created when you updated an existing Notepad++ installation) – if you rename that example to `tabContextMenu.xml` in your configuration directory, and restart Notepad++, then the example will be used. In that example, the default replicates the structure released with v8.4.7; if you comment that out and uncomment the second half of the file, it will instead replicate the Tab Context Menu found in v8.4.6 and earlier (useful for those who like “the old way” of doing things in the tab context menu).

To go back to the default tab context menu, you can just rename or delete `tabContextMenu.xml` (or download the fresh copy and save it over the current `tabContextMenu.xml`) then restart Notepad++.

Keyboard shortcuts: `shortcuts.xml`

Defines keyboard shortcuts for various types of commands. The shortcuts are most-easily defined in the various tabs of the **Settings > Shortcut Mapper**.

This file has the following nodes:

1. `<InternalCommands>`: Keyboard shortcuts for Notepad++ menu commands that have been remapped by the user. (Commands that have never changed from their default shortcuts are *not* listed here.)
2. `<Macros>`: Keyboard shortcuts for the macros listed in the lower part of the **Macro** menu. Also defines what commands those macros execute.

3. `<UserDefinedCommands>` : Keyboard shortcuts for the Run menu entries. Also defines what actions those entries take.
4. `<PluginCommands>` : Keyboard shortcuts for plugin commands that have been remapped. (Commands that use their default shortcuts are not listed here.)
5. `<ScintillaKeys>` : Keyboard shortcuts for Scintilla commands, most of which relate to selecting text and moving around in the editor. (Commands that use their default shortcuts are not listed here.)

The definitions of the `<Macros>` and `<UserDefinedCommands>` are generally all that benefit from manual editing of the `shortcuts.xml` . It is much safer to edit the shortcuts using the **Shortcut Mapper**

Virtual Key Number

All the types of commands in `shortcuts.xml` have a `key` attribute, which uses the Windows virtual key number as the value. This is *not* necessarily the same as the ASCII code or Unicode codepoint. In standard English locales, the virtual key usually lines up with the ASCII code for the character, but that is not universally true. The complete list of base virtual key code is to be found on keys.h. Because of this reliance on OS-defined virtual keys, many letters in your native alphabet cannot be used, though for characters that are entered directly with a key on your keyboard, it may be possible (with some effort) to determine the virtual key number for the key. (Some hints are given in the Notepad++ Community Forum at this post and this other post.)

`<Macros>`

When not empty, this node is made of `<Macro>` nodes, each of which represents an individual macro. Each `<Macro>` holds a non-empty list of `<Action>` tags which represent individual macro steps. These steps are either Scintilla commands or Notepad++ commands, not raw keystrokes. For more details on macro recording, see Macros.

Attributes for the `<Macro>` node

Position	Name	Value format	Meaning
1	name	string	The name of the macro. Several macros may have the same name

Position	Name	Value format	Meaning
2	Ctrl	“yes”/“no”	The key being mapped to has the Control modifier
3	Alt	“yes”/“no”	The key being mapped to has the Alt modifier
4	Shift	“yes”/“no”	The key being mapped to has the Shift modifier
5	Key	integer	The base virtual key number, in the 1..255 range
6	FolderName	string	[optional] Can be used to group one or more macros into a named sub-menu (new to v8.4.8)

Note: FolderName can only be entered by hand-editing the `shortcuts.xml` file; it is not available in the Shortcut Mapper or any other GUI element.

Although it is possible for several macros to share the same name or shortcut, this practice is highly discouraged.

Attributes for the `<Action>` tag

Position	Name	Value format	Meaning
1	type	integer	`0` for Scintilla messages that do not pass a string as second parameter
			`1` for Scintilla messages that pass a string as second parameter
			`2` for Notepad++ defined commands
			`3` for search and replace recording
2	message	integer	`0` if `type=2`, otherwise use the message id

Position	Name	Value format	Meaning
3	wParam	integer	Command id when `type=2` or `type=3`, else actual first parameter of the message. Use `0` if the message or command doesn't require a wParam.
4	lParam	integer	`0` unless `type=0` and the second parameter of the message is actually used, or scalar value used when `type=3`.
5	sParam	string	`""` unless `type=1` or `type=3`, in which case this is the string pointed by the second parameter of the message.

The full list of Scintilla messages for `type=0` and `type=1` Scintilla messages, as well as a concise documentation, can be found in Scintilla.h and/or Scintilla.iface. More details on those messages can be found in the Scintilla Docs. You can use any Scintilla message that does not return a value, that passes an integer in `wParam`, and uses either an integer or string in `lParam`. There are some messages that require strings in the `wParam`, or various data structures in one or both parameters: those will not work in a macro. (For more on the messaging system, see Plugin Communication.)

