CHAPTER TEN – CREATING YOUR OWN TOOLS

10.1 INTRODUCTION

The facility for creating new tools in the AKT software is provided for the user whose particular needs are not met by existing tools. This chapter introduces the basic principles of creating a new tool, and gives some examples.

10.2 TOOL FILES

10.2.1 CREATING A NEW TOOL FILE

If you wish to create a new tool you must first create a tool file in which to save the new tool(s). To do so, select **Tools** from the main menu and then **New Tool File**. The following screen will appear (*Figure 10.1*), requiring that the new tool file be given a name. The suffix is always automatically '.mcr' (for 'macro'). Once the file has been named, press **Save** and the following message will appear (*Figure 10.2*):

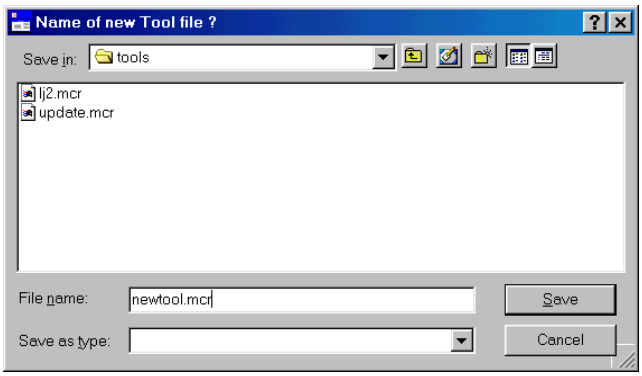


Figure 10.1 *Naming a new tool file*

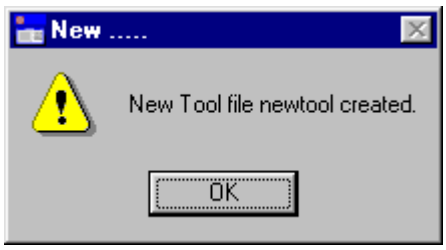


Figure 10.2 *Confirming that a new tool file has been created*

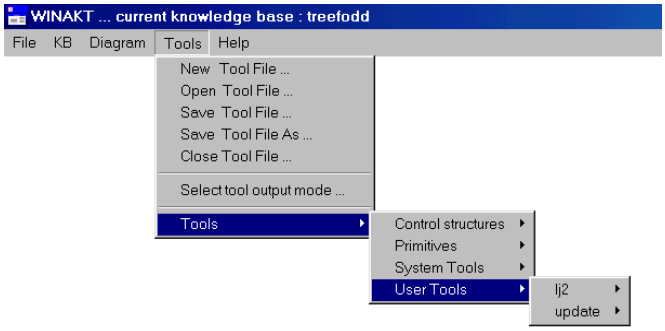
10.2.2 OPENING A PREVIOUSLY CREATED FILE

To open a previously created tool file, select **Tools** from the main menu, then **Open Tool File**. Then browse through your folders and files to select the correct file. Highlight the file required and press **Open**. The tool file will then automatically open.

It is always advisable to keep your knowledge bases and tool files in folders separate from the AKT application

10.2.3 KEEPING TRACK OF THE TOOL FILES

It is possible to have more than one tool file loaded at a time, simply by creating or opening more tool files. To change the tool file in use, select **Tools** from the main **Tools** menu, then **User Tools** and then the tool file required (*Figure 10.3*).



Only the user tools in the *selected* Tool File will be available for use

Figure 10.3 *Changing the tool file in use*

10.2.4 SAVING TOOL FILES

It is only necessary to save a tool file if you have created or edited existing user tools. When you save the new or edited tool, a message appears asking if you wish to save an updated tool file as well as keeping a copy in memory. If you wish to preserve the newly created tool, you *must* save the tool file as well, therefore select **Yes**.

If you wish to save the new tool but not in the current tool file, then save the tool file under a new name.

It is possible to save a tool file at any time by opening the main **Tools** menu and selecting **Save Tool File**. To save it under a different name, or in a different folder or different directory, select **Save Tool File As**.

