V2S Editor User’s Guide

FileNet System Monitor 4.0.0

FileNet Corporation

Table of Contents

List of Screenshots

Chapter 1. Copyright Notice

Trademarks

Notice

References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to Tivoli Systems or IBM's valid intellectual property or other legally protectable right, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user.

Chapter 2. Notices

FileNet Corporation 3565 Harbor Boulevard Costa Mesa, California 92626 USA
800.FILENET (345.3638) Outside the U.S., call: 1.714.327.3400
www.filenet.com (http://www.filenet.com)

Chapter 3. About this document

Who Should Read This Guide

The target audience for this guide are system administrators who use the FSM CALA. Users of the guide should have some knowledge about Unix and/or Windows operating system and the FSM CALA.

List of documents

FileNet System Monitor CALA Guide
: Datatypes that can be processed by the FSM CALA

FileNet System Monitor Monitoring Guide
: Description of all monitors contained in FileNet System Monitor

FileNet System Monitor Task Guide
: Description of all tasks contained in FileNet System Monitor

FileNet System Monitor Users Guide
: Installation guide

FileNet System Monitor Release Notes
: Description of changes and bugfixes

General information

Where you find this guide

You can find this documentation on the FSM installation CDROM in the following folder:

UNIX: `<Mount point>/INSTALL/docs`

Windows: `<Drive letter>:\INSTALL\docs`

Typeface Conventions

The guide uses several typeface conventions for special terms and actions. These conventions have the following meaning:

code Keywords and code examples occur like this

`varname` Variable names occur like `this`

`filename` File names occur like `this`

`constant` Constants and names of tasks, monitors etc. appear like `this`

command Command names appear like **this**

`parameter` Parameters and options for commands apperar like *`this`*

`userinput` Values that th user must provide appear like **`this`**

`Computer output` Output from programs appears like `this`

guilabel Names of windows, dialogs, and other controls appear like this

Programlistings appear like this:

```
# a program listing
echo "This is an example program listing (shell script) with nothing bu ↙
t an extremly long echo command"
exit 0
```

> **Note:** The character ↙ at the end of a line in a `computer output` or program listing shows, that the line has been wrapped and is continued in the next line.

Contacting FileNet Support

We are very interested in hearing from you about your experience with the product. We welcome your suggestions for improvements.

If you encounter difficulties with the FSM please contact the FileNet support (http://www.filenet.com).

Chapter 4. V2S Editor Overview

What is V2S?

V2 is a language for logfile description. It is used by the CALA's v2 format filter to recognize events from complex logfiles and to transform them into the CALA event format FIR.

A V2 syntax description file (V2S) is a text file containing a logfile description in the v2 format.

For further information about the v2 format filter and FIRs refer to the FSM CALA User's Guide, the V2 syntax is described later.

The V2S Editor

The V2S Editor is a graphical tool, that helps you to create and modify V2 syntax description files. You can also test V2S files against logfiles to see which events are recognized and which FIRs would be created if CALA processes this file.

The V2S Editor also provides a BAROC export feature for CALA/Tivoli integration.

Chapter 5. Installation

Installation Requirements

Supported V2S Editor platforms

For detailed information about supported V2S Editor platforms and required Java JRE / JDK versions check the latest release notes

Installation

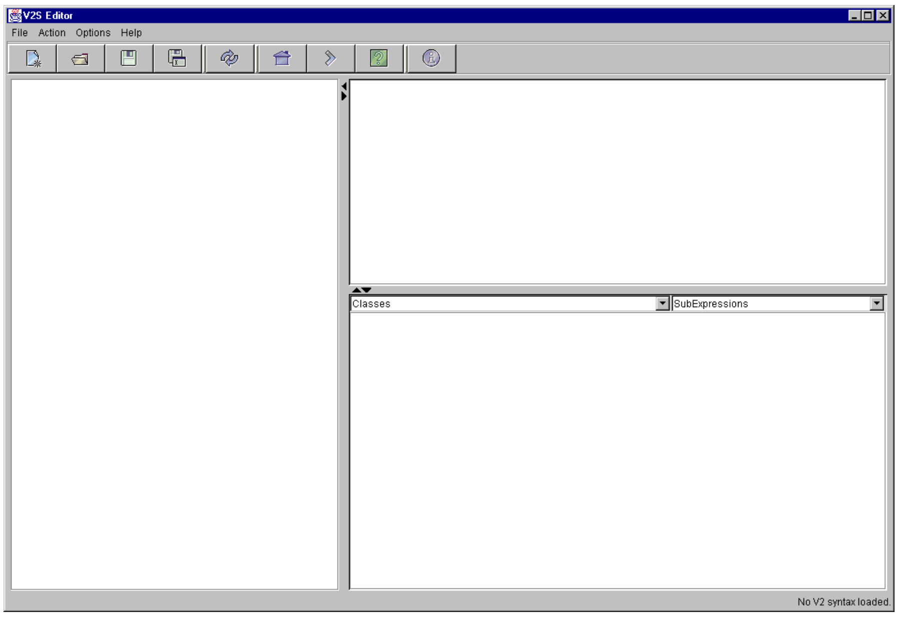
The V2S Editor can be installed by using the CALA’s Tivoli integration for detailes information refer to the CALA User s Guide.

To install the V2S Editor manually, the program **install_cala.sh** from the CALA installation CD can be used. This program is described in the appendix of the CALA Users’s Guide.

Chapter 6. Using V2S Editor

V2S Editor Basics

After starting the V2S Editor you get the following window:



The V2S Editor main window

Note: The screenshot in this guide were taken from a V2S Editor with the Windows look & feel configured.

If you are using any other look & feel manager, the look of your user interface may differ in some details.

The V2S Editor main window consists of three parts:

- the logfile area (left)
- the result area (upper right)
- and the v2s area (lower right).

The logfile, for which a v2 specification should be created, can be load into the logfile area by choosing the Open logfile item from the File menu. The logfile area is writable, so that events can be added manually.

The result area is a read-only area which is used to display the resulting FIR when a v2 specification is tested against a logfile.

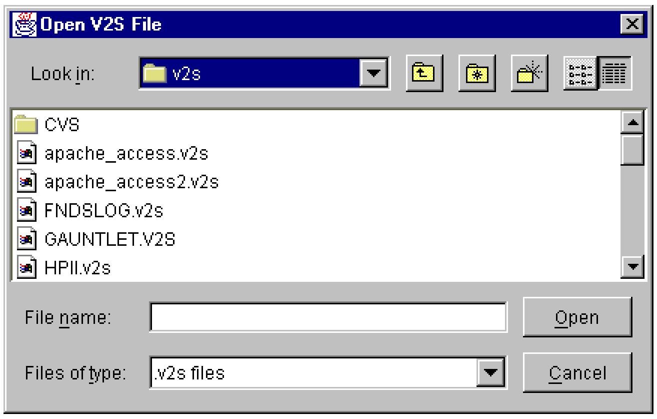
The v2s area is a editable text area which holds the v2 specification to be edited. It can be filled with an existing v2s file by choosing the item Open V2S from the File menu. The v2s area also contains two list boxes for direct access to class and subexpression definitions.

The size of the three areas can be changed by moving the separators with the mouse pointer.

Editing a V2S file

Loading a V2S file into the editor

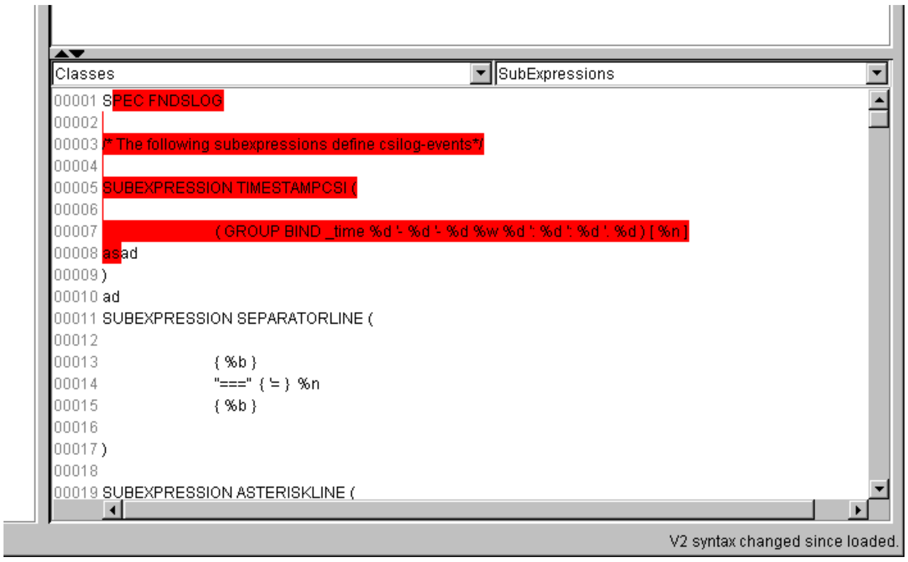
To load a v2s file, choose Open V2S from the File menu or press the corresponding toolbar button and a file dialog appears.



The filechooser dialog

When a v2s file was chosen and the Open button is pressed, the file dialog disappears and the chosen file is loaded into the v2s area, where it can be edited.

If the loaded v2s file is syntactically incorrect, a error dialog is displayed and the region where the error is suspected is highlighted.



```
SPEC FNDSLOG

/* The following subexpressions define csilog-events*/

SUBEXPRESSION TIMESTAMPCSI (

                    ( GROUP BIND _time %d '- %d '- %d %w %d ': %d ': %d '. %d ) [ %n ]
asad
)
ad
SUBEXPRESSION SEPARATORLINE (

                    { %b }
                    "===" { '= } %n
                    { %b }

)

SUBEXPRESSION ASTERISKLINE (
```

V2 syntax changed since loaded.