Each `type=2` message runs a corresponding menu entry from Notepad++'s menu system. The `wParam` command IDs for `type=2` Notepad++ messages can be found as the `IDM` constants in the source file menuCmdID.h, or you can look at the `localization\English.xml` (or your language of choice), which lists the `<Item id="...">` next to the text of the command (as it is labeled in the localized menu system); the value of the `id` attribute is the "command ID" for that menu command, which is used as the `wParam` command ID for the `type=2` macro command.

For `type=3` search-and-replace macros, see the detailed description in "Searching > Searching actions when recorded as macros".

If your automation task requires conditional execution, counters, variables, using the results of one action to influence the next, or other complex behavior, the macro system will not be sufficient for your needs: you will need a Plugin: there are scripting plugins available in Plugins Admin that allow automating Notepad++ with the full power of a variety of programming languages behind them, or you might find a pre-existing plugin that already accomplishes your automation task, or you could write your own plugin).

`<UserDefinedCommands>`

When not empty, this node contains `<Command>` tags, which have the command string as contents. Their order is reflected in the Run menu, otherwise it doesn't matter.

Attributes for the `<Command>` tag

Position	Name	Value format	Meaning
1	name	string	The name of the Run command.
2	Ctrl	"yes"/"no"	The key mapped to has the Control modifier
3	Alt	"yes"/"no"	The key mapped to has the Alt modifier
4	Shift	"yes"/"no"	The key mapped to has the Shift modifier
5	Key	integer	The base virtual key number, in the 1 - 255 range
6	FolderName	string	Can be used to group one or more Run menu entries into a named sub-menu (new to v8.4.8)

Note: FolderName can only be entered by hand-editing the `shortcuts.xml` file; it is not available in the Shortcut Mapper or any other GUI element.

Although it is possible for several commands to have the same name, this is confusing and thus discouraged.

The run command may contain any valid command for the Windows OS. If you use a command that can be found in your PATH (like `cmd.exe`), then you don't need to specify the full path to the command. If it's not in your path, then you *should* specify the full path. Note that Windows will launch your default browser if you put a URL in this. If the command, or one of its arguments, has an embedded space, then put quotes around it (like you would for any command line environment). For example, `<Command name="Run Putty" ... >"c:\program files\putty\putty.exe" -ssh -load "my session"</Command>` shows the quotes around the executable and one of the arguments, because both have spaces. (That same command in the **Run**-menu's **Run…** action dialog would need to be `"c:\program files\putty\putty.exe" -ssh -load "my session"` to encapsulate all the paths with spaces inside double-quotes.)

Variables for Run Commands

There are a number of Notepad++-specific variables available for Run commands, which are accessed in the form `$(VARIABLE_NAME)`, which can be used to supply portions of the command entry (whether creating the command in the `shortcuts.xml` or through the **Run > Run…** entry).

Variable	Description	Example
FULL_CURRENT_PATH	The full path to the active file	`E:\My Web\main\welcome.html`
CURRENT_DIRECTORY	The active file's directory	`E:\My Web\main`
FILE_NAME	The active file's name	`welcome.html`
NAME_PART	The filename without extension	`welcome`
EXT_PART	The extension (with the `.`)	`.html`
CURRENT_WORD	the active selection in Notepad++, or the word under the caret	
CURRENT_LINE	the line number where the caret is currently located in the editor window	1
CURRENT_LINESTR	the text of the current line (added v8.3.2)	`The quick brown fox jumps over the lazy dog`
CURRENT_COLUMN	the column number where the caret is currently located in the editor window	5
NPP_DIRECTORY	the directory where the `notepad++.exe`	`c:\Program Files\notepad++`

Variable	Description	Example
	executable is located	
NPP_FULL_FILE_PATH	the full path to the `notepad++.exe`	`c:\Program Files\notepad++\notepad++.exe`

If the variable contains one or more space characters in the path, like `$(FULL_CURRENT_PATH)` being `C:\Path With Spaces\Goes\Here.txt`, then you should include quotes around the variable, like `"c:\Program Files\App Name\blah.exe" "$(FULL_CURRENT_PATH)"`

If you want access to a Windows environment variable (like `TMP`), use the standard `%`-notation for Windows variables (like `%TMP%`). If the variable references a path with one or more spaces in the name, then putting quotes around environment variables is required: for example, if `%TMP%` is defined as `C:\Users\First Last\Temp\`, then you will need to use `C:\AppDir\blah.exe "%TMP%"` as your command.

v8.5.3 `shortcuts.xml` updates

In Notepad++ v8.5.2 and earlier, if you had a "special character" in your macro or run-menu command – whether it was in the name of the macro/command or in text that it uses – then inside the XML, it would be stored as an XML entity. For example, `☺` would be stored as `&#x263a;` or `π` as `&#x03C0;`.