10.2.5 CLOSING TOOL FILES

In order to close a tool file select **Close Tool File** from the main **Tools** menu. If you have more than one tool file open, the dialog box that appears (*Figure 10.4*) allows you to select the tool file to be closed from the 'Files' list. Once selected, press **OK**.

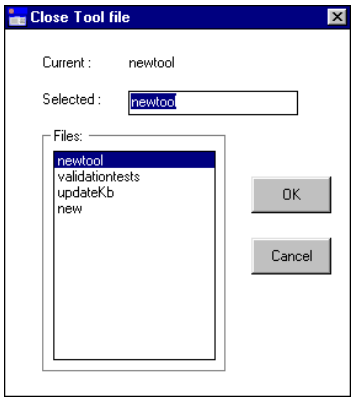


Figure 10.4 *Closing a tool file*

WARNING

If you wish to save a tool file, you must save it *before* closing it.

10.3 AN EXAMPLE OF A USER DEFINED TOOL

Below we give an example of a user defined tool, from a user defined tool file, Ghana Tools[1] which we will run on the treefodd knowledge base. Make sure you have the Ghana Tools tool file loaded and then select the tool **search_Topic_Statements**. This tool allows the user to search for a term, or several terms in a given topic. First we will run the tool, and then we will analyse how it has been built up.

To run the tool, press **Run** (*Figure 10.5*). A dialog box appears, requesting you to select a topic (*Figure 10.6*). Select your topic, in this case 'scientists' and press **OK**.

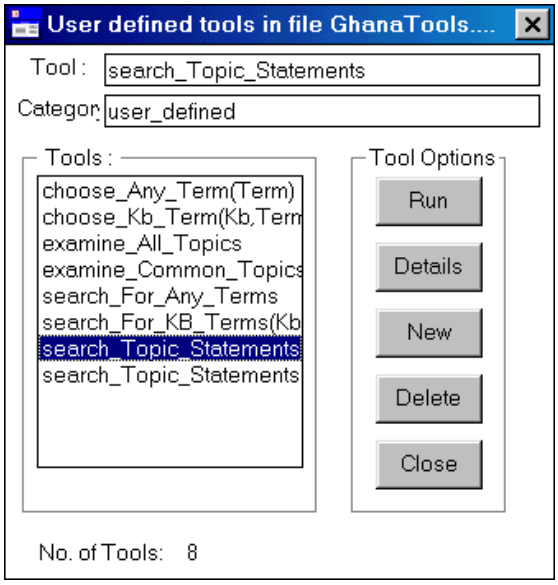


Figure 10.5 *User-defined tools*

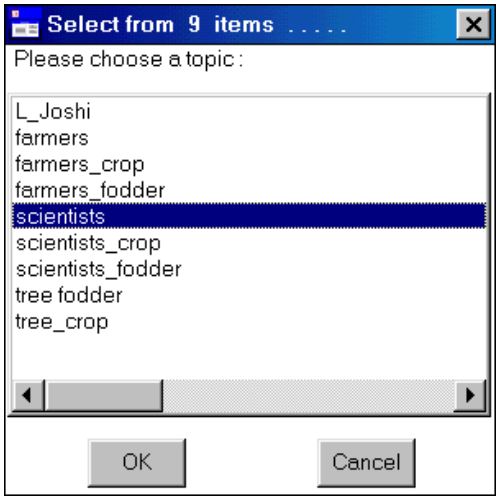


Figure 10.6 *The tool* ***search_Topic_Statements*** *in action*

A new dialog appears, requesting you to type in the formal term/terms that you want to search for, in this case 'soil' (*Figure 10.7*). Press **OK**. The tool output (*Figure 10.8*) will then give you all statements containing the term 'soil', in natural language, from the topic 'scientists'.

