4.900

VisiRule Tutorial

by Clive Spenser

VisiRule Tutorial

The contents of this manual describe the product, VisiRule, and are believed correct at time of going to press. They do not embody a commitment on the part of Logic Programming Associates (LPA), who may from time to time make changes to the specification of the product, in line with their policy of continual improvement. No part of this manual may be reproduced or transmitted in any form, electronic or mechanical, for any purpose without the prior written agreement of LPA.

Written by Charles Langley and Clive Spenser

Logic Programming Associates Ltd
Studio 30
The Royal Victoria Patriotic Building
Trinity Road
London SW18 3SX England

phone:	**+44 (0) 20 8871 2016**
fax:	**+44 (0) 20 8874 0449**
email:	**support@lpa.co.uk**
web:	**http://www.lpa.co.uk**

5 July, 2007

Contents

About This Document

This document describes the ideas behind VisiRule, and how to best combine the different constructs to build potentially complex charts.

.

You should also read the VisiRule User Guide which is supplied by LPA

Intelligent Flowcharts

What is VisiRule?

VisiRule is a tool for creating decision support software purely by drawing flowcharts. The end result is Flex or Prolog code which is automatically generated, compiled and ready to run, but which can also be copied and used in a separate program.

Not only can VisiRule be used by people with minimal programming skills. Once you are familiar with the tool, building an application is like creating a *graphical machine*. VisiRule also enhances productivity by considerably reducing the time it takes to produce a decision support system.

VisiRule is an *intelligent* flowcharting tool in two senses. Firstly, it is used to create knowledge-based systems and, secondly, it intelligently guides the construction process by constraining what you can and can't do on the basis of the semantic content of the emerging program. This means for example, that you cannot inadvertently construct invalid links.

As well as this real time semantic checking, VisiRule also checks the syntax of expressions as they are entered.

VisiRule provides the automatic construction of menu dialogues from questions. These are populated by items inferred from expression boxes throughout the flowchart tree which have a path to the question.

VisiRule also offers:

- a wide variety of question types including single and multiple choice, numeric and integer entry, text and set entry
- a powerful expression handling logic
- statement boxes for computable answers which are not decided by questioning the user
- code boxes for procedural code and external functions
- modularity allowing multiple charts to define one executable program

In this manual we will see how much can be done simply by using question boxes and expressions which test the answers to those questions. In particular we can build decision trees, classifiers and diagnostic systems of arbitrary complexity using these simple tools.

We will then go on to look at more complex applications which involve the use of statement boxes. These function like question boxes, but instead of asking the user for information, this information is inferred from what is already known.

Finally, we will look at the use of code boxes which allow the full power of Prolog and its associated toolkits to be embedded in a chart.

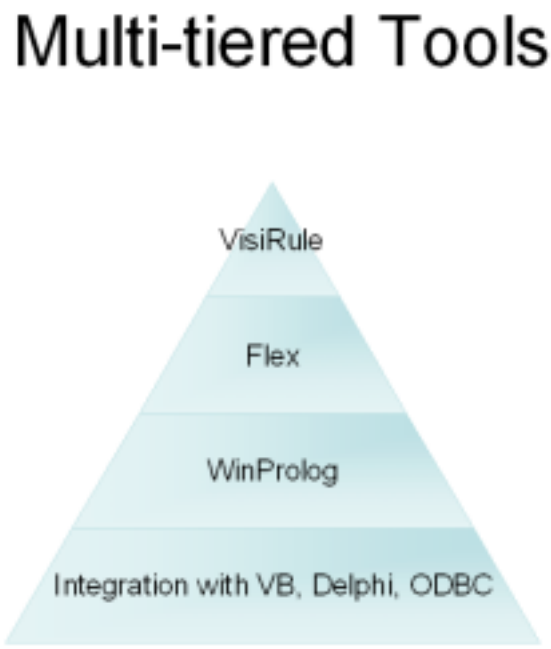


Figure 1 - VisiRule architecture

VisiRule lets you generate code in Flex which in turn gives you access to Prolog

Simple Chart

The simplest VisiRule charts consist of a **start box**, one or more **question boxes**, some **expression boxes** and some **end boxes** which are the conclusions drawn from the answers to the questions.

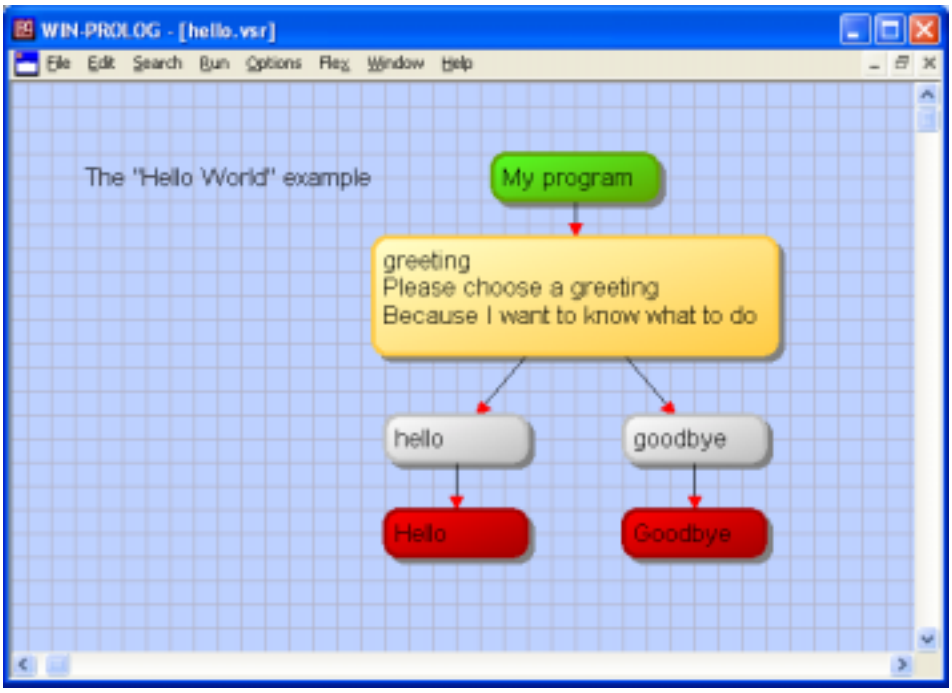


Figure 2 - Simple Chart

Extended Chart

We can extend a chart to include further questions as in the following chart.

Now we have three questions, all single-choice, five expression boxes, one of which is shared, one start box, one end box and one continuation box. And eleven links.

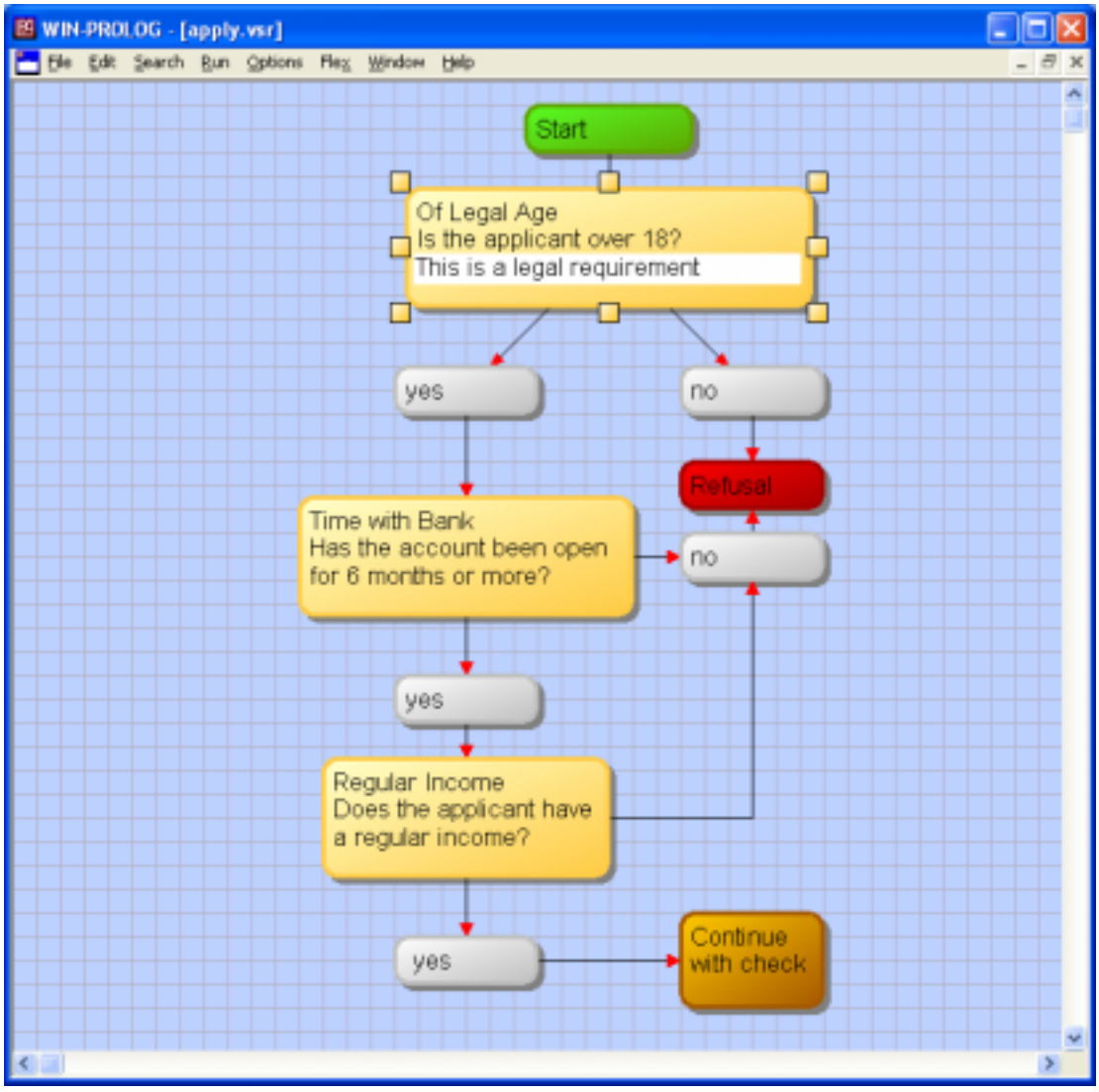


Figure 3 - extended chart

Modularity using Continuation Boxes

Each of the questions in previous chart can be seen as part of the premise of a rule. For example the first question represents the premise of the following rule:

1. **If** the applicant is under 18
2. **then** reject the overdraft application
3. **else** check time with bank

The other two rules implicit in figure2 are:

1. **If** the account is less than 6 months old
2. **then** reject the overdraft application
3. **else** check regular income

1. **If** the applicant does not have a regular income
2. **then** reject the overdraft application
3. **else** check amount applied for

We can break down the chart into separate charts, one for each rule, as shown below.

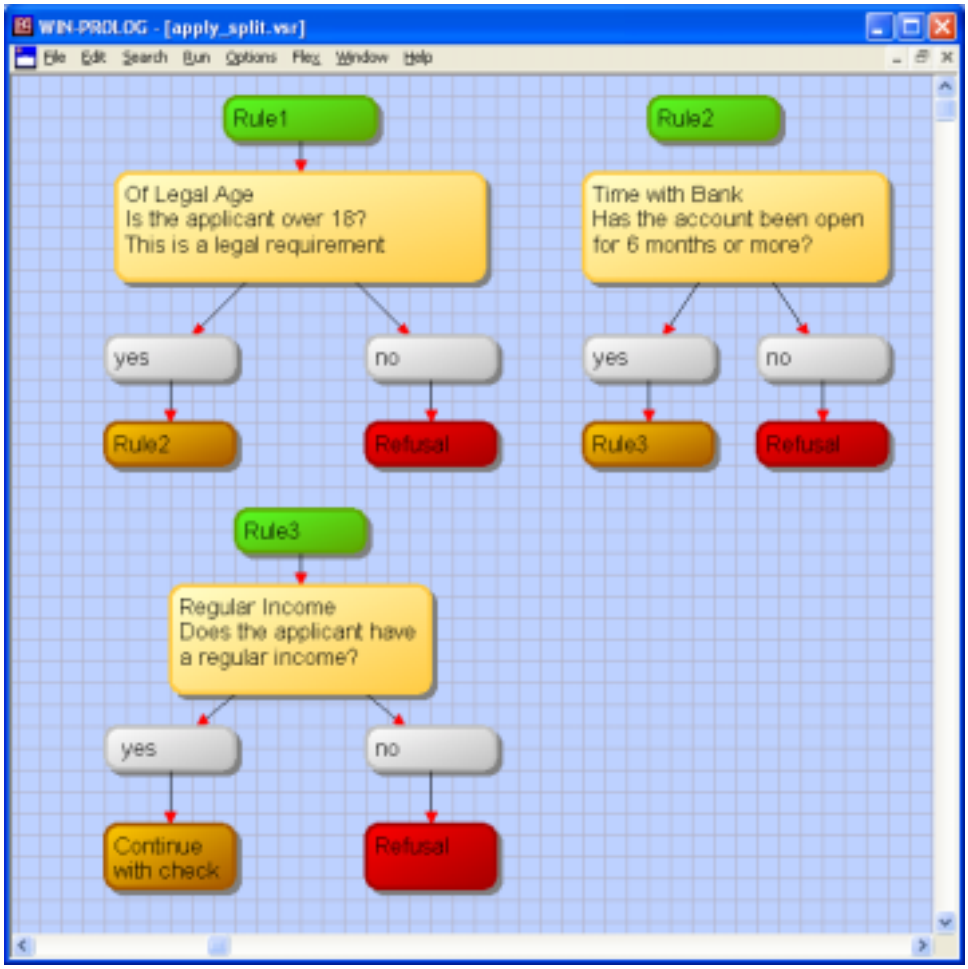


Figure 4 - modularised chart

Question Types

All the questions in the previous charts have been single choice questions which provide a menu of items with the user asked to choose one.