In Notepad++ v8.5.3, it changed to storing those characters as actual UTF-8 encoded characters, and it treats nearly all entities as raw text. Thus, `&#x03C0;` is interpreted by v8.5.3 (and newer) as 8 characters, not an entity-representation of the underlying character.

There are still a small number of entities that *will* be recognized inside macros, even in v8.5.3 and newer:

- The 5 predefined XML named-entities: `&amp;` `&lt;` `&gt;` `&apos;` `&quot;`
- Any of the hexadecimal entities that only include two hex digits, like `&#x0D;` and `&#x0A;` for `CR` and `LF`, and `&#x22;` for double-quotes. This notation works for all ASCII codepoints from 1 (`&#x01;` for the `SOH` character) through 127 (`&#x7F;` for the `DEL` character)

This change in interpretation may make it so your old macros or run-menu commands don't work as expected. You can follow the instructions in the Notepad++ Community Forum FAQ entry to see how to update your `shortcuts.xml` to be compatible with Notepad++ v8.5.3 and newer.

User Interface settings: `config.xml`

The following sections are defined:

1. `<GUIConfigs>` : user interface settings (usually set in the **Settings > Preferences**).
2. `<FindHistory>` : most of the latest state of the Find/Replace dialog box.
3. `<History>` : the list of recently used files.
4. `<ProjectPanels>` : associates workspace files with a given project panel

There are some non-UI options for advanced users, please check **Preferences for Advanced Users** to get more details.

Keyword lists: `langs.xml`

This file contains the keyword lists for syntax highlighting languages.

Attributes for the `<Language>` node

Position	Name	Value format	Meaning
1	name	string	The name of the language.
2	ext	string	The list of file extensions associated to this language by default. Lists are space separated, without leading periods.
3	commentLine	string	The character(s) that prelude a comment extending to the end of the physical line when using the **Edit > Comment/Uncomment >**

Position	Name	Value format	Meaning
			Single line actions. Set to "" if single-line comments are not supported for the current Language.
4	commentStart	string	The character(s) that start a block comment when using the **Edit > Comment/Uncomment > Block Comment/Uncomment** actions. Set to "" if block comments are not supported for the current Language.
5	commentEnd	string	The character(s) that end a block comment when using the **Edit > Comment/Uncomment > Block Comment/Uncomment** actions. Set to "" if block comments are not supported for the current Language.
6	exclude	`"yes"` / `"no"`	Set to `"yes"` to hide this Language from the Language menu; otherwise, set to `"no"` or don't include this attribute. Reflects the **Settings > Preferences > Language** having the Language in the **Disabled** list (`"yes"`) vs **Available** list (`"no"`).
7	tabSettings	integer	If present, the attribute value encodes the number of spaces a tab is equivalent to, reflecting **Settings > Preferences > Indentation > Indent Settings** for the active language. (More on this attribute, below)

Position	Name	Value format	Meaning
8	backspaceUnindent	`"yes"` / `"no"`	If present, reflects the state of the **Settings > Preferences > Indentation > Indent Settings > ☐ `Backspace key unindents...`** checkbox. (New to v8.6.9)

The value used in the `tabSettings` attribute value depends on the **Settings > Preferences > Indentation > Indent Settings**:

- If it is set to use `☑ Tab character`, then the value stored in this XML attribute is the same as in the preference's **Indent Size** value.
- If it is set to use `☑ Space character(s)`, then the value stored in this XML attribute is 128 higher than the preference's **Indent Size** value (so a value of `4` in the preference dialog would be stored here as `tabSettings="132"`).
- If it is set to use `☑ Use Default Value`, then this attribute will be `tabSettings="-1"`.
- If this `<Language...>` entry is missing the `tabSettings` attribute, it will behave as if it had `tabSettings="-1"`.

Inside each of the languages, you *could* add keywords. However, it's better to use **Settings > Style Configurator** and make use of the user-defined keywords box for a given category (when available). These user-defined keywords are stored in `stylers.xml` (described below).

The order of the `ext` list here determines the order of extensions in the file-type pulldowns of the Windows-common-dialogs like **Open**, **Save**, and **Save As** dialogs. The automatically-added extension will be the first extension from this `ext` list.

The `commentLine`, `commentStart`, and `commentEnd` attributes are used by the Edit Menu's **Comment/Uncomment** actions for adding or removing the comment syntax to the selected lines of text. These are independent of the selected programming **Language**'s syntax lexer, so changing these attributes will *not* affect what code sections will get highlighted as comments by the active syntax highlighter. (Because of the way the Scintilla library that Notepad++ uses for the lexers, in order to change what gets highlighted as a comment, one would need to edit and re-compile the lexer for the chosen and then rebuild the Notepad++ project; in other words, it would have to be changed by the developers for a new release, not changed by the user in the `langs.xml` configuration file.) These attributes can only define one type of line and block comments; if your language has multiple types of either, you will have to choose the *one* type that you'd like Comment/Uncomment to work with.

