```
*xpt-snippet-syntax*

                        XPT Snippet Syntax
                                                                    by drdr.xp
                                                                drdr.xp@gmail.com
================================================================================

        Content table:
        |xpt-snippet-sample|

        |xpt-snippet-header|
                |xpt-snippet-priority|
                        |xpt-priority-value|
                        |xpt-priority-format|
                |xpt-snippet-keyword|
                |xpt-snippet-mark|
        |xpt-snippet-variable|
        |xpt-snippet-function|
        |xpt-snippet-XPTinclude|
        |xpt-snippet-embed|

        |xpt-snippet|
                |xpt-snippet-name|
                |xpt-snippet-setting|
                |xpt-snippet-hint|
                |xpt-snippet-hidden|
                |xpt-snippet-alias|
                |xpt-snippet-synonym|
                |xpt-snippet-wrap|
                |xpt-snippet-wraponly|
                |xpt-snippet-abbr|


        |xpt-snippet-body|
        |xpt-snippet-include|
        |xpt-snippet-XSET|
        |xpt-snippet-XSETm|
        |xpt-snippet-ComeFirst|
        |xpt-snippet-ComeLast|
        |xpt-snippet-postQuoter|


        |xpt-snippet-item|
        |xpt-snippet-placeholder|
        |xpt-placeholder-mark|
                |xpt-placeholder-left-mark|   |‘|
                |xpt-placeholder-right-mark|  |^|
        |xpt-placeholder-edge|
        |xpt-snippet-leading-placeholder|

        |xpt-placeholder-special|
        |xpt-placeholder-cursor|

        |xpt-snippet-expression|
        |xpt-snippet-instant-value|
        |xpt-snippet-preset-value|
        |xpt-snippet-default-value|
        |xpt-snippet-post-filter|
                |xpt-snippet-default-post-filter|
                        |xpt-placeholder-optional-ph|

        |xpt-placeholder-default-value|
        |xpt-placeholder-post-filter|
        |xpt-placeholder-ontime-filter|

        |xpt-snippet-expandable|
        |xpt-snippet-repetition|

                                                                *xpt-snippet-sample*
```



```
        A snippet file looks like this : >
                XPTemplate priority=lang keyword=$       | |xpt-snippet-header|

                let s:f = XPTfuncs()                     | |xpt-snippet-function|

                XPTvar $TRUE          true               | |xpt-snippet-variable|
                XPTvar $FALSE         false              |
                XPTvar $NULL          null               |
                XPTvar $UNDEFINED     undefined          |
                                                         |
                XPTvar $CL  /*                           |
                XPTvar $CM   *                           |
                XPTvar $CR   */                          |
                                                         |


                XPTinclude                               | |xpt-snippet-XPTinclud
e|
                      \ _common/common                   |
                      \ _comment/doubleSign              |
                      \ _condition/c.like                |

                fun! s:f.js_filename()                   | |xpt-snippet-function|
                        return expand( "%" )             |
                endfunction                              |

                XPTemplateDef                            | |XPTemplateDef|


                XPT cmt hint=/**\ @auth...\ */           | |xpt-snippet|
                XSET author=$author                      | |xpt-snippet-XSET|
                XSET email=$email                        | |xpt-snippet-XSET|
                /**                                      |\
                * @author : ‘author^ | ‘email^           | \
                * @description                            |  +|xpt-snippet-body|
                *     ‘cursor^                           | /
                * @return {‘Object^} ‘desc^               |/
                */


                XPT for hint=for\ (var..;..;++)
                for ( i=0; i<‘len^; ++i ) { ‘cursor^ }
<

                                                                *xpt-snippet-header*
        Each snippet file starts with a XPTemplate declaration for
        snippet-scope setting, duplication load check, etc.

        Format : >
        XPTemplate [priority=lang] [keyword=#] [mark=‘^]
<
        There are 3 optional settings for ’XPTemplate’ :

                                                              *xpt-snippet-priority*
        Priority affects |xpt-snippet| and |xpt-snippet-variable|;
        |xpt-snippet-function| is defined directly, so it isn’t controlled by
        |xpt-snippet-priority|. Snippets with a lower |xpt-snippet-priority|
        override higher ones. Format : >
                XPTemplate priority=<priority-value>
<       See |xpt-priority-value|.
                                                              *xpt-priority-value*
        Snippets are defined with a certain priority. One buffer might load
        snippets with the same name. Only the snippet with the lowest priority
        is used, others are ignored.
        Priorities can be in the range from 0 to +oo. 0 is the highest
        priority. Some predefined symbols stand for numeric priority
        values : >
                all             : 64
                spec            : 48
                like            : 32
```

```
                lang            : 16
                sub             : 8
                personal        : 0
<       Default priority is "lang" or 16.

        You can set priority for each snippet by using |xpt-priority-format|.
        Or set priority for all templates in the current file with
        |XPTemplatePriority()|.

