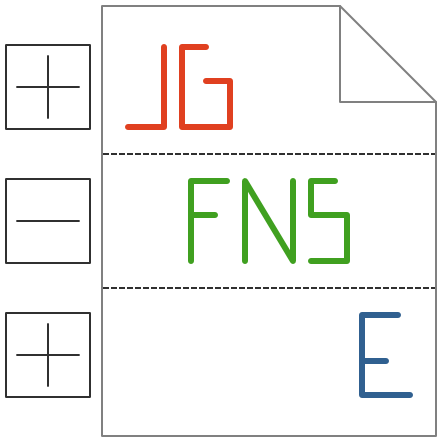

Just Great Software

File Navigation Scheme Editor

Manual

Version 3.0.3 — 24 January 2020

Table of Contents

1. Introduction

With the Just Great Software File Navigation Scheme Editor, you can build and edit file navigation schemes for EditPad Pro. Version 3 of the scheme editor creates schemes compatible with EditPad Pro 8.0.0 and later. These schemes are used by the File Navigator, which you can open via its menu item in the View menu. The File Navigator is only available when you have associated a file navigation scheme with the current file's file type. You can do so via the Configure File Types item in the Options menu. Select the file navigation scheme on the Navigation tab.

EditPad Pro only allows you to select which file navigation scheme to use. It doesn't offer any configuration or customization options. The only way to customize the way the file navigator works is by creating your own file navigation scheme, or editing one of the schemes included with EditPad Pro. This is also the reason why EditPad Pro ships with multiple schemes for certain file types. While they process the same files, they create different file navigation trees.

You can find the file navigation schemes that ship with EditPad Pro in the folder where you installed the software. Each .jgfns file contains one file navigation scheme. It is a good idea to study some of those schemes before attempting to create your own. You can download schemes created by other EditPad Pro users at https://www.editpadpro.com/fns.html.

Schemes that you have downloaded are placed in product-specific folders under %APPDATA%\JGsoft. You can navigate to this folder by typing or pasting %APPDATA%\JGsoft in the address bar in Windows Explorer. You should place your own custom schemes in that folder too. You can do this easily by selecting EditPad Pro 8 in the drop-down menu of the **Save As** button in the scheme editor. If a .jgfns file with the same file name exists in both the AppData folder and the Program Files folder, then the file in the AppData folder takes precedence.

2. File Navigation Scheme Structure

Just Great Software file navigation schemes use a structured format that is four levels deep. The top level is the scheme itself. At this level, you'll enter general information about the scheme. This information helps people select the scheme they're looking for.

All schemes have at least one layout. But most schemes have many layouts. Layouts are used for grouping elements. Typically, your scheme will have one layout for each kind of region in your files that follows different syntactic rules, or that you just want to lay out differently in the file navigation tree.

Each layout in your scheme has one or more elements. Elements do the actual work of parsing files into a file navigation tree. Each element has one regular expression. All elements in a layout are applied at the same time. When an element matches, the element's rules decides whether the file navigator switches layouts, creates foldable regions, or adds nodes to the tree.

Finally, each element can have any number (including zero) of nodes attached to it. Nodes can be nested into a tree of nodes. These nodes are the nodes that are put into the actual file navigation tree displayed by the file navigator in EditPad Pro. All nodes below an element are added to the tree when the element's regular expression matches. Therefore, you can view the whole tree of nodes below an element as a single level in the scheme's four level structure.

As an example, open the file C.jgfns that is included with EditPad Pro in the scheme editor. The scheme is "C". It has four layouts. The first layout is also named "C". This layout has 10 elements. The elements "Include", "Define", and "Function" have nodes. The nodes don't have names. They just show the text they add to the tree (with the backreference `\1` not yet substituted). The other elements in the first layout do not have nodes. The "Preprocessor", "typedef", and "start of block" elements serve to toggle the other three layouts in the scheme. The remaining elements match comments and strings to prevent the other elements from finding matches inside comments and strings.

3. Scheme

Click on the root node in the structure tree to enter general information about the scheme. This information helps people select the scheme they're looking for.

Scheme Name: The name of your file navigation scheme. This name is displayed on the Navigation tab in the Configure File Types screen in EditPad Pro. Each scheme should have a unique name. Otherwise you won't be able to distinguish between schemes with the same scheme name in EditPad Pro. EditPad Pro identifies schemes by their file name. Entering a new scheme name won't break any file type configurations that reference the scheme. Renaming the file on disk will.

File Extensions: The file extensions typically used by files to which your file navigation scheme should be applied. Note that this item is purely informational. Any scheme can be applied to any file, no matter its extension. You should list extensions as full file masks, delimited by semicolons, e.g.: *.html;*.htm;*.shtml

File Type Web Site: Complete URL of the web site that is the central source of information about the file type your scheme enables file navigation for. Try linking directly to a page with relevant information, rather than a top-level domain, unless the top-level domain is specific to the file type.

Author Name: Your name.

Author Email: Email address that can be used to send you comments about your scheme. Your email address is not shown on the EditPad Pro web site when you upload your file navigation scheme.

Author Web Site: Complete URL of your web site.

Comments: If there's anything important that people should know before using your scheme, you can enter that information into the Comments box. Particularly if you're creating a scheme for a file type for which one or more schemes are already available, explain what makes your scheme different. Since this text is displayed on the web site when people download file navigation schemes, please be brief, so the web page listing the schemes doesn't become too long.

