VISRAL L.P.

Python Solutions and Windows Publishing Studio

Visral® 3.0 User Manual

Volume 2

18 April 5, 2016

www.visral.com

P.O. Box 646

Rockwall, Texas 75087

Preliminary

Document Revisions

Date	Version Number	Document Changes
05/02/2014	0.1	Initial Draft
03/05/2015	0.2	Beta release
04/18/2016	0.30	Includes multiport diagram information 3.0.2.7

Notes:

1. This guide may describe some features that have not been enabled.
2. The representative images may in some cases appear slightly different from those of application itself.
3. References to right click or left click means respectively, pushing right mouse button or pushing left mouse button.

Contents

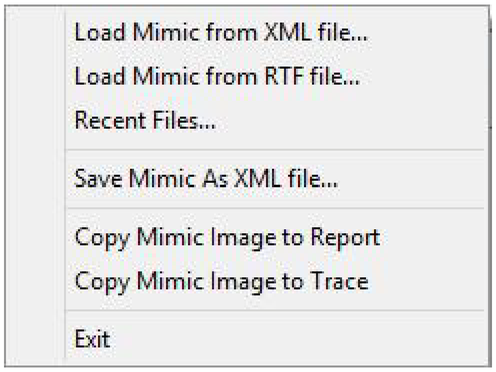
Utilities and Tools

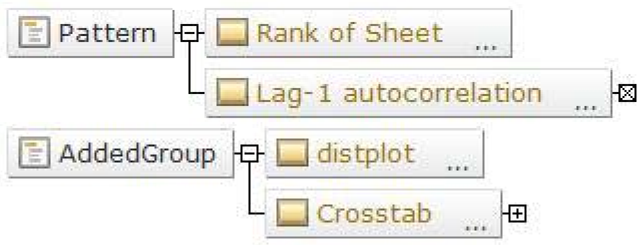
Covers:

- ✓ Mimicker
- ✓ Dependency Trees
- ✓ Maps
- ✓ Passage Encryption

Mimicker

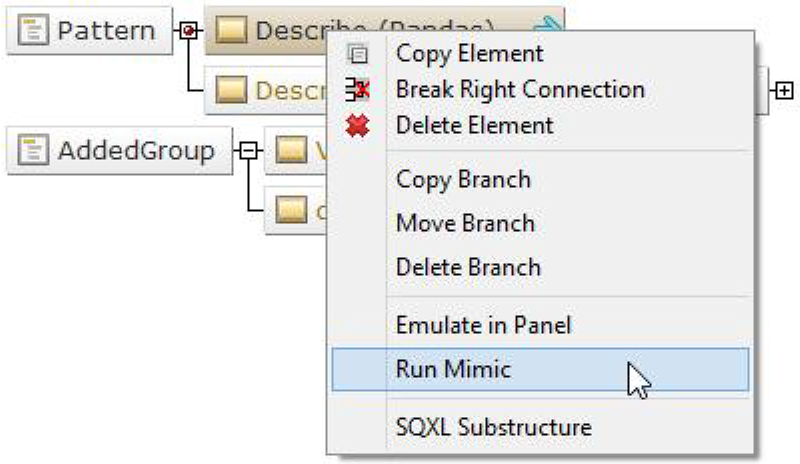
The mimicker, which operates concurrently tracking just user's computational instructions, allows rerunning previous activity. Its contents, may be saved to and reloaded from XML files, as well as extracted from Reports (RTF files) that have embedded Panels and Code.


Load Mimic from XML file...
Load Mimic from RTF file...
Recent Files...
Save Mimic As XML file...
Copy Mimic Image to Report
Copy Mimic Image to Trace
Exit



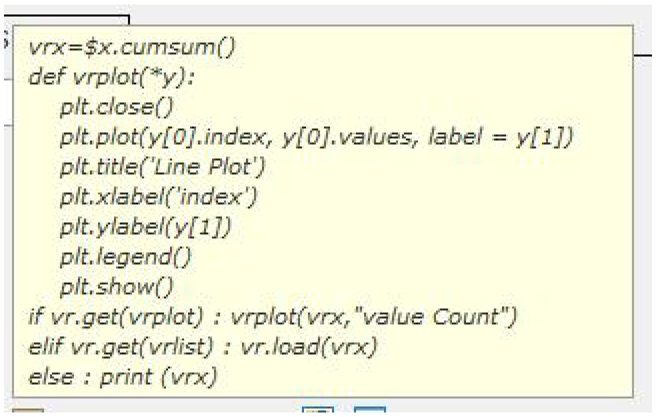
The clapper icon in the Morphing menu enables and disables the tracking. This icon is visible when either the Panel or Mimic has focus.

The Mimic can be started from any Operator element by right clicking on the element and the left clicking on the Run Mimic entry.



Element	Level	Description
Procedure	1	This element can be added to establish a separate listing of executed Operators.

Elements list in the Mimic can be edited before they are executed.



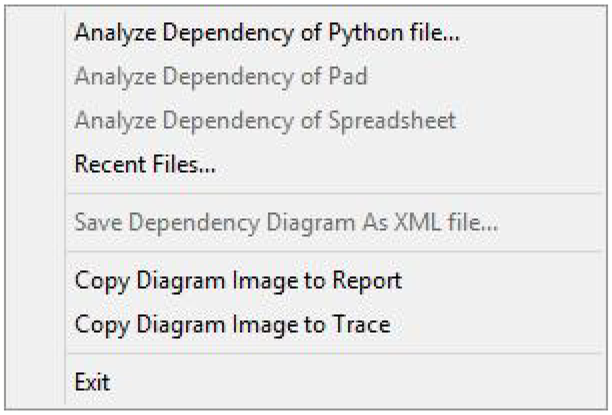
```
vrx=$x.cumsum()
def vrplot(*y):
    plt.close()
    plt.plot(y[0].index, y[0].values, label = y[1])
    plt.title('Line Plot')
    plt.xlabel('index')
    plt.ylabel(y[1])
    plt.legend()
    plt.show()
if vr.get(vrplot) : vrplot(vrx,"value Count")
elif vr.get(vrlist) : vr.load(vrx)
else : print (vrx)
```

Dependency Trees

The computational muscle of Python melded with spreadsheets means a single cell worth of code can populate an entire worksheet. The downside of this power is the ease with which circular references can unintentionally arise. To alleviate this potential problem Visral can analyzes all the functional and parameter relationships and create dependency tree graphics to simplify and correct the condition.

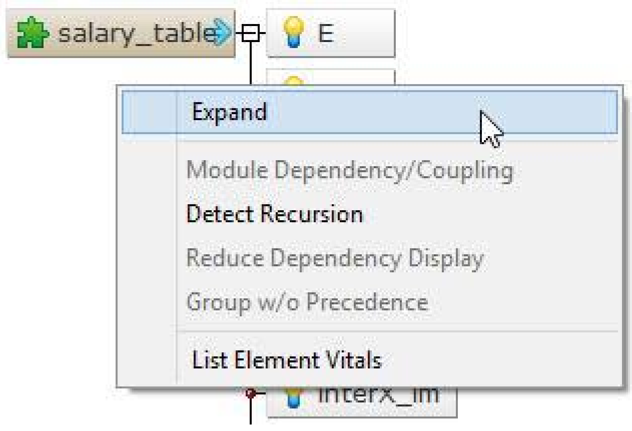
Element	**Description**
CxProc	This is the source element of a dependency. All the elements to the right represent other modules, routines, or functions that use it.
ProStmt	This is an example of a module, routine, or function that is used by the source element. All the elements to the right represent other modules, routines, or functions that use this one. The ones to the left are the ones that use it.
(null)	This is represents a module, routine, or function that could not be located. Therefore there will not be any to the right of it.

Initially only the first level of dependency is shown. A right click on the source module element (rule head) will display a menu with an entry Module Dependency/Coupling. Selecting this entry will cause all of the remaining levels of coupling to be filled out.


Analyze Dependency of Python file...
Analyze Dependency of Pad
Analyze Dependency of Spreadsheet
Recent Files...
Save Dependency Diagram As XML file...
Copy Diagram Image to Report
Copy Diagram Image to Trace
Exit

Visral's Generate Dependency Diagram command analyzes all the functional and parameter relationships and then creates a dependency tree diagram to highlight potential cyclic issues. (This process is used by the spreadsheet calculate routine to determine order of execution and report detected conflicts if any.)

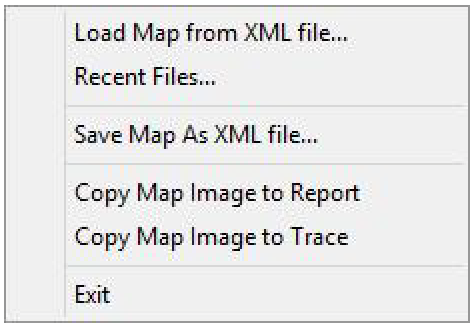
Initially only the first level of dependency is shown. A right click on the source module element (rule head) will display a menu with an entry Expand. Selecting this entry will cause the next levels of coupling to be filled out.

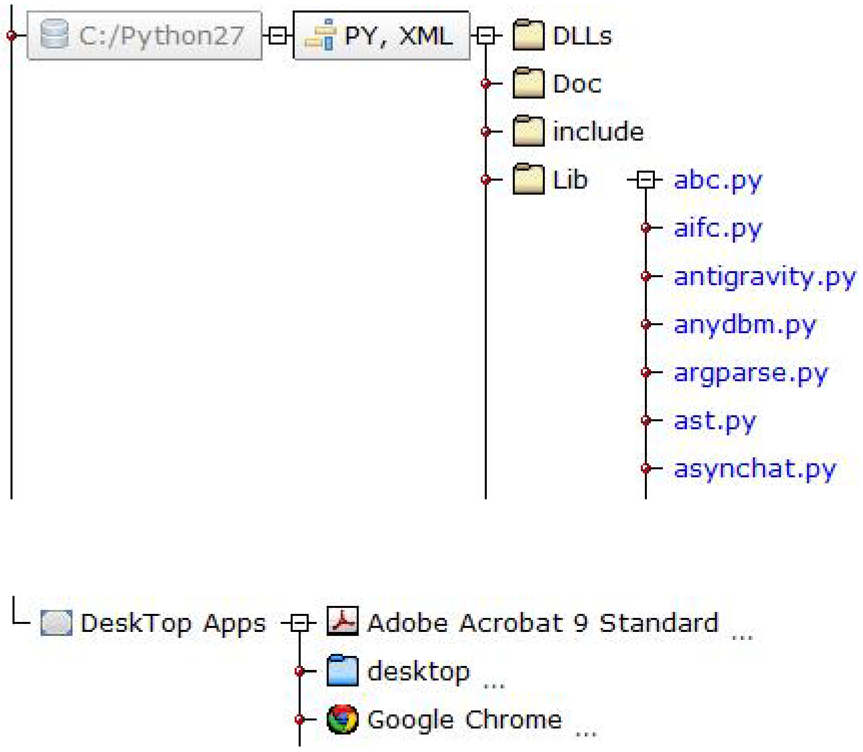


Maps

- ✓ Allows combining files from multiple local and remote directories into virtual directories.
- ✓ Provides specifying means for data retrieval, caching, and mapping from big memory sources (Big Data).
- ✓ Provides specifying means for remote concurrent processing services.
- ✓ Offers filtered hierarchical view of folders and files.