Highlighting schemes: `stylers.xml`

This file sets the color scheme for the default theme. The other themes are stored in `themes\*.xml`, which follow the same format at `stylers.xml`. In general, use **Settings > Style Configurator** for easier maintenance of styles.

Each lexer type has its own `<LexerType>` section, with multiple `<WordsStyle>` entries. Each lexer from the **Language** menu has it's own list of available `<WordsStyle>` entries. Trying to add a new `<WordsStyle>` to a lexer to try to get more categories of keywords will *not* be successful, because the underlying code which does the syntax highlighting has no internal rules for mapping the entries found to that style.

If you have added user-defined keywords in the **Settings > Style Configurator**, they will be stored as the contents of the `<WordsStyle>`, as a space-separated list (for example, `<WordsStyle>fancyKeyword1 fancyKeyword2</WordsStyle>`).

The `<WordsStyle> fontStyle` attribute encodes the setting of the **Bold**, **Italic**, and **Underline** checkboxes from the **Styler** dialog, using the sum of **Italic**=1, **Bold**=2, and **Underline**=4 (so something with all three checkboxes set would have `fontStyle="7"`).

The `<WordsStyle> colorStyle` attribute decides whether to use the defined colors from `fgColor` and `bgColor` attributes, or to use the default color setting (from **Settings > Style Configurator > Global Styles > Default Style**). The attribute should be set to one of the following:

- No `colorStyle` attribute: this style will use both the `fgColor` and `bgColor` attributes from this `<WordsStyle>` item (standard behavior)
- Set `colorStyle="2"`: this style will inherit the foreground color from the Default style, and use the `bgColor` value as the background color (equivalent to right-clicking the foreground color in the UDL styler dialog box)
- Set `colorStyle="1"`: this style will inherit the background color from the Default style, and use the `fgColor` value as the background color (equivalent to right-clicking the background color in the UDL styler dialog box)
- Set `colorStyle="0"`: this style will inherit both the foreground and background colors from the Default style (equivalent to right-clicking both the foreground and background colors in the UDL styler dialog box)

Function List

Defines what counts as a "function" for **View > Function List**. There are some comments in the file(s), and lots of examples of the builtin languages, which you can customize.

Function List Definitions

The `functionList` folder contains a separate XML file (function list parser definition) for each language's function list capability. Each function list parser definition links to a language with the language default name. For example the file name of php language parse rule is `php.xml`, the file name of Java language parse rule is `java.xml`, Check overrideMap.xml for the naming list of all supported programming languages.

For built-in languages that already have Function List behavior, editing overrideMap.xml is optional: if you edit the existing `functionList\XYZ.xml` for that language, you don't need to edit `overrideMap.xml`; however, if you create a new `functionList\my_XYZ.xml` that you want to use instead of the default function list definition for XYZ, then you need to copy the an `<association id="{filename.xml}"... />` for language XYZ from inside the `<!-- ... -->` comment block, paste it outside that block, and point the to the new file using `id="my_XYZ.xml`. If you need to link to a function list parser name "udl_ABC.xml" for your User Defined Language (UDL) named "ABC", you need to modify this file to add an `<association id="udl_ABC.xml" userDefinedLangName="ABC">`: the `id` must match the filename exactly, and the `userDefinedLangName` must match the name of your UDL, as seen in the **Language** menu and UDL dialog.

- Example: Override default parser definition files for two languages:

  ```
  <association id= "anotherPhpParser.xml"                    langID= "1" />
  <association id= "myPerlRule.xml"                          langID= "21"/>
  ```

  If those two are in `overrideMap.xml` (and not commented out), then function list will load your parsers `anotherPhpParser.xml` and `myPerlRule.xml` instead of loaoding `php.xml` and `perl.xml` while showing PHP and Perl function lists, respectively.

- Example: Set the parser definition file for your UDL called "KRL":

```
<association id= "krl.xml"                              userDefinedLangNa
```

Here you define a parser rule file name for your KRL UDL. When you open a file which is recognized as a KRL file, then the function list engine will load `functionList\krl.xml` to show the KRL function list. If you have no KRL UDL defined in your Notepad++, you have to define a dummy one (with the name "KRL") to make it work.

The `functionList\` _languagename_ `.xml` parser file itself, whether it's for a builtin language or a UDL-based language, requires the structure `<NotepadPlus><functionList><parser...>...</parser></functionList></NotepadPlus>`, where the attributes and contents of the `<parser>` are described in the documents section about How to Customize Function List. You can look at any of the default parser files for examples of working Function List configurations.

