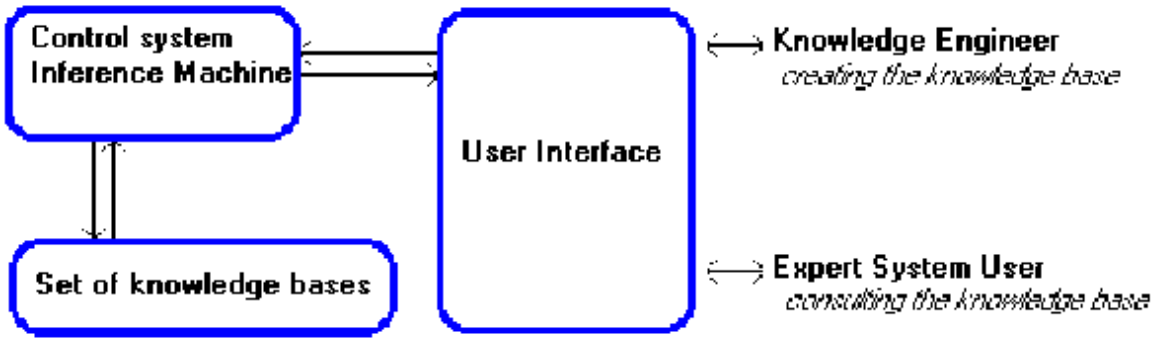
ESTA (**E**xpert **S**ystem Shell for **T**ext **A**nimation)

An expert system is a computer system, which mimics the behavior of a human expert in some specific subject area. The idea behind creating an expert system is that it can enable many people to benefit from the knowledge of one person - the expert. And not only will it make specialist knowledge more readily available, but also it may free the expert for the really difficult problems.
By providing it with a knowledge base for a certain subject area, ESTA can be used to create an expert system for that subject:

ESTA + Knowledge Base = Expert System

Each knowledge base contains rules for a specific domain. Thus, for a car fault diagnosis expert system the knowledge base will contain rules relating certain symptoms, such as excessive engine vibrations or lack of power at high speeds to possible causes. Similary, a knowledge base for an expert system to give taxadvice might contain rules relating marital status, mortgage commitments and age to the advisability (from the point of view of legally paying as little tax as possible) of taking out a newlife insurance policy, since these sometimes alter an individual's tax position beneficially. And likewise, given a plant classification knowledge base, an expert system could be built to recognize plants.



ESTA has all facilities to write the rules that will make up a knowledge base. Further, ESTA has an inference engine, which can use the rules in the knowledge base to determine which advice is to be given to the expert system user or to initiate other actions. ESTA also features the ability for the expert system user to obtain answers to questions such as 'how' and 'why', etc.
The separation of the control and inference parts from the knowledge base is a fundamental feature of an expert system shell:

ESTA is used by a knowledge engineer to create a knowledge base and by theexpert system user to consult a knowledge base.

Knowledge representation in ESTA is based on following items:

· Sections

· Parameters
· Title

To make the knowledge representation easier to read, comments can be included anywhere.

SECTIONS

The top level of knowledge representation in ESTA consists of *sections*. The first section in any knowledge base must be one called `start`. A section consists of a *name*, a *textual description* and a *number of paragraphs*. ESTA deals with the paragraphs in a section working from top to bottom, one paragraph at the time. If a paragraph contains a boolean expression, then the expression is evaluated first. If the expression is true - or the paragraph does not contain a boolean expression at all - the relevant list of actions are executed in the order given in the paragraph. When a consultation is started via the Begin Consultation command, ESTA starts by evaluating the boolean expressions in the paragraphs of the start section.

The syntax for a section is:

```
<section>        ::= section <section-name> [:] <description-
text>
<paragraph-list>
<paragraph-list> ::= <paragraph> [<paragraph-list>]
<paragraph>      ::= [if <boolean-expression>] <action>
                     [if <boolean-expression>] ( <actions> )
<actions>        ::= <action> [,<actions>]
<action>         ::= advice |
                     assign |
                     call |
                     chain |

                     do |
                     do_section_of |
                     exit |
                     stop
```

Example
The following example consists of a knowledge base with one section and one parameter:

```
section start : 'the first section to be executed'
 if car_color = 'red'
   ( advice 'Your car is red, try to sell it to the fire
brigade ' ,

     call sound(200,100)
   )
 if car_color  <> 'red' and car_color <> 'blue'
     advice 'Your car is not one of our favourite colors !'

 advice 'That''s all folks!'
```

```
parameter car_color 'the color of the car'
type text
question 'What is the color of your car ?'
```

The section start contains three paragraphs. The first paragraph contains a list of two actions separated by commas and embedded in parantheses. The next paragraph has only one action in which case the parantheses may be omitted. The last paragraph consists of a single action without any preceding boolean condition.
Notice that evaluation of a boolean expression typically involves the use of parameters - as in `car_color = 'red'` in the above example. If a parameter does not have a value at the time of evaluation, ESTA will first establish a value for it either by questioning the user or by using a rule - depending on the parameters declaration.

BNF Syntax Notation

The *syntax* for the ESTA language is written in the formal description language called BNF. The following symbols are meta symbols with a special meaning as listed below:

`::=` Is interpreted as 'consists of' or 'is defined as'.
`{ }` Curly brackets indicate that the sequence in the brackets may be repeated zero or more times.
`[ ]` Square brackets state that the sequence in the brackets is optional.
`< >` Names embedded in angle brackets are syntactic elements such as `<integers>,<names>` etc. All syntactic names will be defined i.e. they will appear on the left side of the `'::='`

symbol in a grammar rule.
| A vertical bar is read 'or' and is used to separate alternatives.

Example

```
<symbol> ::= <letter>{<letter>|<letter>|<digit>|_}

<letter> ::= a|b|c|...|x|y|z
             A|B|C|...|X|Y|Z

<digit>  ::= 0|1|2|3|4|5|6|7|8|9
```

Name

A *name* consists of letters, digits and underscore characters. The first character must be a letter. The syntax for a name is:

```
<name> ::= <letter> {<letter>| <letter> | <digit> | _ }
```

Example

```
car  michael a38 another_name_with_1_digit
```

The following names are pre-defined keywords and must not be employed as user-defined identifiers:

abs	cos	log	save_values
and	description	ln	section
advice	display	max	showpic
arctan	do	min	sin
assign	do_section_of	not	sound
boolean	ending	number	system
call	endstr	or	sqrt
category	exit	parameter	startstr
cf_add	exp	picture	strlen
cf_and	explanation	power	tan
cf_not	false	question	text
cf_or	hyperadvice	quit	true
chain	if	range	Trunc
clear_all	int	restore_values	type
clear_value	known	rules	unknown
concat			

String

Strings are a number of characters between two quotes. A string may itself contain a qoute, in which case this is indicated by a sequence of two quotes. Strings may contain newlines. The syntax for a string is:

```
<string> ::= ' { <char> | '' } '
<char>   ::= any printable character at the keyboard
```

Examples

```
'this is a string'
'this is
 also
 a
 string with a single quote '' and newlines'
```

Boolean Expression

Boolean expressions - or *conditions* - guide the logic of ESTA's dialog. They are used as part of if-statements in section declarations and rules fields of parameters. Boolean expressions are *true*, *false* or *unknown*. They consist of:

- boolean parameters
- comparisons between number expressions
- comparisons between names and strings

· compound boolean expressions

Comparisons between number expressions or between names and strings may be written using one of the *relational operators*:

< smaller than
<= smaller than or equal to
= equal to
> greater than
>= greater than or equal to
<> different from

When strings are compared, the characters of the two strings are compared one by one by replacing the characters by their ASCII values. Thus, the comparison 'A' < 'B' is true because 65 (the ASCII value of 'A') is less than 66 (the ASCII value of 'B'). The comparison goes from left to right until a pair of letters gives a result.