The error message when loading an invalid v2s file

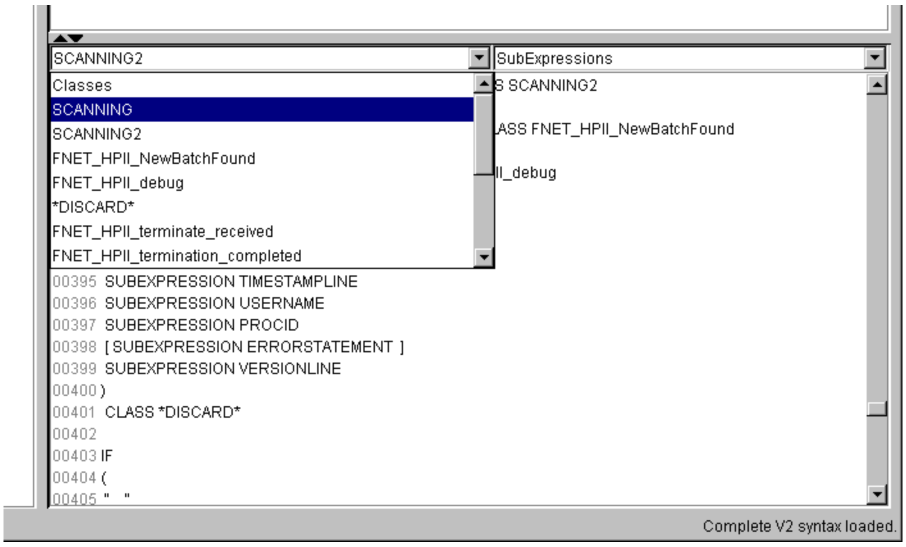
After the error has been corrected, the new specification has to be loaded into the v2s parser by choosing Reload V2 specification from the Action menu or by pressing the reload button from the toolbar.

Editing the V2 specification

An experienced user may like to use the V2S Editor like a normal text editor to create and modify v2s files manually, for less experienced users all modifications can also be done using the graphical user interface explained below.

Direct jump to class and subexpression definition

You can directly jump to the definition of a class or subexpression by selecting the corresponding entry in the listboxes above the v2s text area. The text view is scrolled to the selected class or subexpression definition .



```
SCANNING2                                SubExpressions
Classes                                  S SCANNING2
SCANNING
SCANNING2                                ASS FNET_HPII_NewBatchFound
FNET_HPII_NewBatchFound
FNET_HPII_debug                          II_debug
*DISCARD*
FNET_HPII_terminate_received
FNET_HPII_termination_completed
 SUBEXPRESSION TIMESTAMPLINE
 SUBEXPRESSION USERNAME
 SUBEXPRESSION PROCID
 [ SUBEXPRESSION ERRORSTATEMENT ]
 SUBEXPRESSION VERSIONLINE
)
 CLASS *DISCARD*

IF
(
"  "
                                            Complete V2 syntax loaded.
```

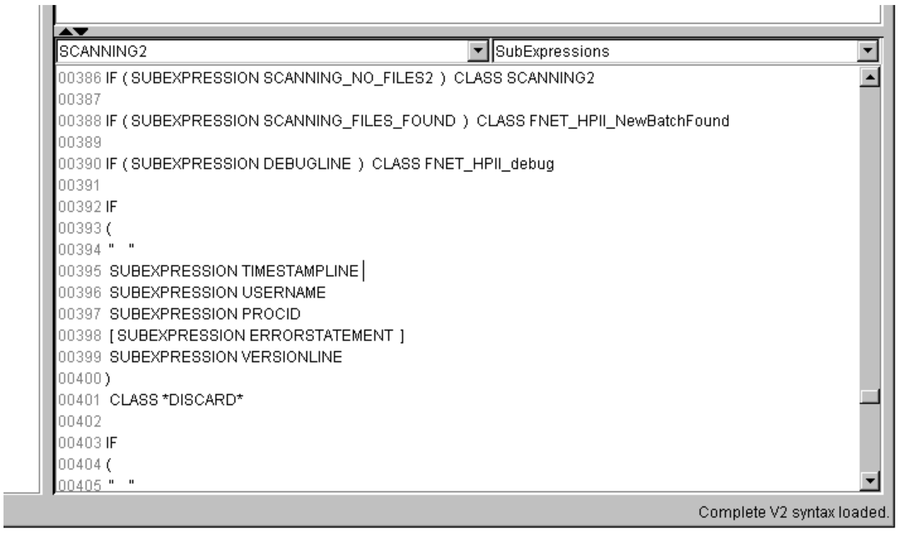
Jumping to a class definition by selecting it's name from the listbox

Editing expressions properties

The properties of any expression can be edited by selecting the Properties item in the popup-menu. The popup-menu appears when the mouse is placed over an expression and the right mouse button is pressed.

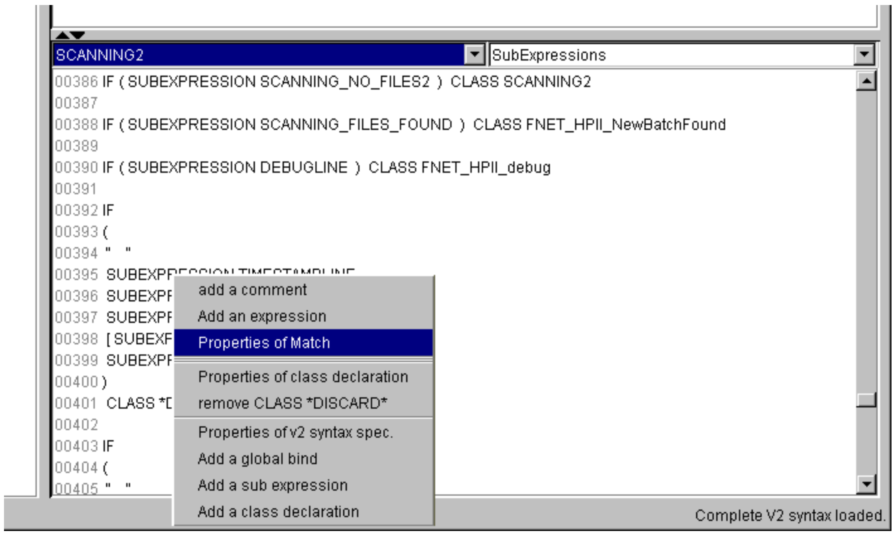
The contents of the properties window depends on the selected expression. If any property has been changed and the ok button is pressed, the properties windows disappears and the expression is changed.

The screenshots below shows an example property change: the match for the subexpression `TIMESTAMPLINE` is changed into a match for the subexpression `VERSIONLINE`.



```
SCANNING2                                SubExpressions
IF ( SUBEXPRESSION SCANNING_NO_FILES2 ) CLASS SCANNING2

IF ( SUBEXPRESSION SCANNING_FILES_FOUND ) CLASS FNET_HPII_NewBatchFound

IF ( SUBEXPRESSION DEBUGLINE ) CLASS FNET_HPII_debug

IF
(
"  "
 SUBEXPRESSION TIMESTAMPLINE
 SUBEXPRESSION USERNAME
 SUBEXPRESSION PROCID
 [ SUBEXPRESSION ERRORSTATEMENT ]
 SUBEXPRESSION VERSIONLINE
)
 CLASS *DISCARD*

IF
(
"  "
                                            Complete V2 syntax loaded.
```

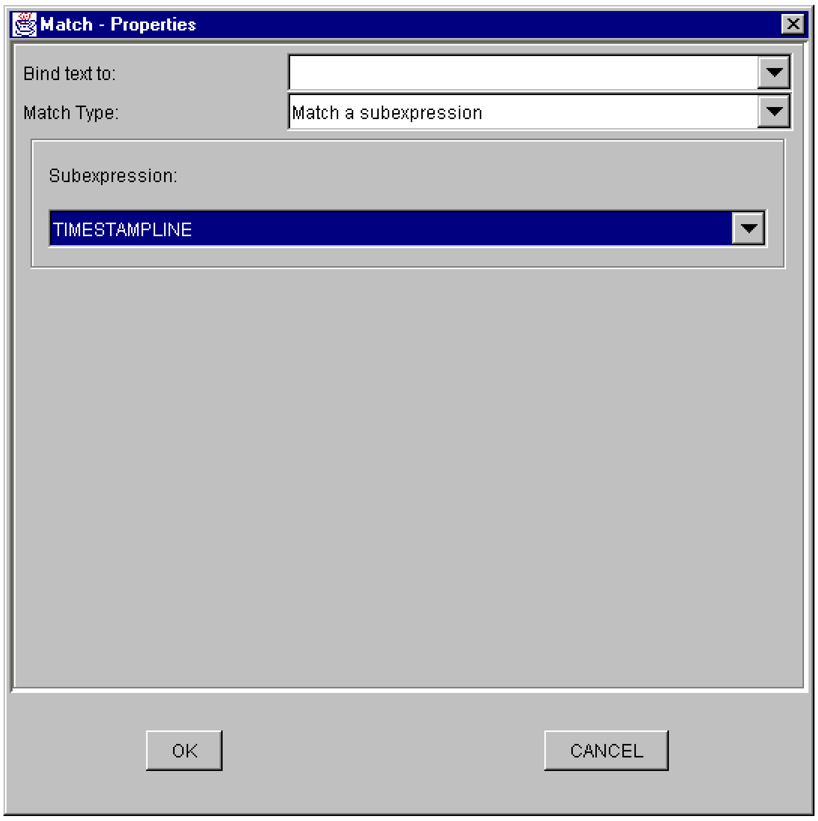
*Editing expression properties: The sub expression *DISCARD* before the change*



```
IF ( SUBEXPRESSION SCANNING_NO_FILES2 ) CLASS SCANNING2

IF ( SUBEXPRESSION SCANNING_FILES_FOUND ) CLASS FNET_HPII_NewBatchFound

IF ( SUBEXPRESSION DEBUGLINE ) CLASS FNET_HPII_debug

IF
(
 "  "
 SUBEXPRESSION TIMESTAMPLINE
 SUBEXPF
 SUBEXPF
 [ SUBEXF
 SUBEXPF
)
 CLASS *D

IF
(
 "  "
```

Complete V2 syntax loaded.

Editing expression properties: Opening the properties window.