                                                        *xpt-priority-format*
        Priority setting format : "priority[+/-[offset]]".
        Following formats are all valid : >
        3               3
        lang            16
        like+4          36 // like=32
        all-            63 // all=64
        all-1           63 // all=64
<
                                                        *xpt-snippet-keyword*
        |xpt-snippet-keyword| specifies what other characters can be used as
        |xpt-snippet-name|.
        Format : >
                XPTemplate keyword=...
<       For example, for C language keyword is set as : >
                XPTemplate keyword=#
<       So that "#if" can be used as |xpt-snippet-name|.

                                                      *xpt-snippet-mark*
        To specify which 2 characters are used as |xpt-placeholder-mark|
        instead of the default |'| and |^|. Format : >
                XPTemplate mark=~^
<
                                                                  *xpt-$*
                                                                *XPTvar*
                                                    *xpt-snippet-variable*
        |xpt-snippet-variable| can be used in :
                |xpt-snippet-instant-value|
                |xpt-snippet-preset-value|
                |xpt-snippet-default-value|
                |xpt-snippet-post-filter|
                |xpt-placeholder-ontime-filter|
        Format : >
                XPTvar $VAR_NAME something
<       Or with single quoter : >
                XPTvar $VAR_NAME 'something'
<       The only difference is that in single quoted strings space can be
        freely used.  Like this : >
                XPTvar $VAR_NAME ' '
<       Instead of using the escaped format "\ ".

        It's also possible to set a variable to the empty value: >
                XPTvar $VAR_NAME ''
<
        Another example from C language : >
                XPTvar $TRUE 1
<       and a snippet defined as : >
                XPT while1
                while ('$TRUE^) {
                    'cursor^
                }
<       will generate : >
                while (1) {
                    'cursor^
                }
<
        |xpt-snippet-variable| is used widely in XPTemplate, such as comment
        definition, format customization, etc.

        Personal information variables should be defined by using
```



```
        |g:xptemplate_vars|. Variables defined with |g:xptemplate_vars|
        override variables defined in any snippet files.

        NOTE    By convention the names of |xpt-snippet-variable| supplied by
                XPTemplate start with an upper-case letter. User defined
                variable name should start with a lower-case letter.

        NOTE    Override control of |xpt-snippet-variable| is affected by
                |xpt-snippet-priority|.

                                                      *xpt-snippet-function*
        |xpt-snippet-function| can be used in :
                |xpt-snippet-instant-value|
                |xpt-snippet-preset-value|
                |xpt-snippet-default-value|
                |xpt-snippet-post-filter|
                |xpt-placeholder-ontime-filter|
        To define |xpt-snippet-function|, the function container must be
        fetched by using |XPTfuncs()|. For example : >
                let s:f = XPTfuncs()
                fun! s:f.c_printfElts( v )
                        ...
                endfunction
<       Then function 'c_printfElts' can be used in the snippet : >
                XPT printf      hint=printf\(...)
                XSET elts=c_printfElts( R( 'pattern' ) )
                printf( "'pattern^"'elts^ )
<
        NOTE    By convention the names of |xpt-snippet-function|s supplied by
                XPTemplate start with an upper-case letter. User defined
                function names should starts with a lower-case letter.

                                                              *XPTinclude*
                                                    *xpt-snippet-XPTinclude*
        Syntax : >
                XPTinclude <folder_name_in_ftplugin>/<filename>
<       The <filename> is only the file root, without ".xpt.vim".
        XPTinclude can include several snippets at one time.

        NOTE do NOT use |:runtime| to include other snippet files. XPTinclude
        handles snippet priority, but |:runtime| does NOT.

                                                                *XPTembed*
                                                          *xpt-snippet-embed*
        Syntax : >
                XPTembed <folder_name_in_ftplugin>/<filename>
<       |XPTembed| acts like |XPTinclude|, except it includes only snippet
        files for nested languages, like JavaScript in HTML, or HTML in PHP.

        NOTE TODO differences between XPTinclude by example.




                                                  *XPTemplateDef* *xpt-snippet*
        The snippet part starts with the command "XPTemplateDef". Any content
        after this command is no longer Vim script, but a XPTemplate snippet.

        Each |xpt-snippet| defines one code snippet with the following
        syntax : >
                XPT <snippetName> [name=value] [name=value] ..
                <snippet body>..
                ..XPT
<       'XPT' is the start of a snippet. '..XPT' is the end of a snippet, it
        is optional. If '..XPT' is not present the snippet body ends at the
        last non-empty line.

                                                            *xpt-snippet-name*
        <snippetName> is the name the user has to type to trigger this
```

```
        snippet. It can only contain characters defined in 'iskeyword' and
        |xpt-snippet-keyword|. Custom snippets (not provided by XPTemplate)