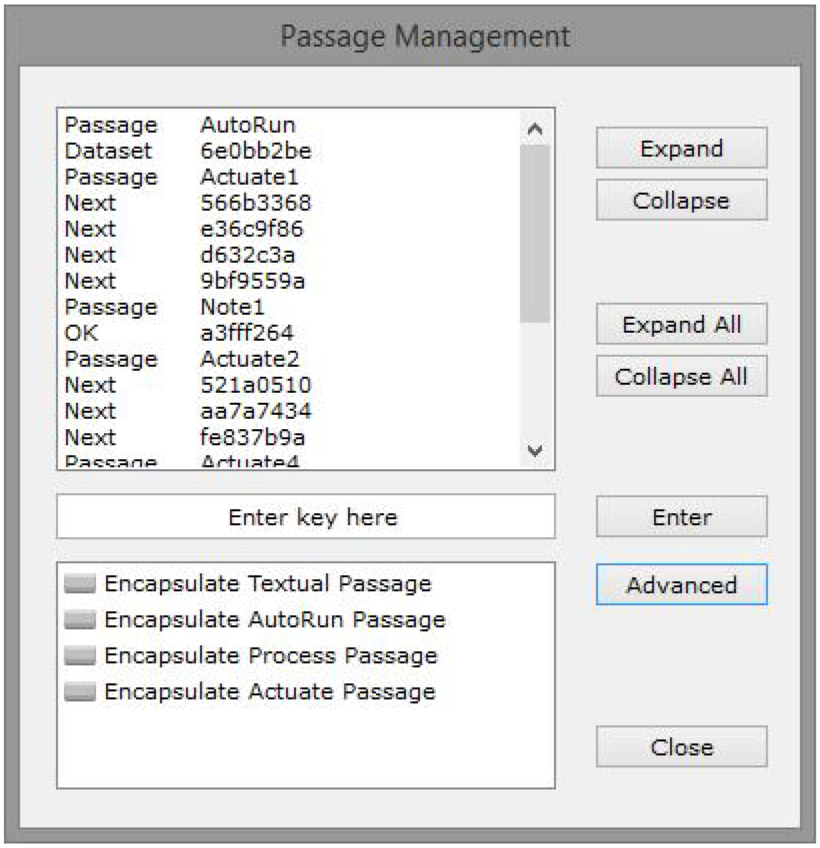
Element	Level	Description
Directory	1	
Extensions	2	



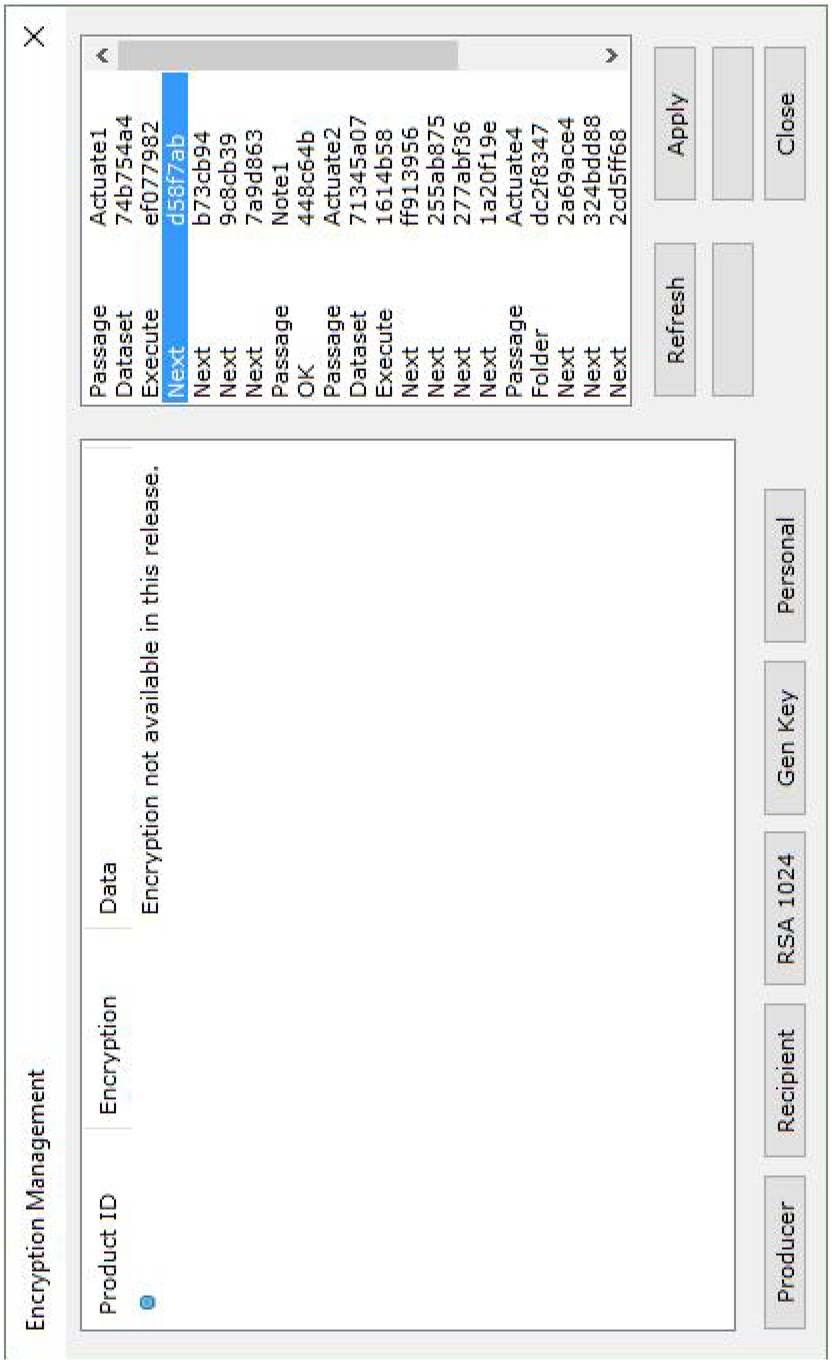


RSA and Symmetric Encryption for Document Passages

Both private-public and symmetrical encryption facilities are available to secure passages and code components. The Producer refer to is the one who creates encrypt content, Recipient to those receiving that encrypted content, and Personal refers to encrypted content received from others.


Passage Management
Expand
Collapse
Expand All
Collapse All
Enter key here
Enter
Advanced
Encapsulate Textual Passage
Encapsulate AutoRun Passage
Encapsulate Process Passage
Encapsulate Actuate Passage
Close

Passage	AutoRun
Dataset	6e0bb2be
Passage	Actuate1
Next	566b3368
Next	e36c9f86
Next	d632c3a
Next	9bf9559a
Passage	Note1
OK	a3fff264
Passage	Actuate2
Next	521a0510
Next	aa7a7434
Next	fe837b9a
Passage	Actuate4



Product ID	Encryption	Data
●		Encryption not available in this release.

Passage	Actuate1
Dataset	74b754a4
Execute	ef077982
Next	d58f7ab
Next	b73cb94
Next	9c8cb39
Next	7a9d863
Passage	Note1
OK	448c64b
Passage	Actuate2
Dataset	71345a07
Execute	1614b58
Next	ff913956
Next	255ab875
Next	277abf36
Next	1a20f19e
Passage	Actuate4
Folder	dc2f8347
Next	2a69ace4
Next	324bdd88
Next	2cd5ff68

Visral Diagrams

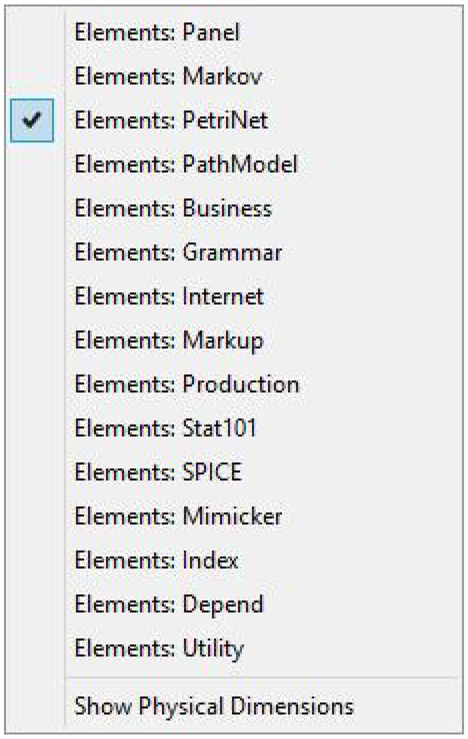
Covers diagrams:

- ✓

Selecting Elements

The content of each diagram is assessed when loaded, setting its type and selecting the appropriate element list and available operations. (The selection is determined by the first rule head encountered.)

Right clicking on the background will bring up the element selection menu. This allows different element groups and their associated operations to be applied to any diagram.



Petri Net Diagrams

Element	Description
PetriNet	
FIFO	
LIFO	
Sort	
Index	
Random	
Transition	
TestSet	
Token	
Delay	
Alarm	
Time	
Date	
Cloud	
Network	
Auto	
Spawn	

Markov Chain Diagrams

Element	Description
Markov	
State	
TestSet	

Path Model Diagrams

Element	Description
Model	
X Exogenic	
Y Endogenic	
Z Residual	

Business Model Diagrams

Element	Description
Rule	
SubRule	
Compare	
Request	
Response	
Timeout	
Delay	
Time	
Date	
Message	
[ ] Matrix	
Encrypt	
Filter	
Print	
Cloud	
Network	
Auto	
Spawn	

Production Model Diagrams

Element	Description
Get Rules	
Condition	
Compare	
Yes	
No	
Trigger	
Cloud	
Network	
Auto	
Spawn	

Circuit Analysis

Element	Description
Resistance	
Inductance	
Capacitance	
Impedance	
Admittance	
H-Parameter	
T-Parameter	
Conductance	
S-Parameter	
Source	

Stat101

Element	Description
Samples	
r.v. Pipe	
Mean	
STD	
Variance	
Median	
Factorial	
LChart	
XYChart	
PieChart	
AreaChart	
BarChart	
BoxChart	
ScatChart	
HistoChart	
HiLoChart	

Syntax/Grammar Parsing

Element	Description
Rule	This element is the non-terminal name that identifies the rule. No two rules may have the same. Must be left most elements.
NonTerm NonTerm	NonTerm is a reference to a non-terminal rule. If the reference is to an existing rule, either user created or built-in, the displayed icon will be , otherwise it will be indicating a container. If the rule is later added, the icon of all referencing non-terminals will then change to .
Terminal	Terminal; the value within means nothing but itself. Used for compare testing such as parsing signatures. This element must contain an argument. Atomic elements (terminal) can accept printable ASCII characters or hexadecimal values, permitting both printable characters and binary values. In atomic elements, the “space” is indicated by a centrally located small blue dot. When creating EBNF production rules, number values are displayed without quotes. The same is true in the other direction; number values outside of quotes are displayed with the H superscript.
RE [A-Za-z0-9_]	This acceptor recognizes ASCII alphanumeric characters plus underscore. It will accept a single character (byte).
RE [A-Za-z0-9]	This acceptor recognizes ASCII alphanumeric characters. It will accept a single character (byte).
RE [A-Za-z]	This acceptor recognizes ASCII letters, both lower and upper case. It will accept a single character (byte).
RE [a-z]	This acceptor recognizes and accepts lower case ASCII letters. It will accept a single character (byte).
RE [A-Z]	This acceptor recognizes and accepts upper case ASCII letters. It will accept a single character (byte).
RE [0-9]	This acceptor recognizes the ASCII digits 0 through 9. It will accept a single character (byte).
RE [0-9a-fA-F]	This acceptor recognizes the ASCII digits 0 through 9 and letters A through F (upper and lower case). It will accept a single character (byte).
RE .	This acceptor recognizes and accepts any ASCII character. Careful because any means any including ‘0’ (zero), the normal string terminator.

Beyond

ScanBytes	If this element is places in a column of optional terminal values their icons will change to a magnifying glass and they will scan the input signature for a match. If the ScanBytes word is blank the scan will be to the end of the signature; if it is a number the scan will be for that number of bytes.
SkipBytes	Skip the specified number of signature bytes.
Compile	
Code	
Test Rule	Must be left most elements.
MSET	
Set Register	
Get Register	
Save for CPU	
Job Ticket	
Auto	
Spawn	
Note	This can be used anywhere to keep notes. It has no effect on the behavior of a rule's execution but is merely to provide information to the viewer of the rule. The element is passive, having no effect on function except as a pass-thru.

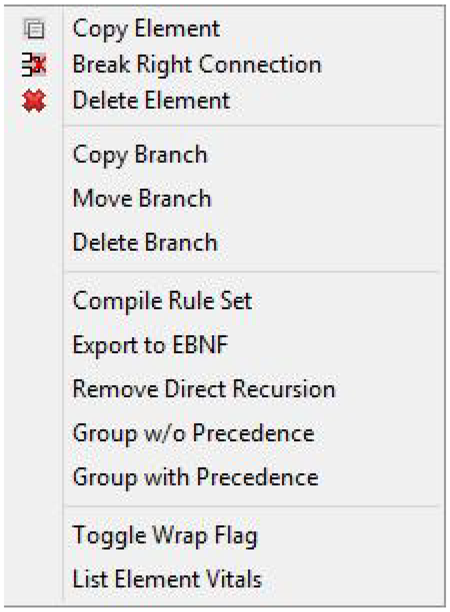
Other Syntax/Grammar Operators

Built-in	**Acts Like**	**Form / Use**
_ws	Rule	If there is a whitespace rule defined with this name then whitespaces will be not be removed from the input signature. Otherwise they will be removed before sentence is parsed. The pre-parser whitespace is defined as (0x20 | 0x0D | 0x09), but may be alter in the source code. (Does not require a rule body to cause the action.)
.letter	Atom	This acceptor recognizes ASCII letters, both lower and upper case. It will accept a single character (byte).
.locase	Atom	This acceptor recognizes and accepts lower case ASCII letters. It will accept a single character (byte).
.upcase	Atom	This acceptor recognizes and accepts upper case ASCII letters. It will accept a single character (byte).
.chr imm	Atom	Must be followed by a number (imm). It will test a single byte represented by that number against the current byte of the signature. If the match is successful the byte of the signature will be consumed. The number can be integer or hexadecimal (0x…).
.arc imm	Atom	Must be followed by a number (imm). It will test a single byte represented by that number against the current byte of the signature. Unlike ".chr", this function does not consume anything from the signature. The number can be integer or hexadecimal (0x…).
.digit	Atom	This acceptor recognizes the ASCII digits 0 through 9. It will accept a single character (byte).
.hex	Atom	This acceptor recognizes the ASCII digits 0 through 9 and letters A through F (upper and lower case). It will accept a single character (byte).
.any [imm]	Atom Rule	This acceptor recognizes and accepts any ASCII character. Careful because any means any including '0' (zero), the normal string terminator. If there is an immediate value (imm) it will then act as ".sup" and suspend the Parser/State Machine for external processing and then continue where it left off. The value can be either decimal or hexadecimal (0x…).