Compound boolean expressions are formed using the logical operators: and, or and not. Not has the highest priority of the logical operators - actually even higher than the mathematical operators. On the other hand, and and or have lower priority than the mathematical operators. As in number expressions parantheses may be used to implement any order of evaluation.

The syntax for a boolean expression is:

```
<boolean-expression>   ::=
    <number-expression> <relational-op> <number-expression> |
    <text-expression> <relational-op> <text-expression> |
    <boolean-expression> and <boolean-expression> |
    <boolean-expression> or <boolean-expression> |
    not [(] <boolean-expression> [)] |
    true |
    false |
    unknown |
    <parameter-name> |
    known(<parameter-name>)

<relational-op>        ::= <  |  >  |  <=  |  >=  |  =  |  <>
```

The function known returns true if the parameter has a value, otherwise it returns false.

Examples

```
number_of_days_required / 7 < weeks_required
favourite_colour = 'red'
not pays_highest_rate_tax
has_retired or is_unwell
date_of_birth < 1960 and owns_a_car

x and not y or z    /* (x and not(Y)) or z) */
```

Advice

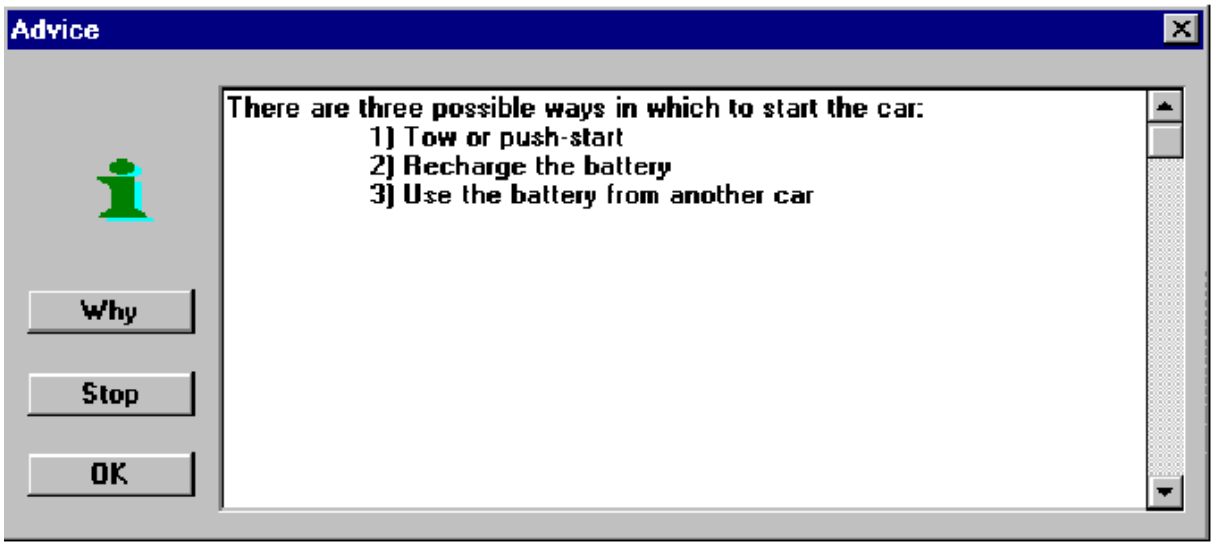
The *advice* action is executed by first establishing the values of all parameters involved in the text expression and then writing the resulting text in a window on the screen. Pictures can be included anywhere in the advice and thereby be shown on the screen as part of the advice. The syntax for an advice is:

```
<advice>        ::= advice <specification>
<specification> ::= <advice-item> {<specification>}
<advice-item>   ::= text-expression | picture <picture-name>
```

Examples

```
advice
'There are three possible ways in which to start the car:
        1) Tow or push-start
        2) Recharge the battery
        3) Use the battery from another car'
```

The advice will look like this in a consultation:



Assign Action

The *assign action* is used to assign values to parameters. The syntax for the assign action is:

```
<assign> ::= assign <parameter-name> := <expression>
```

To be well formed, the `<expression>` must result in a value that conforms to the type of the parameter.

NOTE
Readers familiar with AI techniques may already have realized that use of assign makes it possible to combine forward chaining capabilities of sections with the backward chaining capabilities in the parameter rules.

Examples

```
section start 'a very short section to illustrate assign'
    assign n := 7 + 8 / 2
    advice 'evaluation of 7 + 8 / 2 gives ' n
    assign n := (7 + 8) / 2
    advice 'evaluation of (7 + 8) / 2 gives ' n
           'as / has higher priority than +.' &
           'Notice that the parameter indeed did change the
value'

parameter n 'n'
type number
```

Note the ESTA always begins evaluation from the start section and further on. Thus if you for instance change the value of parameter 'n' by choosing the menu item "What if parameter changes its value" in the Parameter menu during a consultation, this changed value will always be overwritten by the assignments in the start section, meaning that the what-if change command would not have any effect.

Call Action

The *call action* is used to call one of the built-in or user-defined procedures in ESTA. The arguments to the procedures are allowed to be expressions as long as they correspond to the type of the argument. The syntax for the call action is as follows, illustrated with the built-in procedures:

```
<call> ::= call clear_all() |
           call clear_value(<parameter_name>) |
           call display(<filename>) |
           call hyperadvice(<filename>,<name>) |
           call restore_values(<filename>) |
           call save_values(<filename>) |
           call showpic(<picture_name>) |
           call sound(<Duration>,<Frequency>) |
           call system(<string>)
```

CLEAR_ALL Procedure

The built-in procedure : `clear_all()` clears all parameter values in the current knowledge base - in other words returns them to the unevaluated state.

CLEAR_VALUE Procedure

The built-in procedure: `clear_value(<parameter-name>)` clears a parameter value in the current knowledge base - in other words it returns it to its unevaluated state. The next time the value of the parameter is required it is reevaluated.

DISPLAY Procedure

The built-in procedure: `display(<filename>)` displays the contents of a text file in a window on the screen. The file may contain more text than can be accomodated in the window, in which case the expert system user can browse through the text using the scrollbar.

Examples
The file inform.txt from current directory will be shown in a window:

```
call display('inform.txt')
```

HYPERADVICE Procedure

From ESTA it is possible to activate a specific node in a Windows helpfile. The node will be shown as a topic in the Windows hypertext help system. The procedure is called like this:

```
hyperadvice(<filename>,<node_specification>)
<node_specification>  ::= <number>
```

The node_specification is the context number of the node you want to see.

Examples

```
section brakes 'problems with the brakes'
if brake_long_travel call hyperadvice('hypercar.hlp',1)
if brake_poor call hyperadvice('hypercar.hlp',2)
if brake_veer call hyperadvice('hypercar.hlp',3)
```

You can see the example as a whole in CAR.KB. Choose brakes as the answer when asked which kind of problem, your car has. Then you can try out an example of this kind of hypertext integration.