        should not start with "_" by convention as these snippets are used by
        XPTemplate internally, see also |xpt-snippet-include|.

                                                        *xpt-snippet-setting*
        The 'name=value' defines snippet settings, including
                |xpt-snippet-hint|
                |xpt-snippet-hidden|
                |xpt-snippet-alias|

                                                          *xpt-snippet-hint*
        Set the 'menu' entry for the pop up menu; as a short description other
        than |xpt-snippet-name|. Like the following C language popup menu : >
                | #if           #if ...           |
                | #ifdef        #if ...           |
                | #ifndef       #ifndef ..        |
<
        Syntax to set up hints: >
                XPT for hint=for\ (..;..;++)
<       Or : >
                XPT for " for (..;..;++)
<       The quote-hint must be at the end of |xpt-snippet| declaration.

        Some characters( space, left quote and "$" ) need to be escaped in
        hint text: >
                                <space> (       $
                hint=           yes     yes     yes
                "               no      yes     yes
<
        With only the 'hint=' way:
                space needs to be escaped.

        With both these two ways :
                "(" needs to be escaped if you do NOT want it to be evaluated
                as a function call.
                "$" needs to be escaped if you do NOT want it to be evaluated
                as variable.



                                                        *xpt-snippet-hidden*
        Set to 1 to prevent the snippet from being triggered by the user, but
        it CAN be triggered from internal API, or included by other snippet.
        Syntax: >
                XPT for hidden=1 hint=...
<       Or: >
                XPT for hidden hint=...
<
        See |xpt-snippet-include| and |xpt-api|.



                                                        *xpt-snippet-alias*
        Make the snippet an alias to another snippet. Syntax : >
                XPT forin hint=for\ ..\ in\ ..\ ..\ endfor
                for 'value^ in 'list^
                    'cursor^
                endfor

                XPT foreach alias=forin hint=for\ ..\ in\ ..\ ..\ endfor
<       This makes "forin" and "foreach" the same, but with possible different
        settings.

        NOTE |xpt-snippet-alias| can be used to create shortcuts.

                                                        *xpt-snippet-synonym*
        Like |xpt-snippet-alias|, synonym gives a snippet another name.
        Syntax : >
                XPT snippetName synonym=a|b|c...
<       Where a, b and c are all the names of this snippet. For example : >
```



```
                XPT forin synonym=fin|fi hint=for\ ..\ in\ ..\ ..\ endfor
                for 'value^ in 'list^
                    'cursor^
                endfor
<       This makes "forin", "fin" and "fi" the same.

        NOTE |xpt-snippet-synonym| can be used to create shortcuts.



                                                          *xpt-snippet-wrap*
        Wrapper snippets can be triggered in visual mode, place holder marked
        as "wrap" is replaced with the text selected in visual mode.

        Definition of wrapper snippet has no differences from normal
        snippet except it declaring a place holder as wrapping holder. For
        example: >
                        ________/---------------------| wrapper declaration
                XPT if wrap=job
                if ('condition^){
                    'job^
                }
<
        Wrapping can be block-wise or line-wise. Wrapper place holder with
        |xpt-placeholder-edge| is line-wise, or it is block-wise.

        For example: >
                XPT comment wrap=what
                '/* 'what' */^
<       This is line-wise wrapper, it will result in: >
                /* line1 */
                /* line2 */
<       But not: >
                /* line1
                line2 */
<
        See also |xpt-wrapper-snippet| and |xpt-snippet-wraponly|.



                                                      *xpt-snippet-wraponly*
        Normally, wrapper snippet can also be triggered in insert-mode, unless
        "wraponly" declared. For example: >
                                ________/------------| wrap only
                XPT if wrap=job wraponly
                if ('condition^){
                    'job^
                }
<
                                                          *xpt-snippet-abbr*
                                                                {default:0}
        Set to 1 to create |abbreviations| for this snippet.
        Example: >
                        ____/------------------------| create abbr
                XPT if abbr
                if (  ) {  }
<       Snippet defined as above will be triggered by typing "if<space>" or
        "if<C-]>".

                                                          *xpt-snippet-body*
        <snippet body> is all the OTHER text except the first line : >
                XPT for hint=for\ (..;..;++)
                for ('i^ = '0^; 'i^ < 'len^; ++'i^) { |
                    'cursor^                          | snippet body
                }                                     |

<       |xpt-snippet-body| contains snippet text and :
                |xpt-snippet-XSET|
                |xpt-snippet-XSETm|
        Snippet with XSET command : >
```

```
                XPT printf      hint=printf\(...)
                XSET elts=c_printfElts( R( 'pattern' ) )
                printf( "'pattern^"'elts^ )

<       NOTE XSET/XSETm commands can be placed anywhere inside a snippet.


                                        *':::^* *Include:* *xpt-snippet-include*