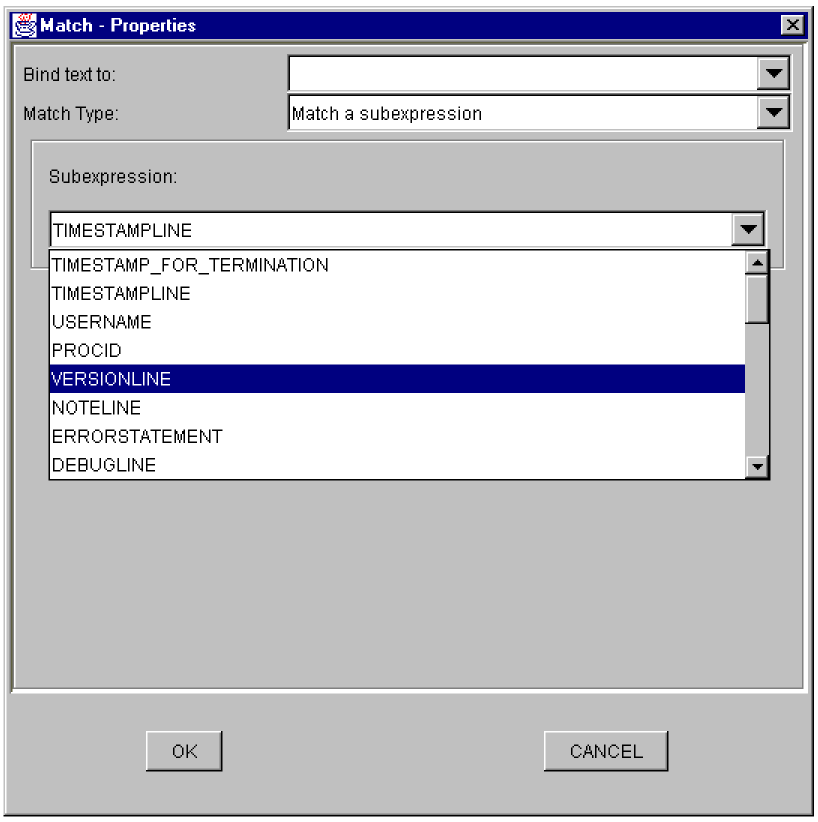
Bind text to:

Match Type: Match a subexpression

Subexpression:

TIMESTAMPLINE

Editing expression properties: The properties window before the change.



Bind text to:

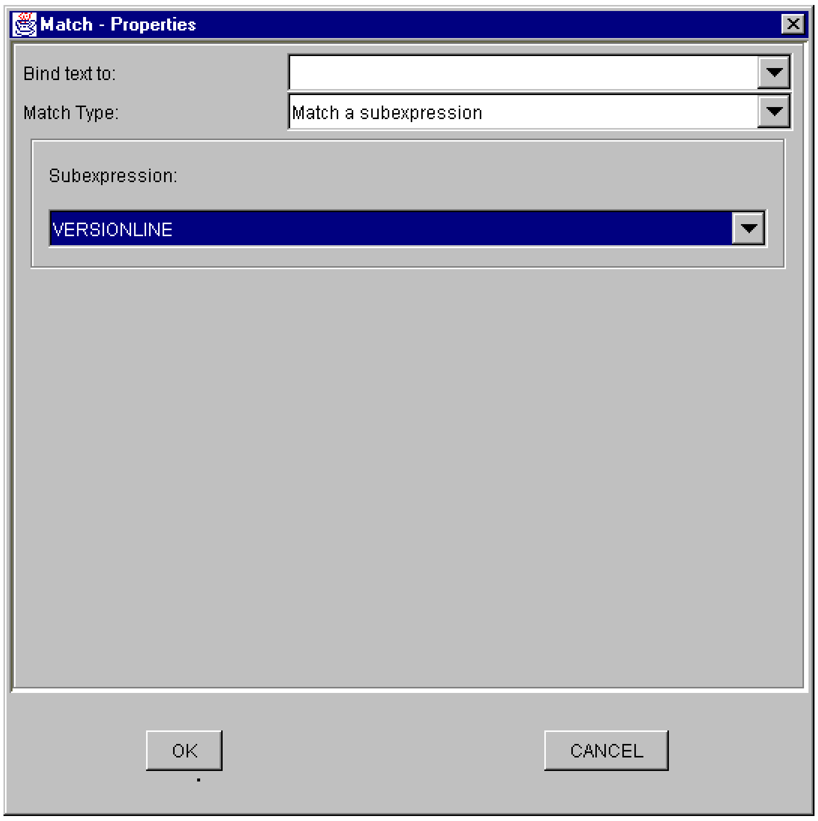
Match Type: Match a subexpression

Subexpression:

TIMESTAMPLINE

TIMESTAMP_FOR_TERMINATION
TIMESTAMPLINE
USERNAME
PROCID
VERSIONLINE
NOTELINE
ERRORSTATEMENT
DEBUGLINE

Editing expression properties: Changing `TIMESTAMPLINE` *to* `VERSIONLINE`



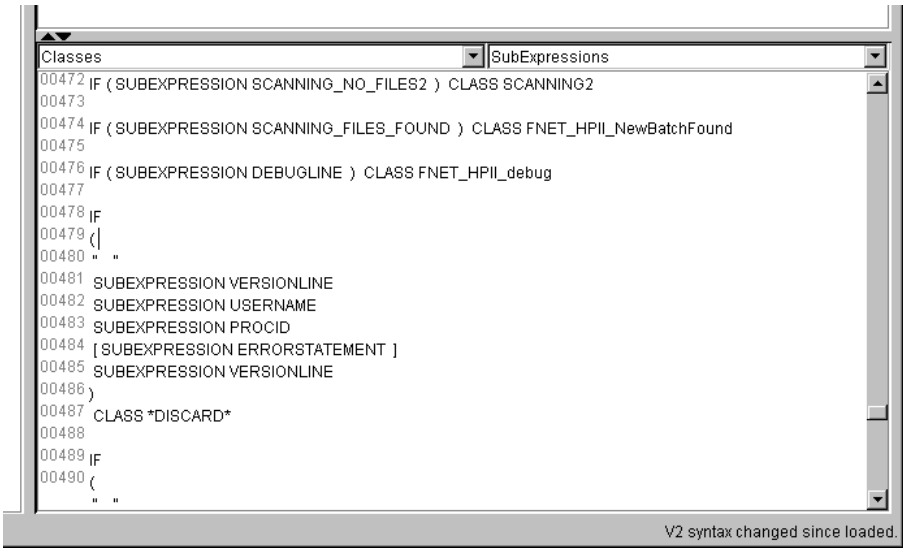
Bind text to:

Match Type: Match a subexpression

Subexpression: VERSIONLINE

Editing expression properties: The properties window after the sub expression `VERSIONLINE` has been selected.

After pressing OK, the modifications are applied to the v2 specification.



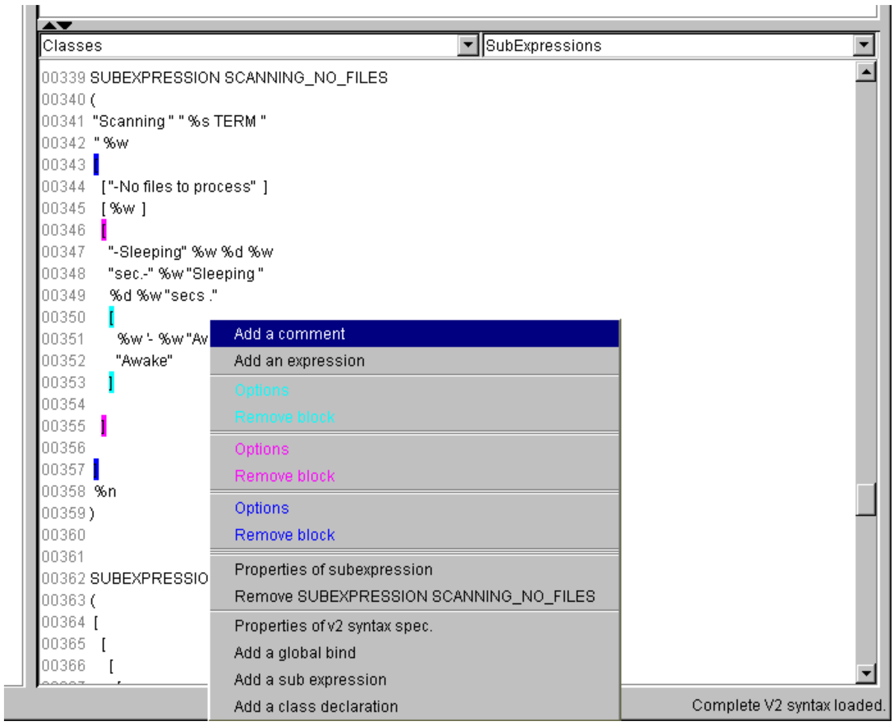
```
IF ( SUBEXPRESSION SCANNING_NO_FILES2 ) CLASS SCANNING2

IF ( SUBEXPRESSION SCANNING_FILES_FOUND ) CLASS FNET_HPII_NewBatchFound

IF ( SUBEXPRESSION DEBUGLINE ) CLASS FNET_HPII_debug

IF
(
" "
 SUBEXPRESSION VERSIONLINE
 SUBEXPRESSION USERNAME
 SUBEXPRESSION PROCID
 [ SUBEXPRESSION ERRORSTATEMENT ]
 SUBEXPRESSION VERSIONLINE
)
 CLASS *DISCARD*

IF
(
 " "
```

V2 syntax changed since loaded.

Editing expression properties: The modified class deinition.