VisiRule has six distinct question types, each with a different colour as shown below

Figure 5 - question types

Single Choice	This is the default option. The menu produced will only allow the user to select one of the items on the menu.
Multiple Choice	This allows the user to select any or none of the items on the menu.
Number Input	Instead of a menu, this option provides an input box into which the user can enter any number
Integer Input	This is like Number Input, but only allows the user to enter an integer.
Set Input	An input box is also provided by this option. The user can type in a list of items, separated by a space character. For example: red amber green.
Name Input	Another input box is provided into which the user can type a word or phrase.

Integer Input and Number Input Questions

In the chart shown in figure 5 we have an **integer input** question box for the amount applied for and a **number input** question for the income.

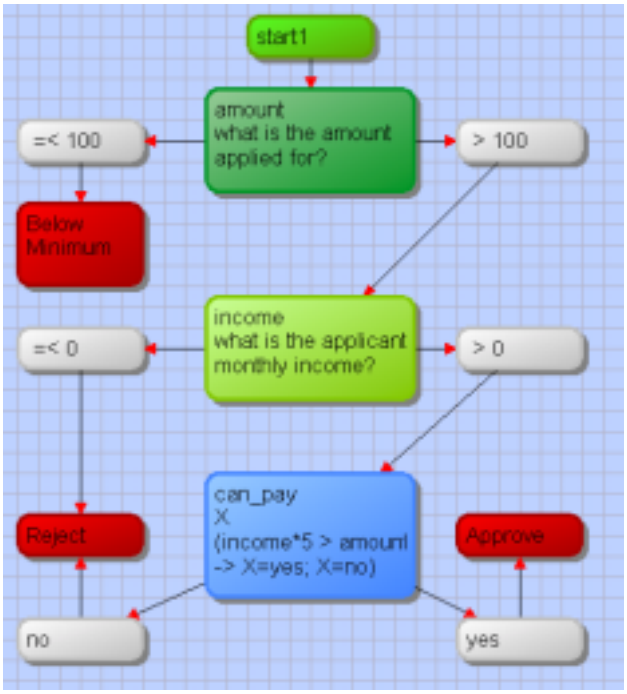


Figure 6 - numbers and integers

Instead of generating a menu, these question boxes generate a number input prompt as below.

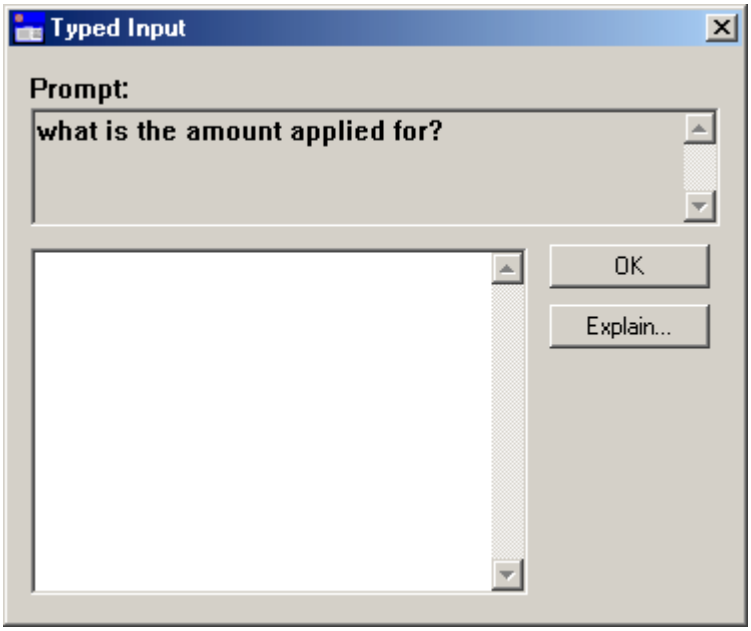


Figure 7 - simple question

Single Choice and Multiple Choice Questions

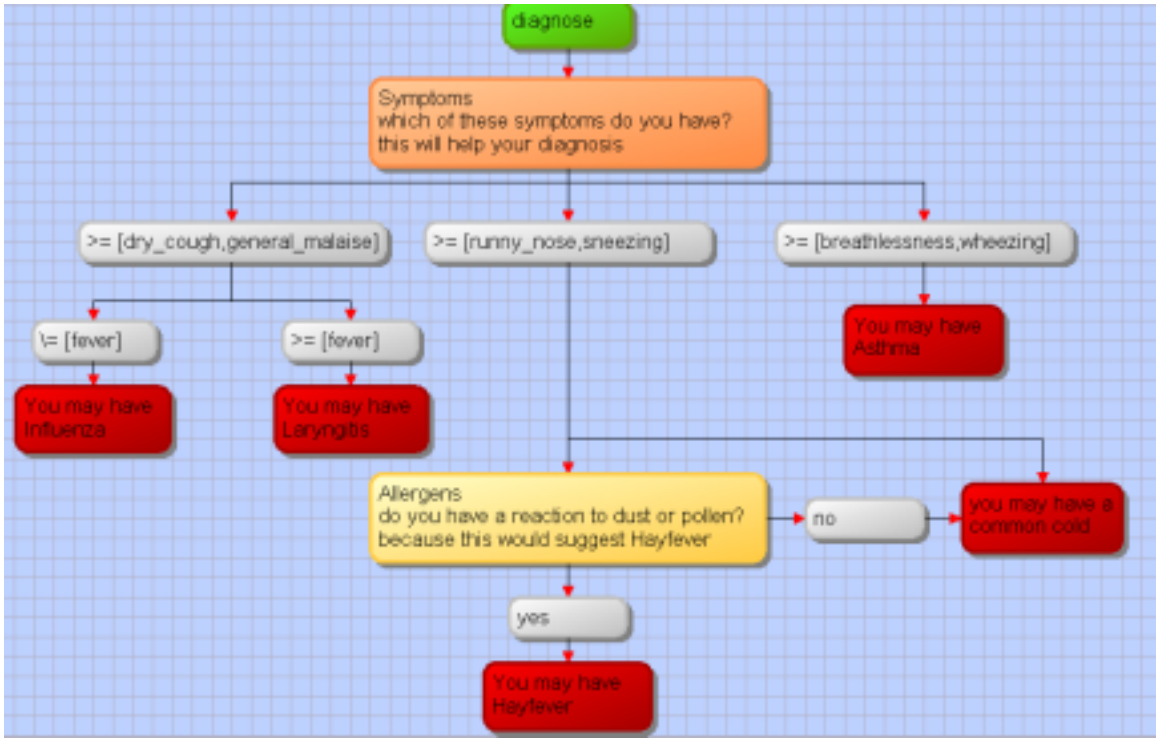


Figure 8 - Multiple Choice questions

You can use a multiple choice question to allow people to chose more than one symptom. Then the expression handler uses set interpretation semantics when applying operators. You can also use backtracking to find possible alternative diagnoses.

Set and Name Input Questions

Name Input

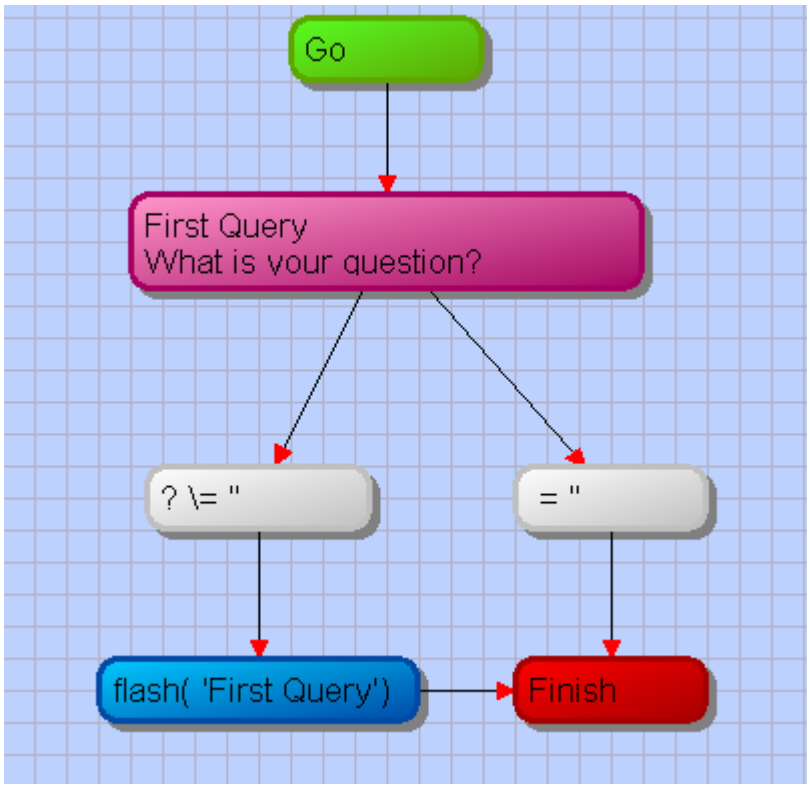


Figure 9 - Name input

A Name input question can be used to capture some free text which could be parsed or used as a database lookup, etc.

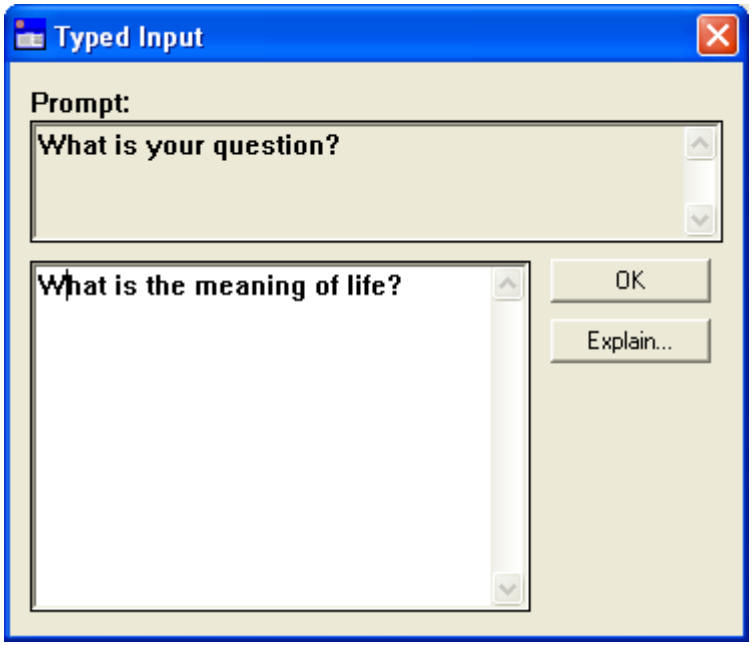


Figure 10 - Typing in free text

And then you get:

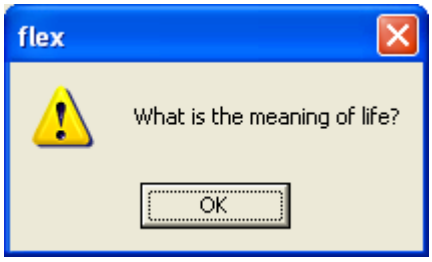


Figure 11 - displaying what has been typed

Set Input

Set input is similar to name input but returns a list of items

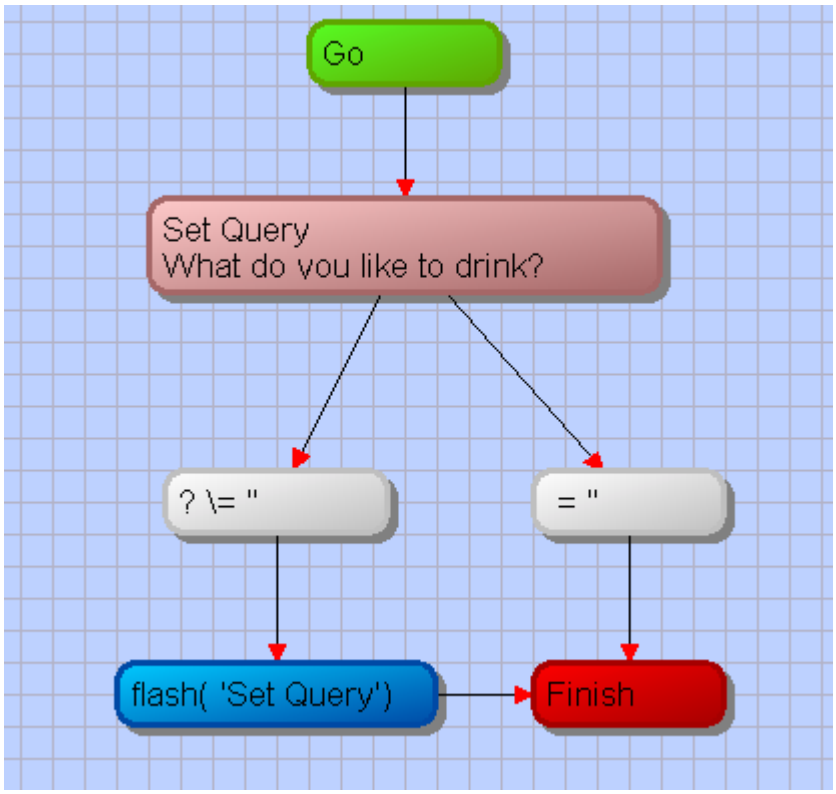


Figure 12 - A simple set input chart

and now we get a slightly different dialog

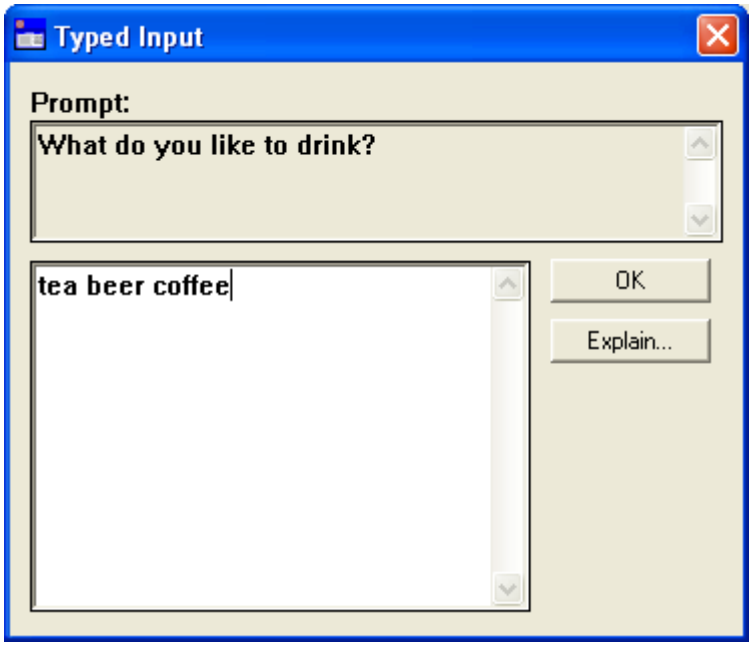


Figure 13 - Set input dialog

and the result is returned as a comma separated list

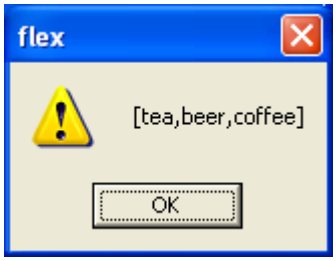


Figure 14 - result of set input

Types of Inference Task in VisiRule

Here we look at three types of inference task using VisiRule: Advisory, Classification and Diagnostic, all of which are typically represented by decision trees. Firstly we need to look at the similarities and differences of these three tasks.

Advisory Systems

function: to provide advice on options given preferences or facts
examples: advise holiday destination given choice of temperature and terrain
advise length of holiday entitlement given time worked and legal requirements

Classification Systems

function: to provide advice on facts given other facts
examples: advise on coin type given coin colour and shape
advise on illness type given symptoms

Diagnostic Systems

function: to provide advice on remedies given symptoms or facts
examples: advise on car repair needed given malfunction
advise on medicine needed given illness type

In fact, each of these inference tasks is a type of classification since advisory systems classify options according to preferences or facts and diagnostic systems classify actions according to symptoms or facts. The difference between them is ***what it is*** that they classify and ***on the basis of what***.

Decision Trees and Decision Tables

A decision tree can be graphically visualised by a tree like structure of nodes and arcs or it can be textually represented by a decision table.

Take the decision table below (figure 17) which recommends a holiday destination based on preferred temperature, continent and terrain. The graph to its right shows the structure of the decision tree which is equivalent to the table.

Although the tree is equivalent to the table it has the advantage of compactness. Some combinations in the table have no leaf nodes in the tree (those rows marked N/A in the destination column), perhaps because the travel agent does not offer these combinations. These combinations simply do not exist as paths through the tree, so the resulting series of questions asked does not include them.

hot or cold	continent	terrain	destination
cold	n_america	mountains	N/A
cold	n_america	desert	N/A
cold	n_america	snow	Rockies
cold	n_america	sea	Boston
cold	n_america	rivers	Mississippi
cold	n_america	mixed	Victoria
cold	asia	mountains	N/A
cold	asia	desert	N/A
cold	asia	snow	Ladahk
cold	asia	sea	Thailand
cold	asia	rivers	Kashmir
cold	asia	mixed	India
mixed	europe	mountains	Alps
mixed	europe	desert	N/A
mixed	europe	snow	N/A
mixed	europe	sea	Baltic
mixed	europe	rivers	Rhine
mixed	europe	mixed	Ireland
hot	s_america	mountains	N/A
hot	s_america	desert	Venezuala
hot	s_america	snow	N/A
hot	s_america	sea	Bahia
hot	s_america	rivers	Amazon
hot	s_america	mixed	Peru
hot	africa	mountains	N/A
hot	africa	desert	Sahara
hot	africa	snow	N/A
hot	africa	sea	Gambia
hot	africa	rivers	Nile
hot	africa	mixed	Nigeria

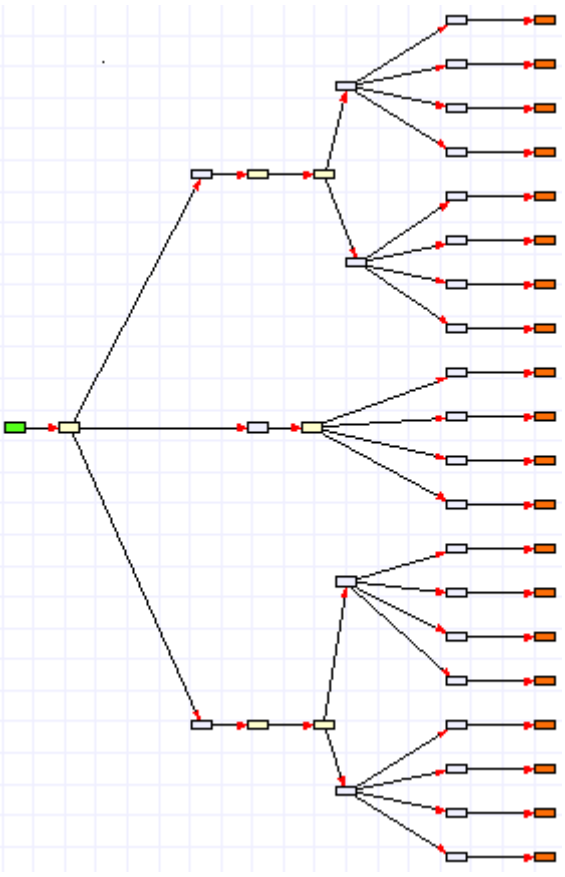


Figure 15 - Two representations

Advisory Systems

It is very easy to build such a tree in VisiRule, in fact the tree above is a VisiRule chart with all of the box fields set to hidden. The reason that they have been hidden is that the tree is too large to otherwise show on this page. However, the use of continuation boxes makes the chart more compact, effectively dividing the tree into a number of sub-trees.

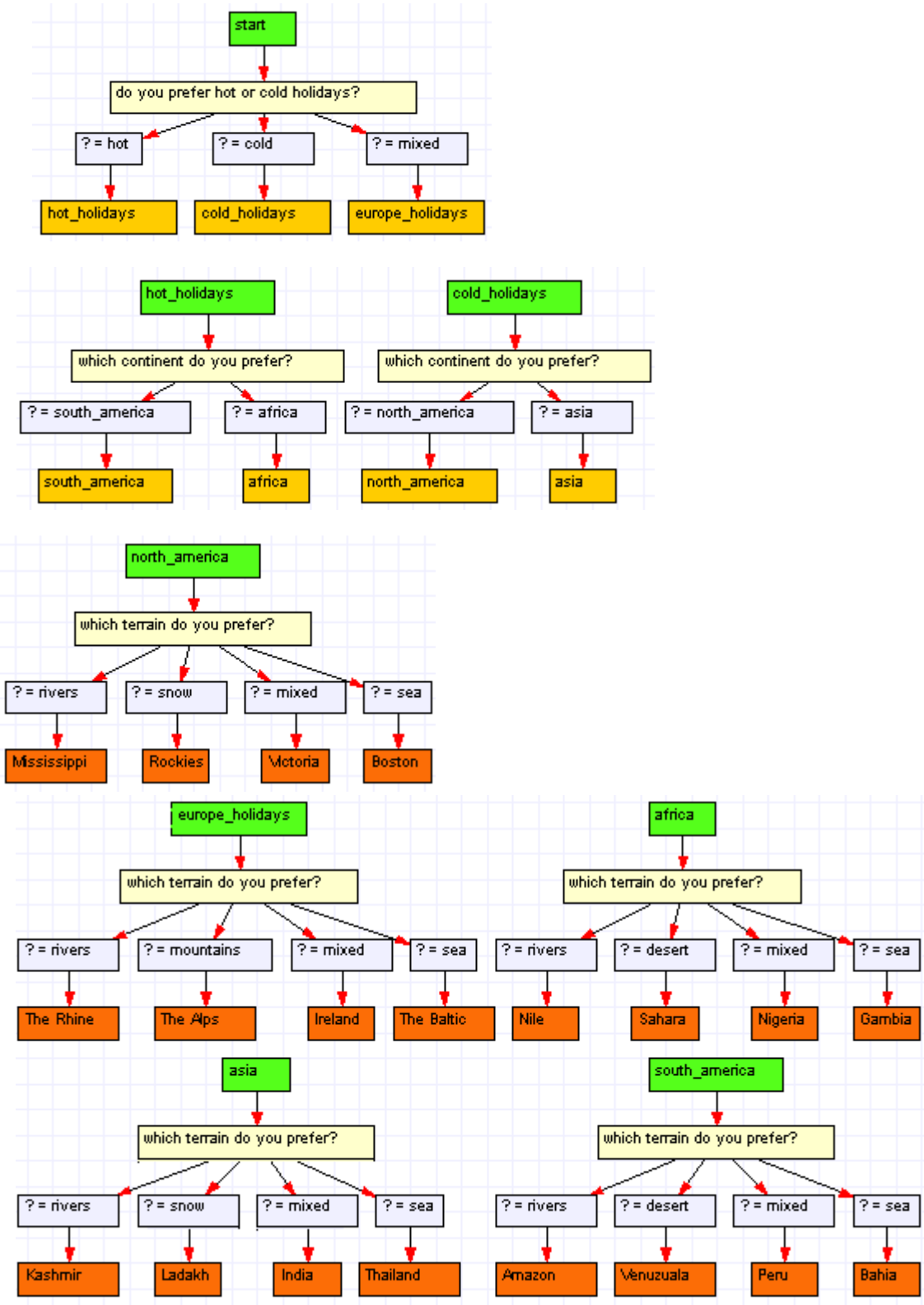


Figure 16 - Modularilty

In the next example we have a number of cases where one of the choices, ***ventilation = poor or ventilation is fair***, is redundant. This redundancy causes unnecessary duplication in the table.

material class	ventilation	rubber	solvent
1	poor	yes	polysol
1	poor	no	galaxy
1	fair	yes	polysol
1	fair	no	galaxy
1	good	yes	polysol plus
1	good	no	pulverizer
2	poor	yes	mtz 80
2	poor	no	cloripro 1
2	fair	yes	mtz 80
2	fair	no	cloripro 1
2	good	yes	machine saver
2	good	no	cloripro 2
3	poor	yes	dgrease
3	poor	no	acd 100
3	fair	yes	dgrease
3	fair	no	acd 100
3	good	yes	grime stopper
3	good	no	banish

To eliminate this redundancy we merge ***poor*** and ***fair*** into one so that there are only two choices for ventilation.

material class	ventilation	rubber	solvent
1	poor or fair	yes	polysol
1	poor or fair	no	galaxy
1	good	yes	polysol plus
1	good	no	pulverizer
2	poor or fair	yes	mtz 80
2	poor or fair	no	cloripro 1
2	good	yes	machine saver
2	good	no	cloripro 2
3	poor or fair	yes	dgrease
3	poor or fair	no	acd 100
3	good	yes	grime stopper
3	good	no	banish

Here is the decision table as a chart below.