.act imm	Rule	This name in a reference element is indented to cause the activation of an external function. Must be followed by a number to indicate which external function is being called. The exception is “.act 0” which will terminate the parsing process and is useful when the input signature might be longer than defined by the grammar but is still okay. Place it at the end of the starting rule. This rule consumes nothing and is used to implement state, flow and Petri functions; as well as the requirements of a strict grammar. Causes a return to the same state it left from, which means it will re-exit if conditions at the state have not changed.
.sup imm	Rule	Similar to .act except does not return to the same state and is indented to suspend the Parser/State Machine for external processing and then continue where it left off. ".sup 0" behaves the same as ".act 0".

Rule Operations

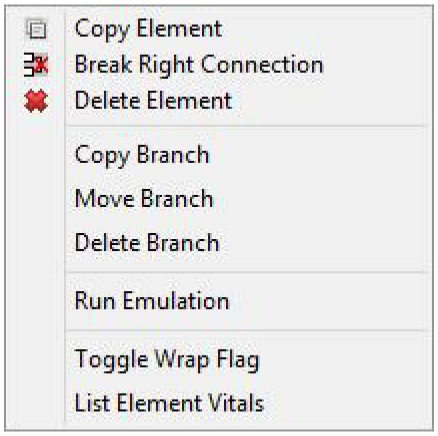
Right click when the cursor is on rule head will produce a menu with the following operations. Notice that all the elements of the selected rule are lit indicating that the operation could affect any or all of them.



Operation	Description
Compile Rule Set	Build a parser image for the rule set whose root is this rule. Algorithms compute the states required to implement the grammar, which are then added to the diagram. A $ will appear on elements that have the possibility of generating an output term in intermediate postfix results. If the element has a name, that be used as a default value. Otherwise, a unique one should be entered into the formula if it is to be recognized by the parser.
Export to EBNF	This operation will convert all the selected rule to EBNF production rules and display it in the Trace window.
Remove Direct Recursion	This operation removes left recursive terms that are found in the selected rules. After this operation, running a reduce operation may provide a cleaner looking presentation. (RecursiveRules.xml has a collection of recursive rules for practice)
Group w/o Precedence	This operation combines terms where possible and reformats the rule to have a left to right, top down form.
Group with Precedence	Same as above but does not make any changes that would disrupt the process order of terms.

Test Rule Operations

Right click when the cursor is on rule head will produce a menu with the following operations. Notice that all the elements of the selected rule are lit indicating that the operation could affect any or all of them.



Operation	Description
Run Emulation	Once a parser rule set has been built, test rules can be run and traced to determine if the grammar is performing as expected. Selecting the entry will cause that to happen.

Compiling Grammar Automaton

A grammar could be created from scratch but to speed the understanding of the whole process it is best to start with a known quantity. Use the button to locate and load the Excel.xml grammar file.

Right click when the cursor is on rule head of the root rule of the grammar. Moving the cursor down over the Rule Operations bar will cause the drop-down to appear.

Continuing down until over the Compile Rule Set and left clicking will cause the parser image to be constructed. The diagram will also change to show all of the assigned states and the scroll window will list the results of the build.

Some elements will show a $ (dollar sign), which indicates that this element has the possibility of generating an output term in postfix results. If the element has a name, that be used as a default value. Otherwise, a unique one should be entered into the formula if it is to be recognized by the parser. This can be done by clicking on the Formula entry on the context menu when in edit mode.

The printout in the scroll window resulting from the build lists some of the various checks the compiler performs.

The report shows testing for direct and indirect recursion indicates none were found. (This check looks throughout the entire diagram not just the rules that are part of the grammar.)

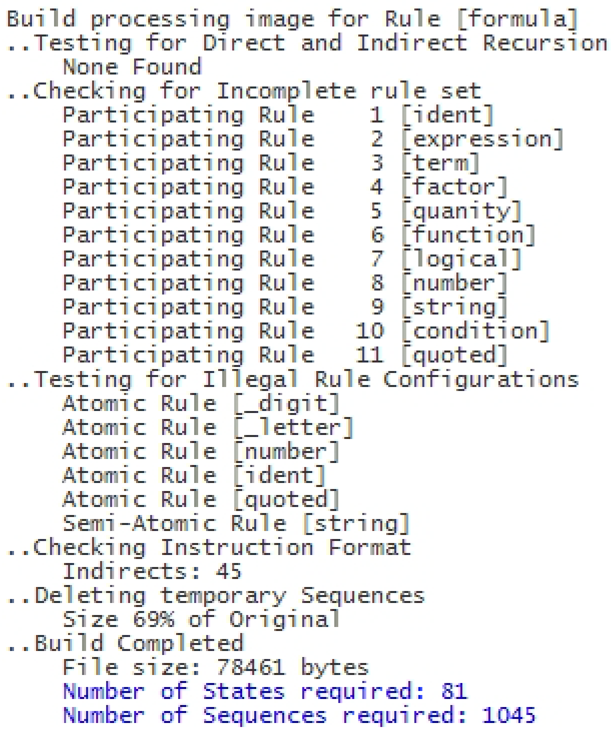
It is determined if all the required rules are present and prints a list; then checks them for illegal configurations.

Indicated next are those rules that are equivalent to terminal values.

The build takes place here followed by checking the resulting instruction format and a count of the number indirects required. (pointers to fixed values)

Then redundant sequences are removed showing the effective size reduction.

And finally the results, which show 81 states and 1045 sequences, were needed to construct a parser that matched the grammar.



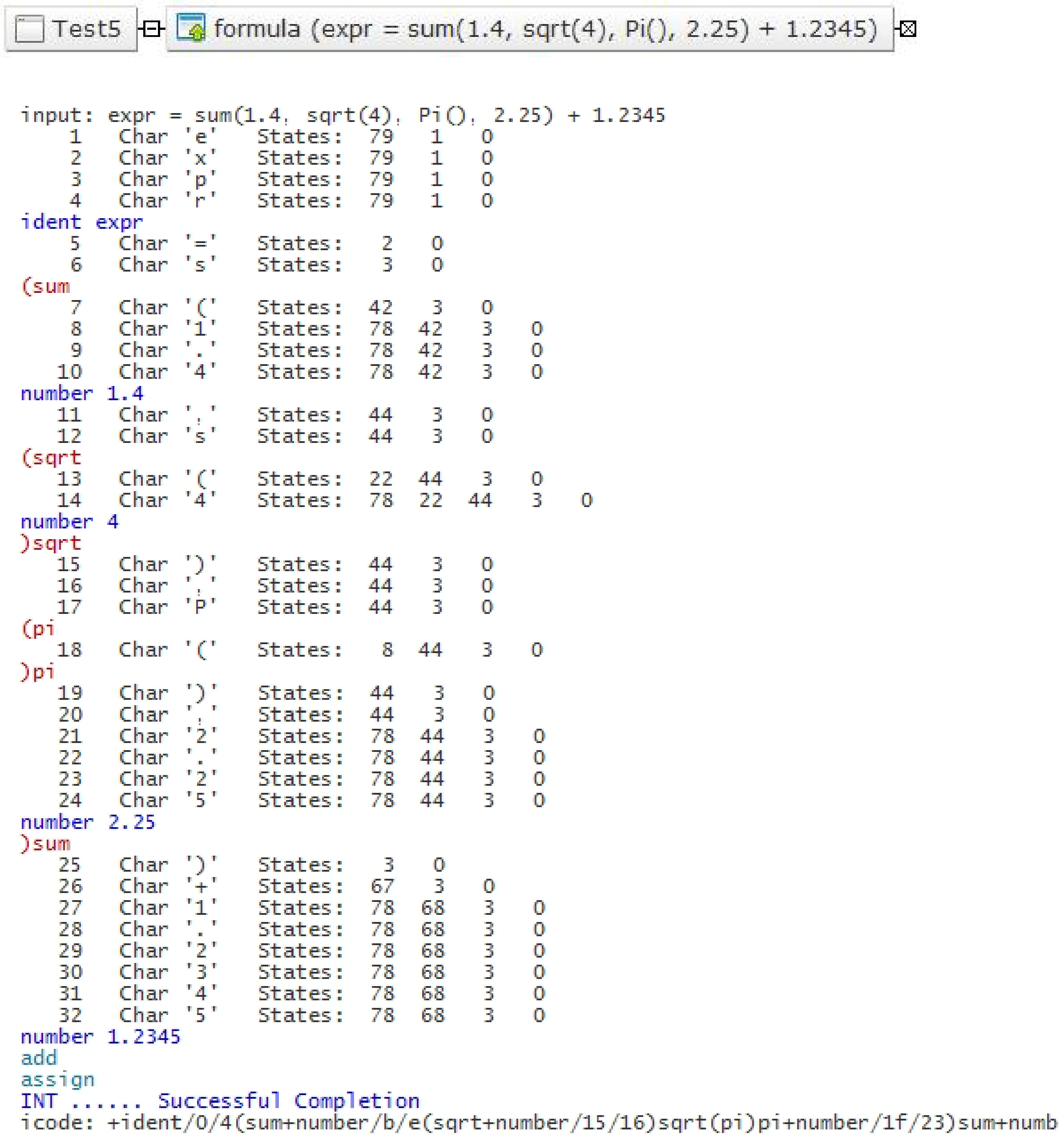
```
Build processing image for Rule [formula]
..Testing for Direct and Indirect Recursion
    None Found
..Checking for Incomplete rule set
    Participating Rule    1 [ident]
    Participating Rule    2 [expression]
    Participating Rule    3 [term]
    Participating Rule    4 [factor]
    Participating Rule    5 [quanity]
    Participating Rule    6 [function]
    Participating Rule    7 [logical]
    Participating Rule    8 [number]
    Participating Rule    9 [string]
    Participating Rule   10 [condition]
    Participating Rule   11 [quoted]
..Testing for Illegal Rule Configurations
    Atomic Rule [_digit]
    Atomic Rule [_letter]
    Atomic Rule [number]
    Atomic Rule [ident]
    Atomic Rule [quoted]
    Semi-Atomic Rule [string]
..Checking Instruction Format
    Indirects: 45
..Deleting temporary Sequences
    Size 69% of Original
..Build Completed
    File size: 78461 bytes
    Number of States required: 81
    Number of Sequences required: 1045
```

There are 8 test rules provided with excel.xml. <u>Some may look a little strange and will fail but were designed to stress certain functions of the parser compiler itself and not the actual grammar.</u>

Note: The test rule is composed of one non-terminal with the name of the root rule of the grammar to be tested and what is termed a payload. The content inside the payload's outer most parentheses is fed to the parser when emulation is run.