4. Layout

The HTML.jgfns file included with EditPad Pro is an example of a scheme with a single layout. The Delphi.jgfns scheme has a very large number of layouts. How many your scheme needs depends on the complexity of the file format you're working with, and how detailed you want the file navigation tree to be.

When EditPad Pro begins applying your file navigation scheme, it starts with the first (topmost) layout in your scheme. Other than that, the first layout is not treated differently in any other way compared with the other layouts in your scheme.

Click the Layout button in the toolbar to add a new layout to your scheme. Click the Up and Down buttons to change the order of the layouts. The order of the layouts is irrelevant, except when multiple regular expressions reference the same named capturing group. Right-click on a layout in the scheme structure to cut, copy and paste layouts. Layouts are copied and pasted in their entirety, including elements and their nodes. You can start multiple instances of the Just Great Software File Navigation Scheme Editor, and copy and paste layouts between them.

Each layout has a **name** that identifies the layout in the scheme editor. Names among layouts should be unique so that you can select the correct layout when an element needs to toggle to a layout.

Each layout must have at least one element. The scheme editor enforces this by automatically adding a blank element to each layout you add. When you delete the last element from the layout, a new blank element is automatically added.

Layouts can **include the elements of other layouts**. If two layouts need to contain the same elements, you could put those into a separate third layout and then have the two layouts include the third. The Include Layouts list shows all layouts other than the active layout that do not include other layouts. Tick the checkboxes next to the layouts that you want to include in the active layout.

Only one layout is active at any time while EditPad Pro applies your scheme. EditPad Pro scans the file for the first (leftmost) regular expression match. While advancing through the file, EditPad Pro attempts to match the regular expression of all the elements in the layout at each character position. The elements are tried in the order you gave them in the scheme editor, from top to bottom. If multiple elements match at different character positions in the file, the element that matches at the earliest (leftmost) character position is considered the matching element. If multiple elements match at the same character position, the topmost element in the layout structure is considered the matching element.

Basically, this means that if the regular expressions of two elements can match at the same position in the file, the more specific element should be placed above the generic element. An element matching a list of keywords, for example, should be placed above an element matching any word. Otherwise, the element matching any word would also match all the keywords, effectively disabling the element with the list of keywords.

If two the regular expressions of two elements could never match starting at the same position, such as an element matching a comment and an element matching a keyword, then the order of those elements is irrelevant. If the keyword occurs in a comment, the comment element will always match first, at the spot where the comment starts. The keyword element will fail at that position, and thus never get a chance to match the keyword inside the comment.

If a layout includes another layout then the elements of the included layout are considered to be placed below the elements that are directly part of the active layout. If a layout includes multiple layouts, then the order of the layouts in the scheme determines the order of the elements included from different layouts.

When a matching element has been found, EditPad Pro will apply the rules you set for that element, and add its nodes to the file navigation tree.

If no matching element can be found, EditPad Pro stops scanning the file, and considers the file navigation tree to be complete. No other layouts will be attempted.

5. Element

To add an element to a layout, click on the layout or one of its elements, and then click the Element button on the toolbar. Click the Up and Down buttons to change the order of the elements in the layout. As explained in the layout{linkID=50} topic, the order of the elements is significant if the regular expressions of multiple elements can match at the same position in the file. Right-click on an element to cut, copy or paste it, including all its nodes (if any). You can copy and paste elements between layouts to move or reuse them.

Elements are where all the action is. They provide a lot of options.

Name: The name of the element. This is only used to identify the element in the scheme structure while editing the scheme.

Text & Background: The text matched by this element's regular expression will be highlighted using these colors when you preview the scheme in the scheme editor. This makes it easier to debug your scheme. The colors are not used in EditPad Pro.

Comment: Anything you want to remind yourself or others of when editing the scheme.

Regex: The regular expression for this element, when EditPad Pro applies the layout. See the layout topic to learn how this regular expression is used.

Case insensitive: When turned on, the regex is applied without regarding any difference to uppercase and lowercase letters.

Free-spacing: Turn this on to ignore spaces and line breaks in the regular expression, unless they're part of a character class or preceded by a backslash. You can use this to make complex regular expressions more readable by using multiple lines with indentation. In this mode, # starts a comment unless it is in a character class or preceded by a backslash.

Dot matches line breaks: In a regular expression, the dot . matches any character except line break characters. If the dot should match everything, including line breaks, tick "dot matches line breaks".

Layout mode: If your scheme consists of multiple layouts, setting the "layout mode" for elements is the only way to make EditPad Pro switch between layouts. Select "toggle at the start/end of the match" to switch to a particular layout whenever the element's regex is matched. Select "toggle back and the start/end" to switch back to the layout that switched to the element's layout, whichever layout that was. By combining "toggle at" and "toggle back", you can nest layouts to any depth. The XML.jgfns file included with EditPad Pro uses this method to parse nested XML tags.

Instead of toggling, you can select "detail the match with a layout". This will apply the selected layout to the selected group, rather than to the remainder of the file. If the detail layout toggles, the layouts it toggles to will also be limited to the group. When no further matches can be found inside the group, EditPad Pro continues at the end of the group with the current element's layout.