        ':<snipname>:^          Simple Include without "cursor".
        'Include:<snipname>^    Include with "cursor" place holder.

        ':<snipname>():^        Simple inclusion with parameter.
        'Include:<snipname>()^  Inclusion with parameter.

        Snippet can include another snippet, through inclusion place holder : >

                'Include:snippetName^

<       When inclusion occurs, |xpt-snippet-post-filter|,
        |xpt-snippet-default-value| and |xpt-snippet-preset-value| is imported
        too, if it does not override.

        Short inclusion format : >
                ':snippetName:^
<       Only two ":" around snippet name are needed.

        NOTE    The only difference between "Include:" and "::" is "Include:"
                keeps "cursor" place holder but "::" does not.

        Take "if" snippet in file "_condition/c.like.xpt.vim" for example(
        snippets are simplified for reading ): >
                XPT _if hidden
                if ( 'condition^ ) {
                    'job^
                }

                XPT if hint=if\ (..)\ {..}\ else...
                ':_if:^' 'else...{{^ 'Include:else^'}}^

<       The real "if" includes the "_if" and "else" snippets.

        NOTE    Inclusion is literal, so that snippets with different
                |xpt-snippet-mark| can not include each other.

        NOTE    By convention snippets of name started with "_" are internal
                snippets. Normally these snippets are set with
                |xpt-snippet-hidden| flag on and used for inclusion only.

        Parameters of Inclusion:

        Inclusion can have parameters passed to included snippet. Parameters
        are name-value pairs. Names are placeholder name. Name-Values presents
        in form of |Dictionary|:
                ':<snipname>( { '<phname>' : '<new_phname>', ... } ):^
        Place holders in sub-snippet presents in parameter are replaced.


                                                                *xpt-snippet-XSET*
        In |xpt-snippet-body| XSET commands can be used anywhere to set :
                |xpt-snippet-preset-value|
                |xpt-snippet-default-value|
                |xpt-snippet-post-filter|
        XSET syntax to set |xpt-snippet-preset-value|: >
                XSET itemname|pre=<expression>
<       XSET syntax to set |xpt-snippet-default-value|: >
                XSET itemname|def=<expression>
<       or : >
                XSET itemname=<expression>
<       XSET syntax to set |xpt-snippet-post-filter|: >
```



```
                XSET itemname|post=<expression>
<       <expression> is |xpt-snippet-expression|. For example : >
                XPT #include_user       hint=include\ ""
                XSET me=fileRoot()
                #include "'me^.h"
<       Item "me" is set to the file name without extension.

                                                                *xpt-snippet-XSETm*
        "XSETm" is similar to |xpt-snippet-XSET| except it uses "\n" instead
        of "=" in |xpt-snippet-XSET| and ends with "XSETm END". For
        example : >
                XPT if          hint=if\ (..)\ {..}\ else...
                if ('condition^) {
                    'job^
                }' 'else...^
                XSETm else...|post
                else {
                    'cursor^
                }
                XSETm END
<

                                                        *xpt-snippet-ComeFirst*
                                                         *xpt-snippet-ComeLast*
        Special XSET keys "ComeFirst" and "ComeLast" specify the item render
        order. Their value is a list of place holder names separated by space.
        For example : >
                XPT for hint=for\ (..;..;++)
                XSET ComeFirst=0 len
                for ('i^ = '0^; 'i^ < 'len^; ++'i^)'$BRloop^{
                    'cursor^
                }
<       So that "0" is focused first, then "len" and then "i".

                                                         *xpt-snippet-postQuoter*
        The key "postQuoter" is designed to specify quoter do define
        |xpt-snippet-expandable|. Default is "{{,}}".

                                                              *xpt-snippet-item*
        In one snippet a group of |xpt-snippet-placeholder|s with the same
        name is an "item". For example : >
                XPT for hint=for\ (..;..;++)
                for ('i^ = '0^; 'i^ < 'len^; ++'i^) {
                    'cursor^
                }
<       In this snippet there are 4 items : >
                i, 0, len, cursor
<       Item "i" has 3 |xpt-snippet-placeholder|s, the others have only 1.

                                                        *xpt-snippet-placeholder*
        A place holder is a segment of a snippet which can be changed by the
        user. It's tracked by XPTemplate to update the user input of
        |xpt-snippet-placeholder|s within the same |xpt-snippet-item|.

        The place holders are defined by |xpt-placeholder-mark|, by default
        |'| and |^| are used. For example : >
                XPT for hint=for\ (..;..;++)
                for ('i^ = '0^; 'i^ < 'len^; ++'i^) {
                    'cursor^
                }
<       Sequentially, the place holders in this snippet are : >
                i, 0, i, len, i, cursor
<

                                          *xpt-placeholder-left-mark*  *'*
                                         *xpt-placeholder-right-mark*  *^*
                                                    *xpt-placeholder-mark*
        |xpt-placeholder-mark| are the characters used to define
        |xpt-snippet-placeholder|s of a snippet, by default |'| and |^|.
        Or the |xpt-placeholder-left-mark| and |xpt-placeholder-right-mark|.
```