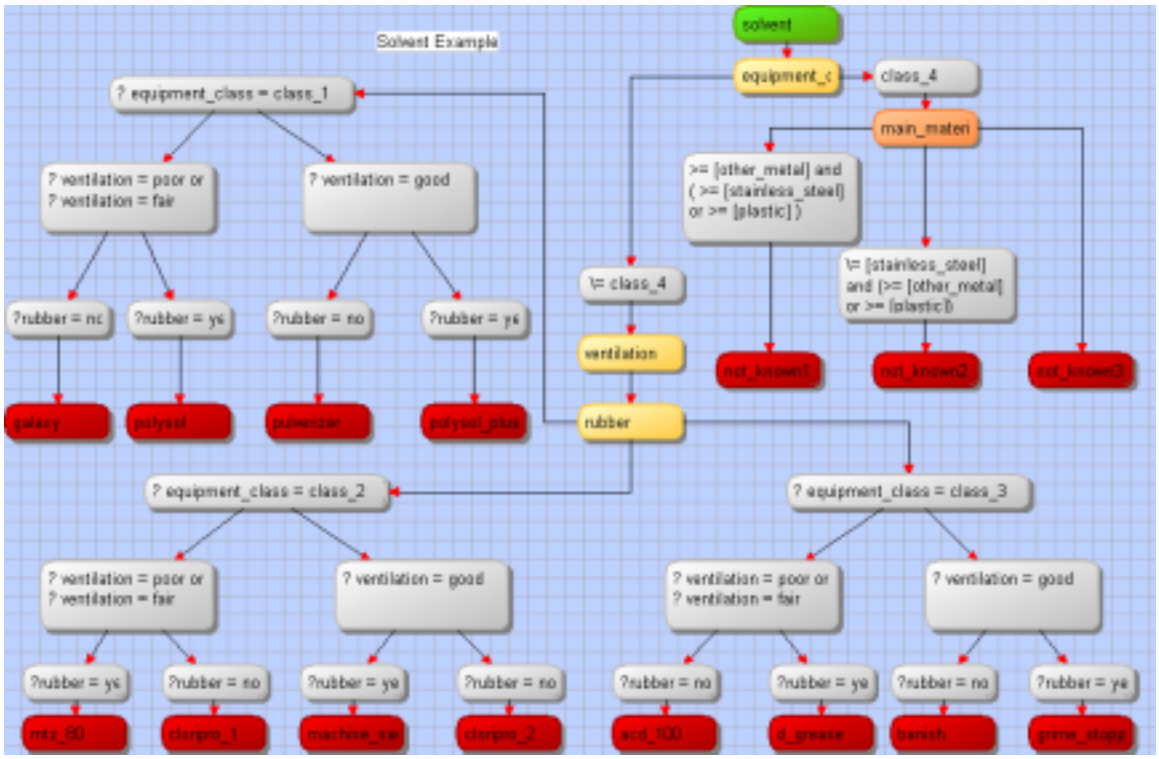


Figure 17 - Solvent Chart

Classification

Classification is another task we can easily perform with VisiRule. The following chart classifies any British coin.

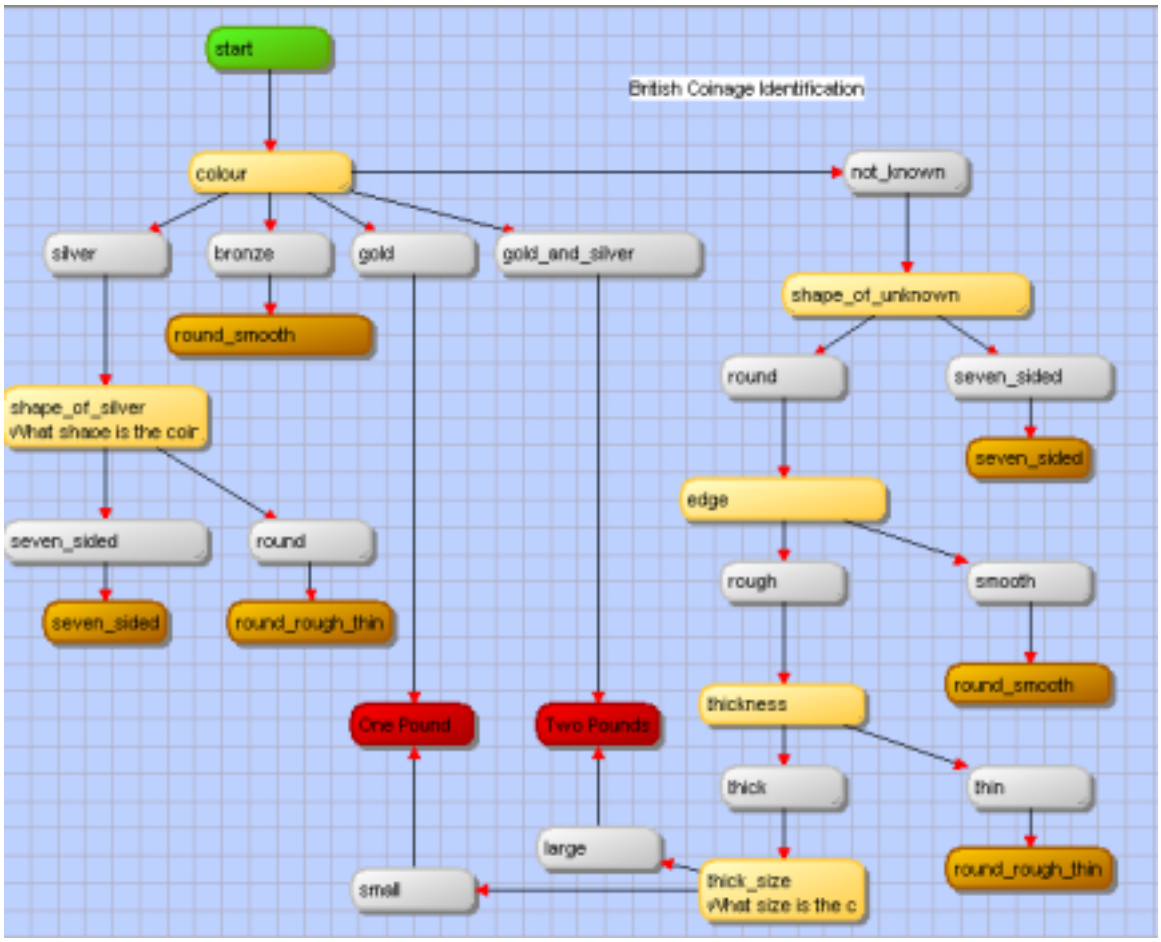
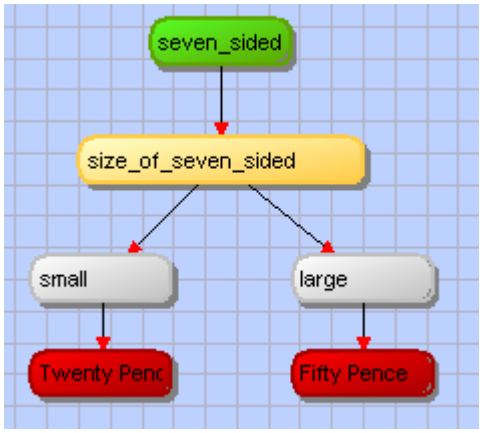
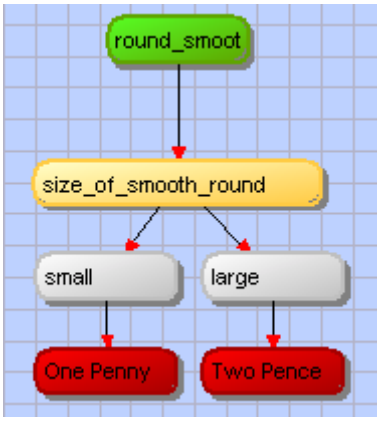


Figure 18 - Coin classification





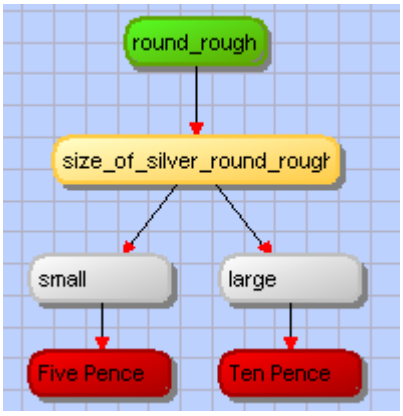


Figure 19 - Coin Sub-trees

Again we have a tree structure (actually a series of sub-trees) as shown below in figure 20.

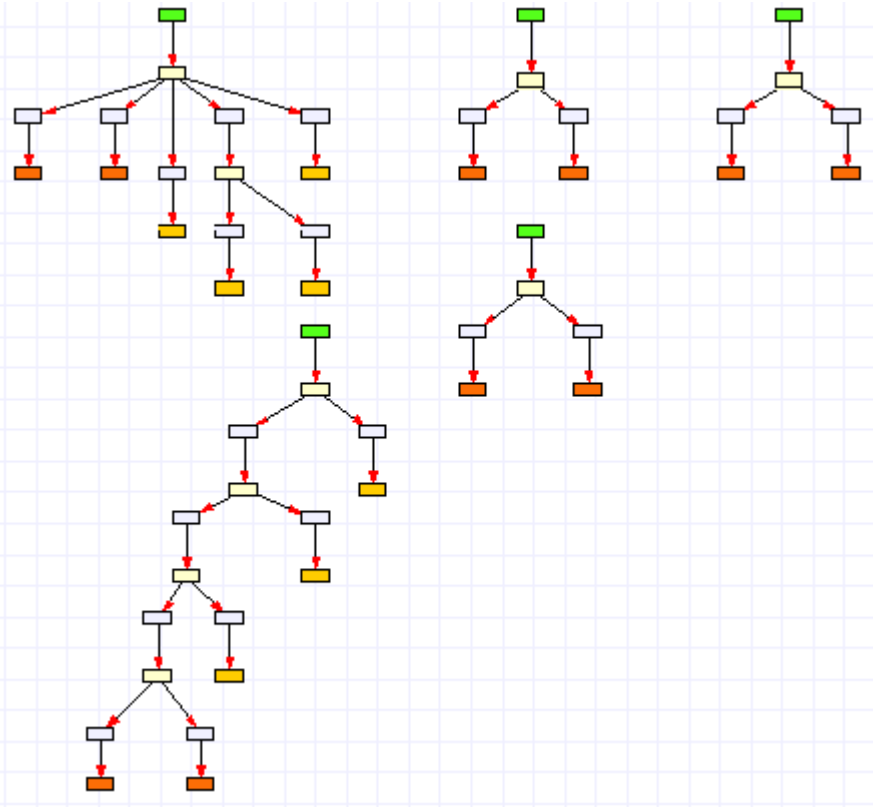


Figure 20 - Tree View

Diagnostics

Here is another decision tree, this time for a car diagnostic system.