Editing syntactically blocks

When the popup-menu is shown for an expression which is within one or more syntactically blocks, there are two menu entries for each of these blocks. The brackets enclosing each of these blocks are marked with different colors and so are the corresponding menu entries. This simplifies finding the correct menu entry for the block to modify.



```
SUBEXPRESSION SCANNING_NO_FILES
(
 "Scanning" " %s TERM "
 " %w
 [
  ["-No files to process" ]
  [ %w ]
  [
   "-Sleeping" %w %d %w
   "sec.-" %w "Sleeping"
   %d %w "secs ."
   [
    %w '- %w "Av
    "Awake"
   ]

  ]
 ]
 %n
)


SUBEXPRESSIO
(
 [
  [
   [
```

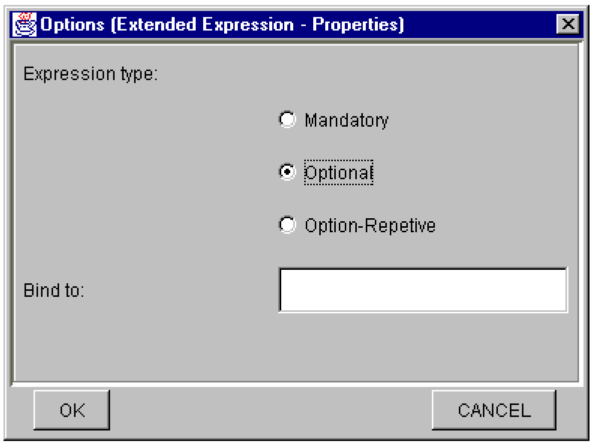
Complete V2 syntax loaded.

The v2s area popup window

There are two menu items for each block:

- Options
- Remove block

Options is chosen, a dialog similar to the properties dialog of expressions appears, where the type of options and a optional bind slot (a FIR slot which receives the complete text matched within this block) can be set.


Options (Extended Expression - Properties)

Expression type: Mandatory [✓ Optional] Option-Repetive

Bind to:

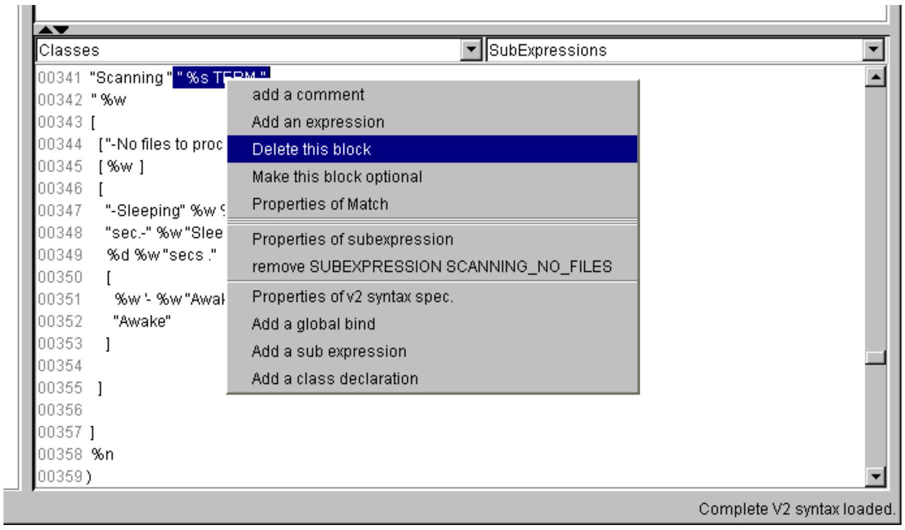
The block options dialog

If the Remove block item is selected, a confirmation dialog appears and the corresponding block is deleted (if confirmed).

A new block can be inserted by marking one or more expressions and choosing the popup-menu entry Make this block optional (which only appears if a block is marked). If done so, the Options dialog for the new block is displayed and if OK is pressed, the selected expressions are put into a block with the choosen properties.

Deleting expressions

One or more expressions can be deleted by marking them and choosing Delete this block from the popup-menu.



```
"Scanning " " %s TERM "
" %w
[
 ["-No files to proc
 [ %w ]
 [
  "-Sleeping" %w 
  "sec.-" %w "Slee
  %d %w "secs ."
  [
   %w '- %w "Awal
   "Awake"
  ]

 ]

]
%n
)
```

Complete V2 syntax loaded.

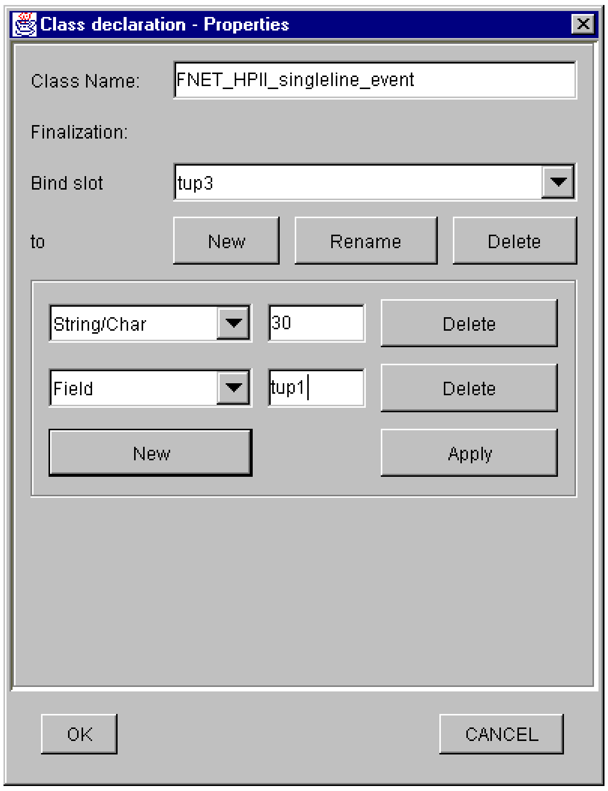
Deleting an expression block

Adding new expressions and comments

New expressions or comments can be added by selecting the popup-menu items Add an expression or Add a comment . If one of these items has been selected, a properties menu appears and the new expression or comment is inserted at the cursor position.

Adding, modifying and deleting classes, subexpressions and global binds

The property dialog of a class, subexpression or global bind is available in the popup-menu of each expression belonging to the class, subexpression or global bind definition (see screenshots above).



Class Name: FNET_HPII_singleline_event
Bind slot: tup3
String/Char 30
Field tup1

The class properties dialog

The classes properties dialog has one specialty: With listbox Bind slot to you can choose the slots definition you want to edit in the frame below. If a new slot definition has to be added, the New button must be selected.

The bind-to frame contains a list of strings and fields which are bound to the slot. If any changes are made in this frame, they have to be committed by pressing the Apply button before changing the selection of the listbox. To change the class declaration permanently, the OK button of the dialog windows must be pressed.

The menu entry for removing a class, subexpression or global bind is always available, when the properties dialog for this definition is available.

The popup-menu items for adding new classes, subexpressions and global binds are available everywhere in the v2s file. After choosing one of these items, a window with the existing classes, subexpressions or global binds appears where the position for the new definition can be set (before or after the chosen definition).

If this has been selected, a properties dialog for the new definition appears. If it has been committed with OK , the new definition is inserted into the v2 specification.

A new class or subexpression definition is initialized with a default expression, which can be changed by the user afterwards.

Properties of V2 specification

The menu item Properties of v2 specification is available everywhere in the v2 specification area. The only property which can be changed is the specification name.

Reloading the v2 specification after modifications

The menu item Reload V2S or the toolbars Reload button tells the V2S Editor to update its internal information about the loaded v2 specification.

If the v2 specification has been changed, a reload is needed before any tests on this specification are performed. If a reload is needed, the menu item and toolbar button are enabled, otherwise they are disabled.

After manually changing the v2 specification a reload is also needed to verify the syntax and update the markers for the context menus.

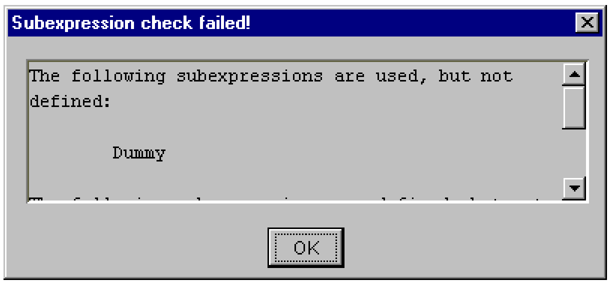
Checking the V2S file for correctness

The syntax of the v2 specification is checked each time a new file is loaded or the Reload button is pressed (see *Reloading the v2 specification after modifications*).

If the syntax check has been passed, the `v2fmtfilt` will accept the v2s file as a correct v2 format syntax.

The V2SEdit gives you an additional test feature to check for unused or undefined subexpressions. To perform this check, choose the item Check subexpression from the Action menu.

If any error is found, a information window is displayed (see screenshot below), if the v2s file is ok, the message `Subexpression check ok` is displayed in the status bar on the lower left of the V2S Editor window.



Subexpression check failed!

```
The following subexpressions are used, but not
defined:

        Dummy
```

The Subexpression check failed dialog

Saving an edited V2S file

To save an edited V2S file, choose one of the menu items Save V2S or Save V2S as from the File menu or one of the corresponding toolbar buttons.

Creating a new V2S file

A new V2S file can be created by selecting New V2S from the File menu. A properties dialog for the new V2 specification appears and, if committed, a new V2 specification is loaded into the v2s area.

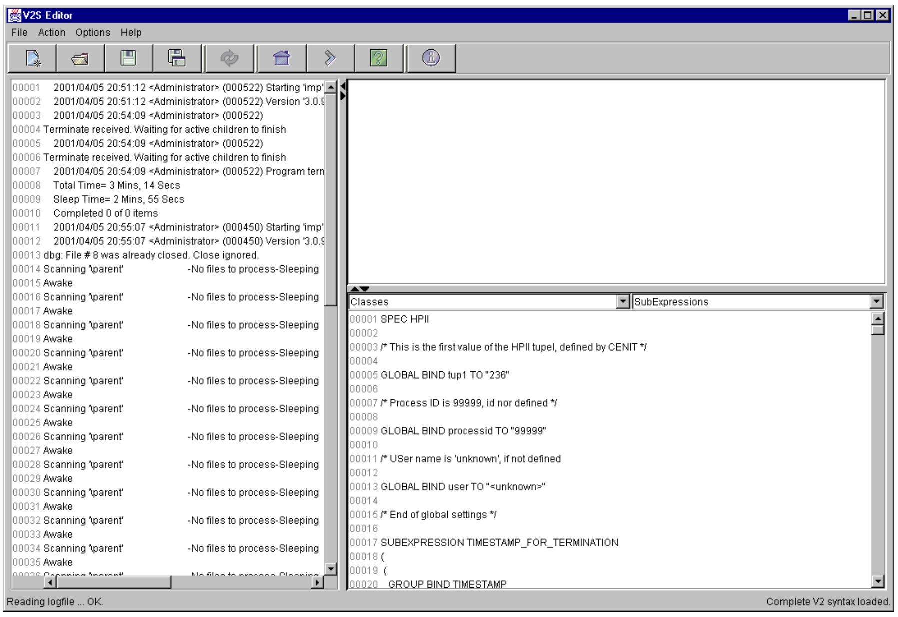
Each new v2 specification has one standard class for Unknown events. In most cases you should leave this class in the specification (as the last class declaration) to match events which are not matched by any other event class.