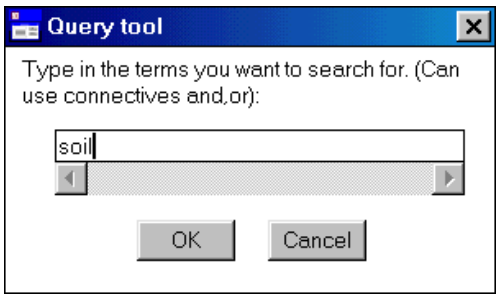


Figure 10.7 *The tool* ***search_Topic_Statements*** *in action continued*

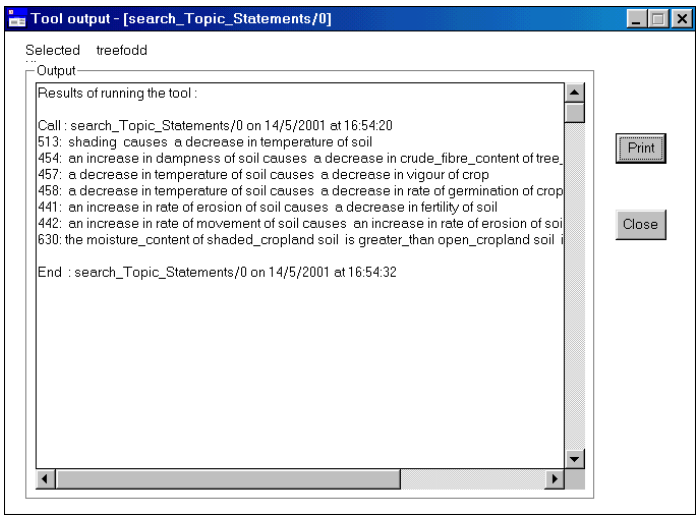


Figure 10.8 *The Tool output*

[1] These tools were created for the Atwima knowledge base. Both the knowledge base and tool file can be downloaded from our website: www.safs.bangor.ac.uk/afforum

Now we will look at the body of the tool. If you return to the tool **search_Topic_Statements** and press **Details**, the following dialog box appears (*Figure 10.9*):

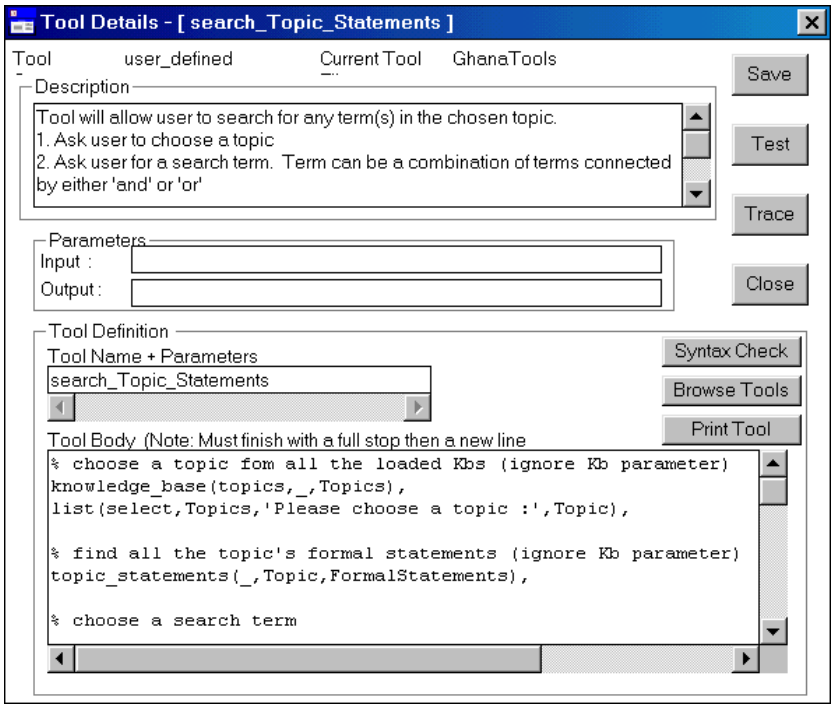


Figure 10.9 *Tool details for the tool* ***search_Topic_Statements***

The full tool body looks like this:

```
% choose a topic fom all the loaded Kbs (ignore Kb parameter)
knowledge_base(topics,_,Topics),
list(select,Topics,'Please choose a topic :',Topic),

% find all the topic's formal statements (ignore Kb parameter)
topic_statements(_,Topic,FormalStatements),

% choose a search term
ask('Type in the terms you want to search for. (Can use connectives and,or)',SearchTerm),

% search the formal statements for selected term (ignore Kb parameter)
statements_search(_,SearchTerm,_,object,FormalStatements,TermStatements),

% list all the numbered translations of the Formal statements
foreach Statement in TermStatements
do ( statements_convert(numbered,_,Statement,Translation),
     show(Translation),show(nl)
   ).
```