They can be changed locally, for the current snippet file, by |xpt-snippet-mark|.

‘’’ *xpt-‘’’* *xpt-placeholder-edge*

Besides |‘| and |^|, additional |xpt-placeholder-left-mark|s can be set inside place holder to add additional information: the edge.

Edge is some text around a place holder that is not selected when the cursor jumps to this place holder, but it still can be edited. For example : >
```
‘(‘xpt‘)^
```
< This place holder is named "xpt" and the edges are "(" and ")". When the cursor jumps onto it : >
```
(xpt)
 ***------ only xpt is selected
```
< Edges help with formatting issues.

Place holder can have only a left edge, for example : >
```
‘(‘xpt^
```
<

NOTE only |_W| characters are acceptable in edges.

xpt-snippet-leading-placeholder

In an item one place holder is the leading place holder which accepts user input. Others are update by XPTemplate automatically.

By default, the first place holder in item is the leading place holder, or the one with a |xpt-placeholder-edge|. This allows it to specify which place holder is the edit area.
For example : >
```
for (‘i^ = ‘0^; ‘i^ < ‘len^; ++‘i^) {
    ‘cursor^
}
```
< In item "i", the first "i" before "=" is the leading one. But in this snippet : >
```
for (‘i^ = ‘0^; ‘‘i^ < ‘len^; ++‘i^) {
    ‘cursor^
}
```
< The second "i", with double |‘| before "<", is the leading one.

xpt-placeholder-special

Special place holders include : |xpt-placeholder-cursor| and |xpt-snippet-wrap|.

*‘cursor^* *xpt-placeholder-cursor*

Sets where cursor the stops after a snippet finished.