Using a logfile for creating and testing a V2 specification

Loading a logfile

To load a logfile the menu item Open Logfile in the File menu must be selected.

The logfile is loaded into the logfile area (left) and can be edited e.g. for removing or duplicating some events.



```
   2001/04/05 20:51:12 <Administrator> (000522) Starting 'imp'
   2001/04/05 20:51:12 <Administrator> (000522) Version '3.0.9
   2001/04/05 20:54:09 <Administrator> (000522)
Terminate received. Waiting for active children to finish
   2001/04/05 20:54:09 <Administrator> (000522)
Terminate received. Waiting for active children to finish
   2001/04/05 20:54:09 <Administrator> (000522) Program tern
   Total Time= 3 Mins, 14 Secs
   Sleep Time= 2 Mins, 55 Secs
   Completed 0 of 0 items
   2001/04/05 20:55:07 <Administrator> (000450) Starting 'imp'
   2001/04/05 20:55:07 <Administrator> (000450) Version '3.0.9
dbg: File # 8 was already closed. Close ignored.
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
Scanning '\parent'                 -No files to process-Sleeping
Awake
```

```
SPEC HPII

/* This is the first value of the HPII tupel, defined by CENIT */

GLOBAL BIND tup1 TO "236"

/* Process ID is 99999, id nor defined */

GLOBAL BIND processid TO "99999"

/* USer name is 'unknown', if not defined

GLOBAL BIND user TO "<unknown>"

/* End of global settings */

SUBEXPRESSION TIMESTAMP_FOR_TERMINATION
(
 (
  GROUP BIND TIMESTAMP
```

Reading logfile ... OK.

Complete V2 syntax loaded.

Loading a logfile

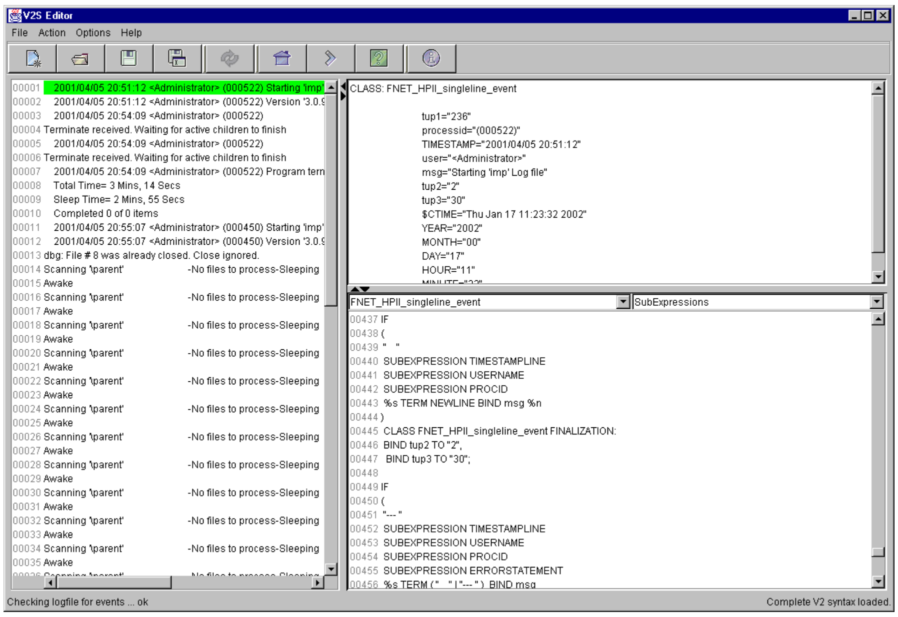
Parsing the logfile with the v2 specification

If a v2 specification and a logfile is loaded, the v2 specification can be tested against the logfile.

If Find first event in the Action menu is selected, the v2 parser tries to match the logfile from the beginning and displays the matched event in the result area. The event is marked in the logfile and the v2s area is scrolled to the definition of the matched class.

When Find next event is choosen, the parser starts event processing at the cursor position. The find actions places the cursor after the last found event, but one can move it to any other position.

If no event could be found, the message `no more events` is displayed in the status line below the logfile area.



```
2001/04/05 20:51:12 <Administrator> (000522) Starting 'imp'
2001/04/05 20:51:12 <Administrator> (000522) Version '3.0.9
2001/04/05 20:54:09 <Administrator> (000522)
Terminate received. Waiting for active children to finish
2001/04/05 20:54:09 <Administrator> (000522)
Terminate received. Waiting for active children to finish
2001/04/05 20:54:09 <Administrator> (000522) Program tern
Total Time= 3 Mins, 14 Secs
Sleep Time= 2 Mins, 55 Secs
Completed 0 of 0 items
2001/04/05 20:55:07 <Administrator> (000450) Starting 'imp'
2001/04/05 20:55:07 <Administrator> (000450) Version '3.0.9
dbg: File # 8 was already closed. Close ignored.
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
Scanning '\parent'            -No files to process-Sleeping
Awake
```

```
CLASS: FNET_HPII_singleline_event

        tup1="236"
        processid="(000522)"
        TIMESTAMP="2001/04/05 20:51:12"
        user="<Administrator>"
        msg="Starting 'imp' Log file"
        tup2="2"
        tup3="30"
        $CTIME="Thu Jan 17 11:23:32 2002"
        YEAR="2002"
        MONTH="00"
        DAY="17"
        HOUR="11"
```

FNET_HPII_singleline_event | SubExpressions

```
IF
(
"   "
 SUBEXPRESSION TIMESTAMPLINE
 SUBEXPRESSION USERNAME
 SUBEXPRESSION PROCID
 %s TERM NEWLINE BIND msg %n
)
 CLASS FNET_HPII_singleline_event FINALIZATION:
 BIND tup2 TO "2",
  BIND tup3 TO "30";

IF
(
"--- "
 SUBEXPRESSION TIMESTAMPLINE
 SUBEXPRESSION USERNAME
 SUBEXPRESSION PROCID
 SUBEXPRESSION ERRORSTATEMENT
 %s TERM ( "   " | "--- " )  BIND msg
```

Checking logfile for events ... ok   Complete V2 syntax loaded.

Parsing a logfile with the v2 specification

To get more information about the parsing process, select Show parse log from the Action menu and the result area shows a detailed trace from the last parsing action.

This information can be helpful to find errors in a v2 specification or simply to understand how v2s works.

Further log file parsing

The V2S Editor has additional features for log file parsing, which can be used for debugging

Search for event

The Search for event menu item in the Action menu is similar to the Find next event , but does also find events which don t start directly at the cursor position. The non-matching text is skipped.

Load partial V2S

This option allows only a part of class definitions to be loaded into the parser. This can e.g. be used in combination with Search for event to find all events of a special class.

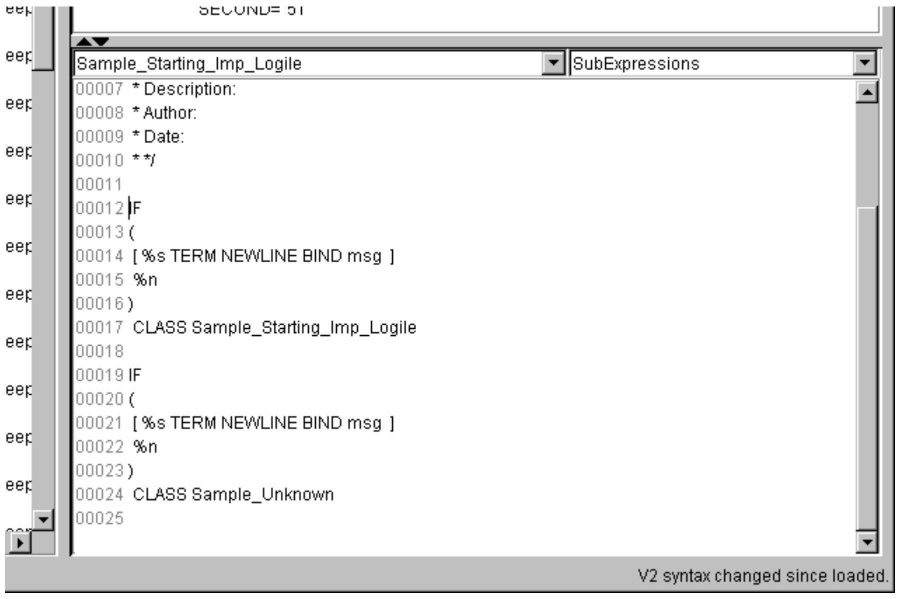
After selecting this menu item, a dialog box appears, where the classes to load can be selected. To load the full specification back into the parser, the v2 specification has to be reloaded (menu item Reload V2S).

Creating an event description from the log file

The V2S Editor helps you to create new event class descriptions from a log file.

First a new event class has to be created, this can be done by choosing the item Add a class declaration in the v2s area popup menu. The new event class is created with the default expression.