Understanding the Emulation Trace

Move the cursor up until it is on rule Test5 and right click. Again slide the cursor down but this time stop at RUN Emulation and left click. The scroll window will have the resulting trace as shown below.



```
input: expr = sum(1.4, sqrt(4), Pi(), 2.25) + 1.2345
     1   Char 'e'   States:  79   1   0
     2   Char 'x'   States:  79   1   0
     3   Char 'p'   States:  79   1   0
     4   Char 'r'   States:  79   1   0
ident expr
     5   Char '='   States:   2   0
     6   Char 's'   States:   3   0
(sum
     7   Char '('   States:  42   3   0
     8   Char '1'   States:  78  42   3   0
     9   Char '.'   States:  78  42   3   0
    10   Char '4'   States:  78  42   3   0
number 1.4
    11   Char ','   States:  44   3   0
    12   Char 's'   States:  44   3   0
(sqrt
    13   Char '('   States:  22  44   3   0
    14   Char '4'   States:  78  22  44   3   0
number 4
)sqrt
    15   Char ')'   States:  44   3   0
    16   Char ','   States:  44   3   0
    17   Char 'P'   States:  44   3   0
(pi
    18   Char '('   States:   8  44   3   0
)pi
    19   Char ')'   States:  44   3   0
    20   Char ','   States:  44   3   0
    21   Char '2'   States:  78  44   3   0
    22   Char '.'   States:  78  44   3   0
    23   Char '2'   States:  78  44   3   0
    24   Char '5'   States:  78  44   3   0
number 2.25
)sum
    25   Char ')'   States:   3   0
    26   Char '+'   States:  67   3   0
    27   Char '1'   States:  78  68   3   0
    28   Char '.'   States:  78  68   3   0
    29   Char '2'   States:  78  68   3   0
    30   Char '3'   States:  78  68   3   0
    31   Char '4'   States:  78  68   3   0
    32   Char '5'   States:  78  68   3   0
number 1.2345
add
assign
INT ...... Successful Completion
icode: +ident/0/4(sum+number/b/e(sqrt+number/15/16)sqrt(pi)pi+number/1f/23)sum+numb
```

The columns of the listing to the right of the character being processed represent the state stack.

The blue highlighted lines have two components and represent terminal values. The first part is the name of the rule representing the type and the second part is the actual value. When processing the results from the parser these values are push onto the stack.

The green lines represent what was originally infix operators (binary) now postfix. That means the last two items on the stack should be popped of, processed with this operator and the result pushed back on.

The red lines with a closing-bracket)function)sum represent function operators which can have any number of arguments. These are also converted to postfix meaning the operator pulls the required number of values from the stack, processes them with the result being push back on.

The other red lines with opening-bracket(function (sum in most cases can be ignored but is there for when dealing with an operator that has a variable number of arguments. This is the case when parsing LISP. It permits popping values of the stack until the matching lead operator is encountered.

The line that is headed with icode: represents the instruction stream that is provided to the hosting system for processing the results of parsing.

```
icode: +ident/0/4(sum+number/b/e(sqrt+number/15/16)sqrt(pi)pi+number/1f/23)sum
+number/27/2d*add*assign
```

The rules for interpreting this string are as follows:

1. A term preceded by a plus (+) indicates a terminal value and should be pushed on the stack or the results of it being processed pushed on the stack. It is always composed of three parts separated by forward slashes. The first part is type, the second is a hex value offset into the input signature where the raw value is located, and the third is the ending offset.
2. Terms preceded by an asterisk (*) indicates a binary operator requiring two values be popped off the stack and one returned.
3. As described above the closing-bracket)function tells the operator to pull the required number of values from the stack or until the matching opening-bracket(function term is encountered and process them with the result being push back on.

Prefix and infix syntax structures are converted to postfix during the parser build. In processing the postfix results (operator follows the arguments) the expression is traverses from left to right using the following procedure:

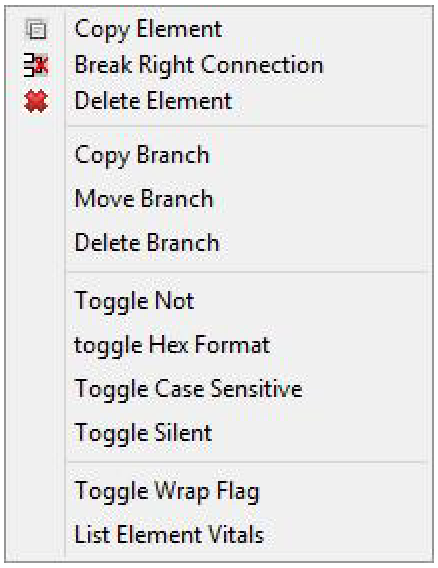
If value encountered, push it onto the stack

If n-ary operator, pop the right number of arguments, apply the operator, and push the result.

At the end of the expression, the result is the only value left on the stack

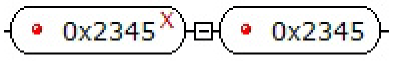
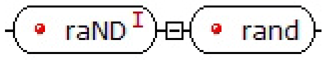
Atomic/Terminal Operations

A right click when the cursor is on terminal element will produce a menu with the following operations. Atom is a terminal symbol; the value within means nothing but itself. Used for compare testing such as parsing signatures. This element must contain an argument.



Atomic elements (terminal values) can accept printable ASCII characters or hexadecimal values, permitting both printable and non-printable (control) characters. In atomic elements, the "space" is indicated by a centrally located small blue dot.

An atomic ASCII element can be made Case Insensitive by placing the cursor on it, pressing the right button of the mouse, and then selecting the Toggle Case Sensitive entry. Repeating this action will toggle it back to case sensitive. When the atom is insensitive to case, a capital "I" will be displayed in the lower right corner of the element.

Operation	Description
Toggle Not	Causes a bar to be placed over the element contents. This condition will allow passing if the signature does not match the element contents. However, if the signature matches it will be consumed; if not, nothing is consumed. Note: Currently this is only functional for a single character identity.
Toggle Hex Format	Atomic elements can accept a values represented by a hexadecimal number. Pressing the right button of the mouse, and then selecting the Toggle Hex Format entry will cause the entered hex value to represent the terminal value and display a capital X or H superscript in the upper right corner.  These two atoms do not represent the same value. The second of the two is looking for a literal match with all six characters, "0x2345". The first is looking for a match to the value composed of two 8-bit characters; the first with the hex value "0x45" and the second with the hex value "023". (The "0x" is optional.)
Toggle Case Sensitive	An atomic ASCII element can be made Case Insensitive by placing the cursor on it, pressing the right button of the mouse, and then selecting the Toggle Case Sensitive entry. Repeating this action will toggle it back to case sensitive. When the atom is insensitive to case, a capital "I" will be displayed in the upper right corner of the element.  The atom with the I superscript will consume (match) any combination of upper and lower case character of the word "rand" for example "rANd", whereas, the other only matches a precise copy.

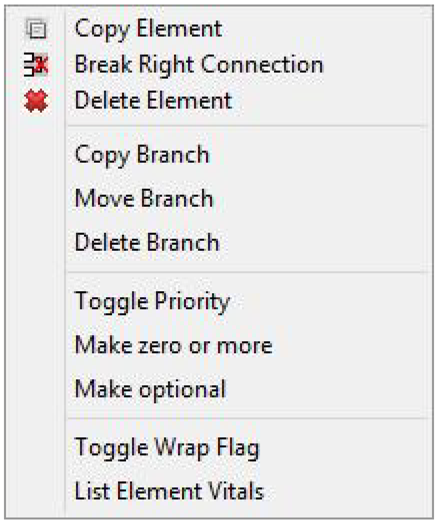
When generating EBNF production rules, number values are displayed without quotes. The same is true in the other direction; number values outside of quotes are displayed with the H subscript.

Element Superscripts & Subscripts

Superscripts	**Element Flag Description**
$	Only present following a parser build (**Compile Rule**). Indicates that this element has the possibility of generating an output term in the postfix results. If the element has a name, it is used as a default value. Otherwise, a unique one should be entered into the formula if it is to be recognized by the parser. (If two such elements have the same name, one should be changed.)
H	In the case of an Atom/Terminal element indicates the hexadecimal value should represent the contents and not the lateral.
I	In the case of an Atom/Terminal element the contents should be interpreted as case insensitive. Refer to **Visral Diagram Elements** for details.
R	Indicates the term/element is part of a direct or indirect recursion.
M	Indicates more than one rule has the same name
Subscripts	**Description**
Value	Function name or Formula description.

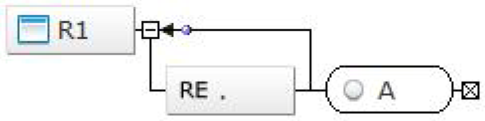
Repeat and Else (aka: Loop and Optional)

Else: (or Optional) The arrow pointing right defines the path to take if all else fails. It makes other elements in parallel with it, optional (in Visral diagram sense). It does not consume anything; it merely expresses a legal direction to view other possibilities. Unlike an atom such as **.letter**, it does not consume (accept) a character/byte.

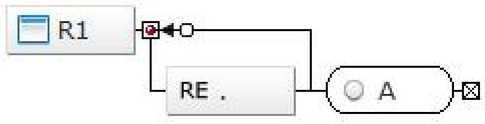


Repeat: The arrow pointing to the left is the means by which recursion is implemented. (The only element that enters from the right and exits to the left.) Like the **.else**, it does not consume anything; it merely expresses a legal direction to view other possibilities. Used for defining loopback, the **.repeat**'s priority can be set with the "**Toggle Priority**" entry in its options menu.

It will toggle back and forth between high and low. Priority is generally not an issue, but as in the example below, the “A” path would never be taken.



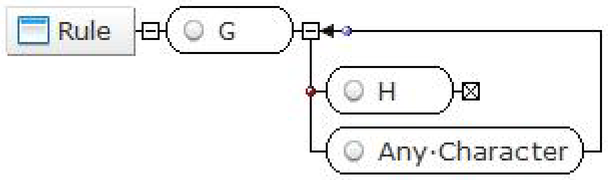
Unless the priority of **.repeat** was changes to low as indicated by the “L” subscript instead of the “H”.



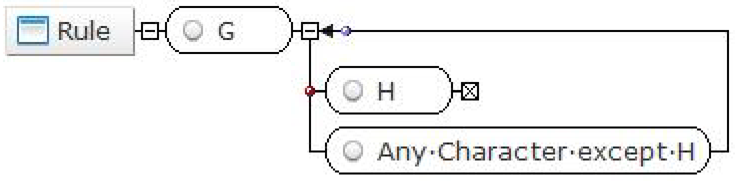
Neither “H” nor “L” is displayed for loops that represent “zero or more” repetitions.

Precedence

Unlike Visral diagrams, Visral elements have precedence in the case of optional paths, with the top-most having the highest priority. This makes it very convenient for creating rules by using exceptions as illustrated below.

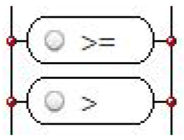


In EBNF this would be: **Rule ::= “G” {<all characters except H>}* “H”** shown below ; which can be difficult to implement.

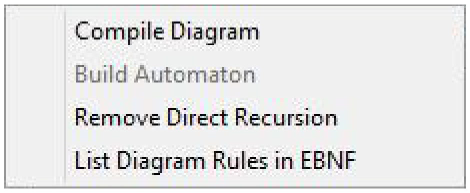


Precedence cannot be represented in EBNF and therefore should be saved in Visral XML format to avoid losing the descriptive information.

This also means that with options the semantic may need to be taken into account, such as those to the right where one is the complete leading subset of another, the longer should be above the shorter.

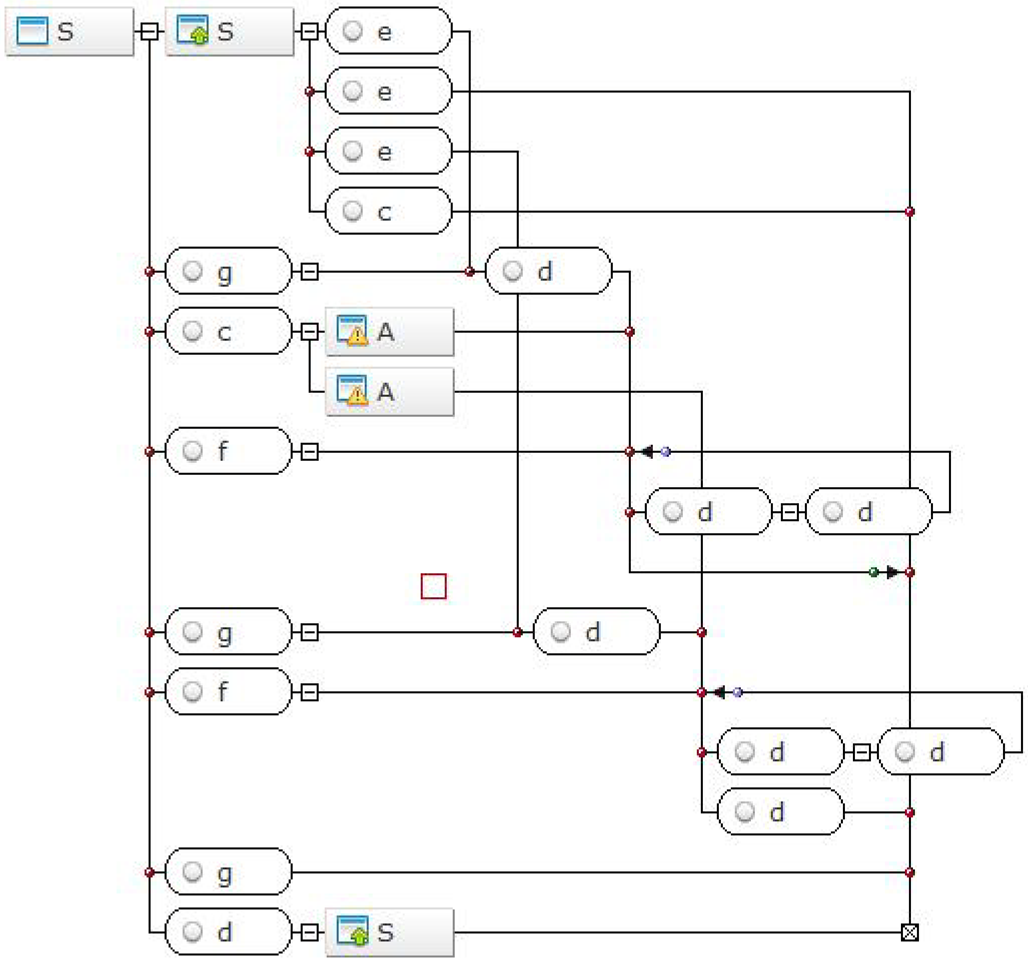


Generate EBNF Production Rules



This operation will generate an EBNF production rules from a diagram rule and display it in the Trace window. To save, the “Save as EBNF” button on the “File” sub-menu will permit that to be accomplished.