Layout: When you set the "layout mode" to "toggle at the start/end of the match", or to "detail the match with a layout", you need to select the layout to toggle to or to detail the match with. This layout can be the same layout as the one containing the current element. If the layout has another element with "layout mode"

set to "toggle back", you can nest layouts into themselves. This is very useful for matching items recursively, which cannot be done with a single regular expression.

The "Block" layout in the Perl.jgfns scheme does this. The "nested block" element toggles (forward) to the "Block" layout itself whenever an opening brace is found. The "Closing }" toggles back to the previous layout whenever a closing brace is found. The text "sub {{ }}" will cause the "sub" element in the "Perl" layout to toggle to the "Block" layout. The second { causes "nested block" to toggle to "Block". The first } causes "closing }" to toggle back to "Block". The second } causes "closing }" to toggle back to the initial "Perl" layout. This way an arbitrary number of nested braces can be matched. The Perl.jgfns scheme uses this to make entire subroutines in Perl scripts foldable.

Layout group: When you set "layout mode" to anything but "continue with current scheme", you can specify the position at which the layout the element toggles (back) to should start. If "layout mode" is set to detail, the group's range (start and end) is used as the range for the detail layout. If your regular expression uses any capturing groups, their numbers will be available in this list. If it uses named capturing groups, the names of the groups will be available too. This means that you can make the next layout start at any position in the regex match, simply by putting that part of the regex inside a capturing group. See the regular expression tutorial in EditPad Pro's help file to learn how to use capturing groups.

Preserve named capturing groups: If the element toggles to another layout, which in turn will toggle back to the current element's layout, you can turn on this option to restore the matches held by named capturing groups to those held at the moment the current element caused the switch to the other layout. Turn on this option when you're using named capturing groups to carry over regex matches between elements, e.g. to carry over the name of an opening tag to the element looking for its closing tag. The XML.jgfns scheme included with EditPad Pro does this. The preservation option works at the level of each toggle, rather than at the layout level. This means that you can nest layouts to arbitrary depths and even recursively, and still preserve capturing groups at each point.

Toggle back node level steps: If the element's nodes change the current node level to be below one of the element's nodes, you can use this option to change the node level back the way it was when the layout the element toggles to, toggles back to the current element's layout. This is useful when nodes can be nested to arbitrary depths. The XML.jgfns scheme included with EditPad Pro does this. Tags are nested inside one another. When an opening tag is matched, the node level is changed to be below the new tag's node in the file navigation tree. When the closing tag is matched, the closing tag element toggles back to the layout with the opening tag element. The opening tag element has "toggle back node level steps" set to one. Therefore, when toggling back, the opening tag's parent node is made the current node again. This way, subsequent matches of the "opening tag" element in the same layout create sibling nodes, while still allowing any number of subnodes to be added under each node.

Short-circuit mode: You can select this option for any element that has its "layout mode" set to "continue with the same layout". If you've specified a short-circuit regex for any element, you can set this option to check all short-circuit regexes of all layouts that are in the stack of layouts to be toggled back to. E.g. if L1 toggles to L2 which toggles to L3 which toggles back to L2 which toggles to L4, then layouts L1 and L2 are in the layout stack. When you turn on "short-circuit mode" for an element in layout L4, the short-circuit regexes for L2 and L1 will be attempted, in that order. If a short-circuit regex matches, EditPad Pro will toggle back all the way to that layout.

Short-circuit group: When you've set "short-circuit mode" to "peek at the start/end of the match", you can select the position at which the short-circuit regexes should be tried. They will only be tried at exactly that position. You can select a numbered or named capturing group, like you can for the "layout group" option.

Short-circuit regex: When you've set the "layout mode" to "toggle at the start/end of the match", you can specify a short-circuit regex. If the layout the element toggles to, or any layout subsequently toggled to, has an element with its "short-circuit mode" set to peek, this element's short-circuit regex will be attempted. It will only be attempted at exactly that spot indicated by the "short-circuit group" in the element that peeks. When the short-circuit regex matches, EditPad Pro immediately toggles back all the way to the layout of the element with the matching short-circuit regex. The layout will continue at the "short-circuit group" position.

When layouts are short-circuited, EditPad Pro will process the "preserve named capturing groups" and "toggle back node level steps" options for all layouts that are skipped over, in addition to the one that is being toggled back to.

If the option "preserve named capturing groups" is turned on, EditPad Pro will also temporarily restore named capturing groups while testing the short-circuit regex.

The XML.jgfns scheme included with EditPad Pro uses short-circuiting. E.g. when a file contains <tag1><tag2><tag3></tag2></tag1><tag4></tag4>, the XML scheme should pair up tag1 and tag2 and treat tag3 as an unclosed tag. Without short-circuiting, tag4 would be seen as being nested inside tag3, since there's no </tag3> closing it.

In the "XML tag" layout, the "closing tag" element uses a backreference to a named capturing group to make sure the proper closing tag is matched (i.e. </tag2> should not be seen as the closing tag for <tag3>). The next element, "improper closing tag" matches </, but only when "closing tag" could not be matched. This means that when "improper closing tag" matches, the scheme has possibly encountered a closing tag that belongs to a previously found opening tag. In the example, "improper closing tag" will match the </ of </tag2>.