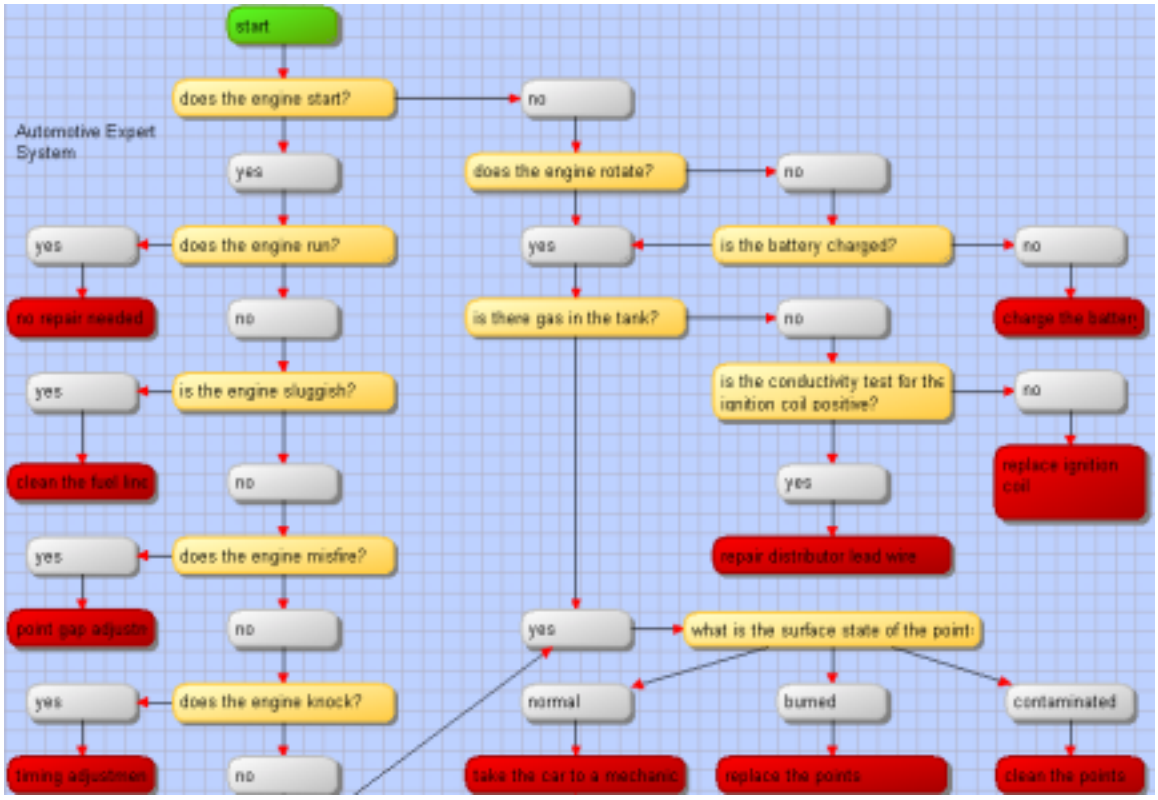
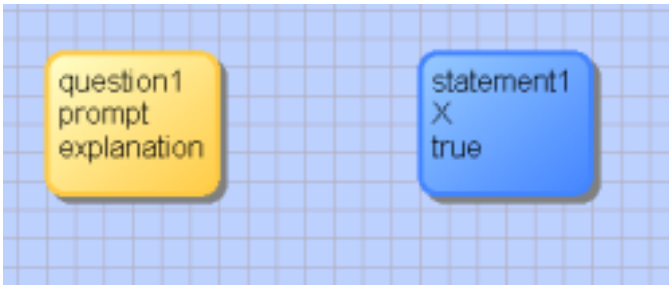


Figure 21 - Car Diagnostic system

Statement Boxes

What is a Statement Box?

A statement box looks superficially like a question box. However, its function is not to ask the user for information, but to calculate a value from information that is already known.



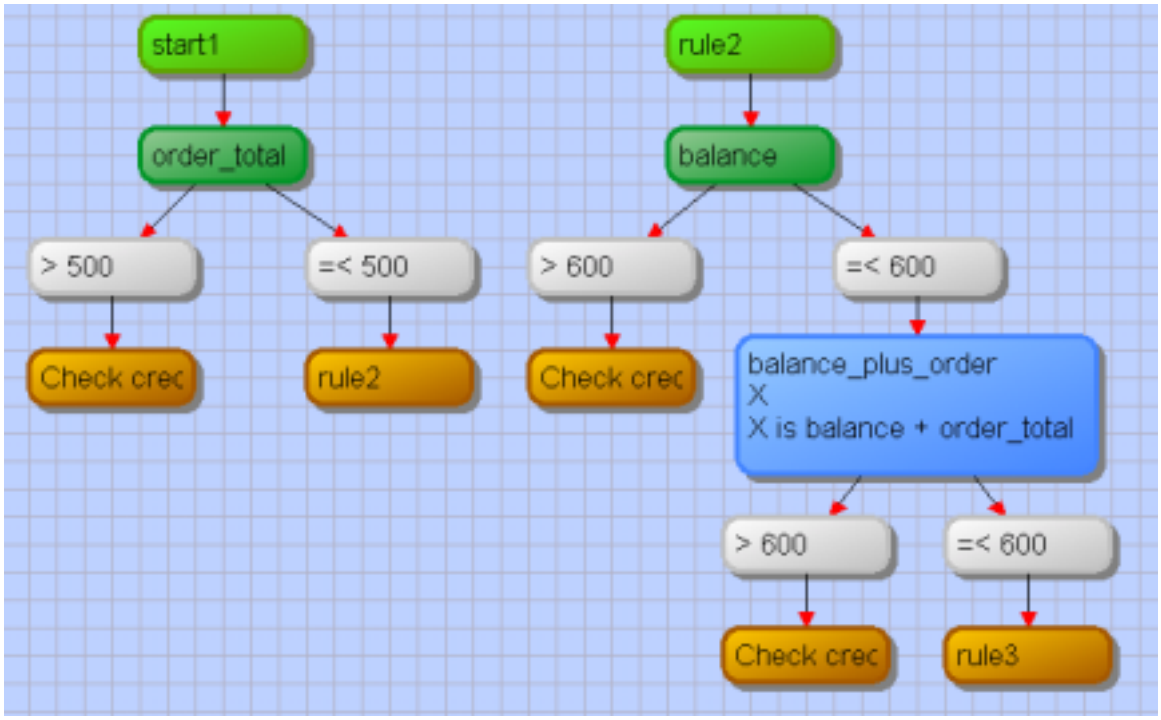


Figure 22 - Statement Box to do some simple maths

Statement boxes have three elements:

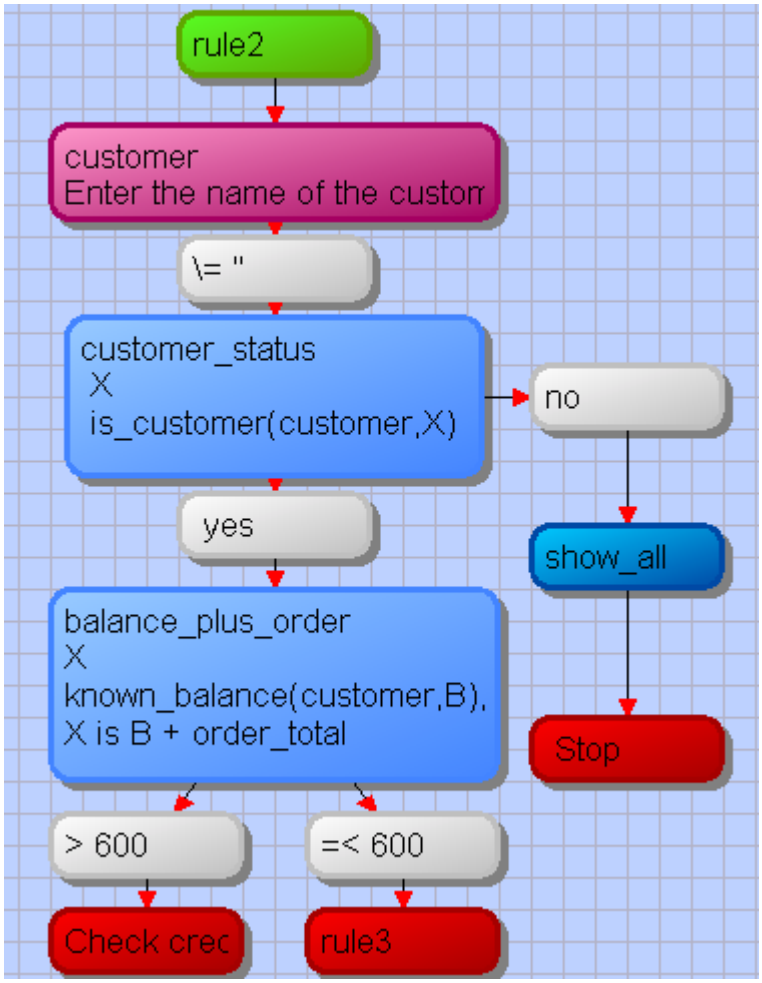
- an editable ***name*** ('balance_plus_order' in figure 2)
- an editable local ***variable*** ('X' in figure 2)
- editable Prolog code which is used to calculate the value ('X is balance+order_total')

Simple Calculations Using Statement Boxes

The above chart shows a typical use of a statement box to calculate a value (**balance_plus_order**) from two previously asked questions (**order_total** and **balance**).

Using Statement Boxes with Prolog Databases

The next example extends the chart seen earlier and requires that the user knows (or should look up) the customer's balance. If this information were held in a database it would be more sensible to look it up automatically. We can do this in VisiRule by loading in a Prolog database which can then be referenced directly by a statement box.




```
% customer.pl

known_balance('John Smith',800).
known_balance('Joe Doe',100).
known_balance('Mary Jones',10).

is_customer(X,'yes'):-
    known_balance(X, Bal).
is_customer(X,'no').

show_all:-
    flash('Here is a list of customers'),
    known_balance(X, Bal),
    write(X),nl,
    fail.
show_all.
```

Figure 23 – Accessing a Prolog database

In this example the external code (shown in the box on the right) is not represented in the chart, but is loaded by means of a code box(not shown).

This code contains both a database and auxiliary code to query the database to determine that the customer exists and lookup the balance. If the name entered is not in the database, it write out a list of all customers who are.

Date Handling using Statement Boxes

A number of applications will use dates, so the next example will look at how we can handle dates using statement boxes.

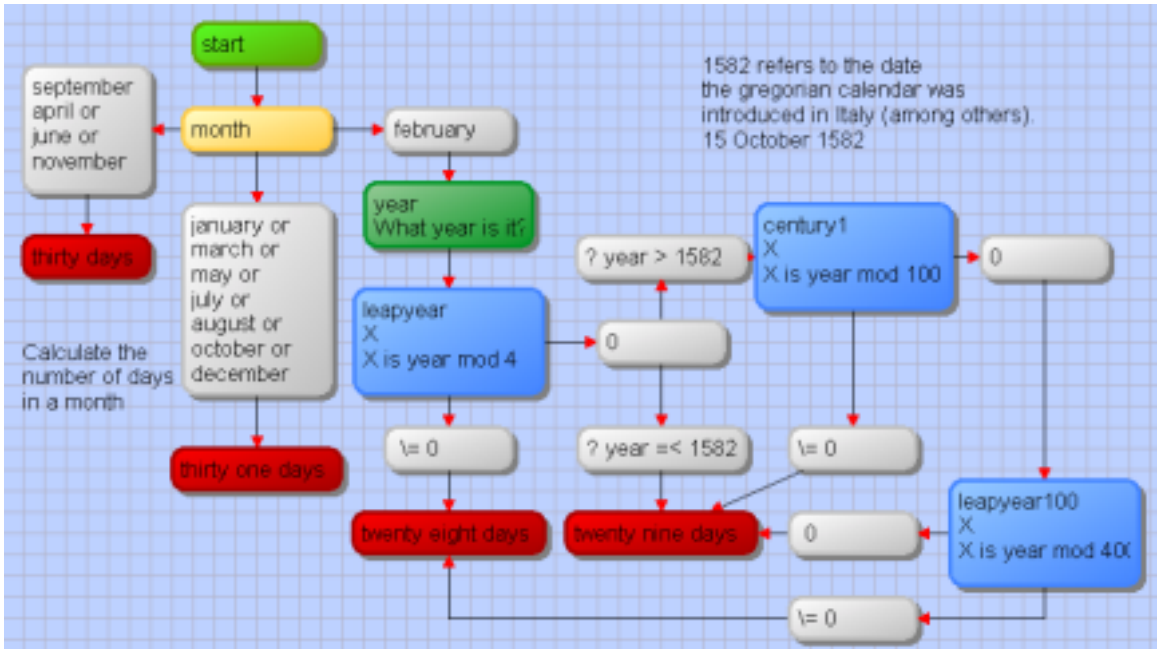


Figure 24 - Chart to calculate leap years

Three statement boxes perform the somewhat complicated calculation which uses the year to determine whether or not it is a leap year.

Note the use of '\= here, meaning 'not equal to'.

We can directly reference the global value **year** in the statement boxes because it is the name of a question box asked earlier in the chart.