Selecting the “List as EBNF” button while the cursor is on the head of the following diagram rule:



Will show the following in the Trace window:

```
S ::= ( "c" A | ( S "e" | "g" ) "d" | "f" ) { "d" "d" }* | ( "c" A | ( S "e" | "g" ) "d"
    | "f" ) { "d" "d" }* "d" | S ( "e" | "c" ) | "g" | "d" S ;
```

Some changes can be made to the EBNF output form through Properties Page:

Function	Default	Notes
Assignment	::=	No more than 7 characters.
Concatenation	space	Not settable in current release.
Termination	;	No more than 7 characters.

EBNF Conversion Constraints

Else's (**.else**) should be in the lowest priority position of any list of optional paths.

Repeat's (**.repeat**) should be in the highest priority position (topmost) of any list of options. (This is not to be confused with the feedback priority of a **.repeat** which is toggled by the "Toggle Priority" entry of it options menu.

No dangling elements (rules that are to be converted to EBNF must have a single termination point)

If these conditions are not satisfied, the results may or may not be correct. If the conversion processor is able to detect a rule it can not convert, it will display in the scroll window "UNABLE TO CONVERT". A correction routine tries to place all **.else**'s in the lowest priority position, but if unable to do so, it may be the reason the conversion could not be performed. If that is the case, the repositioning of **.else**'s will need to be done by hand.

Detect Recursion

This operation will list those rules that are recursive, whether direct or indirect. This operation examines all rules of the current diagram.

Note**:** If a large number of indirect recursive rules are detected, it may be more efficient to modify the rule set to eliminate them than to use substitution and direct recursive removal. The number of elements could grow to an unmanageable size otherwise. As an example, the ATIS grammar rule set with over hundred rules and 16,000 elements produces eleven indirect left recursive rules. As can be seen below the dependences are large. Running the automatic Indirect Left Recursive Removal would result in excessively large rules.

The above listing below indicates substitutions that can be made to reduce dependencies.

```
..Testing for Direct and Indirect Recursion
    Rule with Recursion [NP_NN]
    Rule with Recursion [NP_NP]
    Rule with Recursion [AVP_QL]
    Rule with Recursion [AVP_RB]
    Rule with Recursion [NP_NNS]
    Substitution Sequence for Indirect
    Substituting [NP_NP] into [NP_NN] reduces dependencies from 77 to 76
Indirect: [NP_NP] contains [NP_NN]
    Substituting [AVP_QL] into [NP_NN] reduces dependencies from 77 to 76
Indirect: [AVP_QL] contains [NP_NN]
    Substituting [AVP_RB] into [NP_NN] reduces dependencies from 77 to 76
Indirect: [AVP_RB] contains [NP_NN]
    Substituting [NP_NNS] into [NP_NN] reduces dependencies from 77 to 76
Indirect: [NP_NNS] contains [NP_NN]
    Substituting [NP_NN] into [NP_NP] reduces dependencies from 15 to 14
Indirect: [NP_NN] contains [NP_NP]
    Substituting [AVP_RB] into [NP_NP] reduces dependencies from 15 to 14
Indirect: [AVP_RB] contains [NP_NP]
    Substituting [NP_NNS] into [NP_NP] reduces dependencies from 15 to 14
    Substituting [NP_NN] into [AVP_QL] reduces dependencies from 2 to 1
    Substituting [AVP_RB] into [AVP_QL] reduces dependencies from 2 to 1
    Substituting [NP_NNS] into [AVP_QL] reduces dependencies from 2 to 1
    Substituting [NP_NN] into [AVP_RB] reduces dependencies from 5 to 4
    Substituting [NP_NP] into [AVP_RB] reduces dependencies from 5 to 4
    Substituting [AVP_QL] into [AVP_RB] reduces dependencies from 5 to 4
Indirect: [AVP_QL] contains [AVP_RB]
    Substituting [NP_NNS] into [AVP_RB] reduces dependencies from 5 to 4
    Substituting [NP_NN] into [NP_NNS] reduces dependencies from 92 to 91
Indirect: [NP_NN] contains [NP_NNS]
    Substituting [NP_NP] into [NP_NNS] reduces dependencies from 92 to 91
Indirect: [NP_NP] contains [NP_NNS]
    Substituting [AVP_QL] into [NP_NNS] reduces dependencies from 92 to 91
Indirect: [AVP_QL] contains [NP_NNS]
    Substituting [AVP_RB] into [NP_NNS] reduces dependencies from 92 to 91
Indirect: [AVP_RB] contains [NP_NNS]
```

Remove Direct Recursion

This operation removes left recursive terms that are found in the selected rule. After this operation, running Combine Elements may provide a cleaner looking presentation. The following is an example of left recursion removal.

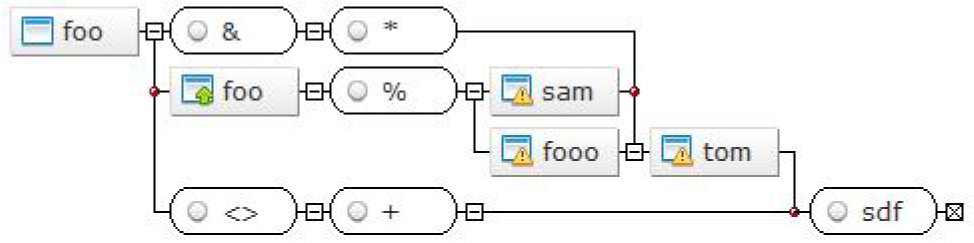


Figure 1

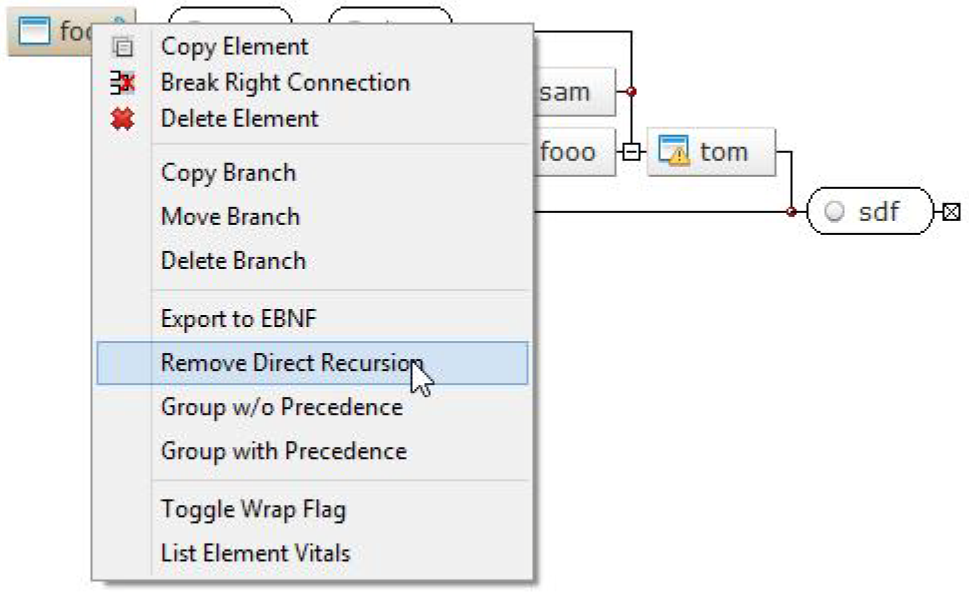


Figure 2

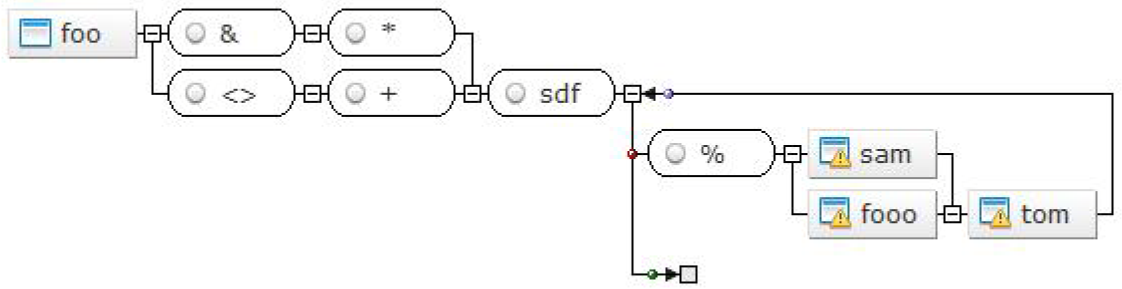


Figure 3

Right clicking on a rule heads will bring up the menu of possible operations. Left clicking **Remove Direct Recursion** will lead to the rewriting of rule with recursion removed.

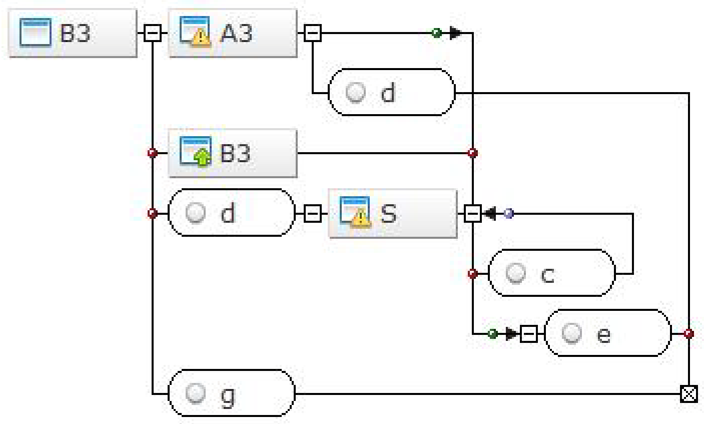


Figure 4

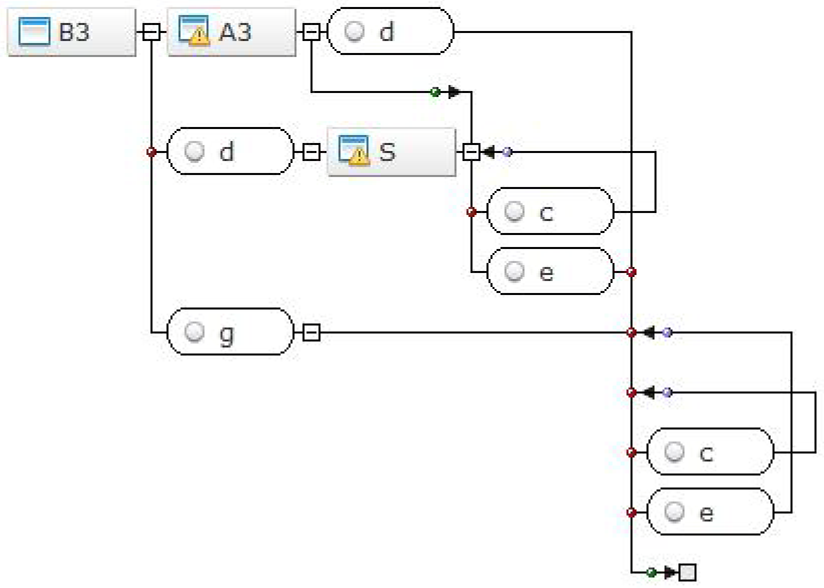


Figure 5

In the above example the B3 recursive term is first removed with **Remove Direct Recursion** and then terms are combined with **Group w/o Precedence** to clean up the results.