The "improper closing tag" element has its short-circuit mode set to "peek at the start of the match". This will cause the short-circuit regex of "opening tag", which toggled to the current layout, to be tested. The short-circuit regex for "opening" tag is the same as the one for "closing tag", except that it uses the preserved capturing groups of another layout. In our example, </tag2> will be matched by the short-circuit regex. After toggling back and restoring preserved capturing groups, </tag2> will be matched by "closing tag".

Foldable range without a node: This option is only available for elements that do not have any nodes. Turn on this option to create a foldable range. If the element does have nodes, you can link the nodes to the text to create a foldable range.

For the position of the start of the range, you can select the start or the end of the overall regex match or a capturing group, similar to the "layout group" and "short-circuit group" options. For the end of the range, you have several additional choices.

when an element uses the "detail the match with a layout" layout mode then that element determines where the detail layout starts and ends. You can make the foldable range start or end at that position with the options "at the start of the detail layout" and "at the end of the detail layout". Those positions do not change if the detail layout toggles to another layout. If the current layout is not a detail layout or was not toggled to by one, then these positions are the start and end of the file.

If this element toggles to another layout, which subsequently will toggle back to this element's layout, you can select "start/end of match after toggling back". The start/end overall regex match of the element that toggles back to the current element's layout is then used as the end of the range.

Select "start of next regex match in any layout" to end the range at the next regular expression match, regardless of which element or layout the regex belongs to.

Select "start of next regex match in this layout" to end the range at the next regex match of an element in the same layout as the current element. The difference between this option and the option "start/end of match after toggling back" is that this option does not take into account nested toggles. E.g. if the current element is in layout L1 and toggles to L2 which then toggles to L3 which then toggles (forward) to L1, the next regex match in L1 occurs at that point. The "start/end of match after toggling back" occurs after L1 toggles back to L3, back to L2 and back to L1.

"Start of next match attempt in this layout" occurs at the same time as "start of next regex match in this layout", but at a different position. It ends the range at the position where EditPad Pro starts looking for the next regex match in the layout, rather than at the start of the actual match. In the example in the previous paragraph, this is the position at which the element in L3 tells EditPad Pro to toggle (forward) to L1.

Select "start of the next match of this element" to end the range at the next match of of the regular expression of the same element as the one that adds the range. It doesn't matter if the regexes of other elements in the same layout or in other layouts find any matches before the regex of this element finds its next match. The range will end at the start of the next match of this element.

Note that folding ranges always work with whole lines. If the start or end of the range is in the middle of a line, EditPad Pro will automatically expand the range to completely include the lines at the start and the end. Therefore, you don't need to bother with matching complete lines, as long as you match part of the line you want the range to start or end on.

If a foldable range starts on the same paragraph on which the previous range ends, the previous range is automatically made one paragraph shorter. Again, this makes things easier because you don't have to end and start ranges at the start of a line, while preventing ranges from overlapping in an odd way. This adjustment is only made when two ranges overlap a single line. If they overlap several lines, the second will be nested inside the first.

Only when detailed folding: If you turn this on, the "foldable range without a node" option is only used when "add detailed automatic folding points from the file navigation scheme" is selected in Options, Configure File Types, Navigation in EditPad Pro. If you turn this off, the "foldable range without a node" option is used when either "add automatic folding points from the file navigation scheme" or "add detailed automatic folding points from the file navigation scheme" is selected.

6. Regular Expressions

Regular expressions are used for searching through (usually textual) data. They allow you to search for pieces of text that match a certain form, instead of searching for a piece of text identical to the one you supply. For example, the regular expression `[0-9]+` allows you to search through a file for any integer number.

It takes some time to get used to the syntax used by regular expressions. In our example, we used square brackets to create the character set `[0-9]`. This character set matches any character that is a digit. The `+` means that the character set must be matched as many times as possible, and at least once. EditPad's help file and printable manual contain a detailed tutorial to regular expressions. The tutorial clearly explains the entire regular expression syntax. It explains the exact regex flavor used by EditPad and the scheme editor.

Regular expressions are not only useful for building your own syntax coloring schemes. You can also use them for powerful search and replace operations in EditPad. Make sure the Regex button is down and you can unleash all their power upon your files. If you're a developer, your programming language very likely has its own regex library. It may even support regexes as a language feature.

If you want more assistance with creating and editing regular expressions, take a look at our product RegexBuddy at https://www.regexbuddy.com/. You can launch RegexBuddy right from within the scheme editor by clicking the RegexBuddy button on the main toolbar. This launches RegexBuddy and transfers the regular expression of the active element and the text you have in the preview editor to RegexBuddy. This allows you to quickly test and debug this regex in isolation in RegexBuddy. After fixing your regex in RegexBuddy you can click the Send button in RegexBuddy to transfer your regex back to the scheme editor. EditPad's Search panel has a similar button that allows you to edit its regex just as easily.

RegexBuddy can accurately emulate the regular expression flavors of a wide variety of applications and programming languages. It has specific support for EditPad's regex flavor. You can select EditPad 8 as your application in RegexBuddy. In RegexBuddy's documentation, EditPad 8's regex flavor is known as "JGsoft V2".

7. Named Capturing Groups

When you work with a single regular expression in EditPad Pro's search-and-replace, there is absolutely no difference between a named or a numbered capturing group, except that you reference one by name, and the other by number. In file navigation schemes, which usually consist of a multitude of regular expressions, there is a significant difference.