The item named "cursor" is a special one. It’s always selected at last and replaced with an empty string. When navigating to the "cursor" item the snippet is complete.

xpt-{} *xpt-mixed* *xpt-snippet-expression*

Expression is a mixture of plain text, |xpt-snippet-variable| and |xpt-snippet-function|.
Expression is used as the value of
- |xpt-snippet-instant-value|
- |xpt-snippet-preset-value|
- |xpt-snippet-default-value|
- |xpt-snippet-post-filter|
- |xpt-placeholder-ontime-filter|.

Functions can be |xpt-snippet-function|s defined as member of |XPTfuncs()| or native Vim functions.

Functions are called as member of rendering context : |xpt-snippet-function-ctx|.

Functions or variables can be enclosed by "{}" to prevent function or

variable names messing up with the surrounding plain text.

Functions are not evaluated if "()" is escaped : >
```
S\(S("abc", ’.’, ’\u&’))
```
< is evaluated to : >
```
S(ABC)
```
<

Escaping the ’$’ stops variable evaluation : >
```
\$author
```
< is evaluated to : >
```
$author
```
<

Escaping the "{}" : >
```
\{S("abc", ’.’, ’\u&’)}
```
< is evaluated to : >
```
{ABC}
```
< While : >
```
{S("abc", ’.’, ’\u&’)}
```
< is evaluated to : >
```
ABC
```
<

Another example, supposing you are editing a file named "your_file_name.ext" : >
```
__{S(E("%:t"),".","\\u&")}__
```
< is evaluated to : >
```
__YOUR_FILE_NAME.EXT__
```
< And : >
```
this is S($author,".","-&")-
```
< is evaluated to : >
```
this is -d-r-d-r-.-x-p-
```
< See also:
- |xpt-snippet-function|
- |xpt-snippet-variable|

xpt-snippet-instant-value

There is a special case for |xpt-snippet-placeholder| when the place holder’s content is a |xpt-snippet-expression|. In this case the place holder is evaluated at once, and no more further editing can happen on this place holder. For example : >
```
XPT filehead
...
 * @since : ‘strftime("%Y %b %d")^
...
```
<

xpt-snippet-preset-value

Preset values are like |xpt-snippet-default-value| but are applied earlier. |xpt-snippet-default-value| are applied before the place holders are focused; preset value are applied just after the snippet is displayed on the screen.
To define preset values: >
```
XSET the_name|pre=<expression>
```
<

xpt-snippet-default-value

By default place holders use their name as the default value but you can choose another text as default value by using : >
```
XSET the_name|def=<expression>
```
<

So that before cursor jumps to leading place holder of name "the_name", the <expression> evaluated and applied to the place holder.

Example : the "#ind" snippet defined as : >
```
XPT #ind
XSET me|def|=fileRoot()
#include "‘me^.h"
```
< In C language, type "#ind<C-\>" you get : >
```
#include "current_file_name.h"
```
<