Combine Terms

The following illustrates some of the reduction and re-formatting performed by the **Group w/o Precedence** command.

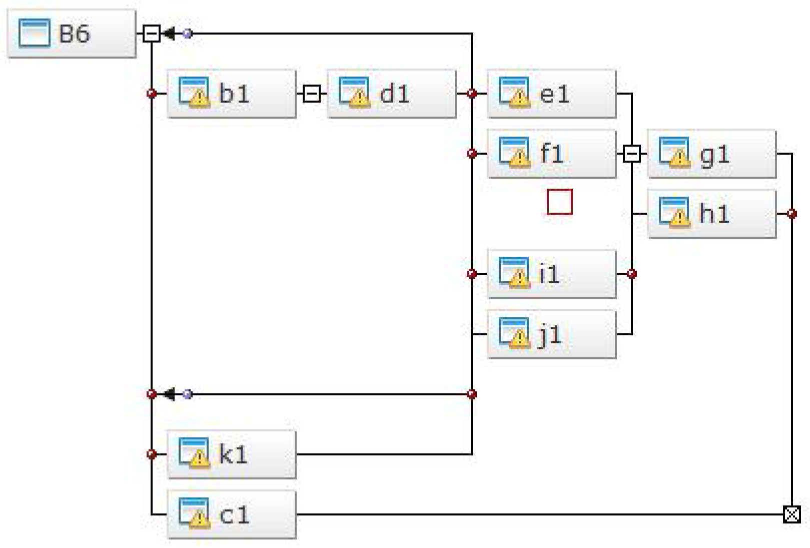


Figure 6 Before Group w/o Precedence

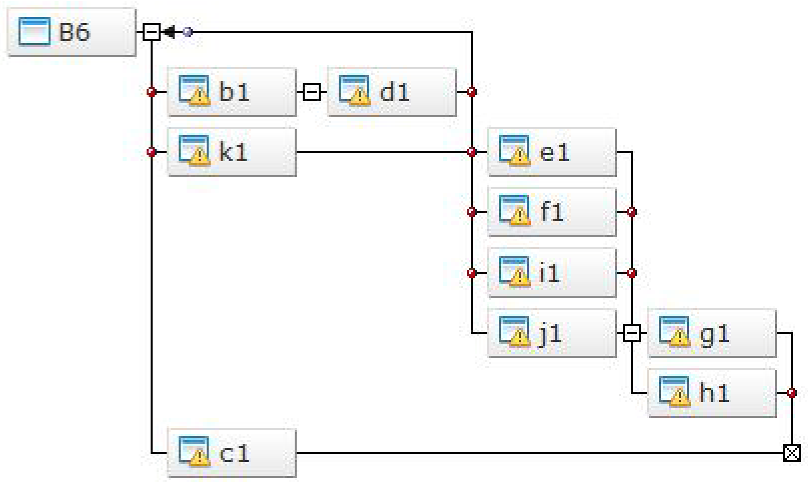
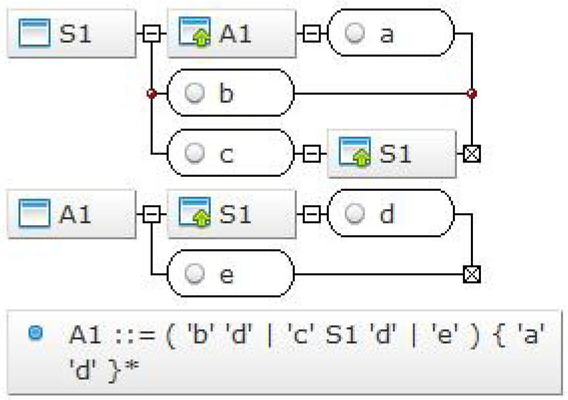


Figure 7 After Group w/o Precedence

Automated Indirect Left Recursive Removal

Visral uses the method of steepest gradient decent to implement an automated substitution and direct left recursive removal strategy.



A1 ::= ('b' 'd' | 'c' S1 'd' | 'e') { 'a' 'd' }*

Markup - XML/HTML/… Creation, Editing, and Inspection

Element	Description
Tag	Element stands for an XML element.
Data	Data holds text that goes/found between elements.
CDATA	Holds data/text that can contain XML markup characters. Only the sequence "]]>" (0x5d 0x5d 0x3e) needs to be avoided. <![CDATA[…CDATA…]]>
Instruction	Instruction holds text for an instruction element; i.e. <?Instruction>
Document	<!Documentation>
Comment	Comment holds text for a comment element; i.e. <!--Comment-->
Note	This can be used anywhere to keep notes. It has no effect on the behavior of a rule's execution but is merely to provide information to the viewer of the rule. The element is passive, having no effect on function except as a pass-thru.

HTML files are converted to XHTML when they are displayed.

XML special characters & < > " ' will show as themselves and can be used directly in Visral elements. They automatically get converted to and from &amp; &lt; &gt; &quot; &apos; when files are either inputted, outputted or appear in the raw window.

A right click on an element will bring up the options menu with two special entries, Generate XPATH Sequence and Generate XPATH Seq with id/class.

If an entry is selected the Clipboard will receive the XPATH description of where the element is located, and a copy will be printed in the Trace.

The editor tries to correct errors such as missing closing tags, out of order tags, missing quotes, and wrong case when converting HTML to XHTML. It also understands HTML specific characteristics, for example the fact that the ampersand (&) and angle brackets (<>) are treated differently depending on whether they are inside or outside of a <script> element.

Note: Visral is not an XHTML checker. Nor is it a pure XML editor because of its need to support graphs as well as trees, and its need to recognize a much larger class of element types than those of XML. However, it does understand such cases as the ampersand (&) and angle brackets (<>) are treated differently depending on whether they are inside or outside of a <script>.

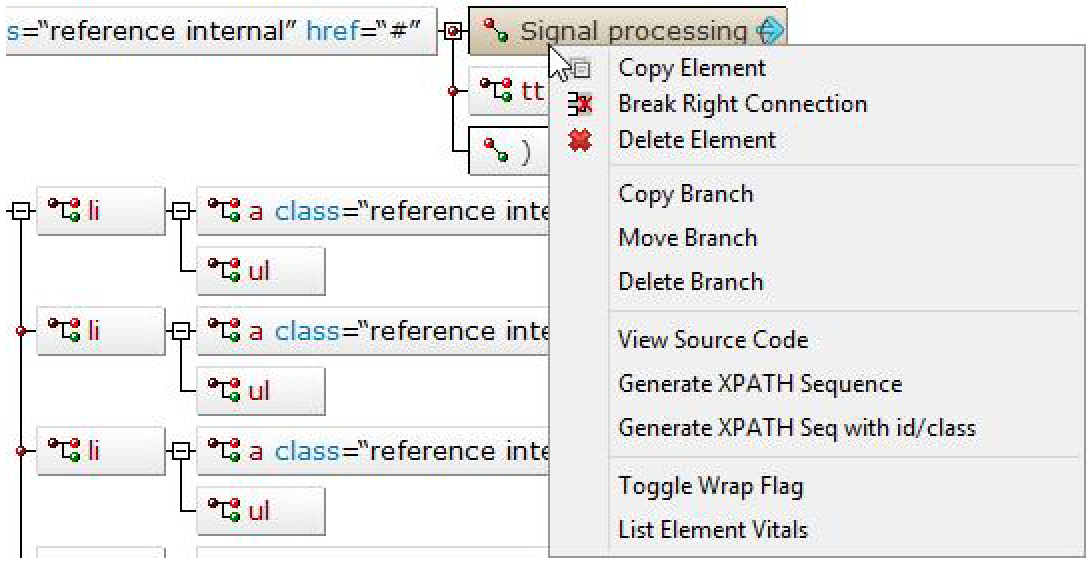
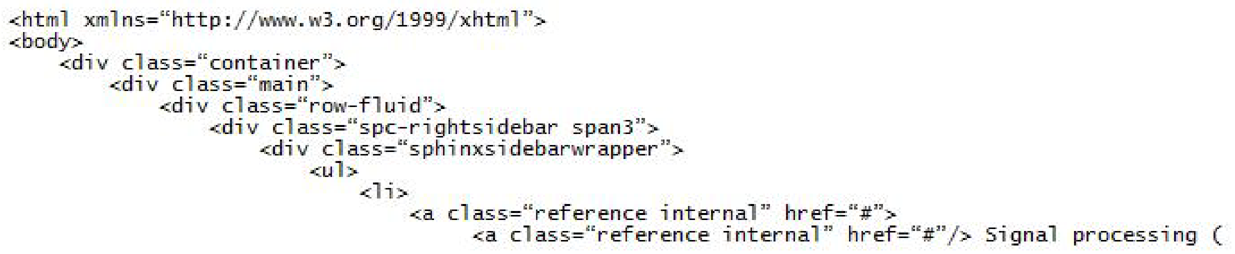


Figure 8

A right click on an element will bring up the menu with entries: View Source Code, Generate XPATH Sequence, and Generate XPATH Seq with id/class.

Selecting the View Source Code entry in menu from the right click on the highlighted element above displays the raw XML structure in the trace similar to the following. The raw image is recomposed to discard all preceding siblings and just show parents.



```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
    <div class="container">
        <div class="main">
            <div class="row-fluid">
                <div class="spc-rightsidebar span3">
                    <div class="sphinxsidebarwrapper">
                        <ul>
                            <li>
                                <a class="reference internal" href="#">
                                     <a class="reference internal" href="#"/> Signal processing (
```

If an entry is selected the Clipboard will receive the XPATH description of where the element is located, and a copy will be printed in the Trace. The following shows the result of selecting one, then the other of the example above.

Selecting the Generate XPATH Sequence will send the following to both the Trace and the clipboard.

- `XPATH: /html/body/div/div/div/div/div/ul/li/a`

Selecting the Generate XPATH Seq with id/class will send the following to both the Trace and the clipboard.

- `XPATH: /html/body/div class="container"/div class="main"/div class="row-fluid"/div class="spc-rightsidebar span3"/div class="sphinxsidebarwrapper"/ul/li/a`

Automatons

Covers:

- ✓

Appendix

Covers:

✓ Accepted EBNF Forms

Accepted EBNF Forms

	EBNF	ISO EBNF	ABNF	XBNF	XML EBNF
Assignment	Name::=E… Name=E… Name:SpaceE… Name->E…	Name=E…	Name=E…	Name::=E…	Name::=E…
Terminal	"…" '…' %bB… %dD… %hH… %xH… D… 0xH…	"…" '…'	"…" %bB… %dD… %hH…	"…"	"…" '…'
Non-terminal	W… <…>	W…	W… <…>	W… <…>	W…
Concatenation (And Then)	Space ,[3] W…-W…	,	Space	Space	& W…-W…
Disjunctive	| /	|	/	|	|
Conjunctive (And Also)	&			&	
Negation (But Not)	Space-W… Space-(E…)			~	
Optional	[E…] W…? (E…)?	[E…]	[E…] *1…	O(E…)	W…? (E…)?

Repeat (zero or more)	{E…}*
	{E…}
	W…*
		{E…}	*W…
			*(E…)
				#(E…)	W…*
					(E…)*

Repeat (one or more)	{E...}+ {E...}- W...+	{E...}-	1*W... 1*(E...)	N(E...)	W...+ (E...)+
Grouping	(E...)	(E...)	(E...)	(E...)	(E...)
Rule termination	; . LF	;	LF	.	LF
Range[5] Selection[5]	[E...- E...]* [E... E... E...]+				[^A-B] [ABC]
Comment[2]	(*...*) /*...*/ ;...EOL ...EOL %%...EOL	(*...*)	;... <EOL>		/*...*/
Special Expressions	?...? ~...	?...? n*...	n*n n*m		[WFC: E...] [VC: E...]

Legend: E... = Expression, W... = Word, D... = Decimal, H... = Hexadecimal, B... = Binary, ... = Literal, Space = One or more whitespaces.

Notes on EBNF Forms:

1. The column titled EBNF indicates the expressions that are recognized by the editor. The first entry in each group represents the default EBNF output form.
2. Comment fields will show up as inline green text, including if placed within the rule, excluding ...EOL, which is not passed through. (Their presents within rules have no effect on any of the operations.)
3. Any undefined character can be used as a concatenation token. Terminal values may not even require concatenation symbol, as long as there is a unique character to differentiate adjacent non-terminal values.
4. Care should be used when terminal values contain quotes within quotes to make the outside quotes different than the inside ones; i.e. double and single. However, ‘’’ and “”” are recognized properly.
5. Not available in this version.