First, recall that (group) creates a named capturing group, while (?<name>group), (?'name'group) and (?P<name>group) are three different syntaxes for creating a named capturing group. \1 is a backreference to numbered group 1, while \k<name> and \k'name' are backreferences to named groups in the regular expression, and ${name} is a backreference to named groups in node captions (using the syntax for the replacement in a search-and-replace).

Difference in Functionality

Numbered groups can only be referenced inside the regular expression that defines them, and in the captions of the nodes below the element that regular expression belongs to.

Named groups, however, can be referenced inside the regular expression that defines them, and in all regular expressions in all elements below the element that holds the regular expression, including elements in layouts below the layout of the element that holds the regular expression. "Below" means according to the vertical order the elements and layouts have in the structure tree. In addition, named capturing groups can be referenced in node captions below all aforementioned elements. Simply put: named capturing groups persist all the way down.

There is no performance penalty for using named capturing groups instead of numbered capturing groups, unless you turn on the option to preserve named capturing groups for one or more elements. That option will save and restore all named groups used in any regular expression throughout the entire file navigation scheme, not just the ones in the element or its layout. Obviously, preserving and restoring more named groups requires more time and memory.

Groups Blanked Out Before Match Attempt

You can use (i.e. capture part of the match) the same named capturing group more than once in the same regex, and in multiple regular expressions, even across layouts. There is however one caveat. As explained in the layout topic, EditPad Pro applies all regular expressions of all elements in the current layout at once. Before each match attempt, all named and numbered capturing groups used to capture something in those regular expressions are cleared. Backreferences to named groups do not cause them to be cleared.

If you have, in the same layout, one regex that captures something into a named group, and another regex that references the same named group, the reference will always be blank. The group is cleared when EditPad Pro attempts both regexes (at the same time), and it isn't filled until the first regex matches. When the first regex matches, the second isn't attempted. If you change their order, the regex with the reference would be attempted with the blank group, and fail. (Backreferences groups that did not match yet always fail.)

Named Group Tag-Teams

The solution is to use a tag-team of named capturing groups. The XML.jgfns scheme included with EditPad Pro does this. The initial layout "XML" has an element "opening tag". This element captures the tag into the named group "tag", and toggles to the layout "XML tag". The "XML tag" layout also has an "opening tag" element. However, this element uses the capturing group "tag2" to store the tag it matches. The "closing tag" element in the "XML tag" layout references the "tag" group saved by the "XML" layout.

The "opening tag" element in the "XML tag" layout toggles to the "XML tag alternate" layout. This layout does the opposite. Its "opening tag" element captures "tag" and toggles to "XML tag", while its "closing tag" element references "tag2".

This effectively creates a tag-team of two layouts. "XML tag alternate" is nested inside "XML tag", and "XML tag" is nested inside "XML tag alternate" (in addition to the initial layout). Both layouts capture into one named group, and reference the other. Therefore the referenced group is never cleared when the match is attempted.

The initial "XML" layout, without a "closing tag" element, is necessary. If it was omitted, the "closing tag" element of "xml tag" would reference a named capturing group that was not used in a regular expression in an element above it in the structure tree.

All three "opening tag" elements have the "preserve named capturing groups" option turned on. This way, each layout always references its own copy of the tag in the "closing tag" element, no matter how deeply nested the XML file is. This is also true for the short-circuit regex of each "opening tag" element. Because "preserve named capturing groups" is on, the named groups are restored temporarily as the short-circuit regex is attempted.

You can look at the "XML tag" and "XML tag alternate" layouts as a tag-team of two runners, where both runners alternate between running one lap, and waiting for the other to run one lap.

8. Node

To actually build up a file navigation tree, you will need to add one or more nodes below one or more elements in your file navigation scheme. The INI.jgfns scheme is an example of a fully functional scheme with just one element and just one node. It builds a flat list of section headers, turning each section into a foldable range. The Delphi.jgfns scheme, while complex, creates a very straightforward file navigation tree. All the nodes the scheme defines start from the top level. The XML.jgfns scheme, while defining only a few nodes, creates a file navigation tree just as deep as the depth of the nested tags in the XML file the scheme is applied to.

To add a node to an element, click on the element and click the Node or Subnode button. Both have the same effect. If you select a node in the scheme structure, then the buttons have a different effect. Click the Node button to add a sibling to the selected node. Click the Subnode button to add a child to the selected node. You can add as many nodes to an element as you want. However, be careful not to clutter the file navigation tree. A tree with too many nodes is just as hard to navigate as the original file.

When the regex of an element matches, all the nodes under that element, if any, are added to the file navigation tree. The "**relative to**" option of the first node under the element defines where all the nodes are added. All other nodes under the element are added relative to the first node.

"Current level" creates a node below whichever node is current. Initially, the current level is the top level, and nodes will be created in the root. When adding nodes, the current level can be changed to one of the nodes that was added. Then, future nodes relative to "current level" will be added below that node.

"Above current level" creates a node below one of the current node's parents. E.g. if "node3" is current, and this node is a child of "node2" which is a child of "node1", specifying "above current level" with "levels" set to 1 will create the new node under "node2" instead of under "node3" as the "current level" option would do. Set "levels" to 2 to create a new node under "node1". Any higher number for "levels" will create the new node at the top level, since there are no further nodes to climb up to.