For plain text files (ones with **Language > None (Normal Text)** selected), you can have a function list definition in `functionList\normal.xml`, or you can use `langID="0"` in your `functionList\overrideMap.xml` to assign it to some other file. For example, if you've defined your normal-text functions in `functionList\someOtherNormalText.xml`, you would use the following override:

```
<association id= "someOtherNormalText.xml"              langID= "0" />
```

Upgrading old Function List entries

If you previously had a v7.9-or-earlier style function list entry in `functionList.xml`, and you want to use it in a modern v8.x version of Notepad++, you can extract the pieces to the right locations in the new multi-file format:

1. Open the old `functionList.xml`
2. Open the `functionList\overrideMap.xml`
3. Copy the `<association...>` tag from the old `functionList.xml` to the `functionList\overrideMap.xml`, and place near the end of the `<associationMap>` section. Make sure it follows the rules for modern `<association>` tag syntax
4. Open the `functionList\blah.xml` for your particular language
    - If you don't have `blah.xml` yet, copy another language's XML file from the most-recent Notepad++ version to `blah.xml`, and remove the whole `<parser...> ... </parser>`

section

5. Copy the `<parser...>...</parser>` section from the old `functionList.xml` to the `functionList\blah.xml`
    - Please note that the `blah.xml` should *not* contain a `<parsers>` section, *just* the `<parser>` section. It will cause problems with the Function List if you wrap it in the `<parsers>...</parsers>` block. Make sure it ends up with the structure described above.

Toolbar Customization

Toolbar Icon Visibility: `toolbarButtonsConf.xml`

This configuration file allows you to override the visibility of any of the default or plugin toolbar icons. (New to v8.7.8)

The installation directory comes with a file called `toolbarButtonsConf_example.xml` : you can copy that to `%APPDATA%\Notepad++\toolbarButtonsConf.xml` (or other appropriate Config Files Location). Once copied, edit `toolbarButtonsConf.xml` , edit it (see details below), and save, then restart Notepad++ to see the changes go into effect.

Set the `<Standard hideAll="...">` attribute to `yes` or `no` to decide whether or not to hide all of the built-in toolbar icons. Set `<Plugin hideAll="...">` attribute to `yes` or `no` to decide whether or not to hide all of the toolbar icons from all plugins.

You can change the `hide="no"` to `hide="yes"` (or vice versa) on any of the `<Button>` elements to toggle the visibility on specific toolbar buttons.

- For the `<Standard>` buttons, you can edit the `hide` or `name` attribute values, but **do not** edit the `index` or `id` values (which are required in order to properly map). The `name` is based on the default English name for that command, but you can change that value if it helps you understand which command you are editing – the `name="..."` value you set will *not* affect the Notepad++ GUI at all.
- For the `<Plugin>` buttons, you can edit the `hide` value to be `yes` or `no` as appropriate; and the name, as with `<Standard>` buttons, is solely for your reference and never used by Notepad++.

- The *order* of the `<Button>` tags corresponds exactly to the order that the plugin buttons are on the toolbar; there is no `index` or `id` attribute to help with mapping, and the `name` is ignored (and is just a useful reference for you).
- You must keep the `hide="yes"` entries lined up with the appropriate plugin buttons, as you install or upgrade your plugins, otherwise you will likely end up hiding the wrong buttons at some point.

Toolbar Icon Customization

Notepad++ also allows customization of the icons used for the toolbar buttons (new to v8.4.2). This allows you do define your own set of icons to use on the toolbar, and is useful (for example) if none of the five different icon sets available through **Settings > Preferences > General > Toolbar** are sufficient for your needs.

- In v8.4.2 - v8.7.9, the configuration file `toolbarIcons.xml` was used
- In v8.8 and newer, this was merged into the `toolbarButtonsConf.xml` (above), so the same will be used for both Toolbar Customization tasks.
  - If you had `toolbarIcons.xml` in an earlier version, and you upgrade to v8.8 or newer, you will have to copy the `<ToolBarIcons icoFolderName="myAwesomeIcons" />` from the old `toolbarIcons.xml` file into `toolbarButtonsConf.xml`

Aside from the config file, you need to create icons. You will populate and place the configuration file and icon files as described below:

1. Put the file `toolbarButtonsConf.xml` (Note 1) in the main configuration folder (Note 2) (called `<CONFIGDIR>` in the examples below).
2. Create a new sub-folder called `toolbarIcons\` in that same folder.
3. Edit the file `toolbarButtonsConf.xml`: put the icon set name you want in the `icoFolderName` attribute (Note 3). for example: `<ToolBarIcons icoFolderName="myAwesomeIcons" />`
4. Go into `toolbarIcons\` folder and create a new folder with the exact name of the icon set name you provided in `icoFolderName`, for example `<CONFIGDIR>\toolbarIcons\myAwesomeIcons\`.
5. Put all your customized icons into `<CONFIGDIR>\toolbarIcons\myAwesomeIcons\`.
6. Now it is the magic moment: Relaunch Notepad++ and you'll see your icon set instead of the default icons.
   - This overrides the icons for any of the **Settings > Preferences > General > Toolbar** icon-set selections.