RESTORE_VALUES Procedure

The built-in procedure `restore_values(<filename>)` restores the ESTA parameters to settings previously saved in a data file (See save_values). The single argument is the full file name of this datafile. This may be used in several situations: when a standard set of parameter values are known in advance; if an external program, called from ESTA via the system procedure, needs to communicate with ESTA; or in combination with chaining from one knowledge base to another.

Examples

```
call restore_values('c:\esta\setup.dat')
```

SAVE_VALUES Procedure

The built-in procedure `save_values(<filename>)` saves in a datafile the settings of all parameters currently bound to a value. The single argument is the full filename of this datafile. Its typical use is described under restore_values. A data file generated by save_values can be inspected to discover in which format external programs must write parameter settings for ESTA. For instance if the external program will terminate by defing a number of parameter values, and let ESTA decide the remaining relevant parameters based on rules and/or a dialog with the user.

Examples

```
call save_values('c:\esta\setup.dat')
```

SHOWPIC Procedure

The built-in procedure `showpic(<picture-name>)` displays a picture from the picture database in a window.

Examples

```
call showpic('cars')
```

SOUND Procedure

The built-in procedure `sound(Duration,Frequency)` generates a sound through the computers loudspeaker lasting the given duration and with the given frequency.

Examples

```
sound(10,100)
```

SYSTEM Procedure

The built-in procedure `system(command_string)` is used to call other applications from ESTA. This can either be a DOS or Windows application.

Examples

For instance, a program in the file myprog.exe can be executed from within ESTA using the procedure call:

```
system("myprog.exe")
```

Chain Action

The *chain action* enables ESTA to consult new knowledge bases. If the knowledge can be structured so that it is possible to divide it up into several smaller knowledge bases, it will enhance the overview of knowledge. The syntax for the chain action is:

```
<chain> ::= chain <filename>
```

Parameter values may be transferred between two knowledge bases by calling save_values before chaining and let the first action in the new knowledge base be a call of restore_values.

Note that only parameter values can be transferred between knowledge bases. The definitions get lost when a knowledge base is chained.

If the filename parameter is an empty text, eg, chain '', ESTA will chain to the current knowledge base, in effect restarting it.

Examples
The action `chain 'car.kb'` starts a consultation of the knowledge base `car.kb`. Notice that the new knowledge base will be loaded into memory and that the 'old' knowledge base thereby will be lost.

Do Action

The *do* action simply transfers control to a new section. The do action enables 'data-driven' search to be carried out between sections. In the AI litterature this is also referred to as 'forward chaining'. The syntax for the do action is:

```
<do> ::= do <section-name>
```

Examples

```
section start 'first section to be executed'
   if answer = 'yes' (do positive_section, do next_section)
   if answer = 'no' do negative_section
```

Depending on the value of the parameter answer, control is transferred to either `positive_section` followed by `next_section`, if the answer was yes; or to the `negative_section`, if the answer was no.

Do_Section_of Action

The action *do_section_of* takes one argument: a parameter of the type category. The action is an alternative to the often used sentences in section declarations:

```
if <category parameter name> = <option1> do <option1>
if <category parameter name> = <option2> do <option2>
...
```

Instead of listing all the if-statements corresponding to the possible options, you can simply write:

```
do_section_of <category parameter name>
```

In a consultation session the value of the parameter will determine to which section, the control will be transferred. If the parameter hasn't got a value, ESTA evaluates the parameter first and the obtained value is then used as reference to the corresponding section. The functionality of the `do_section_of` and the `do` action is exactly the same. The difference is that you don't have to write all the equal sentences with the do-statements, but instead only one sentence. Remember that the option names of the category parameter must correspond to section names in the knowledge base.

Examples
This example is taken from the supplied knowledge base CAR.KB:

```
section start 'main section'
do_section_of problem

parameter problem : 'the problem with your car'
type category
explanation
'Identify the problem with your car as closely as you can.
Only those listed are dealt with in this knowledge base.'
options
starting_problems - 'starting problems'
overheating - 'the engine gets overheated'
smell_of_gasoline - 'the car smells of gasoline'
bad_running - 'the engine runs badly'
brakes - 'the brakes'
vibration - 'excessive vibrations'

wiper_motor - 'the wiper motors'
light_problem - 'the lights'
horn_problem - 'a faulty horn'.

question
'What is the problem with your ' car ' ?'

picture 'car'
```

Exit Action

The *exit* action simply terminates consultation of the current knowledge base. The syntax is:

```
<exit> ::= exit
```

Stop Action

The *stop* action may be used to optimize rules written in a section. Execution of a stop action indicates that no more actions are to be executed or boolean expressions to be evaluated in the containing section. The stop action helps the knowledge engineer to avoid repeating complicated conditions.
The syntax for stop is simply:

```
<stop> ::= stop
```

Examples
A simple example using stop:

```
section start 'a section without any action'
stop
```

```
advice 'This advice will never be given due to the stop
action'
```

An example showing two sections, one with stop and one without, which operate identically:

```
section start1 'start section without stop'
if p1 and p2 and p3 and p4 and p5 and p6 and p7
   advice 'realizing that p1 - p7 is true , this advice is
given'
if not (p1 and p2 and p3 and p4 and p5 and p6 and p7)
   advice 'this second advice is given since one of p1 - p7
was false'

section start2 'start section with stop'
if p1 and p2 and p3 and p4 and p5 and p6 and p7
   (advice 'realizing that p1 - p7 is true , this advice is
given',
   stop)
advice 'this second advice is given since one of p1 - p7 was
false'
```

Comands for Sections' manipulating

- **New Section**

This command is used to create a new section. When activated ESTA prompts for a name. An editor window with the keyword section followed by the name will then appear on the screen. The definition of the section can be entered using the editor and insert facilities.

- **Edit Section**

This command allows you to edit an existing section. When the menu option is selected, a list box is displayed containing all existing section names in alphabetical order. After selecting a section, ESTA opens an editor window with the text defining the selected section, which may then be changed as required.

- **Update Section**

This command updates a section in the active knowledge base in memory. ESTA automatically checks that the syntax for the section is correct. If an error is detected, a dialog box containing the error message will appear. Once the dialog box is closed, the caret will be positioned in the editor window at the detected error position. In the middle status area at the bottom of the main window, the error message can be reviewed. Correct the error and try to update again.

If no errors are detected, the edited version of the section is then updated in the current knowledge base in memory. To save the current knowledge base into a file, choose Save in the file menu.
Note

After updating, the editor window will still remain on the screen. To close it, choose Close in the system menu of the window.

- **Delete Section**

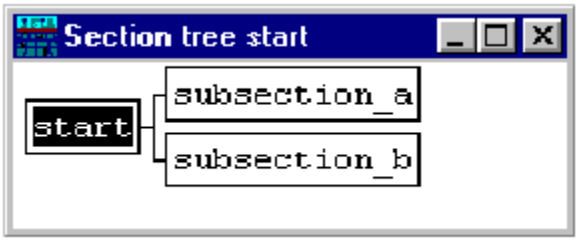
This command deletes a section from the current knowledge base in memory, after presenting a dialog box allowing you to confirm your intention to delete it.