The "top level" option creates the node as a root node in the file navigation tree, regardless of which level is current. Root nodes are always visible, since they do not have parent nodes that can be collapsed.

"Below top level" works just like "above current level", except that is counts the other way around. In the same situation where "node3" is current under "node2" and "node1", setting "levels" to 1 creates a node under "node1", while a setting of 2 creates a node under "node2". Any higher value creates the node at the current level, which is "node3". Even if "node3" has children below it, the "below top level" option will never navigate deeper than "node3", the current level.

When adding nodes, you can also change the current level with the "**change current level**" option. The current level affects where the nodes of the next element that matches will be added, regardless of which element that is, or which layout it is part of.

The "no change" option leaves the current level where it is. If the next matching element adds a node to the current level, it will be added as a sibling to the current element's node. Choose "below node" if you want to make a particular node the current level. You can select any node that the current element creates. The next element will add its nodes as children to that element.

The other three options for "change current level" work just the way the "relative to" options work.

There is one more way to change the current level. As described in the element topic, you can turn on the option "**toggle back node level steps**" for each element that toggles to another layout, and expects that layout to toggle back. The key difference between this element option and the "change current level" node option is that the element option does not require you to add nodes, and also works when layouts are short-circuited. The XML.jgfns scheme makes use of this option to properly nest the XML tags in the file navigation tree, even when certain opening tags are missing their closing tags.

For each node you add, you can specify several options as to how it should appear in the file navigation tree.

Caption: Enter the text for the node. You can use backreferences to numbered groups in the element's regular expression. E.g. \1 references the first capturing group. You can also use backreferences to any named group filled by the element's regular expression, or the regular expression of any of the elements above the current element in the structure tree, including elements in layouts above the current layout. E.g. ${name} references the named group "name".

Initial state: When the node is first added to the tree, its state can be either collapsed or expanded. Setting this to "user choice" for one or more nodes enables the checkbox "expand nodes for which the initial state in the scheme is user choice" in the Navigation section in the file type configuration in EditPad Pro. That checkbox then determines whether the node is initially collapsed or expanded. You can select "user choice" for some nodes and "collapsed" or "expanded" for other nodes. Then the user will be able to control the initial state of only some of the nodes. If you select "user choice" for one node it's best to select it for most nodes. Otherwise the user may think the checkbox in EditPad Pro isn't working. If you select "collapsed" or "expanded" for all nodes, then EditPad Pro disables its checkbox to make it obvious it has no effect.

Sort: Set this to "alphanumeric order" to have the new node sorted alphanumerically among its sibling nodes. Set it to "file order" to add the new node as the last sibling. If multiple elements add nodes to the same parent node, you should give all those sibling nodes the same sorting setting. Otherwise, the first sibling determines whether the nodes are sorted or not. Setting this to "user choice" for one or more nodes enables the checkbox "sort nodes alphanumerically if their order in the scheme is user choice" in the Navigation section in the file type configuration in EditPad Pro. That checkbox then determines whether the node is sorted alphanumerically with its siblings or if the nodes are added in the order they are found by the scheme.

Combine: If you set this to "combine sibling nodes with the same caption" then the node is not added to the tree if adding it means that two siblings will have the same caption. When nodes are combined, their children are all added to the remaining node. If the nodes are linked to the text, the remaining node will have multiple links to different parts of the text. If nodes are sorted in file order, then the first node with the same caption determines the order.

If you select "combine successive nodes with the same caption" then the node is not added to the tree if it has the same caption as the most recent sibling. This way there will never be two sibling nodes with the same caption right next to each other. But there may be sibling nodes with the same caption with siblings with different captions in between.

The option "combine siblings if sorted; successive if not sorted" does what either of the two previous options do depending on how the node is sorted. It combines a node with any of its siblings if the node is sorted alphanumerically by the Sort option or by user choice. If the nodes are in file order, then the node may only be combined if it has the same caption as the most recent sibling.

Case insensitive: When combining nodes with the same caption, turn on this option to consider captions that differ only in case to be the same caption. The capitalization of the node that was added to the tree first

will prevail. If nodes are sorted alphanumerically, then captions that differ only in case are always considered to be the same.

Link node to part of the text: Turn on this option to link the node to a range in the file the scheme is being applied to. When you click on the node in the file navigation tree, the text cursor will be moved to the start of the range. If you press the Shift key while clicking, the range will be selected.

Primary link: When combining nodes with the same caption, the remaining combined node may have multiple ranges. When you click on such a node repeatedly, the text cursor or selection will move among those ranges. If you turn on "primary link" for one of the combined nodes, that range will be the first one the cursor or selection moves to. The other ranges will require additional clicks. If several nodes are set to be primary links, the last one to be combined will be the primary one. If none of the nodes is set to be the primary link, the node that was added to the tree first will be the primary link.

Range from: If you select "start/end of match", you can make the range start and the start or the end of the regex match of the node's element. You can also select one of the numbered or named capturing groups in the regex to start the range at. Select "end of previous sibling" to start the range and the end of the range of the node above it in the file navigation tree. If the node does not have a sibling above it, the end of the range of the parent node is used instead. Select "start of parent" to make the range start at the same position as the range of the node's parent node in the file navigation tree. Similarly, "end of parent" makes the range start at the position where the parent node's range end.