- If you have **Settings > Preferences > General > Toolbar** set to any of the three “small” choices, it will scale the icon to 16x16; if you use one of the two “large” choices, it will scale to 32x32. So if you are going to use a “large” icon set, you should make sure the icons are defined as 32x32.
- The v8.4.2 release page allows you to download a bundle that contains the legacy 32x32 icon set along with the v8.4.2-style `toolbarIcons.xml`; unzip that bundle into the directory described on that page (equivalent to the main configuration folder [Note 2]), merge `toolbarIons.xml` into the `toolbarButtonsConf.xml` as described above, restart Notepad++, and you will have the pre-v8.0-style 32x32 “Big Icons” (but they are different icons than just a big version of the “standard icons”, sorry).
- Troubleshooting: if you started Notepad++ with one of the “small” choices selected but have 32x32 icons in your custom icons, it will scale them to 16x16, which is fine for small modes; but if you switch to a “large” option, it might scale the 16x16 back to 32x32 rather than using the true-32x32 from the icon file, which makes for a pixelated (“klunky”) large icon: if this happens, leave it with “large” selected, then exit Notepad++ completely and restart: on the subsequent starts, it will use the full 32x32 resolution.
- Transparency: To get part of the icon transparent, you need to make sure your icon editor is able to save in 32bpp mode (8-bits each for alpha/transparency, red, green, and blue). Some icon editors only support 24bpp (no alpha/transparency information). Alternately, some editors will allow you to use a 8bpp (256-color palette) with 1 of those 256 indexes allocated to transparency; using this format will allow creating much smaller `.ico` files, assuming 255 colors is sufficient for your icon needs.
- Recommendation: Every `.ico` file used for custom toolbars should include a 16x16 image and a 32x32 image, with color depth set to either 32bpp or 8bpp-with-transparency, so that you have full control of the exact appearance on large and small toolbars, and allow transparent pixels.

Notes:

- Note 1:
  - For v8.8 and newer, `toolbarButtonsConf.xml` will have:

```
<?xml version="1.0" encoding="UTF-8" ?>
<NotepadPlus>
    <ToolbarButtons>
        ...
    </ToolbarButtons>
    <ToolBarIcons icoFolderName="myAwesomeIcons" />
</NotepadPlus>
```

You can copy the example from `<installationDir>\toolbarButtonsConf_example.xml` to use as a start for `toolbarButtonsConf.xml`. This example file has plenty of comments, to help you understand what's needed.