- **Draw Section Tree**

ESTA features a graphical presentation of sections. This command will draw a tree for the individual section calls starting at a given section. By means of illustration, consider the following knowledge base:

```
section start : ' the start section '
do  subsection_a
do  subsection_b

section subsection_a : ' subsection a '
advice 'This is subsection a'

section subsection_b  ' subsection b '
advice 'This is subsection b'
```

A section tree starting with 'start' would be displayed as follows:



The generated tree is displayed in a window, which can be. You can use the mouse to move around and scroll to see all parts of the tree. Double-clicking in a section box will activate an editor window with the section definition. The tree will be updated dynamically whenever changes are made to the involved sections.
By default the maximum level for a tree is equal to 10. You can change the level in the file "winesta.def".

- **List All Sections**

This command lists all sections in the current knowledge base. The name and the description for each section will be listed. It may be useful to combine the report-facilities with the list feature for documentation purposes (See Print). The list will be updated dynamically whenever changes are made to any of the involved sections.

Parameter

Parameters are like variables which determine the flow of control between the sections. Any parameter consists of a declaration field and a type field. Further a number of optional fields, depending of the parameter type, may be used to describe the parameter. A parameter can be one of the following four types:

- boolean parameter
- text parameter
- number parameter
- category parameter

Any parameter can obtain a value in one of the following ways:

- from an answer to a question
- as a result of following some rules
- an assignment resulting from an assign action

Examples

```
parameter name 'the name of the user'
type text
question 'What is your name ?'

parameter pc_colour 'The user''s favourite screen colour'
type category
options
  red,
  blue,
  green.
question 'Which screen colour do you prefer when using an
editor ?'
picture 'colour_screen_picture'

parameter colour 'a word describing the colour'
type text
explanation
'Humans normally use words to describe colours rather than
specifying the frequency of the light wave. This parameter
represents colours as red, blue, etc.'

rules
  'blue' if frequency < 1000,
  'red'  if frequency > 2000 and frequency < 3000,
  'invisible'.
```

Declaration Field

The first line of a parameter *declaration* takes the form of the keyword *parameter*, the *name* of the parameter, optionally followed by a *colon*, : and then some *text* describing the parameter. The syntax for the declaration field is:

```
<declaration field> ::=  parameter <parameter-name> [:]
<description-text>
```

The description text is used to generate replies when the expert system user employs ESTA's facility to ask HOW a certain value was established or WHY a certain piece of advice was given. Thus, the person responsible for writing a knowledge base should take care that the descriptions read well given the sentence-schemes for explanations automatically generated by ESTA.

For *boolean* parameters the ESTA phrase is:
The objective is to find out whether `<description-text>`

For *number*, *text* or *category* parameters the ESTA phrase is:
The objective is to establish the value of `<description-text>`

Examples

Using the following parameter declarations:

```
parameter owns_car 'you own a car'
type boolean
parameter height_cm 'your height in cm'
type number
```

ESTA would generate the following explanations on command from the user:

```
The objective is to find out whether you own a car
The objective is to establish the value of your height in cm
```

Boolean Parameter

Boolean or *logical* parameters are used when a parameter is restricted to one of the values true, false or unknown - i.e. when the answer to a question is to be either *Yes*, *No* or *Unknown*. By default ESTA automatically generates a listbox corresponding to the values: true, false and unknown. Because of the special use of the value unknown and because in some boolean expressions the unknown value is not appropriate, it is possible to declare a listbox where unknown is omitted. This kind of listbox declaration can be done in an options field attached to a category parameter.

The syntax for a boolean parameter is:

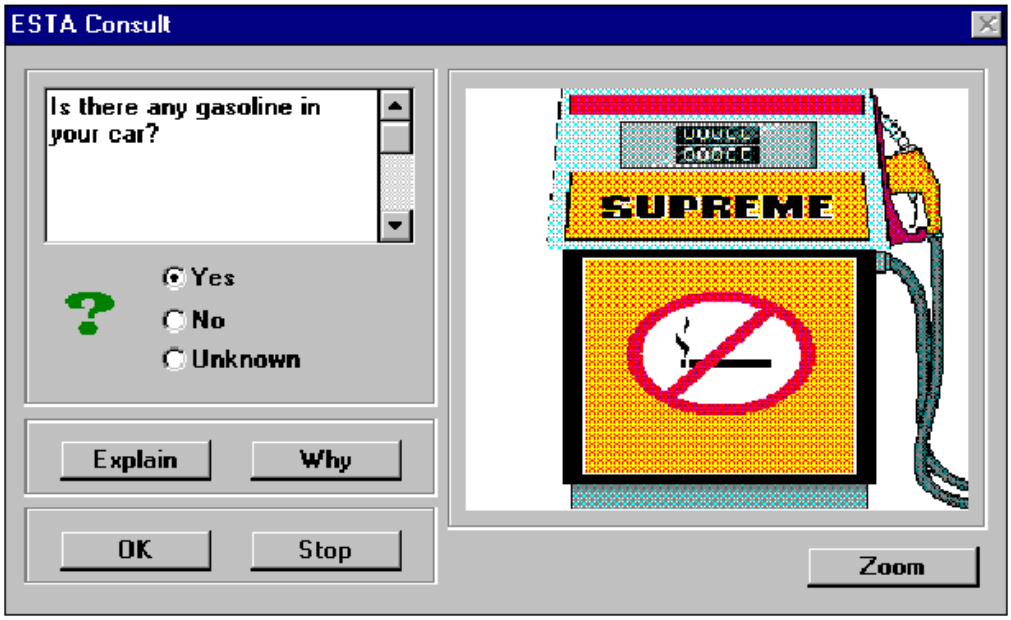
```
<boolean parameter> ::=
             <declaration field>
             type boolean
             [<explanation field>]
             [<rules field (with boolean expressions)>]
```

```
            [<question field>]
            [<picture field>]
```

Examples

```
parameter gasoline_ok : 'there is gasoline in your car'
type boolean
explanation 'Turn on the ignition and look at the fuel gauge'
question 'Is there any gasoline in your car ?'
picture 'gasoline'
```

The parameter gasoline_ok will look like this in a dialog:




```
parameter wet_weather 'the weather is wet'
type boolean
explanation 'It has been raining within the last few days or
it is foggy'
question 'Is the weather wet ?'
```

The parameter `wet_weather` will look like this in a dialog:

Explanation Field

The value of a parameter may be determined by the answer to a question. When such a question is posed during consultation, the expert system user will have the option to ask ESTA to 'explain the question'. This explanation must be provided by the knowledge engineer in the explanation field. This field consists of the keyword explanation followed by an explanatory text. The syntax for the explanation field is:

```
<explanation field> ::= explanation <text expression>
```

Examples

```
parameter marital_status 'the people are married'
type category
explanation
   'Marital status is determined by whether' &
   'there exists a marriage certificate for' &
    name ' and ' name_of_partner ' or not'
options
   married,
   unmarried.
question 'What is the marital status of ' name ' ?'
```

Notice that the explanatory text may contain parameters already associated with a value as well as parameters currently unevaluated. In the latter case ESTA will evaluate the parameter first and then use its value as part of the explanation.

Rules Field

The *rules* field is used when a parameter value should be determined by rules and not by a question. The rules field has the following syntax :

```
<rules field>     ::=  rules  <parameter rules> .
<parameter rules> ::=  <parameter rule> {,<parameter rules>}
<parameter rule>  ::=  <expression> [if <boolean-expression>]
```

For instance consider a rules field like this:

```
rules
  <expression> if <boolean-expression>,
  <expression> if <boolean-expression>,
  <expression> if <boolean-expression>,
  <expression> .
```

The parameter will in this case receive the value of the first `<expression>` if the first `<boolean-expression>` is satisfied, and so on. Notice that the optional last `<expression>` has no corresponding `<boolean-expression>`. It is a catch all `<expression>` which is used to determine a value for the given parameter if none of the `<boolean-expression>`s are fullfilled.