(See class `Sample_Starting_Imp_Logfile` in the screenshot)

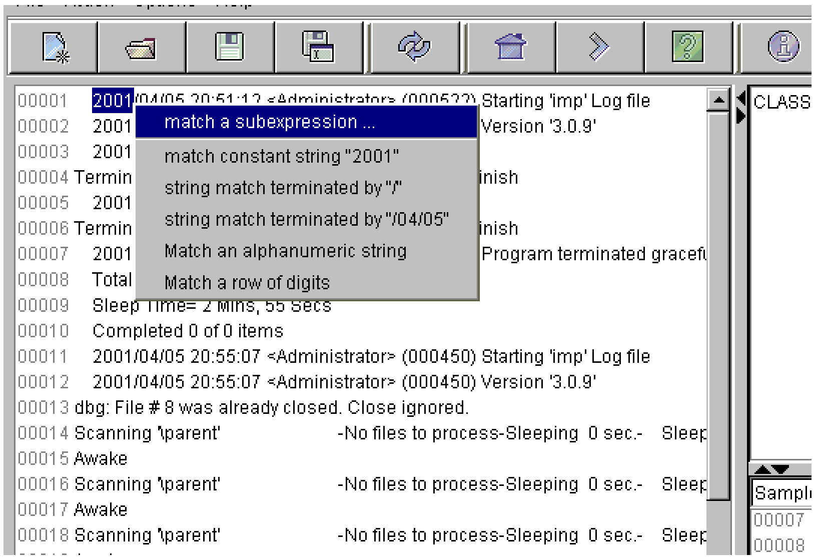


```
 * Description:
 * Author:
 * Date:
 **/

IF
(
 [ %s TERM NEWLINE BIND msg ]
 %n
)
 CLASS Sample_Starting_Imp_Logile

IF
(
 [ %s TERM NEWLINE BIND msg ]
 %n
)
 CLASS Sample_Unknown

```

Creating an event description from the log file - step 1

Moving the cursor has behind the opening brace of this class definition tells the editor, that new expression have to be placed there.

Now take a look at the log file area and mark the first field of the event you want to define. If done so, the popup menu shows you a list of expression the marked text would match with.


match a subexpression ...
match constant string "2001"
string match terminated by "/"
string match terminated by "/04/05"
Match an alphanumeric string
Match a row of digits
CLASS
Sampl

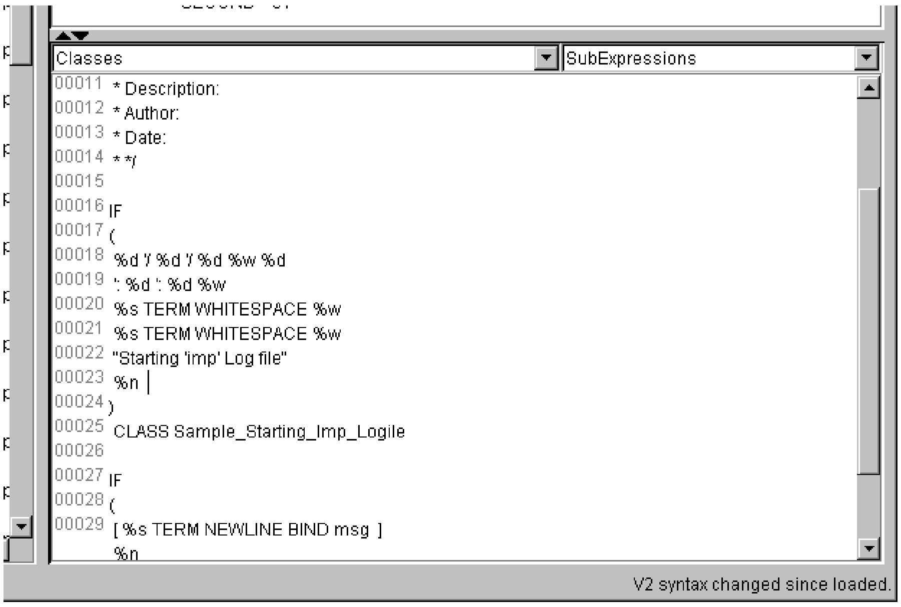
```
00001  2001/04/05 20:51:12 <Administrator> (000522) Starting 'imp' Log file
00002  2001                                        Version '3.0.9'
00003  2001
00004 Termin                                        inish
00005  2001
00006 Termin                                        inish
00007  2001                                         Program terminated gracefu
00008  Total
00009  Sleep Time= 2 Mins, 55 Secs
00010  Completed 0 of 0 items
00011  2001/04/05 20:55:07 <Administrator> (000450) Starting 'imp' Log file
00012  2001/04/05 20:55:07 <Administrator> (000450) Version '3.0.9'
00013 dbg: File # 8 was already closed. Close ignored.
00014 Scanning '\parent'          -No files to process-Sleeping  0 sec.-   Sleep
00015 Awake
00016 Scanning '\parent'          -No files to process-Sleeping  0 sec.-   Sleep
00017 Awake
00018 Scanning '\parent'          -No files to process-Sleeping  0 sec.-   Sleep
```

Creating an event description from the log file - step 2

After choosing the text type, the v2 representation of this text is inserted at the cursor position in the v2s area.

There is one special item in this popup menu the item Match a subexpression checks the marked text against all defined subexpressions and gives you a list of the ones which match. After selecting the a subexpression, a match for it is inserted into the v2s file.

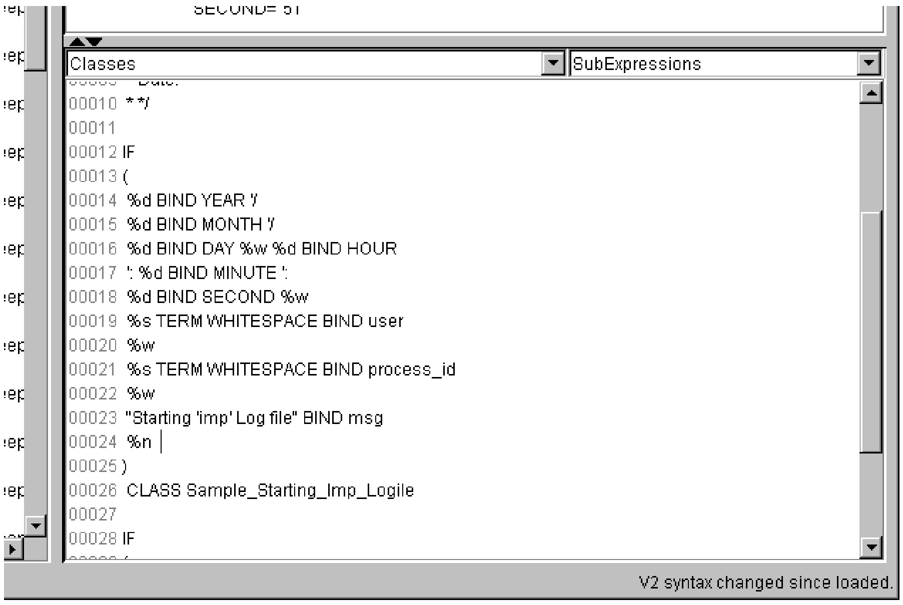
Using this feature, the complete class definition can be build. (Don't forget to remove the standard expression from the definition, if it is no longer needed.)



```
 * Description:
 * Author:
 * Date:
 * */

IF
(
 %d '/' %d '/' %d %w %d
 ':' %d ':' %d %w
 %s TERM WHITESPACE %w
 %s TERM WHITESPACE %w
 "Starting 'imp' Log file"
 %n
)
 CLASS Sample_Starting_Imp_Logile

IF
(
 [ %s TERM NEWLINE BIND msg ]
 %n
```

Creating an event description from the log file - step 3

Finally there may be some options and slot assignments to make and the definition is complete.



```
SECOND= 51
**/

IF
(
 %d BIND YEAR '/'
 %d BIND MONTH '/'
 %d BIND DAY %w %d BIND HOUR
 ': %d BIND MINUTE ':
 %d BIND SECOND %w
 %s TERM WHITESPACE BIND user
 %w
 %s TERM WHITESPACE BIND process_id
 %w
 "Starting 'imp' Log file" BIND msg
 %n
)
 CLASS Sample_Starting_Imp_Logile

IF
```

V2 syntax changed since loaded.

Creating an event description from the log file - step 4

Chapter 7. Additional command line programs

There are two additional command line programs shipped with the V2SEdit:

- **v2scheck**
- **v2s2baroc**

These programs are available as Unix shell scripts (`v2scheck.sh`, `v2s2baroc.sh`) and Windows batch files (`v2scheck.bat`, `v2s2baroc.bat`). This document describes the handling of the Unix shell scripts, but the usage of the batch files is exactly the same.

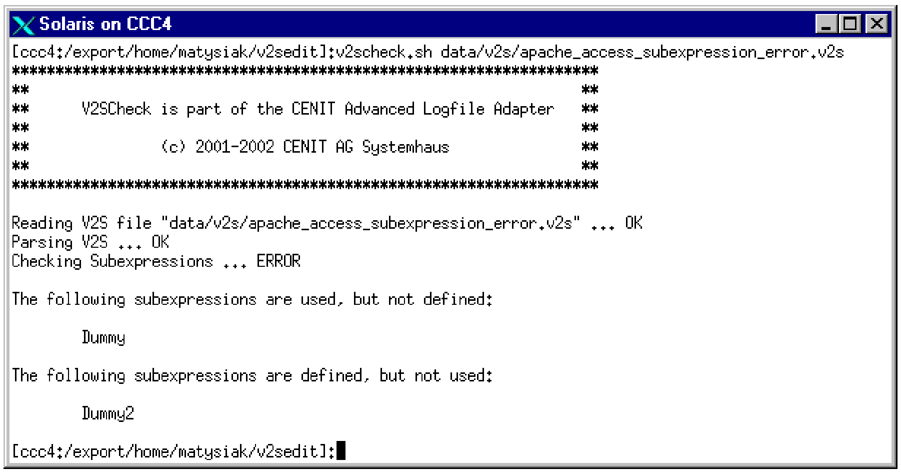
On all systems a java runtime environment >= JRE 1.3 must be reachable over the `PATH` environment variable to execute the programs.

v2scheck - Check a v2s file for correctness

The **v2scheck** program performs a syntax check and a subexpression check for a given v2s format file. If any check failes, detailed error messages are printed on the screen.