- The content of the v8.4.2-v8.7.9 `toolbarIcons.xml` is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<NotepadPlus>
    <ToolBarIcons icoFolderName="myAwesomeIcons" />
</NotepadPlus>
```

- Note 2: This is the same folder descibed in Configuration Files Locations where `config.xml` goes, and will generally be the `%APPDATA%\Notepad++\` directory, unless you are using local configuration or cloud configuration or overriding the configuration directory with `-settingsDir`.

- Note 3: If the `icoFolderName` value is an empty string, the path of icons will be `<CONFIGDIR>\toolbarIcons\default\` folder.

Set of Icons

For each of the 45 toolbar icons that can be customized, use the specific file name listed below. (Some toolbar buttons have two icons, one for when the button is enabled and one when the button is disabled.)

index	Normal icon	Disabled icon
1	new.ico	
2	open.ico	
3	save.ico	save_disabled.ico
4	save-all.ico	save-all_disabled.ico
5	close.ico	

index	Normal icon	Disabled icon
6	close-all.ico	
7	print.ico	
8	cut.ico	cut_disabled.ico
9	copy.ico	copy_disabled.ico
10	paste.ico	paste_disabled.ico
11	undo.ico	undo_disabled.ico
12	redo.ico	redo_disabled.ico
13	find.ico	
14	replace.ico	
15	zoom-in.ico	
16	zoom-out.ico	
17	sync-vertical.ico	
18	sync-horizontal.ico	
19	word-wrap.ico	
20	all-chars.ico	
21	indent-guide.ico	
22	udl-dlg.ico	
23	doc-map.ico	
24	doc-list.ico	
25	function-list.ico	
26	folder-as-workspace.ico	
27	monitoring.ico	monitoring_disabled.ico

index	Normal icon	Disabled icon
28	record.ico	record_disabled.ico
29	stop-record.ico	stop-record_disabled.ico
30	playback.ico	playback_disabled.ico
31	playback-multiple.ico	playback-multiple_disabled.ico
32	save-macro.ico	save-macro_disabled.ico

It is not necessary to have a complete set of 45 icons in the directory: any icons not in the directory will just use the built-in icon instead.

You can have multiple icon set directories; to switch between icon sets, you just edit the `icoFolderName`, save the config file, and relaunch Notepad++.

Other Configuration Files

- `asNotepad.xml`: This is a zero-byte file that is used as an indicator to Notepad++. When this file is present, Notepad++ will run as if the `-multiInst -nosession -notabbar` command line options were used, allowing you to focus on editing a single file at a time. (But please note that opening or creating a file from one of those “single file” instances will still allow you to have multiple files open in that instance of Notepad++, even when there is no tabbar to show them to you; you can navigate to the other files by the **Window** menu, **View > Tab…**, and their equivalents.) This config file *must* go in the Notepad++ installation folder; it will not be recognized in the `%AppData%\Notepad++` hierarchy or in the cloud settings folder.

- `autoCompletion\*.xml`: Files for defining per-language auto-completion. This config folder *must* go in the Notepad++ installation folder; it will not be recognized in the `%AppData%\Notepad++` hierarchy or in the cloud settings folder.

- `cloud\choice` file: If this folder and file are present, the file contains the path to the Cloud folder. This config file is created when you define the Cloud preferences. If you are in a normal installation, this config file will be `%AppData%\Notepad++\cloud\choice`; if it is deleted from there, Notepad++ will *not* look for `<installdir>\cloud\choice`. If you are in a portable / local configuration (`doLocalConf.xml` exists), it will be in `<portable>\cloud\choice`.

- `config.model.xml`: If it exists, Notepad++ will use the `config.model.xml` from the installation directory as the default configuration when creating a new per-user `config.xml`; if the `config.model.xml` does not exist in the installation directory, Notepad++ has a set of default internal values that it will use to populate the per-user `config.xml`. This is useful if you have multiple users on the same PC, and want them all to start with the same configuration choices when they first use Notepad++; it can also be used if you want to bundle your own customized Notepad++ installation directory to distribute to multiple PCs, then when each user first uses Notepad++ on the new computer, they will start with those values in their user `config.xml`.

- `disableLineCopyCutDelete.xml`: This is a zero-byte file that is used as an indicator to Notepad++ to not enable the context-aware "line copy / cut / delete" feature in v8.6.1. This file needs to be in `%AppData%\Notepad++\` for a normal installation, and in the portable directory for a local configuration. The only version of Notepad++ to pay attention to this file is v8.6.1; in earlier or later versions, this file is ignored and meaningless. (It was added because v8.6 had some strange behavior with the context-aware Cut/Copy/Paste, so v8.6.1 added a workaround for that difficulty by allowing users who had problems with it to disable the context-aware actions. However, v8.6.2 fixed the underlying bug, so this workaround was no longer necessary and hence removed.)

- `doLocalConf.xml`: This is a zero-byte file that is used as an indicator to `notepad++.exe` to not go looking for `%AppData%`. This will only exist on local installations of Notepad++ (when you tell the installer to not use `%AppData%`, or when you install from the zipfile, or when you manually create the zero-byte file). This config file *must* go in the Notepad++ installation folder; it will not be recognized in the `%AppData%\Notepad++` hierarchy or in the cloud settings folder. (If this file is deleted, the instance will go back to using the normal `%AppData%` location the next time Notepad++ is run.)

- `enableSelectFgColor.xml`: This is a zero-byte file that is just used as an indicator to the **Settings > Style Configurator > Global Styles > Selected text color** to honor the foreground color, not just the background color. This config file *must* go in the per-user `%AppData%\Notepad++` hierarchy or in the cloud settings folder or `-settingsDir` folder: it will *not* be recognized if it's in the Notepad++ installation folder (unless `doLocalConf` is in effect). (Available on v8.0.0 - v8.7.9. Replaced by **Settings > Preferences > Editing 1 > ☐ `Apply custom color to selected text foreground`** in v8.8.)