Note that all the rules must be wellformed and thus, when evaluated, every `<expression>` must give a value which conforms to the type of the parameter.
Note also that a parameter with a rules field may NOT contain a question field. When there is a rules field, any existing explanation field can never be accessed by the expert system user and it is therefore only present for the convenience of the knowledge engineer.

Examples

```
  rules
      'blue'       if frequency < 1000,
      'orange'     if frequency >= 1000 and frequency < 2000,
      'red'        if frequency > 2000 and frequency < 3000,
      'invisible'.

  rules
      (height * (length + width)) * 2 if shape = box,
       height * width * width / 2     if shape = pyramid,
       4 / 3 * 22 * radius * radius   if shape = ball.
```

Question Field

The *question field* is used when the value of a parameter should be determined by an answer to a question. The syntax for the question field is:

```
<question field> ::= question <text expression>
```

Notice that ESTA does not automatically add a question mark, so the text itself must contain one if you wish it to be grammatically correct. If a parameter is used as part of the question text, then the value of that parameter will be established before the question is posed.
If the question text is omitted, ESTA will generate a default text, based on the description text from the declaration field, according to the type of the parameter. For boolean parameters the default is:

```
Is it true that <description-text> ?
```

For text, number or category parameters, the default is:

```
What is the value of <description-text> ?
```

Examples

```
parameter name 'the name of the user'
type text
question 'What is your name ?'

parameter owns_car 'owns a car'
type boolean
question 'Does ' name ' own a car ?'
```

In the last example, the parameter name will be evaluated, if no value is attached. If name already got a value, this value will be inserted in the question text.

Picture Field

The *picture field* is used to specify a picture, that will be showed in connection with the parameter. The picture can be specified by a name from the picture database in qoutes or by a parametername without quotes. In the latter case the value of the specified parameter determines the picture, that will be showed. The parameter value must be identical to a name in the picture database in order to show the picture.

The syntax for the picture field is:

```
<picture field> ::= picture <parameter name> | '<picture
name>'
```

Examples

```
picture car
picture 'sedan'
```

Text Parameter

Text parameters are used for text objects such as a person's name, a favourite colour, etc. The syntax for a text parameter is:

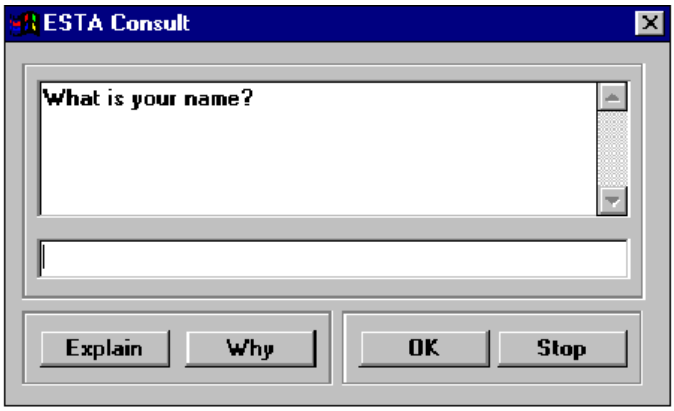
```
<text parameter> ::=
         <declaration field>
         type text
         [<explanation field>]
         [<rules field (with text expressions)>]
         [<question field>]
         [<picture field>]
```

If the text parameter gets its value from an answer to a question, ESTA will show a dialog box, with the question and an edit line for the answer.

Examples

```
parameter name 'the name of the user'
type text
question 'What is your name ?'
```

The parameter name looks like this in a dialog:



Number Parameter

Number parameters may take integer or real values - ESTA converts all values to reals automatically. Number parameters can be used to represent tax rates, interest rates, or the number of parts in a component etc. It is possible to declare a range, that ESTA will validate in response to an assignment of the parameter value.
The syntax for a number parameter is:

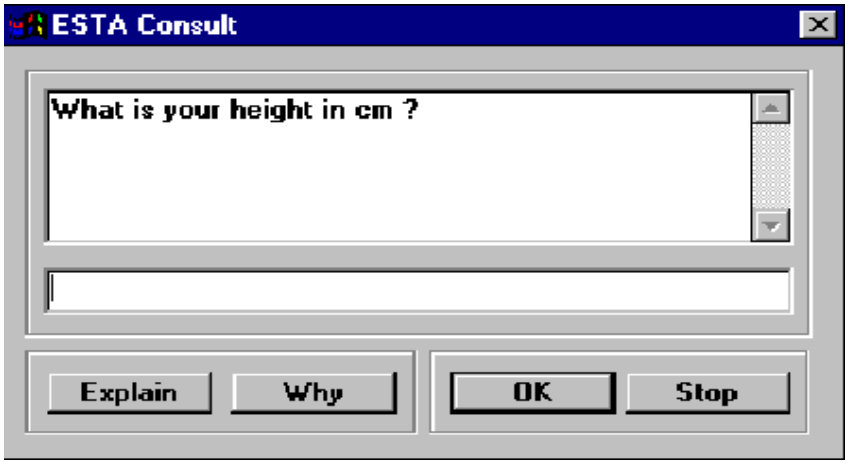
```
<number parameter> ::=
            <declaration field>
            type number
            [<explanation field>]
            [<rules field (with number expressions)>]
            <range field>
            [<question field>]
            [<picture field>]

<range field> ::= range <number> <number>
```

If the number parameter gets its value from an answer to a question, ESTA will show a dialog box, with the question and an edit line for the answer. The input value will be checked automatically for being a number. If a range field is specified, the value will further be checked as being within the range.

Examples

```
parameter height_cm 'your height in cm'
type number
range 100 250
question 'What is your height in cm ?'
```

The parameter will look like this in a dialog:



Category Parameter

A *category parameter* is used, when the parameter is known to take one of a predefined set of values. For instance, it may be decided that color must be one of red, green, blue or purple. The available options are described in the options field. After giving the keyword options, members of the list of options are given separated by commas and terminated by a period. An option is defined by a name and optionally by an explanatory text. Note that ESTA converts everything to lower case before displaying the automatically generated listbox. To avoid unexpected box entries it is therefore a good idea to use lower case letters for the individual entries in an options field.