Usage

The v2scheck program takes only one argument: the name of the file to check.



```
[ccc4:/export/home/matysiak/v2sedit]:v2scheck.sh data/v2s/apache_access_subexpression_error.v2s
********************************************************************
**                                                                **
**      V2SCheck is part of the CENIT Advanced Logfile Adapter    **
**                                                                **
**               (c) 2001-2002 CENIT AG Systemhaus                **
**                                                                **
********************************************************************

Reading V2S file "data/v2s/apache_access_subexpression_error.v2s" ... OK
Parsing V2S ... OK
Checking Subexpressions ... ERROR

The following subexpressions are used, but not defined:

        Dummy

The following subexpressions are defined, but not used:

        Dummy2

[ccc4:/export/home/matysiak/v2sedit]:
```

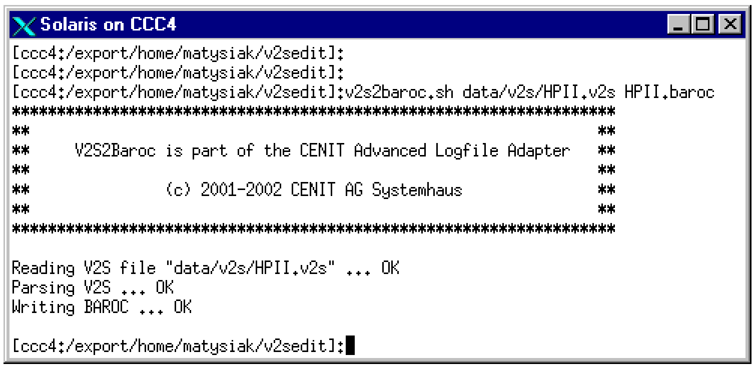
Using the v2scheck program

v2s2baroc - A baroc file generator

This program creates a Tivoli baroc file from a v2s format file. The baroc files contains a event description to be loaded into the T/EC.

Usage

The **v2s2baroc** program takes two arguments: the name of the v2s file and the name of the baroc file to be created.



```
[ccc4:/export/home/matysiak/v2sedit]:
[ccc4:/export/home/matysiak/v2sedit]:
[ccc4:/export/home/matysiak/v2sedit]:v2s2baroc.sh data/v2s/HPII.v2s HPII.baroc
********************************************************************
**                                                                **
**      V2S2Baroc is part of the CENIT Advanced Logfile Adapter   **
**                                                                **
**                (c) 2001-2002 CENIT AG Systemhaus               **
**                                                                **
********************************************************************

Reading V2S file "data/v2s/HPII.v2s" ... OK
Parsing V2S ... OK
Writing BAROC ... OK

[ccc4:/export/home/matysiak/v2sedit]:
```

Using the v2s2baroc program

Appendix A. The v2 format

The v2 format is the description language for complex logfile formats which do not comply with the logfile standard (single-line entries, fixed format).

The v2 format is capable of describing formats which

- possess a multi-line sentence format
- possess a sentence format which cannot be defined in advance without ambiguity, or which contains a repetitive sentence format

Storage form

Format files for V2FMTFILT must be saved as a file.The filename can have any extension, although the extension ".v2s" is recommended.

Identifiers

Identifiers are class names and variables (slots) in the V2S format.

Identifiers in V2S must start with a letter and can contain any sequence of alphanumeric characters.Valid characters include uppercase and lowercase letters as well as digits and the underscore

Identifiers are used directly by V2FMTFILT to set up FIRs (Filter Input Records). Variables with designators starting with a leading underscore are treated as temporary and do not occur in the resulting FIRs.

General design of the v2 format

A V2 format file contains three main sections:

- a header
- definitions of global variables
- declarations of sub expressions and classes

Comments

Comments are allowed before, between and after the sections and between expressions in the declarations section.

There are two different comment types supported, similar to comments in C/C++.

```
/* <comment> */
// <comment terminated by new line>
```

Example A-1. Example of comments in v2s

Header

The header has the format:

```
SPEC <name>
```

Figure A-1. Format of v2s spec expression

`<name>` is any identifier for the format specification.

```
SPEC SNA
```

Example A-2. Example of a v2s spec expression

The header information is obligatory.

Global Variables

Global variables are definitions of general FIR slots, which should occur in each created FIR. The class definition may overwrite or delete this slot.

```
GLOBAL BIND <slot name> TO "<string>"
```

Figure A-2. Format of v2s global bind expression

```
GLOBAL BIND source to cala
```

Example A-3. Example of a v2s global bind expression

The definition of global variables is optional.

Automatically assigned variables

There are some variables, which are automatically assigned by the parser. This variables can also be used in the class finalization.

> **Note:** Fields starting with $ should be accessed in a read only manner only.

field name	description
$HOSTNAME	name of the host the event occurred on

field name	description
$ORIGIN	ip address of the host the event occurred on
$ADAPTER_HOST	name of the host, which read the event
$LOGFILENAME	name of the logfile the event was read from
hostname	name of the host the event occurred on
origin	ip address of the host the event occurred on
adapter_host	name of the host, which read the event

Variables to set timestamp

The event’s timestamp is initialized with the current time (the time when the v2 format filter starts parsing the event). Using the following fields, the timestamp can be adjusted.

field name	description
DAY	day of month (1-31)
HOUR	hour (0-24 or 0-12 in 12-hour mode, see below)
MINUTE	minute (0-60)
SECOND	second (0-60)
TIME_POSTFIX	Setting `TIME_POSTFIX` to any value switches to 12-hour-mode. Sets time to P.M. if `TIME_POSTFIX` is set to any value starting with P or p.

Classes and sub-expressions

Classes

Every format description file must contain a series of class definitions. These definitions were processed top-down at parsing time. This means that if more than one definition matches, the first one is taken.

```
IF expression CLASS name [ FINALIZATION: BIND <slot> TO <any sequence of ↙
 V2S expressions > ]
```

Figure A-3. Format of v2s class expression

```
IF (
SUBEXPRESSION TIMESTAMP
"myprocess shut down"
) CLASS MYPROCSHUTDOWN
```

Example A-4. Example of a v2s class expression

A classname may occur several times in one format description file.

Sub-expressions

Sub-expressions can be defined and called in the same way as macros.

The declaration must take place in the File-Scope, i.e. at the same level as the classes are defined. Ideally, all SUBEXPRESSION definitions should be defined before the list of class definitions.

```
SUBEXPRESSION name ( expression )
```

Figure A-4. Format of v2s subexpression definition

```
SUBEXPRESSION IPADDREXPR ( GROUP BIND IPADDR %d { ’. %d } )
```

Example A-5. Example of a v2s subexpression definition

The "Macro" is called using

```
SUBEXPRESSION name
```

Figure A-5. Format of v2s subexpression call

```
"Text" SUBEXPRESSION IPADDR "Text"
```

Example A-6. Example of a v2s subexpression call

Expressions

Matching types

Character Match (individual characters)

Syntax < ’x > defines a character match.

<x> can be any character for which a match is to be found.

Example:

’A matches the letter A

This syntax makes it possible to match up special characters.

Character Match (individual characters by ASCII code)

Syntax ’\x defines a character match. `x` is the decimal ASCII code of the character to be found.

```
\65
```

Example A-7. Example v2s character match: matching the letter `A`

This syntax makes it possible to match up special characters.

Multi match (multiple match)

Syntax %x [BIND field] matches a sequence of characters and links the result to a specified field (slot).

```
%a BIND FIELD1
```

Example A-8. Example v2s: matching a sequence of alphanumeric chars

If the field name starts with a leading underscore, the field is for local use only and does not appear in the resulting event. Nevertheless it can be used in the finalization section of the class.

Multi match type d (decimal match)

> A sequence with at least one decimal numeral is matched.
>
> e.g. %d BIND NUMBER23

Multi match type a (alphanumeric match)

> A sequence of at least one alphanumerical character is matched. Alphanumeric characters include letters `A-Z`, `a-z`, as well as digits `0-9`.

Multi match type w (white space match)

A sequence with at least one white space character (space or tab key, ASCII characters `SPC` and `HT`, code 32 or 9) is matched.

Multi match type n (new line match)

Precisely one line feed is matched. (`LF`, ASCII-Code 10): where necessary, a `CR` (ASCII code 13) is skipped for this.

Multi match type b (blank line match)

Matches precisely one blank line. This can contain any number of `SPC` and `HT` characters.

Character Match s<number>

Using the notation %`<number>`s, it is possible to read out a definable number of characters. This makes it possible to disassemble an input string into any number of sub-sections, e.g. to generate a standard time format out of any given time stamp.

Multi match type s (string match)

There are six operational modes:
%s TERM ’x
%s TERM ’x BIND field
%s TERM \x
%s TERM \x BIND field

The first format matches all characters up to the specified terminator (not including this character), and links the result to a field when necessary. The character can be given as the character itself (x) or as it s ASCII code (\x).

%s TERM WHITESPACE
%s TERM WHITESPACE BIND field

The second format matches all characters up to the next white space character (`SPC`, `HT`, `CR` or `LF`), and links the result to a field if necessary.
%s TERM NEWLINE
%s TERM NEWLINE BIND field

The third format matches all characters up to the first line break (UNIX and DOS/Windows line breaks) and links the result to a field if necessary.
%s TERM BLANKLINE
%s TERM BLANKLINE BIND field

The fourth mode matches all characters up to the first blank line and links the result to a field when necessary.
%s TERM termination string
%s TERM termination string BIND field
%s TERM (alt. term. string 1 | alt. term. string2
%s TERM (alt. term. string 1 | alt. term. string2) BIND field

The fifth format matches all characters up to the first occurrence of the given termination string and links the result to a field if necessary. It is also possible to give a list of alternative termination strings, which means: match the characters up to the first occurrence of one of the given strings.
%s TERM SUBBEXPRESSION subexpr
%s TERM SUBBEXPRESSION subexpr BIND field

The sixth format matches all characters up to the next occurrence of subexpr (not including this subexpression) and links the result to a field if given.

To ensure the match is successful, **at least 1** character must be matched.

Multi match type S

This special type of string match behaves in the same way as the standard multi-match type **s** with one exception: processing of the string stops at the end of the first line.