NOTE : if default value expression contains only plain string and

```
        |xpt-snippet-variable|, it is used as |xpt-snippet-preset-value|, too;
        for better looking without any side-effect.


                                                    *xpt-snippet-post-filter*
        Post filters are executed after the user presses <Tab> and change the
        typed text. To define a post filter use : >
                XSET the_name|post=<expression>
<       Or use |xpt-snippet-postQuoter| : >
                XPT enum hint=enum\ {\ ..\ }
                enum 'name^'$BRstc^{
                    'elt^;'
                    '...{{^
                    'elt^;'
                    '...^'}}^
                }' 'var^;
<
        Some usually-used post filter functions are defined in
        ftplugin/_common/common.xpt.vim.

        For example for c language, "#ifndef" snippet is defined as follows : >
                XPT #ifndef     hint=#ifndef\ ..
                XSET symbol=S(fileRoot(),'\.','_','g')
                XSET symbol|post=UpperCase(V())
                #ifndef 'symbol^
                #     define 'symbol^

                'cursor^
                #endif '$CL^ 'symbol^ '$CR^
                ..XPT
<
        When you pressing <tab> from the first item "symbol", typed content are
        converted to upper case. Before <tab> pressed : >
                #ifndef __gnu__
                #    define __gnu__
                'cursor^
                #endif /* __gnu__ */
<       After <tab> pressed : >
                #ifndef __GNU__
                #    define __GNU__
                'cursor^
                #endif /* __GNU__ */
<


                                            *xpt-snippet-default-post-filter*
        Place holder with some special has default post filter set. Following
        sections discuss them.

        Place holders have default place holder defined: >
                \V\w\+?         EchoIfNoChange('')
<


                                                *xpt-placeholder-optional-ph*
        If a place holder name matches pattern '\V\w\+?', "EchoIfNoChange('')"
        is assigned as its post filter.
        This makes the place holder optional.

        For example a snippet defined as below: >
                fun('arg^', 'context?^)
<       When you render this snippet, and cursor stays on "context?" : >
                fun(arg, context?)
                        --------
                              \__ selected
<       Pressing <Tab> clears it, and snippet becomes: >
                fun(arg)
<
```



```
                                                    *xpt-placeholder-post-filter*
        For each place holder a private post filter can also be set by using
        the |^||^| syntax : >
                XPT lowerUpper
                lower : 'text^
                upper : 'text^UpperCase(V())^^  <---- double "^"
<       Press <Tab>, this snippet results in : >
                lower : text
                upper : TEXT
<
        NOTE If both |xpt-snippet-post-filter| and
        |xpt-placeholder-post-filter| are set, the place holder filter takes
        effect.

                                                *xpt-placeholder-default-value*
                                                *xpt-placeholder-ontime-filter*
        For each place holder an ontime filter can be set to filter the text
        each time the user types something by using the |^| syntax : >
                XPT lowerUpper
                lower : 'text^
                upper : 'text^UpperCase(V())^   <--- only one "^"
<       Each time the user types something at place holder "text" the second
        place holder is updated with the content converted to upper case.

        NOTE ontime filter is used as |xpt-placeholder-default-value| for the
        leading place holder.

                                                        *xpt-snippet-expandable*
        Sometimes you want to create an additional piece of snippet other than
        the original snippet. For example, add another "else" after an "if"
        block. To do this use expandable : >
                XPT if
                if 'cond^
                    'job^
                'else...{{^else
                    'cursor^
                '}}^
                endif
<       At the place holder "else..." press <Tab> to generate another else
        block, the text quoted by {{ and }}. Press to <Cr> to clear "else...".

        Another way to define expandable is by using XSET command to define a
        post filter: >
                XPT if
                if ('condition^)'$BRif^{
                    'job^
                }' 'else...^
                XSETm else...|post
                else {
                    'cursor^
                }
                XSETm END
<       These 2 methods are the same inside XPTemplate.

                                                        *xpt-snippet-repetition*
        Repetition is only a special case utilizing expandable, that another
        same expandable trigger residing inside the expandable part.

        For example the "case"s in "switch". To specify the repetition part,
        just wrap the part you want it to repeat with '...^. n is a number and
        can be omitted.  Take the case from "switch": >
                XPT switch
                switch ('^) {
                  '...^             | repetition part
                  case '^0^ :       |
                    '^              |
                    break;          |
                  '...^             |
```

```
                  default:
                    `^
                }
<
        When you trigger a repetition template it works as below: >
                switch () {    <------- cursor stays here
                  `...^

                  default:
                     `^
                }
<       Press <Tab>, the |`...^| is selected. Press <Tab> again to expand the
        repetition part. Or press <Cr> |xpt-key-clear| to cancel the
        repetition part. These 4 lines are expanded: >
                switch () {

                  case `^ :         | expanded
                    `^              |
                        break;      |
                    `...^           |

                  default:
                        `^
                }
<       Enter the repetition part. There is another |`...^|, that is the
        another repeat trigger.

        Press <Tab> 3 times: >
                switch () {

                  case 0 :

                    break;

                  case `^ :        | selected repetition part
                    `^             |
                        break;     |
                    `...^          |

                  default:
                        `^
                }
<
        Using named |`...^| allows you define multiple repetition parts in one
        snippet. For example: >
                XPT switch
                switch (`^) {
                  `case...^        | repetition part
                  case `^0^ :      |
                    `^             |
                    break;         |
                  `case...^        |

                  default:
                    `^
                }
<

        See |xpt-repetition| for using repetition.


" vim:tw=78:ts=8:sw=8:sts=8:noet:ft=help:norl:
```