The syntax for a category parameter is:

```
<category parameter> ::=
            <declaration field>
            type category
            [<explanation field>]
            <options field>
            [<rules field (with text expressions)>]
            [<question field>]
            [<picture field>]

<options field> ::= options <name> [ - <string>] {,<name> [ -
<string>]}.
```

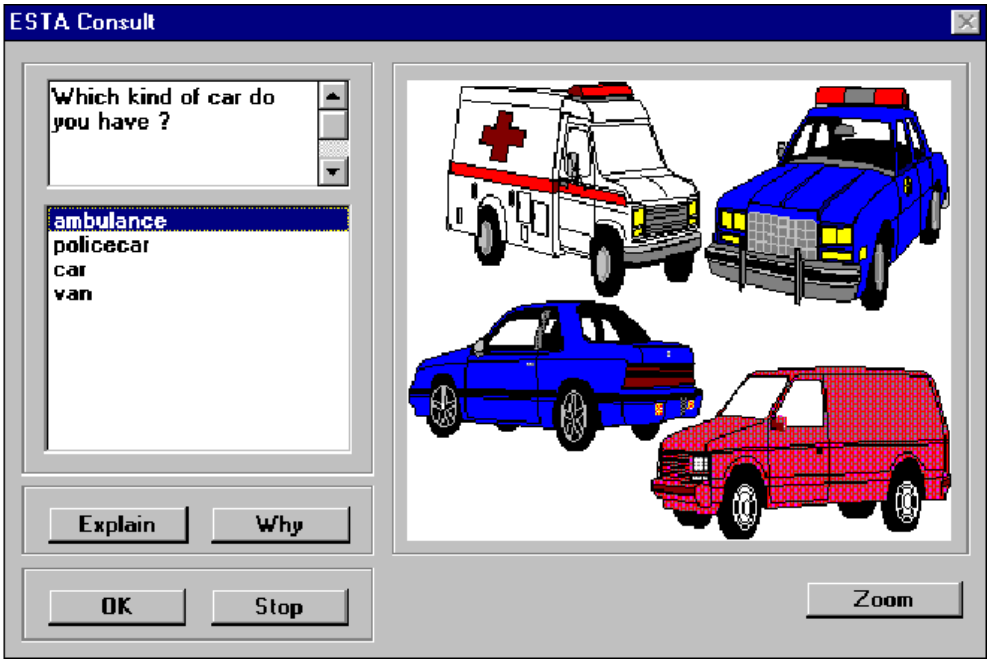
If the category parameter gets its value from an answer to a question, ESTA will show a dialog box, with the question and a listbox with the option names or the explanatory texts, if these are given. The user then selects the answer from the listbox. As an alternative, the user can select an option by clicking on a field in a picture. To do this, the fields in the picture must be defined and linked to the options using the hotspot editor, which is accessed via the menu command Pictures Database.

Examples

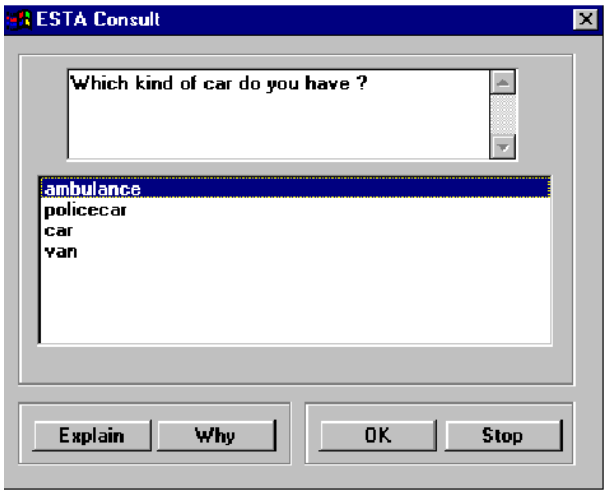
```
parameter car 'the kind of car'
```

```
type category
explanation
'Identify your car with one of the listed types as closely as
you can'
option
     ambulance
     policecar
     sedan - 'car'
     van.
question 'Which kind of car do you have ?'
picture 'cars'
```

This parameter looks like this in a dialog:



Without the picture, the parameter would look like this:



Comands for Parameters' manipulation

- **New Parameter**

This command is used to create a new parameter. When activated, ESTA prompts for a name and a type. An editor window, with a prefilled schema for the parameter, will appear on the screen. Using the editor and the insert facilities, the definition of the parameter can be entered.

- **Edit Parameter**

This command allows you to edit an existing parameter. When the menu option is selected, an alphabetized list box of all existing parameter names will be showed. After selecting a parameter, ESTA will open an editor window with the text defining the parameter, which may then be changed as required.

- **Update Parameter**

This command updates a parameter in the active knowledge base in memory. ESTA automatically checks that the syntax for the parameter is correct. If an error occurs, a dialog box with the error message will appear. When you close the dialog box, the caret will be positioned in the editor window at the detected error position. At this point the error message is displayed in the middle status area at the bottom of the main window. Correct the error and try to update again.

If no errors are detected, the edited version of the parameter is then updated in the current knowledge base in memory. To save the current knowledge base into a file, choose Save in the file menu.
Note
After saving, the editor window will still remain on the screen. To close it, choose close in the system menu of the window.

- **Delete Parameter**

This command deletes a parameter from the current knowledge base in memory. A dialog box confirms that you really wish to do so prior to deleting it.

- **Show Current Parameter**

The command shows the definition of the current parameter in a consultation session in a window.

- **What if - parameter changes its value**

This option provides a direct value-input for parameters. It is typically used after a session is completed to change the value of a parameter, which was previously given an incorrect value. In this case, any advice depending on the specified parameter will be withdrawn.

Note
Only parameters determined by questions can be changed in this way.

- **Draw Parameter Tree**

ESTA features a graphical presentation of parameters. Since the values of parameters may be determined using rules, one parameter may depend on a number of others, which themselves may rely on new parameters, etc. These interrelationships can be displayed as a tree for any given parameter.
As an example the boolean parameter `nice_parameter` defined below will be displayed as shown:

```
parameter nice_parameter  'it is a nice parameter'
type boolean
rules
true if nice_name and letter_count < 20  or easy_to_use.
```

The generated tree is displayed in a window. Use the mouse to move around and scroll to see all parts of the tree. Double-clicking on a parameter box will activate an editor window with the parameter definition. The tree will be updated dynamically whenever changes are made to the involved parameters.

- **List All Parameters**

This command lists all parameters in the current knowledge base. The name, the description and the value for each parameter will be listed. If the parameter doesn't have a value a question mark will be written in place of the value. Ypu may find it useful to combine the report-facilities with the list feature for documentation (See print). The list will be updated dynamically whenever changes are made to the involved parameters.

- **List Parameters with a Value**

This command lists parameters which currently have an assigned value in a window. The list provides a snapshot of the state of the consultation session.

Title

To represent the whole knowledge base a *title* can be used. The title can either be plain text or a picture. The syntax for a title is:

```
<title> ::= text | *<picture-name>*
```

Examples

```
* cars *   /* The picture of cars will be shown as the title
*/
```

```
Car knowledge base  /* The text: Car knowledge base will be
shown */
```

Comments

Comments must start with the /* sequence and end with the */ sequence. Comments may be several lines long.

Examples

```
/* These two lines
are shown so that further comments are superfluous */
```

Pictures in ESTA

ESTA supports *pictures* of the following types: Windows metafiles, device dependent bitmaps(DDB) and device independent bitmaps(DIB).
Pictures are stored in a *pictures database*, called **PICTURES.DBA**. The name of the picture is used as reference, thus when a picture has to be included as part of the expert system, the name of the picture is used. The database can be maintained with the common used database functions like **Add**, **Edit** and **Delete**. You can add a picture either by importing it from a file or by pasting it from the Clipboard. All DDBs, all DIBs and metafiles larger than 64 KB are stored in readonly files. The file has the extension .BMP if the picture is a bitmap(DDBs are converted to DIBs) or .EMF if the picture is a metafile. Using vertical and horizontal scrollbars you can specify where to place the picture in the window. In addition you can specify whether you want the picture to fit in any size of a window or you want to keep the size of the picture, regardless of the window size. Note that pictures are only stretched up to a size, where the original proportions can be maintained. The ESTA system also includes a hotspot editor allowing you to specify selectable subfields in a picture. This feature is used in conjunction with category parameters, in which the options correspond to the fields in the attached picture. The expert system user is thereby able to select an option by clicking on a field in a picture.