%S TERM `<term expression>`
%S TERM `<term expression>` BIND field

> **Note:** The implementation of this match has changed from CALA version 1.1b to CALA version 2.1
>
> *Old implementation (<= CALA 1.1b):* Match the string up to the termination condition or if this condition is not fulfilled until the line ends, match the rest of the line.
>
> *New implementation (>= CALA 2.1):* Match if the termination condition can be fulfilled within the current line.

Constant string match

By specifying

```
"any text"
```

Figure A-6. Format of v2s constant string match

(any text in double quotes), precisely that section of text is matched.

You can also specify a list of alternative strings to match:

```
(" alt string1" |  "alt string2" |  "alt string 3" )
```

Figure A-7. Format of v2s constant string match with alternatives

Escape sequences have not yet been implemented. In an instance of this kind, the special character must take the form of a character match (‘<any character>).

Subexpression match

The following line calls a subexpression match

```
SUBEXPRESSION name
```

Figure A-8. Format of v2s subexpression match

The sub-expression indicated is matched (refer to subexpression section).

Mandatory, optional and repetitive expressions

Mandatory expression

The use of parentheses (round brackets) around any code group (<your code>) indicates that an expression is mandatory.

This means that all matches enclosed in brackets must be performed.

```
( ’A %d )
```

Example A-9. Example for a mandatory v2 expression group

Tihs matches the letter A and one or more numerals.

Examples:

Expression	**Source**	**Match**
(’A %d)	A1PQR	A1
(’A %d)	A2324 XYZ	A2324

Optional expression

The use of square brackets around any code group [<your code>] indicates that an expression is optional.

This means that all matches enclosed in these brackets should be made either 0 or 1 time.

```
’A %d [ ’. %d ]
```

Example A-10. Example for an optional v2 expression group

matches letter `A` and one or more digits and optional a following dot and another sequence of digits.

Expression	**Source**	**Match**
’A %d [’. %d]	A1PQR	A1
’A %d [’. %d]	A1.24XYZ	A1.24

Optional repetitive expression

The use of curly brackets around any code group { <your code> } indicates that an expression is optional, and can be repeated several times.

```
’A %d { ’. %d }
```

Example A-11. Example for an optional-repetitive v2 expression group

matches letter `A` and one or more digits as well as (optional and repetitive) a following dot and another sequence if digits.

Expression	**Source**	**Match**
’A %d { ’. %d }	A1PQR	A1
’A %d { ’. %d }	A1.24.35XYZ	A1.24.35

Group binding

An expression can be started with a group statement: GROUP BIND field

This binds all characters matched by this expression to a field. If a group statement is used within any expression, it must be set in parenthesis.

```
( GROUP BIND IPADDR %d { ’. %d } )
```

Example A-12. Example for a v2s group bind expression

matches a series of numerals interspersed with dots, assigning the field `IPADDR`.

If the field name starts with a leading underscore, the field is for local use only and does not appear in the resulting event. Nevertheless it can be used in the finalization section of the class.

Example of format file sna.v2s

This section provides a description of an SNA server error logfile:

```
SPEC SNA

SUBEXPRESSION TIMESTAMP (
   %d BIND HOUR
   ':
   %d BIND MINUTE
   ':
   %d BIND SECOND
   " "
   %a BIND TIMEZONE
   "  "
   %d BIND DAY
   " "
   %a BIND MONTH
   " "
   %d BIND YEAR
)

SUBEXPRESSION TIMESTAMPLINE (
   SUBEXPRESSION TIMESTAMP
   " "
   %d BIND CODE1
   '-
   %d BIND CODE2
   '(
   %d BIND CODE3
   '-
   %d BIND CODE4
   ')
   " "
   %a BIND CODE5
   " "
   '(
   %a BIND CODE6
   ')
   %n
)

IF (
   "============ Log file initialised " SUBEXPRESSION TIMESTAMP " ===== ↙
=======" %n
) CLASS LOGINIT

IF (
   SUBEXPRESSION TIMESTAMPLINE
   "Abnormal UNBIND request received" %n
   "Sense code" %w '= %w %a BIND SENSECODE %n
   "Local LU name" %w '= %w ( GROUP BIND LOCALLU %a '. %a ) %n
   "Partner LU name" %w '= %w ( GROUP BIND PARTNERLU %a '. %a ) %n
   "Mode name" %w '= %w %a BIND MODENAME %n
   "UNBIND RU :" %n ( GROUP BIND UNBINDRU %a { " " %a } ) %n
) CLASS ABNORMALUNBIND FINALIZATION:
     BIND msg TO "Abnormal UNBIND request received " + SENSECODE,
     BIND SENSECODE TO NOTHING;
```

```
054
```

Example A-13. An example v2s format file

The last class definition described here sub-divides the event into various slots (`SENSECODE`, `LOCALLU`, `PARTNERLU`, `MODENAME` and `UNBINDRU`) and into slots which are defined when sub-expression `TIMESTAMPLINE` is called up. Processing at the end of a class definition (`FINALIZATION`) involves combining slot from text "`Abnormal UNBIND request received`" and the content of the `SENSECODE` slot. The `SENSECODE` slot is deleted afterwards.

Appendix B. An example personal properties file

```
// Filename: V2SEdit_personal.properties
//
// Date: 12.12.2001
//
// Personal settings for V2S editor
//
// The settings in this file are user definable.
// Things which are not configured here (like button and dialog texts)
// are defined in V2SEdit.properties file or any of it’s parents.
// The V2SEdit.properties file also contains a default value for each
// of  the following parameters, which is used if the parameter is
// not configured here.
//
// The settings in this file are given like this:
//
//  [property name]=[value]
//
// For properties expecting color values, the following colors are
// supported:
//
// black
// blue
// cyan
// darkGray
// gray
// green
// lightGray
// magenta
// orange
// pink
// red
// white
// yellow
//

// All lines starting with // are comments.

// Look and Feel: The user interface manager
//
// 0 = use system default (motif on Unix, windows on Windows)
// 1 = motif
// 2 = windows (only in MS operating systems)
// 3 = metal
//
// default: 0
v2sedit.menu.options.lookandfeel.defaultmanager=3

// Line number mode at startup
//
// The line no. mode can be switched on for each
// of the text areas in the main window.
// Setting the property to 0 turns the line no. mode off,
// each other value turns it on.
```

```
//
// Line no. mode is switched on for logfile and v2s by default.
//
v2sedit.menu.options.linenumbers.logfile.startup=1
v2sedit.menu.options.linenumbers.v2s.startup=1
v2sedit.menu.options.linenumbers.result.startup=0

// Text font and size for the logfile area in the main window.
//
// Default: SansSerif, 12
//
v2sedit.logarea.font=SansSerif
v2sedit.logarea.fontsize=12

// Highlight color for found events.
v2sedit.logarea.eventcolor=green

// Text font and size for the v2s area in the main window.
//
// Default: SansSerif, 12
//
v2sedit.v2sarea.font=SansSerif
v2sedit.v2sarea.fontsize=12

// Highlight color for errors in v2 syntax
v2sedit.v2sarea.errorcolor=red

// Show subexpression as tooltip? (0=no, 1=yes)
v2sedit.v2sarea.showtooltips=1

// Text font and size for the result area.
// (This is the area where the FIRs are displayed.)
//
// Default: SansSerif, 12
//
v2sedit.resultarea.font=SansSerif
v2sedit.resultarea.fontsize=12

// Logfile path and extensions
v2sedit.openlogfile.filechooser.path=data/logs
v2sedit.openlogfile.filechooser.extension=.log

// V2S file path and extensions
v2sedit.openv2sfile.filechooser.extension=.v2s
v2sedit.openv2sfile.filechooser.path=data/v2s

// Baroc file path and extensions
v2sedit.exportbaroc.filechooser.extension=.baroc
v2sedit.exportbaroc.filechooser.path=data/baroc

// baroc export: header and footer
//
// This text is added at the beginning (header) or end (footer) of
// the BAROC file. {0} is the name of the v2 syntax.
//
v2sspec.barocexport.header=//\n// Filename:\n//\n//\t {0}.baroc\n//\n// Language:\n//\n//\t Tivoli BAROC\n//\n// Description:\n//\n//\t Created by the V2 format editor.\n//\n//\n\n
v2sspec.barocexport.footer=

// Display an "are you sure" dailog before exit? (0=no, 1=yes)
v2sedit.exit.displaydialog=0

// split lines in v2s windows at this length
v2sspec.splitLinesAtLenght=30

// Creating a new V2S -> default names
v2sedit.specdecl.defaultname=Unknown

// Comment at the beginning of the syntax
// {0} is the syntax name
v2sedit.specdecl.defaultheadcomment=V2 definition {0}\n *\n * (c) 2002 CENIT AG Systemhaus, Stuttgart (Germany)\n *\n * Description:\n * Author:\n * Date:\n *

// Default class name
//{0} is the syntax name
v2sedit.specdecl.addclass.defaultclass={0}_Unknown

// Default subexpression name
//{0} is the syntax name
v2sedit.specdecl.addsubexpression.defaultepxression={0}_UNKOWN

// remove redundant mandatory expressions?
//
// If set to 1, the expression
//
// IF
// (
//   ( %n  )
//    %s TERM NEWLINE
// )
// CLASS mini_Unknown
//
// is replaced by
//
// IF
// (
//    %n
//    %s TERM NEWLINE
// )
// CLASS mini_Unknown
//
// (the mandatory expression is no longer needed)
v2sspec.extendedexpression.removeredundant=1


// colors for colored popup menu in v2s area
// 1 = inner
// 10 = outer
v2sspec.extendedexpression.color.1=blue
v2sspec.extendedexpression.color.2=magenta
v2sspec.extendedexpression.color.3=cyan
v2sspec.extendedexpression.color.4=green
v2sspec.extendedexpression.color.5=yellow
v2sspec.extendedexpression.color.6=pink
v2sspec.extendedexpression.color.7=red
v2sspec.extendedexpression.color.8=orange
```

```
v2sspec.extendedexpression.color.9=darkGray
v2sspec.extendedexpression.color.10=gray
```

Example B-1. An example personal properties file