Quizzing the User

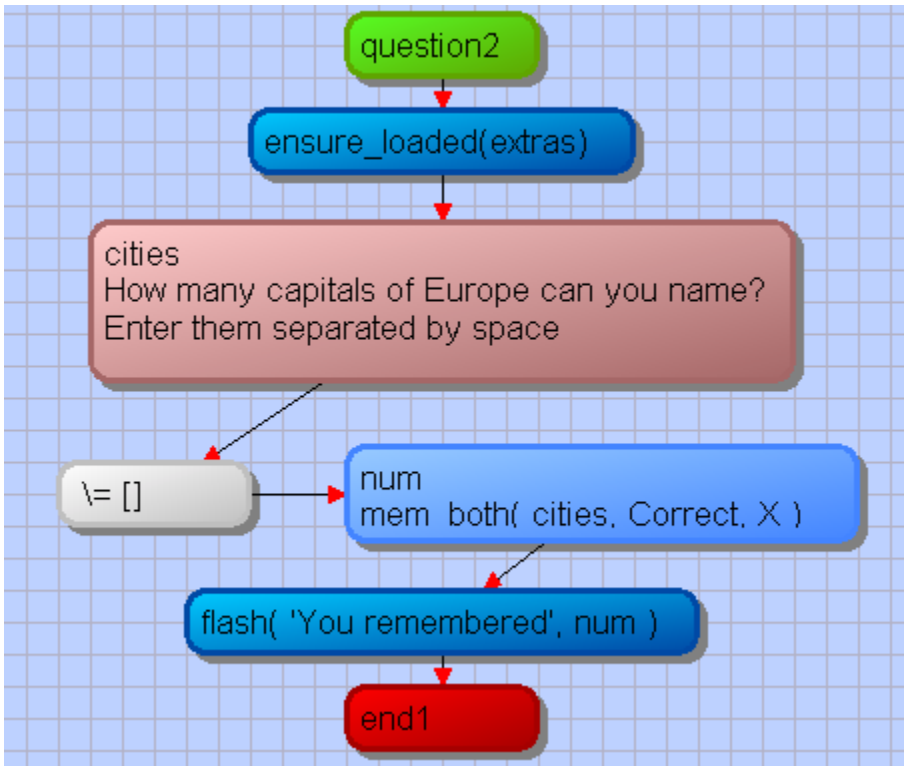


Figure 25 - Asking for capital cities

Here is the Prolog code loaded from the file extras.pl.

```
% given a list L1, this will walk thru that list looking
% to see which are in the database defined by town
% it returns them as a list X and counts its length
mem_both( L1, X, Number ) :-
  lwrupr_list( L1, L2 ),       % always convert entries to lower case
  sort( L2, L3 ),              % sort is an easy way to remove duplicates
  findall( P, (member(P,L3),town(P)), X ),
  length( X, Number ).

% convert all items in the input list into lower case items
lwrupr_list( [], [] ).
lwrupr_list( [Hd|Tl], [LowHd|LowTl] ) :-
  lwrupr( LowHd, Hd ),
  lwrupr_list( Tl, LowTl ).

town( london ).
town( paris ).
town( dublin ).
town( madrid ).
town( amsterdam ).
town( rome ).
```

If we run the chart we get a dialog asking us to name some capitals.

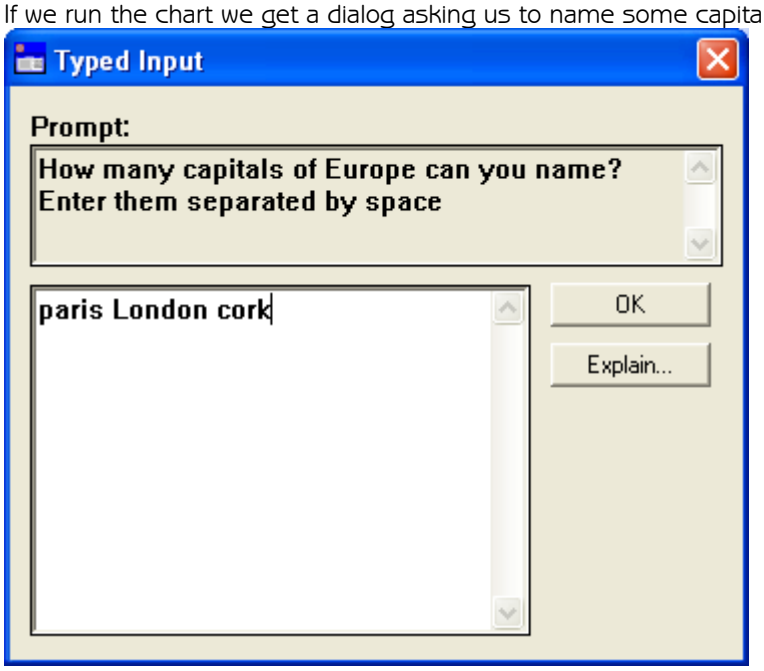


Figure 26 - Entering the names of the capitals

VisiRule will tell us how many are known to be capitals (i.e. in the file)

Figure 27 - Getting a score

Probabilistic Reasoning Using Code Boxes

In this example we load a prolog file which uses Bayesian probability to determine the likelyhood of a car insurance claim being fraudulent.

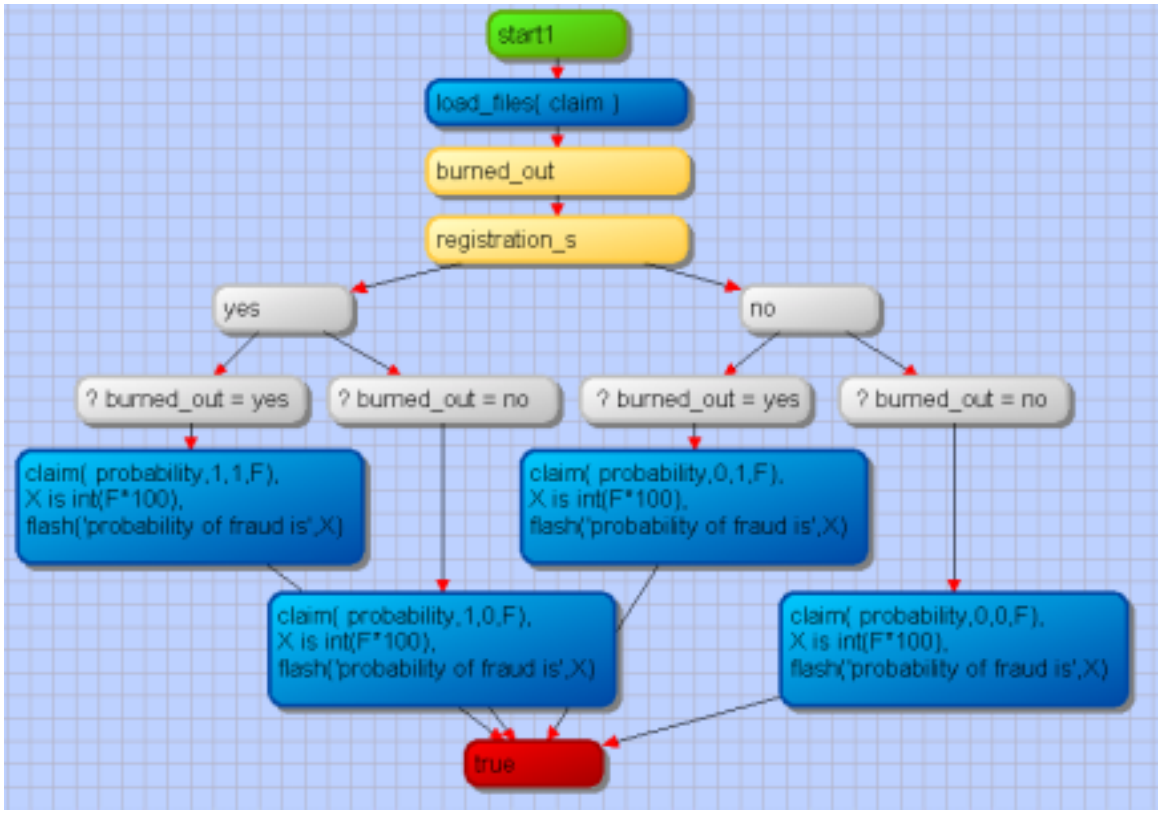
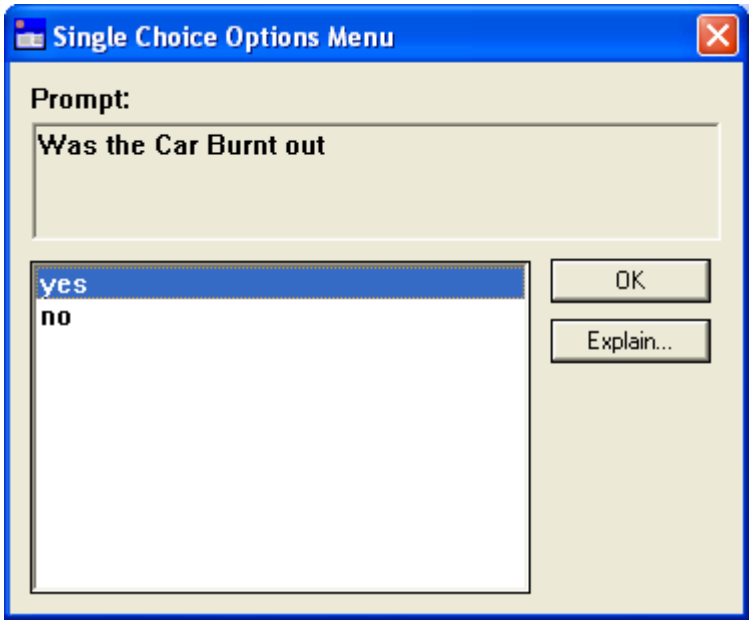


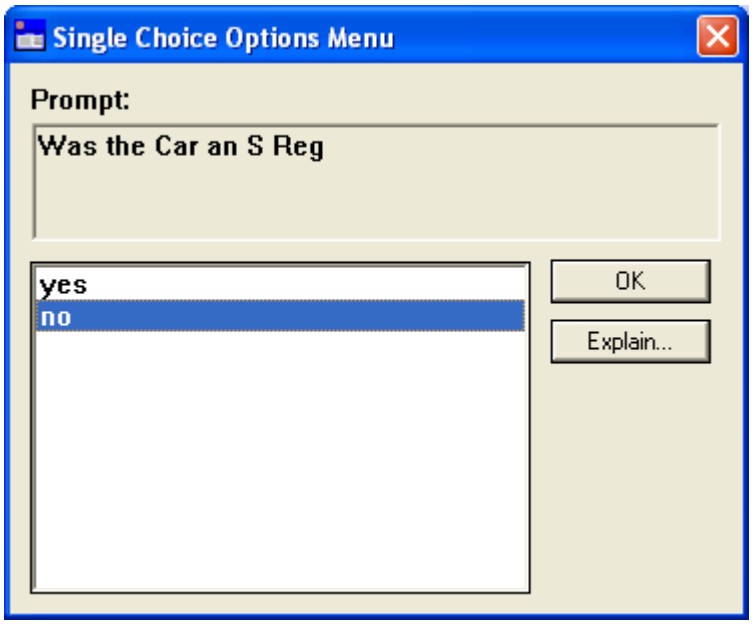
Figure 28 - Probabilistic reasoning

Here is the code loaded.

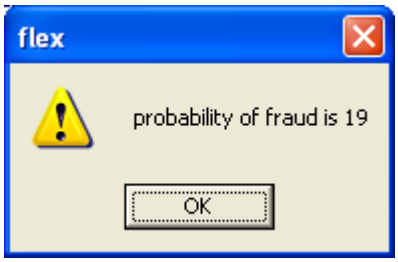
```
:- ensure_loaded(system(flint)).

claim( probability, M1, M2, M3 ) :-
  restart,
  uncertainty_value_reset( probability                                  ),
  uncertainty_value_set(   probability, recovered_burnt_out, yes, M1 ),
  uncertainty_value_set(   probability, registration_s,      yes, M2 ),
  uncertainty_propagate(   probability, [r1,r2]                         ),
  uncertainty_value_get(   probability, fraud,               yes, M3 ).

uncertainty_rule( r1 ) :-
  if    recovered_burnt_out is yes      affirms  2.20 denies 0.20
  then  fraud is yes.

uncertainty_rule( r2 ) :-
  if    registration_s      is yes      affirms  1.20 denies 0.95
  then  fraud is yes.
```



and the answer the next question



and then we get an answer



Fuzzy Logic using Code Boxes

An example which uses fuzzy logic to apply the appropriate brake force to a train given its current speed and distance from the buffers.