	EBNF	**Comments**
Non-terminals may begin with	Any letter Including <	If non-terminal starts with < it must end with > and the second character must be a letter. The <> will subsequently be stripped from the expression when generating the Visral diagram.
Non-terminals may contain	Any letter Any number Including _ and spaces if inside <...>. May contain spaces if concatenation uses comma.	For Non-EBNF Uses: May contain any character except for > if inside <...>. May not contain “:” followed by a space or end in “.”

Non-terminals may end with	Any letter Any number Including _ >	
Terminal	“…”	May contain any character except double quote.
Terminal	‘…’	May contain any character except single quote.
Terminal	May start with a number or percent symbol.	Specifically: %bB… %dD… %hH… %xH… D… 0xH… See legend above.
Concatenation	,	If a comma is detected within a rule but outside a terminal or comment field, the entire file is assumed to use commas for concatenation.

ASCII Control Characters

Hex	Definition
00	NUL (Null)
01	SOH (Start of Heading)
02	STX (Start Text)
03	ETX (End Text)
04	EOT (End Transmission)
05	ENQ (Enquiry)
06	ACK (Acknowledge)
07	BEL (Bell)
08	BS (Backspace)
09	TAB HT (Horizontal Tab)
0A	LF (Linefeed, Newline)
0B	VT (Vertical Tab)
0C	FF (Form Feed)
0D	CR (Carriage Return)
0E	SO (Shift Out)
0F	SI (Shift In)
10	DLE (Data Link Escape)
11	DC1 (X-ON)
12	DC2
13	DC3 (X-OFF)
14	DC4
15	NAK (Negative Acknowledge)
16	SYN (Synchronous Idle)
17	ETB (End Transmission Blocks)
18	CAN (Cancel)
19	EM (End of Medium)
1A	SUB (Substitute)
1B	ESC (Escape)
1C	FS (File Separator)
1D	GS (Group Separator)
1E	RS (Record Separator)
1F	US (Unit Separator)

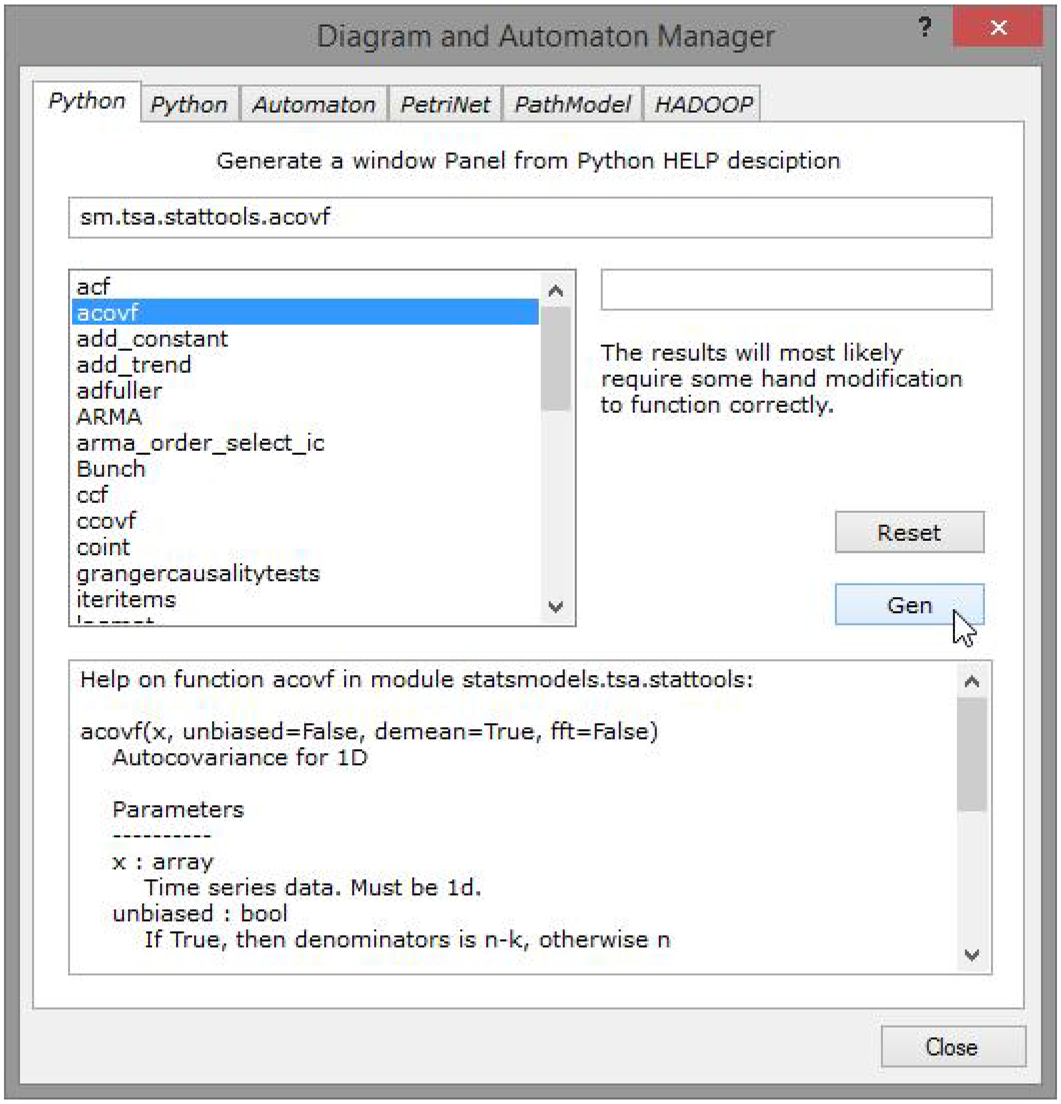
Errors and Warnings

	!WARNING: Auto-router timed out on Rule: [ ]
	!Warning: Dangling element [ ] in rule [ ]
	!Warning: Unable to remove direct recursive from rule [ ]
	!WARNING: Low memory
	!WARNING: Low on Blocks
	!WARNING: Cannot open file:
	WARNING: Saving rule with a reserved system name [ ]
	!WARNING: Cannot open file:
	!WARNING: Outside of Data Section Space:
	!WARNING: Outside of Sequence Section Space:
	ERROR: Overrun
	ERROR: missing argument of element in rule [ ]
	ERROR: missing second argument of element in rule [ ]
	ERROR: Undefined Rule [ ]
	ERROR: Undefined Rule References
	ERROR: Recursion Detected
	ERROR: Actor position in rule [ ]
	ERROR: Errors Detected - Build Aborted
	Error detected: Conversion aborted.
	Error detected: Conversion and file write aborted.
	Error: Cannot create file
	Error: Cannot write to file
	ERROR: Errors Detected - Build Aborted

Panel construction assistance

Although Operators can be created manually, there are three different methods provided to assist in creating them. They extract information from Python Help responses or from online function descriptions. However, because of variations in the way description are written it is not possible to perform one hundred percent correct conversions. In other words, they should only be looked at as providing a starting point and likely to require some level of rework.

Clicking this icon will bring up an assistant in Diagram and Automaton Manager. The first Python tab will display the Generate from Python Help menu.



Generate a window Panel from Python HELP desciption

sm.tsa.stattools.acovf

acf
acovf
add_constant
add_trend
adfuller
ARMA
arma_order_select_ic
Bunch
ccf
ccovf
coint
grangercausalitytests
iteritems

The results will most likely require some hand modification to function correctly.

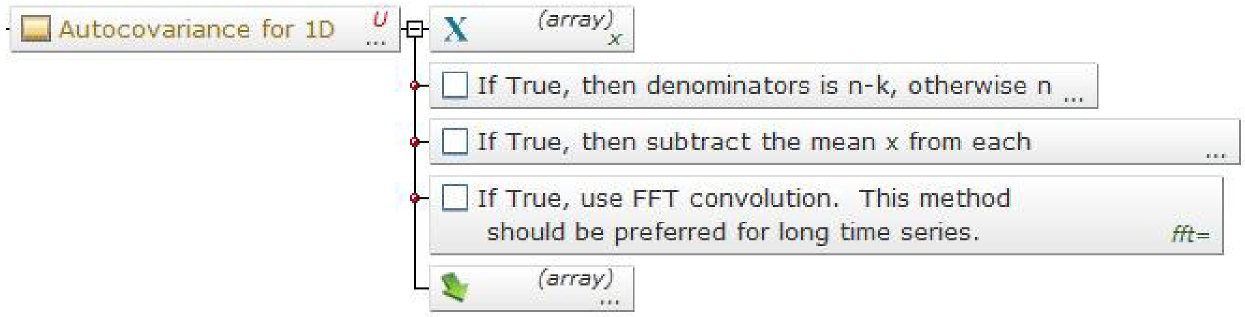
```
Help on function acovf in module statsmodels.tsa.stattools:

acovf(x, unbiased=False, demean=True, fft=False)
    Autocovariance for 1D

    Parameters
    ----------
    x : array
        Time series data. Must be 1d.
    unbiased : bool
        If True, then denominators is n-k, otherwise n
```

Double clicking on lines within the first list box will assemble on the first edit box an expression pointing to the desired function. When reached the Gen button will become enabled and can then be clicked.

At that point, moving the cursor to the diagram will cause it to change to represent an element group. Attaching the cursor symbol to an existing element within the diagram will reveal the graphical interpretation of the Operator including the proposed executable program. The image below is the actual result from clicking the Gen in the illustration above.



Autocovariance for 1D U ...

X (array) x

If True, then denominators is n-k, otherwise n ...

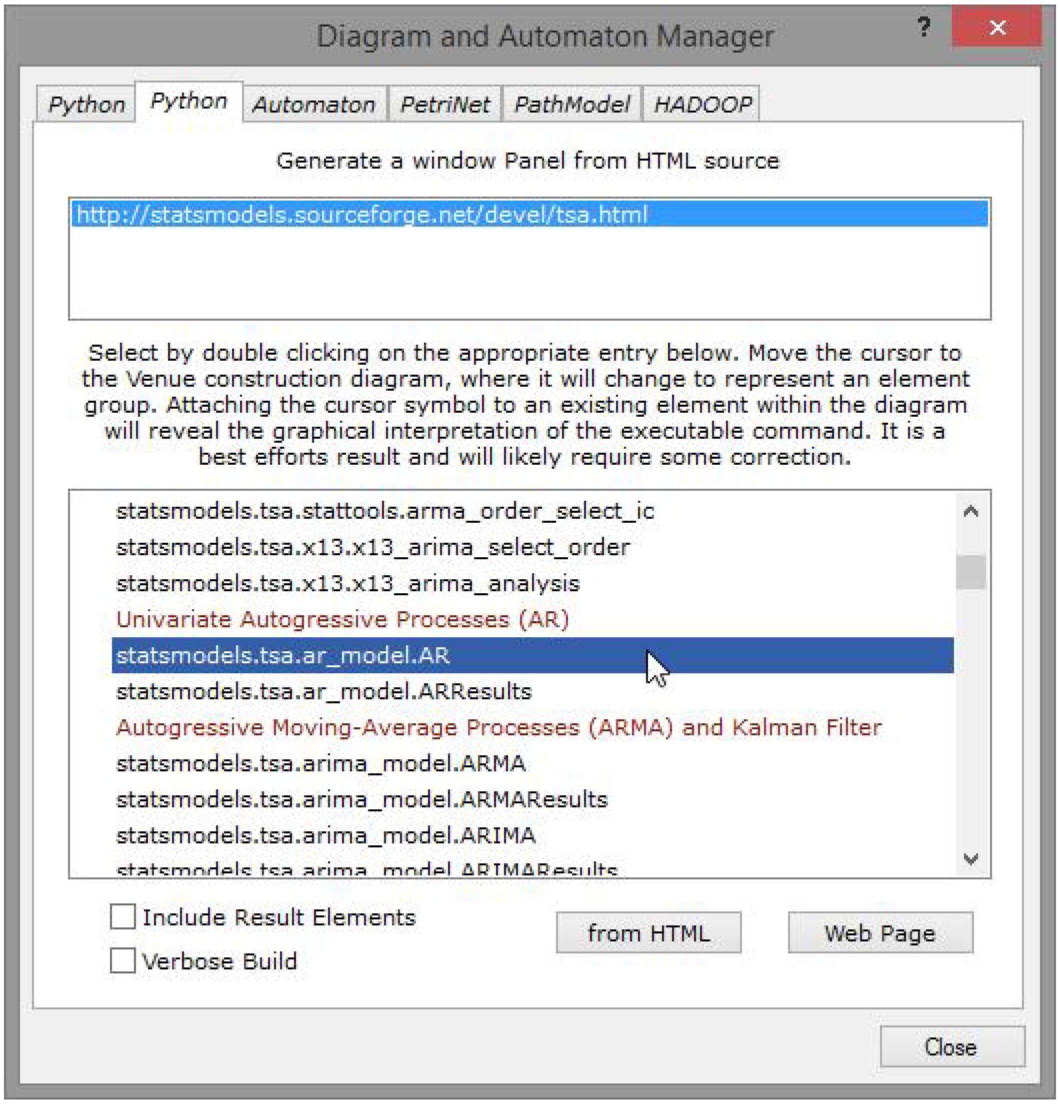
If True, then subtract the mean x from each ...