Range until: To end the range at a position inside the element's regex match, select "start/end of match", and then select the overall regex or the group the range should end at.

when an element uses the "detail the match with a layout" layout mode then that element determines where the detail layout starts and ends. You can make the node's range start or end at that position with the options "at the start of the detail layout" and "at the end of the detail layout". Those positions do not change if the detail layout toggles to another layout. If the current layout is not a detail layout or was not toggled to by one, then these positions are the start and end of the file.

If this element toggles to another layout, which subsequently will toggle back to this element's layout, you can select "start/end of match after toggling back". The start/end overall regex match of the element that toggles back to the current element's layout is then used as the end of the range.

You can also make the range end at a position where another node's range starts or ends. "Start of next sibling node" makes the range end at the start of the range of the node's sibling in the file navigation tree. If the node doesn't have a sibling, the sibling of its parent node is used instead (or the parent's parent if the parent doesn't have a sibling, etc.). Choose "start of first child node" to make the range end at the start of the node's first child node in the file navigation tree. Similarly, "end of last child node" makes the range end at the same position where the range of the node's last child node ends. The "start of next parent sibling" option ends the range at the starting position of the range of the node that is the next sibling to the current node's parent node.

Select "start of next regex match in any layout" to end the range at the next regular expression match, regardless of which element or layout the regex belongs to.

Select "start of next regex match in this layout" to end the range at the next regex match of an element in the same layout as the current element. The difference between this option and the option "start/end of match after toggling back" is that this option does not take into account nested toggles. E.g. if the current element is

in layout L1 and toggles to L2 which then toggles to L3 which then toggles (forward) to L1, the next regex match in L1 occurs at that point. The "start/end of match after toggling back" occurs after L1 toggles back to L3, back to L2 and back to L1.

"Start of next match attempt in this layout" occurs at the same time as "start of next regex match in this layout", but at a different position. It ends the range at the position where EditPad Pro starts looking for the next regex match in the layout, rather than at the start of the actual match. In the example in the previous paragraph, this is the position at which the element in L3 tells EditPad Pro to toggle (forward) to L1.

Select "start of the next match of this element" to end the range at the next match of of the regular expression of the same element as the one that adds the node. It doesn't matter if the regexes of other elements in the same layout or in other layouts find any matches before the regex of this element finds its next match. The node's range will end at the start of the next match of this element.

Range from and range until options that involve parent, sibling and child nodes are resolved after the entire file has been processed, and the entire file navigation tree has been built. This has several implications. When sorting nodes alphabetically, the next or previous sibling is not the previous or next one to be added to the tree, but the previous or next one in the alphabetic order. This is often not very useful, so you'll either want to turn of the alphabetic option, or use one of the "next match" or "toggling back" options to end the range. When combining nodes, a node may end up with multiple ranges. The primary range of the related nodes will be used when resolving ranges relative to other nodes.

If no related node can be found, the selected group or "overall regex match" is used as a failsafe default. EditPad Pro will start the range at the start of the selected group, and end the range at the end of the group selected for the end. You cannot specify any other default.

Foldable range: Turn on this option to make the range foldable. A plus in a box will appear next to the first line in the range, and a vertical line will extend from the box until the last line in the range. When the box is clicked, all lines in the range except the first line will disappear. Another click makes them visible again.

Folding ranges always work with whole lines. If the start or end of the range is in the middle of a line, EditPad Pro will automatically expand the range to completely include the lines at the start and the end. Therefore, you don't need to bother with matching complete lines, as long as you match part of the line you want the range to start or end on.

If a foldable range starts on the same paragraph on which the previous range ends, the previous range is automatically made one paragraph shorter. Again, this makes things easier because you don't have to end and start ranges at the start of a line, while preventing ranges from overlapping in an odd way. This adjustment is only made when two ranges overlap a single line. If they overlap several lines, the second will be nested inside the first.

9. Important Considerations

There are a number of things you should keep in mind if you want to create excellent file navigation schemes.

Handle Invalid Files

Most importantly, there is a key difference between a file navigation scheme for a text editor like EditPad Pro, and a compiler or data processor handling the very same files. While you edit the file in EditPad Pro, most of the time it will be in an invalid state. E.g. an XML file will constantly contain invalid tags, and unmatched opening and closing tags, as you type in new tags. A file navigation scheme that only works well with perfectly formatted XML files would be far from useful. Quick navigation is needed far more when making complex changes that throw off the whole file's structure until you're done editing, than when you're merely viewing a finished file. The file navigation scheme should not stumble on errors in the file, but continue with the remaining valid parts of the file to the best of its ability.

An XML processor, on the other hand, can easily demand a perfect XML file, since there's no point in processing a broken file. All it needs to do is point out the first error in the file with a clear error message. It doesn't need to bother to try to recover from the error or figure out what to do with the remainder of the file.