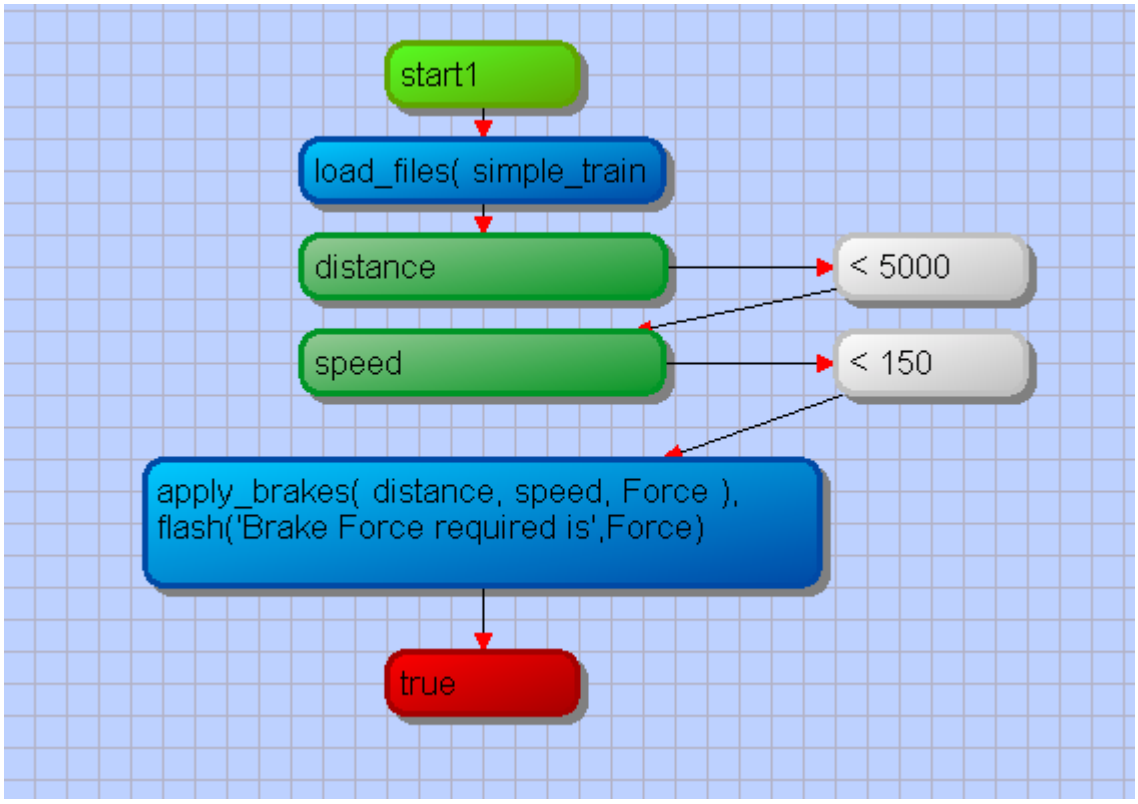


Figure 29 - Fuzzy Logic example

The Prolog file sim_train.pl contains something like:

```
:- ensure_loaded( system( flint ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Propagation of fuzzy values %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
apply_brakes( BufferDistance, TrainSpeed, BrakeForce ) :-
   apply_brakes( Method, BufferDistance, TrainSpeed, BrakeForce ).

apply_brakes( Method, BufferDistance, TrainSpeed, BrakeForce ) :-
   Method = bounded_range,
   uncertainty_value_reset( fuzzy ),
   fuzzify( buffer_distance, BufferDistance ),
   fuzzify( train_speed,     TrainSpeed     ),
   uncertainty_propagate( fuzzy, braking_matrix ),
   defuzzify( brake_force, BrakeForce, Method ).

%%%%%%%%%%%%%%%%%%%
% Fuzzy variables %
%%%%%%%%%%%%%%%%%%%

fuzzy_variable( buffer_distance ) :-
```

```
   [ 0 , 1000 ] ;
   low,    \,  linear, [ 0, 500       ] ;
   medium, /\, linear, [ 0, 500, 1000 ] ;
   high,    /, linear, [    500, 1000 ] .

fuzzy_variable( train_speed ) :-
   [ 0 , 150 ] ;
   low,    \,  linear, [ 0, 75      ] ;
   medium, /\, linear, [ 0, 75, 150 ] ;
   high,    /, linear, [    75, 150 ] .

fuzzy_variable( brake_force ) :-
   [ 0 , 100 ] ;
   very_low,  \,  linear, [ 0, 25             ] ;
   medium,    /\, linear, [    25, 50, 75     ] ;
   very_high,  /, linear, [            75, 100 ] .

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FAM - Fuzzy Associative Memory %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

uncertainty_matrix( braking_matrix ) :-

   buffer_distance  *  train_speed  ->  brake_force ;

   low              *  _            ->  very_high   ;
   _                *  high         ->  very_high   ;
   medium           *  medium       ->  medium      ;
   medium           *  low          ->  medium      ;
   high             *  medium       ->  medium      ;
   high             *  low          ->  very_low    .
```

When we run this chart we get prompted for 2 input values and the will get back an output value.

WebFlex deployment

Let's take the Hello World chart and publish it on the web using WebFlex.

(This presumes that we have already installed and tested WebFlex as per the WebFlex User Guide).

There are a number of steps we must go thru to get the chart published on the internet.

1) Export the chart as KSL code.
This creates a Flex version of our chart on disk called HELLO.KSL

2) Then add into the file HELLO.KS:

```
relation run if 'My program'( Conclusion ) and write( Conclusion ).
```

This line links the main goal in the WebFlex.ini file to the name of our start box.
Remember to save the file after making the edit.

3) Copy/move the HELLO.KSL file into the correct WebFlex directory, probably something like:
C:\InetPub\pws\Exec\flx\examples

4) Update the WebFlex.ini by adding in some extra entry:

[vr2]
main_goal=run

5) Now you can paste into your browser (or create a HTML page with this as an entry):

http://localhost/pws_exec/flx/webflex.exe?webflex=vr2

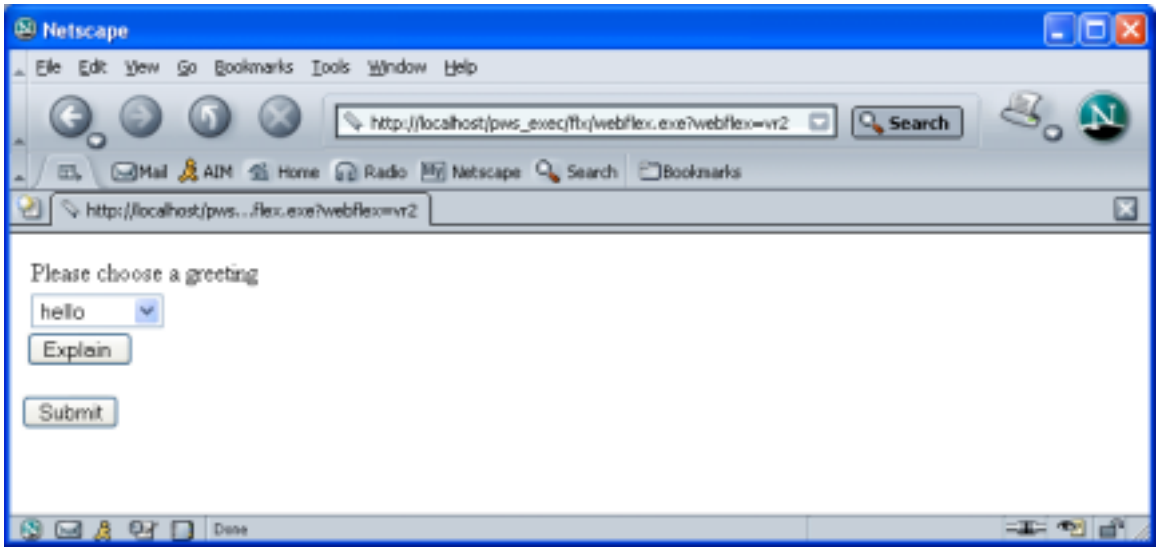
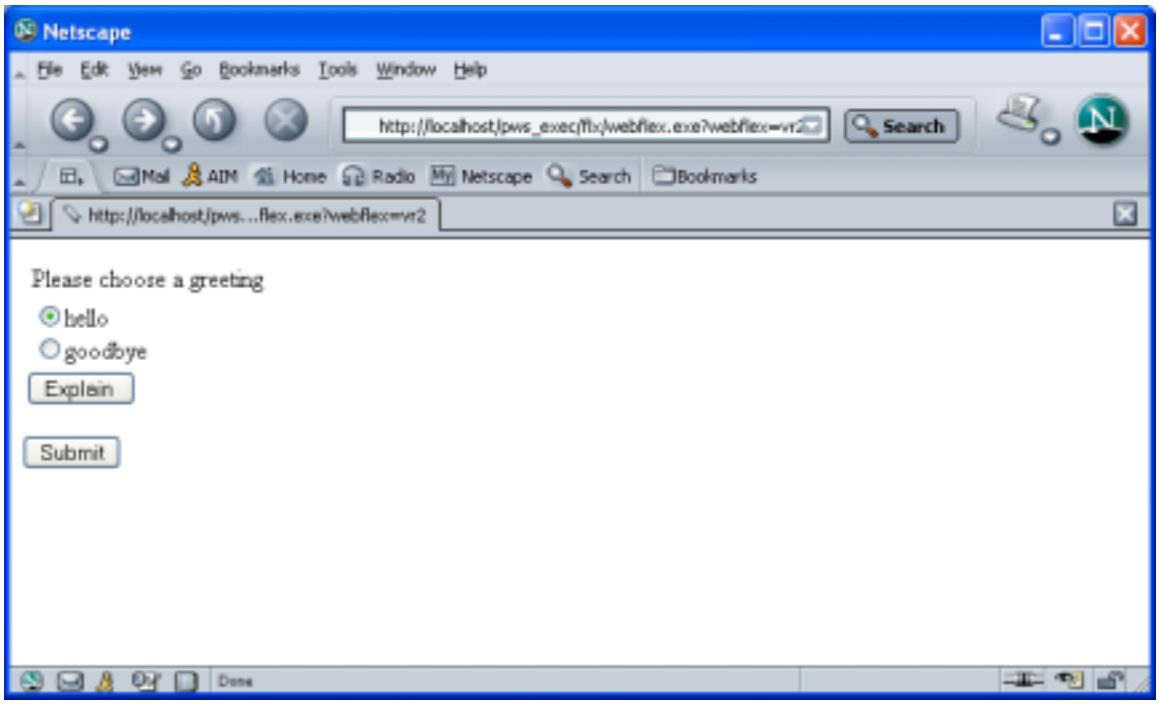


Figure 30 - First WebFlex page

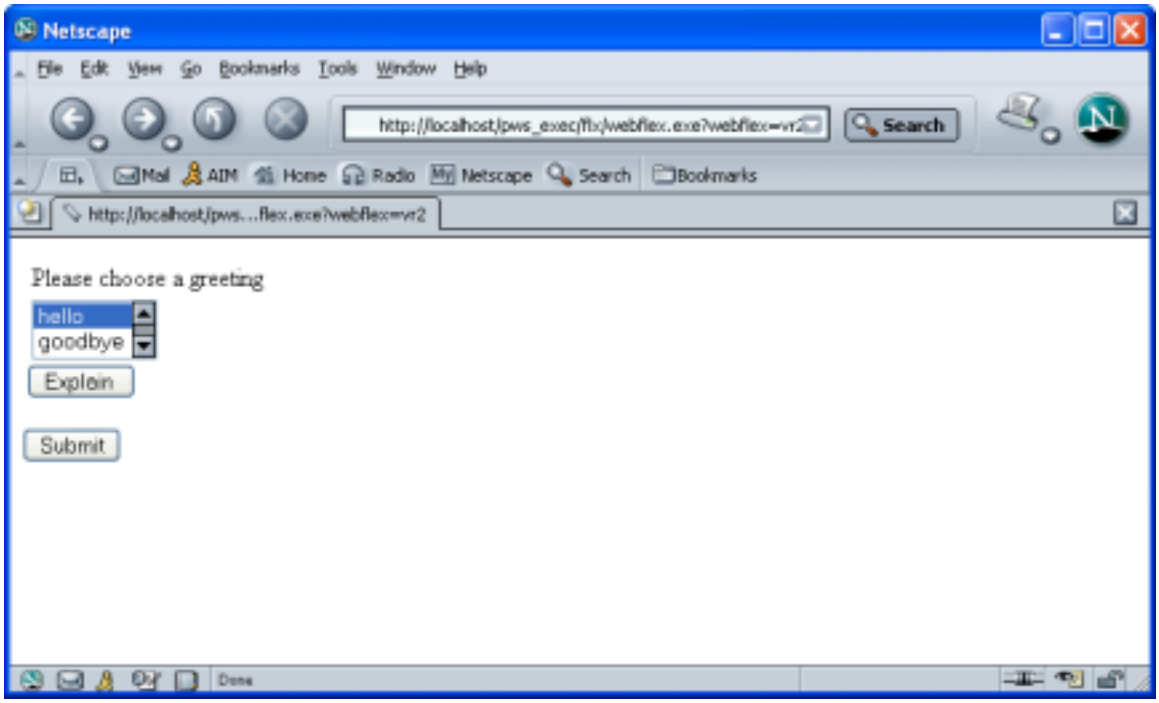
6) We can extend the KSL file with some question styles.

For instance, we can add the following to the KSL file:

```
frame greeting_style ;
  default method is radio .
```




```
frame greeting_style ;
  default rows is 2 .
```



Updating Question Styles

We can use a small Prolog program to update the styles dynamically at runtime.
Let's take the AUTO.VSR chart for example.
Let's save it as KSL.

Now here's some code to define the style frames dynamically.