- `nativeLang.xml` : If you make a selection in the **Settings > Preferences > General > Localization**, Notepad++ will copy the appropriate `localization\*.xml` to `nativeLang.xml` .

- `noColumnToMultiSelect.xml` : This is a zero-byte file that is used as an indicator to Notepad++ to not enable the column-mode to multi-edit conversion feature (new to v8.6.1). This file needs to be in `%AppData%\Notepad++\` for a normal installation and in the portable directory for a local configuration. Starting in v8.6.3, this zero-byte file was replaced by **Settings > Preferences > Editing 2 > ☐ Enable Column Selection to Multi-Editing**; if you used `noColumnToMultiSelect.xml` in v8.6.1-v8.6.2, to maintain the same behavior, you will need to make sure the new setting is not checkmarked; as of v8.6.3, this file is ignored and does nothing.

- `noEasterEggs.xml` : This is a zero-byte file that is used as an indicator to `notepad++.exe` to not show the "Easter Eggs" when trying to access the **About Notepad++** dialog (accessed from the **?** menu or the default keyboard shortcut `F1`). This config file *must* go in the Notepad++ installation folder; it will not be recognized in the `%AppData%\Notepad++` hierarchy or in the cloud settings folder. You can find out more about the Easter Eggs in the Command Line Options and Ghost Typing descriptions.

- `noRestartAutomatically.xml` : Starting in Notepad++ v8.5.8, Notepad++ is a Restarable App. If you would like to *disable* Notepad++ from being restartable, add an empty config file called `noRestartAutomatically.xml` into `%APPDATA%\Notepad++\` (for normal installations) or the Notepad++ installation directory (for other configuration settings). (See also Undoing Portable, above.)

- `nppLogNulContentCorruptionIssue.xml` : This is a zero-byte file that allows Notepad++ to write a logfile to `%AppData%\Notepad++\nppLogNulContentCorruptionIssue.log` in the case of certain crashes, which can help the developers debug issues resulting from the crash. If you want to prohibit that logfile from being created, delete this config file; but that will mean that you will not be able to provide the useful information to the developers if a crash causes issues for you; delete at your own risk. This config file *must* go in the Notepad++ installation folder; it will not be recognized in the `%AppData%\Notepad++` hierarchy or in the cloud settings folder. (New to v8.1.9.3.)

- `nppLogNetworkDriveIssue.xml` : This is a zero-byte file that allows Notepad++ to write a logfile to `c:\temp\nppLogNetworkDriveIssue.log` in the case of certain network drive issues. (New to v8.1.9.3.)

  - This should be used if you are getting “This file has been modified by another program” when dealing with network drives (like a Samba server, or other filesystems access by `\\machinename\path\`).
  - Create this zero-byte file either next to `notepad++.exe` or in the `%AppData%\Notepad++` directory.
  - Create directory `c:\temp\` if it doesn’t exist.
  - Start `notepad++.exe` , and wait until you get the file status (timestamp) detection error from the network drive.
  - If the errors occur, there should be some trace in “C:\temp\nppLogNetworkDriveIssue.log”.
  - This log can be shared with the developers to help them improve Notepad++'s network handling (create an issue here and attach the logfile along with your other details).

- `session.xml` : Stores the current session information. Overwritten on every exit of Notepad++ if **Settings > Preferences > Backup > Remember current session for next launch** is enabled. If you want sessions that you control, use **File > Save Session…** to save it; the file is safe to edit; and you can reload that session at any time using **File > Load Session…**. This config file is confusing for its location: a normal installation will keep it in `%AppData%\Notepad++\` ; and the `-settingsDir` will specify the folder where it will live; a portable installation (one with `doLocalConf.xml`) will keep `session.xml` in the same directory as `notepad++.exe` ; but if you use the Cloud settings, `session.xml` will *not* honor that directory, and will instead go in either the AppData or the portable location.

- `userDefineLang.xml` and `userDefineLangs\*.xml` : Configuration location for the **User Defined Languages** feature.

- `v852NoNeedShortcutsBackup.xml` : Tracks whether your macros definitions in `shortcuts.xml` have been updated per the v8.5.3 rules

  - This file is created the first time you save a recorded macro after upgrading from Notepad++ v8.5.2 (or older) to Notepad++ v8.5.3 (or newer)
  - When this file is created, it will also create `shortcuts.xml.v8.5.2.backup`
  - After this file is created, Notepad++ won’t check any more whether `shortcuts.xml` has been updated
  - See v8.5.3 `shortcuts.xml` updates (above) and this Notepad++ Community FAQ entry for more details

Logfiles

Notepad++ doesn't normally create logfiles. However, there are some circumstances when it does:

- As described above, if `nppLogNulContentCorruptionIssue.xml` exists, then the logfile `%AppData%\Notepad++\nppLogNulContentCorruptionIssue.log` will be created during crash events. (New to v8.1.9.3.)
- As described above, if `nppLogNetworkDriveIssue.xml` exists, then `c:\temp\nppLogNetworkDriveIssue.log` will be created during certain network drive issues. (New to v8.1.9.3.)
- Starting in v8.9, security errors will be automatically logged to `%LOCALAPPDATA%\Notepad++\log\securityError.log`.

Validating Config-File XML

If you are developing a config file by editing the raw XML file and would like to be able to validate that you have correct XML syntax while you are doing so, you can see the instructions in the Notepad++ Community "Validating Config-File XML" FAQ.