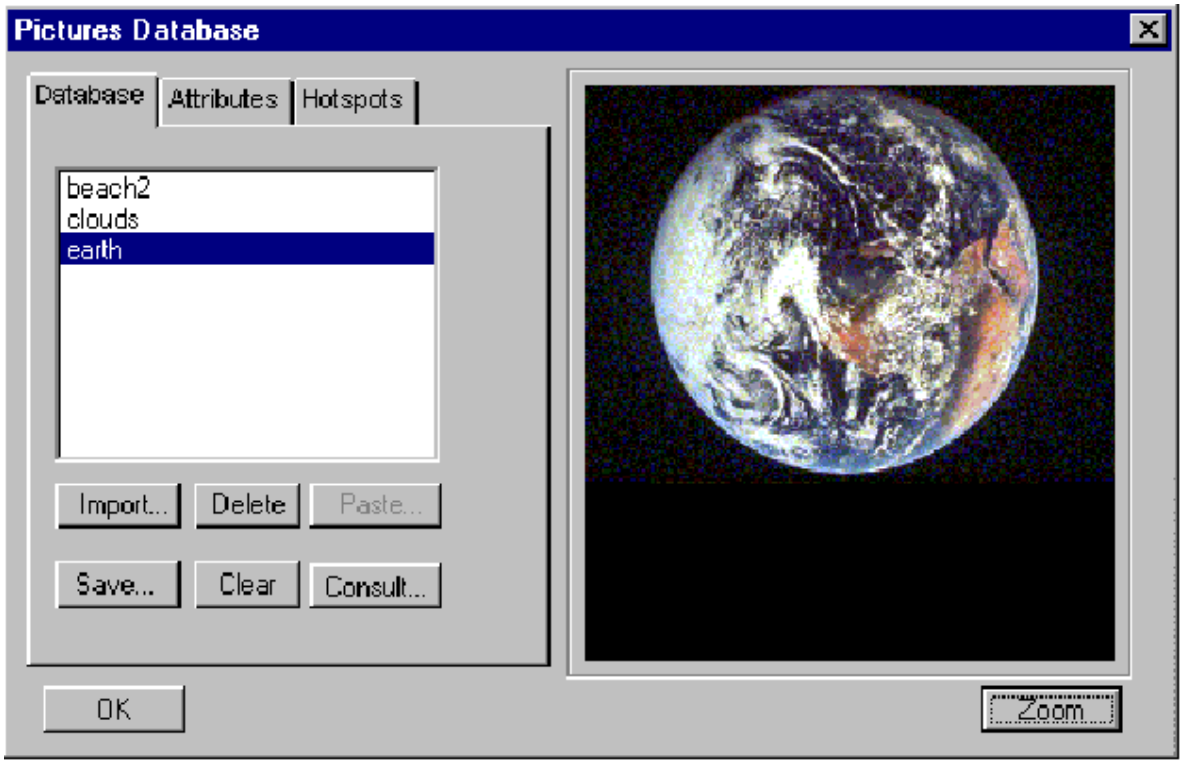
A *picture* can be used in ESTA as:

1. A start picture for the ESTA system
2. A title for a knowledge base
3. A part of a consultation dialog, attached to a parameter
4. A part of an advice
5. A part of an actionlist in a section, showpic(picture)

The knowledge base CAR.KB contain examples of all the mentioned uses.

Pictures Database

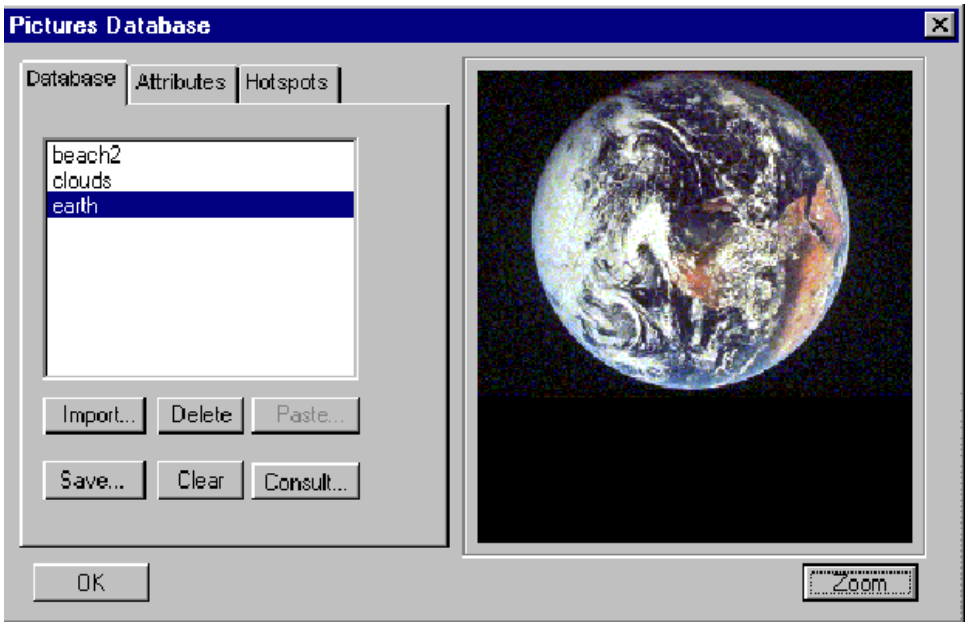
This command is used to modify or view the *pictures database*. This invokes the Picture Database Dialog



This dialog is devided into two parts. The pictures stored in the Pictures Database are listed in the box at the left side of it. And the currently selected picture is shown at the right side. The Zoom button at the right bottom corner is used to display the picture on the full screen.
The dialog has three tabs:

- Database
- Attributes
- Hotspots

Database Tab



This tab is used to view the *database contents*, to insert new pictures in the database, or to delete the existing ones.
Here is the description of the controls:

- Import...

Imports a Windows metafile or a device independent bitmap to the ESTA pictures database. When you choose this command, an Open File dialog box is displayed. You can open a metafile or device independent bitmap file and the file will be shown in a dialog box, where name, position and scale can be specified.

- Delete

This button *deletes* from the pictures database the picture whose name is highlighted.

- Paste...

Pastes the picture from the Clipboard into the pictures database. The picture in the Clipboard must be a Windows metafile, a device dependent bitmap or a device independent bitmap.Note that you can copy pictures to the Clipboard from other Windows programs even while the ESTA pictures dialog box is open. The Paste command button will then be dynamically activated.

- Save...

Saves current pictures database in text format to keep it portable across 16/32 bit platforms.

- Consult...

Consults current pictures database from text format to keep it portable across 16/32 bit platforms.