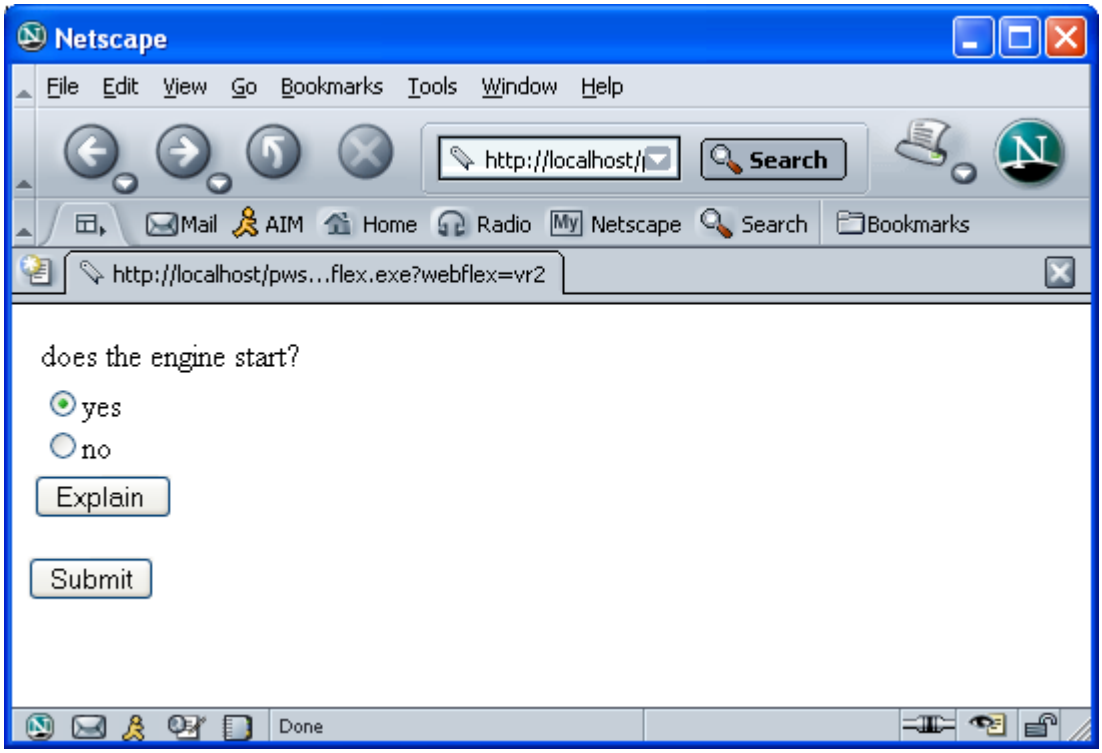
```
go :-
   isa_question( Name, A, L, P),
   L = single( Group ),
   cat([Name,'_style'],FrameName,_),
   new_frame( FrameName, [] ),
   new_default(  method, FrameName, radio ),
   fail .

go.
```

Save this code into a file called, say, update_styles.pl and save it in the SYSTEM folder within the WebFlex installation.
Now, we must add something into the AUTO.KSL file to load this file and call the 'go' program.

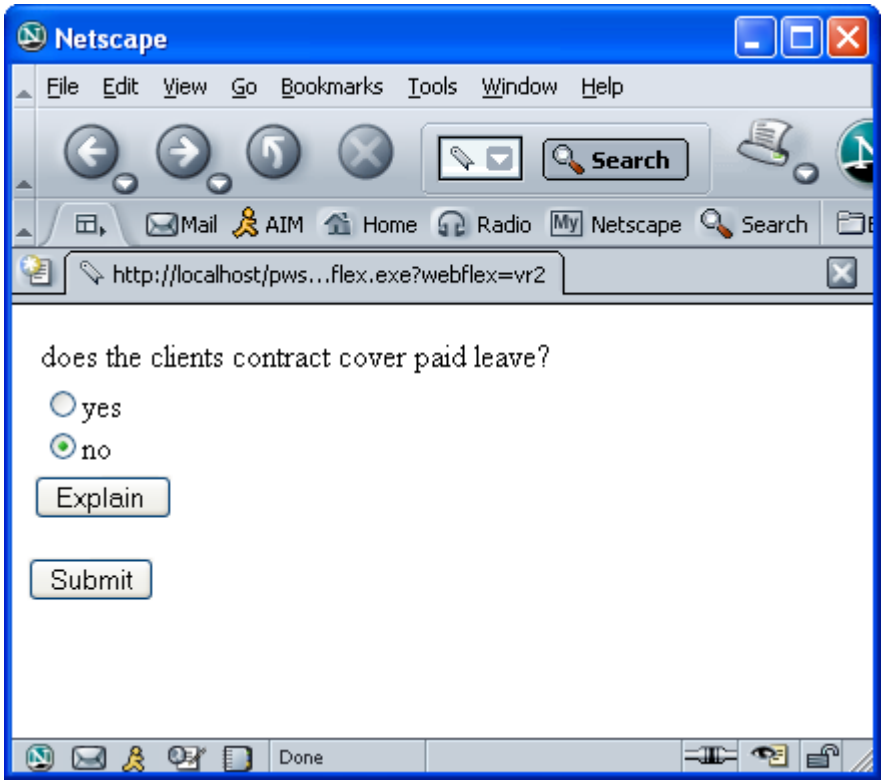
```
do ensure_loaded( system( update_styles) ) .

relation run if
   go and start( Conclusion ) and write( Conclusion ) .
```

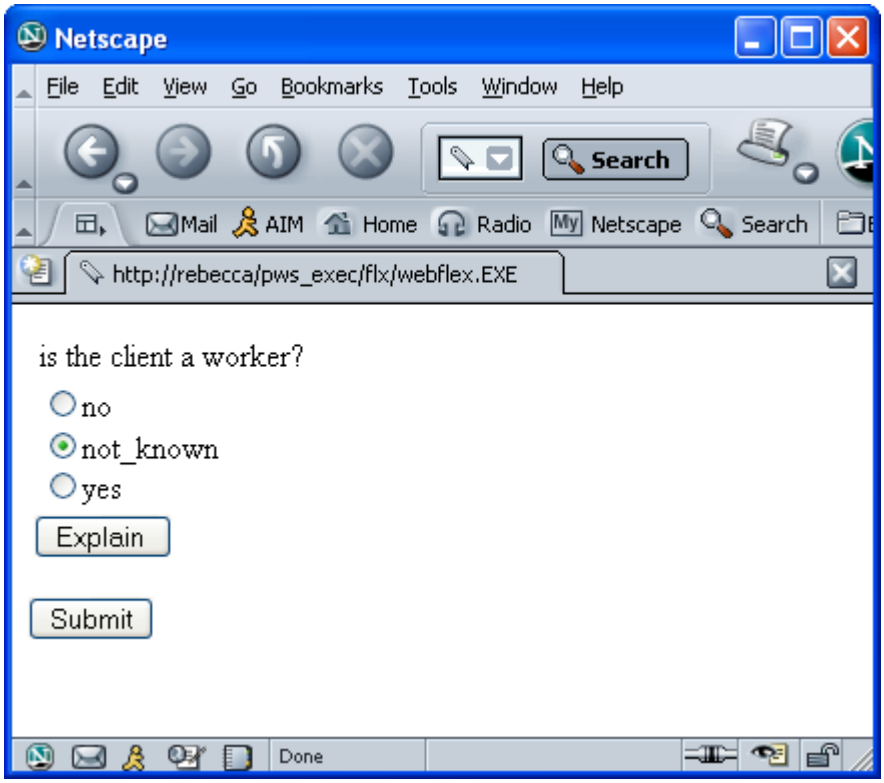
Now either add a new entry into the WebFlex.ini file or save this back into the VR2.KSL file.

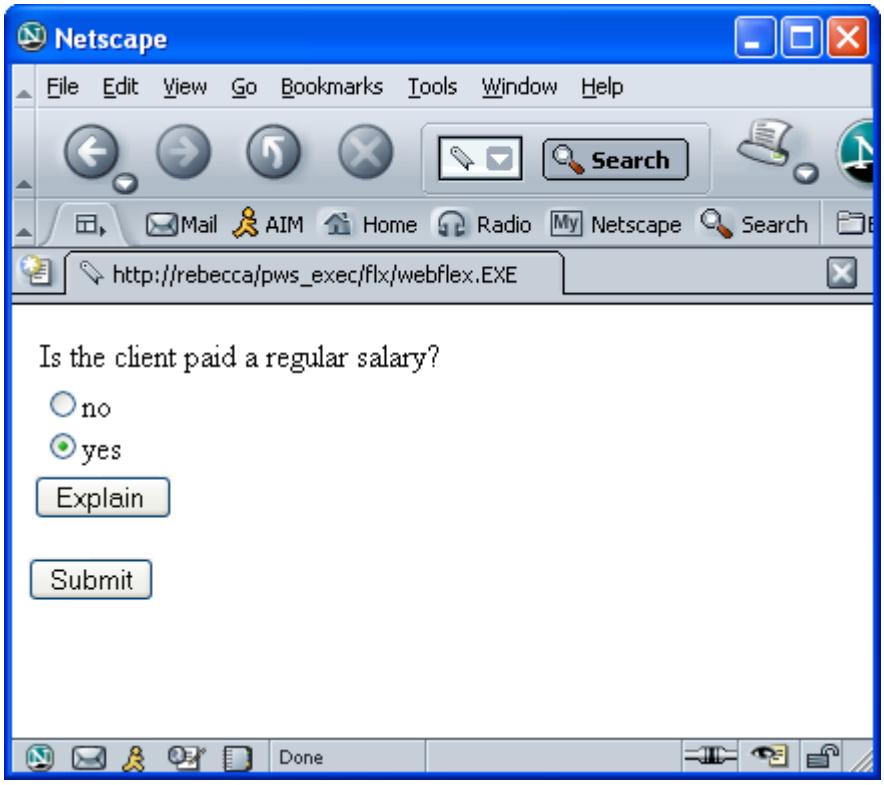
Now when we execute the chart, and all the questions will be presented as radio buttons.



We can use the same utility with the FAB.VSR chart







And we can extend the utility to deal with other question types.

Appendix 1

Creating a New File Type Entry for VisiRule (.VSR) Files

This appendix shows you how to set up Windows so that whenever you double-click on a VisiRule (.VSR) file, it is opened into a VisiRule window within the WIN-PROLOG development environment.

Windows XP

- Double click on the "My Computer" icon. From the window that opens (either the "My Computer" folder or the "Windows Explorer" window) open the 'Tools' menu and select the "Folder Options..." entry.
- From the "Folder Options" dialog that appears, select the "File Types" tab.

Within the Folder Options dialog,

- Click on the "New" button to create a new file type entry for .VSR files.
- When the "Create New Extension" dialog appears, enter "VSR" in the "File Extension:" field and then click OK.
- Within the Folder Options dialog, locate and select the VSR entry in the 'Registered file types:' list. Click on the Advanced button to display the Edit File Type dialog.
- Click on the 'New...' button to display the 'New Action' dialog. Enter into the 'Action:' field something like 'Open With VisiRule' or whatever you want the textual name of the action to be; this will eventually become an entry in the menu that appears when you right-click over a .VSR file.
- Click on the 'Browse' button to locate and select the PRO386W.EXE file. You should now have: **"\program files\win-prolog 4700\pro386w.exe"** in the 'Application used to perform action:' field.
- Click in the 'Application used to perform action:' field and navigate to the end of the line and append the following to the existing text: **ensure_loaded(system(visirule)), system_menu(_,file,open('%1'))**.
- This will load the .VSR file into a VisiRule window within the WIN-PROLOG development environment. Ensure that you leave a space character between the bit in double quotes and the Prolog goal. The bit in double quotes (i.e. **"c:\program files\win-prolog 4700\pro386w.exe"**) is the path to your installation of WIN-PROLOG. The double quotes must be included if spaces are present in the path or file name. The "%1" in "system_menu(_,file,open('%1'))." will get replaced by the full pathname of the .VSR file you later double click on.
- Click the "OK" button. The action you have just added should appear in the 'Actions' list on the Edit File Type dialog. Click OK on the Edit File Type dialog. Click Close on the Folder Options dialog.

Windows ME

- Double click on the "My Computer" icon. From the options that appear, open 'Control Panel' and select the "Folder Options..." entry.
- From the "Folder Options" dialog that appears, select the "File Types" tab.

Within the Folder Options dialog,

- Click on the "New" button to create a new file type entry for .VSR files. Type '.vsr' in the dialog that appears and click on OK.
- Again within the Folder Options dialog, make sure that the file extensions tab is current and select the new .vsr entry in the list box.
- Click on the Change button and click on Other in the dialog which appears. Under File Name type in: **"c:\program files\win-prolog 4700\pro386w.exe" ensure_loaded(system(visirule)), system_menu(_,file,open('%1'))** which will load the .VSR file into a VisiRule window within the WIN-PROLOG development environment.
- Ensure that you leave a space character between the bit in double quotes and the Prolog goal. The bit in double quotes (i.e. "c:\program files\win-prolog 4500\pro386w.exe") is the path to your installation of WIN-PROLOG. The double quotes must be included if spaces are present in the path or file name. The "%1" in "system_menu(_,file,open('%1'))." will get replaced by the full pathname of the .VSR file you later double click on.
- Click on OK and close the folder options dialog.