Suppose you have the following XML file:

<tag1><tag2><tag3></tag2></tag4></tag1><tag5></tag5>

An XML processor will typically output an error that <tag3> and </tag2> don't match up. A file navigation scheme on the other hand, should have additional logic to figure out <tag3> is missing its closing tag, and proceed with the rest of the file as if nothing happened. If the scheme blindly searched for </tag3>, then tag5 would be nested underneath tag3, instead of listing it as a root node. It also shouldn't be pedantic about XML files permitting only one root node. It should list all nodes, and leave it up to the user to further edit the file so it has only one root. The unpaired closing tag </tag4> also shouldn't throw the scheme off its track.

The XML.jgfns scheme included with EditPad Pro has all these qualities. This makes the scheme significantly more complex than a scheme that would only handle perfectly matched tags. Fortunately, Just Great Software file navigation schemes are designed with this complexity in mind, offering features such as short-circuiting used by the XML scheme.

Another good example is the Delphi.jgfns scheme included with EditPad Pro. Almost every layout in this scheme has an element called "next section keyword". These elements match keywords that actually should never occur in the context of that element. What those elements do is to toggle back to the main scheme, assuming they've reached the end of an incomplete class, method or whatever. By using the option to toggle back at the start of the match, the "next section keyword" items can be kept rather simple. The keyword they matched will be matched again by the "implementation" or "interface" layouts, which will then decide what to do with it.

Don't Provide Too Much Detail

The purpose of a file navigation scheme is to summarize the file, making it very easy to jump to different spots in the file. To keep the file navigation tree usable, it should only contain nodes for important locations

in the file. If the list of nodes becomes too long or too deep, it becomes just as hard to navigate as the file itself.

An important feature of the scheme editor in this regard is the ability to alphabetize and/or combine nodes. Often, an alphabetized tree is much easier to navigate than a tree that follows the structure of the file. The programming language schemes included with EditPad Pro are all completely alphabetized. The order of the different parts of the source code files is largely irrelevant, so alphabetizing the list makes it much easier to navigate. The HTML scheme, on the other hand, follows the layout of the file. The order of the headings in an HTML file is obviously significant, since that's the order the readers will see them in.

Combining nodes is a good way to make your scheme more detailed without cluttering it. E.g. if the file format you're working with consists of a declaration section and an implementation section, with each item in the file first declared and then implemented, your file navigation tree will be far more workable if you combine declaration and implementation for each item into a single node. If you set the implementation range as the primary range, you can click the node once to jump to the item's implementation, and again to jump to the declaration. This is far quicker than first expanding an "implementation" node, and then finding the item you want underneath it, only to repeat the search with the "declaration" node if you wanted the declaration instead.

A scheme that uses this technique is the Delphi.jgfns scheme included with EditPad Pro. Click once on a routine or method to jump to its implementation. Click again to jump to its declaration in the interface section of the Delphi unit.

10. Preview

To test your file navigation scheme, the Just Great Software File Navigation Scheme Editor provides a preview area near the bottom of the window. You'll probably want to maximize the scheme editor's window to enlarge the preview area. There's also a splitter bar above the preview toolbar that you can use to further enlarge the preview area.

Click the **Open** button to open a test file. Click the Preview button to apply the file navigation scheme to the test file. Click the Word Wrap button to toggle word wrap on or off.

If you edit the scheme after clicking the **Preview** button, the preview is not automatically updated. You have to click the Preview button again to apply the updated scheme. If you edit the test file or open another one, the file navigation tree is automatically updated, just like would happen in EditPad Pro. However, the automatic updates will continue to use the scheme in the state it was when you last clicked the Preview button. The only way to test the changes you've made to the scheme is to click the Preview button again.

After clicking the Preview button, you can toggle the Detailed, Sorted, and Expanded buttons to test how your scheme works with the various options that EditPad Pro will offer for your scheme in the Navigation section in the EditPad Pro file type configuration. Toggling one of these options resets the file navigation tree to its initial state. But the preview will continue using the scheme that was loaded when you last clicked the Preview button.

Turning on **Detailed** corresponds with selecting "add detailed automatic folding pionts form the file navigation scheme" while turning it off corresponds with "add automatic folding points from the file navigation scheme". The Detailed button is enabled if you turned on "only when detailed folding" for one or more elements.

The **Sorted** button corresponds with the "sort nodes alphanumerically if their order in the scheme is user choice" checkbox in EditPad Pro. It is enabled if you set the "sort" option for one or more nodes to "user choice".

The **Expanded** button corresponds with the "expand nodes for which the initial state in the scheme is user choice" checkbox in EditPad Pro. It is enabled if you set the "initial state" option for one or more nodes to "user choice".

The **Colors** button has a **Dark Theme** item that toggles the entire user interface of the scheme editor between the standard Windows them and a dark theme.

The **Word Wrap** button toggles between wrapping long lines at the edge of the preview editor and not wrapping lines.

To the left of the test file you will see the file navigation tree. It works just like the one in EditPad Pro. Click on a node to move the text cursor to the start of the node's primary range. Click again on the same node to cycle through the node's additional ranges (if the node is a combined node). Shift+click to select the range instead of only moving the text cursor.

There is one big difference between the preview and EditPad Pro. While EditPad Pro uses a separate syntax coloring scheme, the File Navigation Scheme Editor will use the regex matches from your file navigation scheme to highlight different parts of the file. You can specify the colors for each element. Double-click on a

highlighted piece of text to select the element in the editor. The correct element will only be selected if you haven't added, deleted or moved any elements or layouts since last clicking the Preview button.