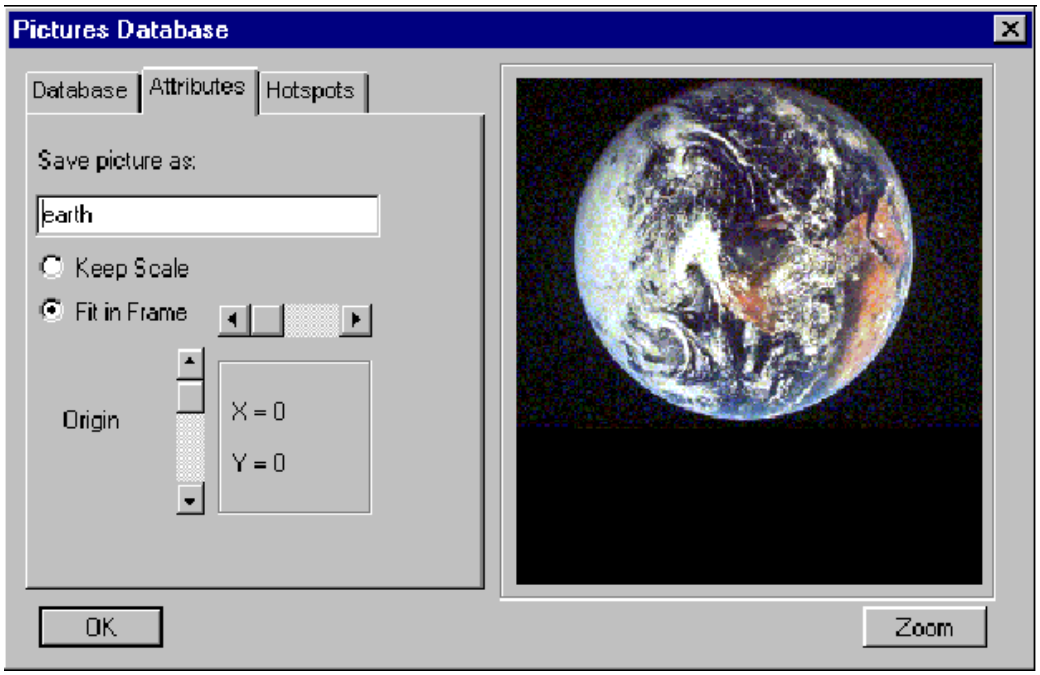
- Clear

Clears the Picture Database.
ESTA Pictures Database is Prolog External Database itself, so the databases created in 16 and 32 bit versions of ESTA are incompatible. Using Save and Consult facility you can avoid this restriction.

NOTE:

· ESTA stores pictures that are large than 64 kylobytes in separate files. After converting pictures database from/to different version you can simple copy this files in a new location.
· Currents version of VPI and ESTA do not support metafiles under OS/2.

Attributes Tab



Allows to modify the name, scaling and positioning of a picture. Here is the description of the controls:

- Save picture as

Allows to change the currently selected picture's name

- Keep Scale

When checked, the current picture will be shown with its original sizes in the Consult dialogs and Title window.

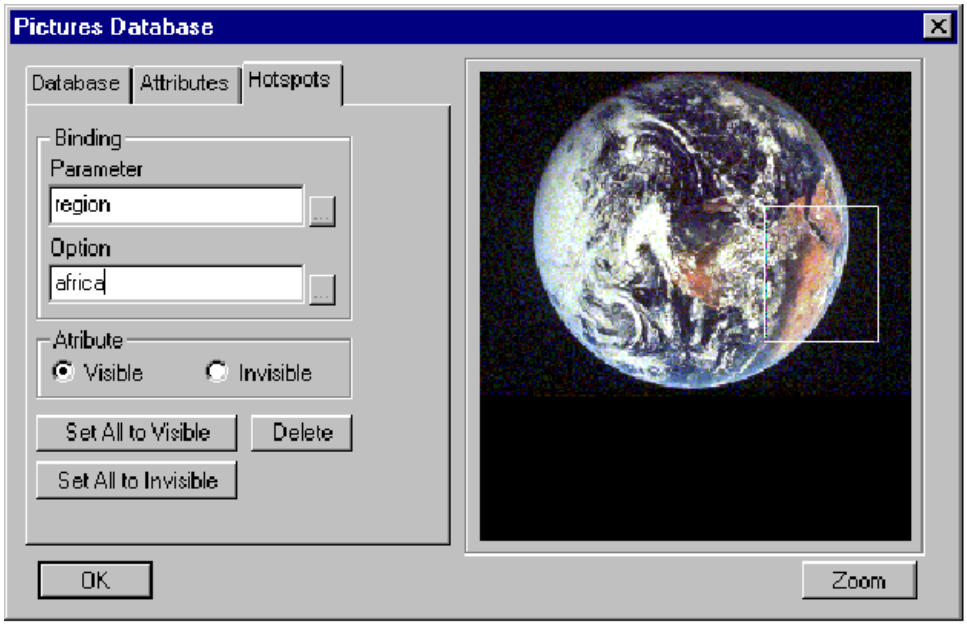
- Fit in Frame

When checked, the current picture will be stretched to the window it shows.

- Origin

Two scrollbars allow to set the picture's vertical and horizontal position.

Hotspots Tab



Allows *specify fields (hotspots)* in a picture. The facility is only useful for pictures which are, or are going to be, attached to category parameters. The fields correspond to the options in a category parameter. When pictures are linked to category parameters in this way, instead of selecting an option from a listbox, the user can select the option by clicking on a field in the picture. You should define the category parameter with its options and the name of the picture first, and then use the hotspot editor to define the fields in the picture. When the hotspot button is activated the current picture will appear in a window. To specify a hotspot field do the following:

1. Position the mouse at the upper left corner of the intended field location and move (drag) the mouse while the left mouse button is held down until the rectangle indicates the size field you want, then release the mouse button.

2. At the Parameter field you enter the name of the category parameter which will include the picture. You can choose the available parameters by clicking at the small button to the right of the Parameter field. Having done so, enter, or select the possible options for the selected parameter.

3. In addition to specifying names for the field you can also specify whether you want the field to be visible or not. The boundaries of a visible field are marked by a rectangle, whereas the rectangle is not draawn for an invisible field. In either case the cursor will change its shape into a hand, when it is positioned over the field, giving visual feedback that this is a selectable region. You can set the visibility for all fields in the current picture by clicking on the button Set All to Visible or Set All to Invisible respectively in the hotspot window.

The description of the controls:

- Parameter

The field to enter the category parameters name here.

- Options

The field to enter one of the posible option for the specified parameter.

- Attribute

Radiobuttons to set weather the specific hotspot is visible or not.

- Set All to Visible

Turns all the hotspots for the current picture to visible state.

- Set All to Invisible

Turns all the hotspots for the current picture to invisible state.

- Delete

Delete the specified hotspot from the current picture.

Take Consultation

Begin Consultation

Clears all parameters and starts the consultation by executing the *start section* of the active knowledge base. The Boolean expressions in the paragraphs of start will be evaluated. Part of these Boolean expressions will be parameters without values, and ESTA will try to establish values for these, either by using rules or by prompting the user for input as appropriate.

Continue Consultation

Restarts an interrupted consultation from the point at which it was interrupted, without resetting any parameter values.

Check Knowledge Base

Checks the current knowledge base. The following types of errors are reported:

Type errors

- Expressions evaluating to non-conforming types are detected.
- Rule cycles
- Cyclical(recursive) definitions of rules, occurring when a parameter is determined by either directly or indirectly referring to itself, are not allowed.
- Section cycles
- Cyclical patterns in general, e.g. that the section leads back to itself, are not allowed either.
- Missing sections

- Do and do_section_of actions referring to non-existing sections are detected and the names of the missing sections are indicated.
- Missing parameters
- All references to undefined parameters are detected and the names of the missing parameters are indicated

Show Knowledge Base

Lists the current knowledge base in a window. Note that you can print the knowledge base on paper by selecting the print command.