If True, use FFT convolution. This method should be preferred for long time series. fft=

(array) ...

The second Python tab brings up an assistant that makes an effort at translating a webpage to a Panel construction diagram. The top listbox contains the top level address. If necessary, double click it will fill the bottom listbox with of pages for translating. Double click an entry in that box will cause the process to take place.

Moving the cursor to the Venue construction diagram, will cause it to change to represent an element group. Attaching the cursor symbol to an existing element within the diagram will reveal the graphical interpretation of the executable command. It is a best efforts result and will likely require some correction.



Generate a window Panel from HTML source

http://statsmodels.sourceforge.net/devel/tsa.html

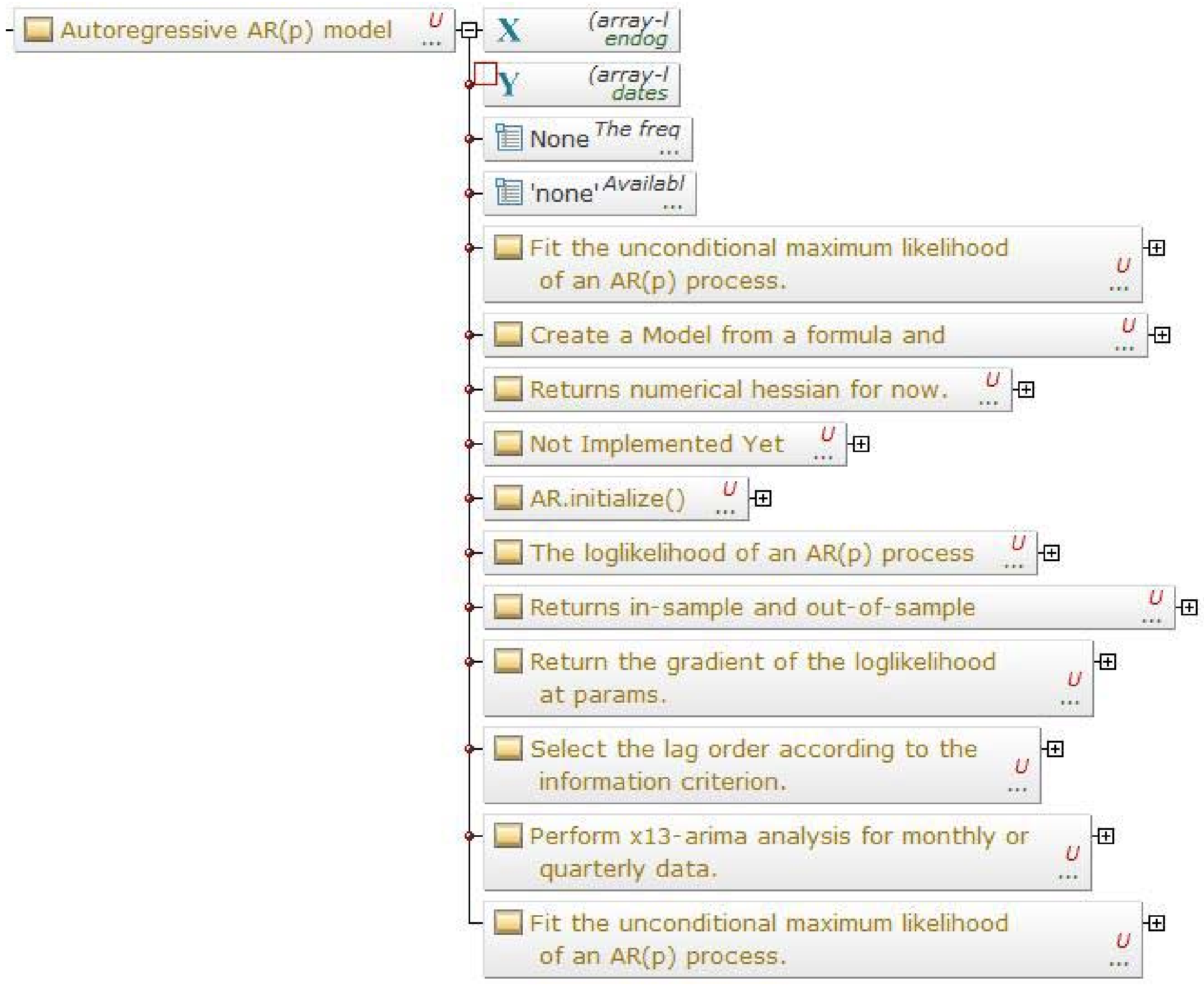
Select by double clicking on the appropriate entry below. Move the cursor to the Venue construction diagram, where it will change to represent an element group. Attaching the cursor symbol to an existing element within the diagram will reveal the graphical interpretation of the executable command. It is a best efforts result and will likely require some correction.

statsmodels.tsa.stattools.arma_order_select_ic
statsmodels.tsa.x13.x13_arima_select_order
statsmodels.tsa.x13.x13_arima_analysis
Univariate Autogressive Processes (AR)
statsmodels.tsa.ar_model.AR
statsmodels.tsa.ar_model.ARResults
Autogressive Moving-Average Processes (ARMA) and Kalman Filter
statsmodels.tsa.arima_model.ARMA
statsmodels.tsa.arima_model.ARMAResults
statsmodels.tsa.arima_model.ARIMA
statsmodels.tsa.arima_model.ARIMAResults

☐ Include Result Elements
☐ Verbose Build

The Web Page button will cause the default browser to go to the page of the entry selected in the lower listbox.

The from HTML button will perform a similar operation as above but on the content captured from the browser.


Autoregressive AR(p) model
X (array-l endog
Y (array-l dates
None The freq
'none' Availabl
Fit the unconditional maximum likelihood of an AR(p) process.
Create a Model from a formula and
Returns numerical hessian for now.
Not Implemented Yet
AR.initialize()
The loglikelihood of an AR(p) process
Returns in-sample and out-of-sample
Return the gradient of the loglikelihood at params.
Select the lag order according to the information criterion.
Perform x13-arima analysis for monthly or quarterly data.
Fit the unconditional maximum likelihood of an AR(p) process.

Copy from website:

class `statsmodels.tsa.ar_model.AR`(*endog*, *dates=None*, *freq=None*, *missing='none'*)[source]

Autoregressive AR(p) model

Parameters:

endog : array-like

1-d endogenous response variable. The independent variable.

dates : array-like of datetime, optional

An array-like object of datetime objects. If a pandas object is given for endog or exog, it is assumed to have a DateIndex.

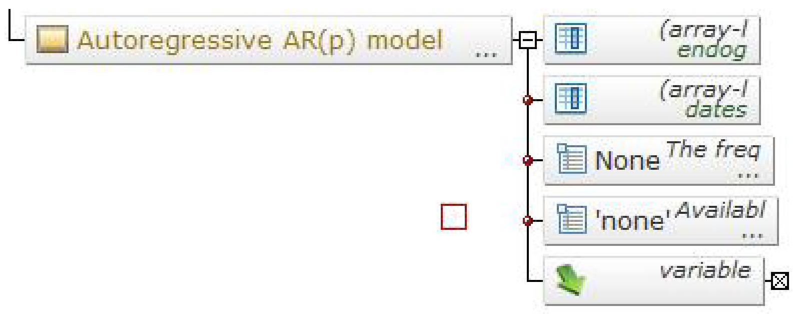
freq : str, optional

The frequency of the time-series. A Pandas offset or 'B', 'D', 'W', 'M', 'A', or 'Q'. This is optional if dates are given.

missing : str

Available options are 'none', 'drop', and 'raise'. If 'none', no nan checking is done. If 'drop', any observations with nans are dropped. If 'raise', an error is raised. Default is 'none.'

While in the diagram, click the



Paste HTML:

```
<dt id="statsmodels.tsa.ar_model.AR">

<em class="property">class </em><tt
class="descclassname">statsmodels.tsa.ar_model.</tt><tt
class="descname">AR</tt><big>(</big><em>endog</em>, <em>dates=None</em>,
<em>freq=None</em>, <em>missing='none'</em><big>)</big><a class="reference internal"
href="http://statsmodels.sourceforge.net/devel/_modules/statsmodels/tsa/ar_model.html
#AR"><span class="viewcode-link">[source]</span></a><a class="headerlink"
href="http://statsmodels.sourceforge.net/devel/generated/statsmodels.tsa.ar_model.AR.
html#statsmodels.tsa.ar_model.AR" title="Permalink to this definition"></a></dt>

<dd><p>Autoregressive AR(p) model</p>

<table class="docutils field-list" frame="void" rules="none">

<colgroup><col class="field-name">

<col class="field-body">

</colgroup><tbody valign="top">

<tr class="field-odd field"><th class="field-name">Parameters:</th><td class="field-
body"><p class="first"><strong>endog</strong> : array-like</p>

<blockquote>

<div><p>1-d endogenous response variable. The independent variable.</p>

</div></blockquote>

<p><strong>dates</strong> : array-like of datetime, optional</p>

<blockquote>

<div><p>An array-like object of datetime objects. If a pandas object is given

for endog or exog, it is assumed to have a DateIndex.</p>

</div></blockquote>

<p><strong>freq</strong> : str, optional</p>

<blockquote>

<div><p>The frequency of the time-series. A Pandas offset or &#8216;B&#8217;,
&#8216;D&#8217;, &#8216;W&#8217;,

&#8216;M&#8217;, &#8216;A&#8217;, or &#8216;Q&#8217;. This is optional if dates are
given.</p>

</div></blockquote>

<p><strong>missing</strong> : str</p>

<blockquote class="last">

<div><p>Available options are &#8216;none&#8217;, &#8216;drop&#8217;, and
&#8216;raise&#8217;. If &#8216;none&#8217;, no nan

checking is done. If &#8216;drop&#8217;, any observations with nans are dropped.

If &#8216;raise&#8217;, an error is raised. Default is &#8216;none.&#8217;</p>

</div></blockquote></td></tr></tbody></table></dd>
```

`numpy.fft.fft`(*a*, *n=None*, *axis=-1*)[source]¶

Compute the one-dimensional discrete Fourier Transform.

This function computes the one-dimensional *n*-point discrete Fourier Transform (DFT) with the efficient Fast Fourier Transform (FFT) algorithm [CT].

Parameters:

a : array_like

Input array, can be complex.

n : int, optional

Length of the transformed axis of the output. If *n* is smaller than the length of the input, the input is cropped. If it is larger, the input is padded with zeros. If *n* is not given, the length of the input along the axis specified by *axis* is used.

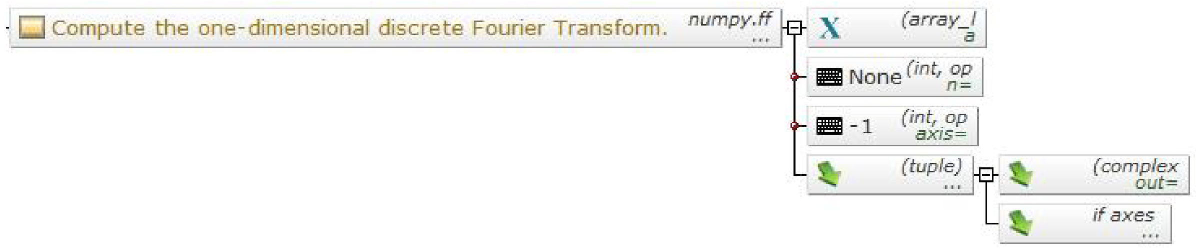
axis : int, optional

Axis over which to compute the FFT. If not given, the last axis is used.

Returns:

out : complex ndarray

The truncated or zero-padded input, transformed along the axis indicated by *axis*, or the last one if *axis* is not specified.


Compute the one-dimensional discrete Fourier Transform.
numpy.ff ...
X (array_l a
None (int, op n=
-1 (int, op axis=
(tuple) ...
(complex out=
if axes ...