We will now go through the tool, step by step. As we go through the primitives that make up the tool, look at each primitive in turn, by using the **Browse Tools** button on the 'Tool Details' dialog box.

Several things to note before we start:

a) A line beginning with '%' is a comment and not an executable part of the tool. Thus '% choose a topic from all the loaded Kbs (ignore Kb parameter)' tells you that the next step will instruct the program to permit you to choose any one topic from all the

available topics in the loaded knowledge bases. (In the following step by step analysis of the tool, these descriptions have been removed).

b) Each primitive or tool may have parameters which can be any combination of input and output parameters. The output parameter can contain a result which is returned as the output from the tool or it can in turn be used as an input parameter to another primitive or tool later in the definition.

NOTE: An **input parameter** is one where the primitive or tool expects the parameter to be specified before starting the tool. An **output parameter** is one where the primitive or tool determines the value and returns this value as the output at completion of the tool.

Here goes:

 Take one

knowledge_base(topics,_,Topics),
list(select,Topics,'Please choose a topic :',Topic),

The first step enables the user to specify the topic that he wishes to investigate. The topic may come from any of the loaded knowledge bases, as the knowledge base parameter is left unspecified.

This primitive is from the **Miscellaneous** category and its generic form is **knowledge_base(Type, Kb, Result)**. In this tool you will see that there is an anonymous variable '_' where the Kb parameter would normally be. This means that the knowledge base parameter will match any loaded Kb name, i.e. the tool will search all the loaded knowledge bases, rather than a specific knowledge base.

list(select,Topics,'Please choose a topic :',Topic),

This primitive is from the **List** category and its generic form is **list(Operation, List, Item, Result)**. The variable 'Operation' is replaced by the term 'select'. (In the Description box of the primitive is a list of all the possible terms that can be used for the variable 'Operation' – 'select' is one of the possibilities). The variable 'List' is replaced by the variable 'Topics' which will gives a list of all the topics found by the previous primitive. The variable 'Item' is specified as the message 'Please choose a topic' which has to be enclosed in inverted commas. Finally the variable 'Topic will contain the name of the topic selected by the user'

Remember: parameters which start with a capital letter are variables (unless they are enclosed in inverted commas), whereas parameters which start with a small letter are atomic terms and do not change.

 Take two

topic_statements(_,Topic,FormalStatements),

This step will find all the formal statements about the topic selected in step one.

The primitive is from the **Statements** category and its generic form is **topic_statements(Kb, Topic, Statements)**. The first term of the primitive is an anonymous variable ('_') and will match with any knowledge base. The output variable 'FormalStatements' contains a list of the statements about the topic.

Take three

ask('Type in the terms you want to search for. (Can use connectives and,or)',SearchTerm),

This step enables the user to specify the search term/terms

This primitive is from the **Display** category and the generic form is **ask(Question, Answer)**. The parameter'Question'is used to contain a message. To distinguish the message from a variable, it is written within inverted commas. The output variable 'SearchTerm', contains the term(s) typed in by the user.

Take four

statements_search(_,SearchTerm,_,object,FormalStatements,TermStatements),

This step will search the formal statements about the selected Topic for the user specified search term.

This primitive is from the **Statement** category and its generic form is
statements_search(Kb,Term,Hierarchy,SearchOption,StatementsIn,ListFormalStatements)

The first and third parameters are anonymous variables because we are not restricting the search to a particular knowledge base or object hierarchy.The variable SearchTerm contains the output from step 3 and the search option is set to object because we are looking for statements that contain the specified search term(s) and not statements that might contain the superobjects or subobjects of the search term. FormalStatements,contains the output of Step 2 and TermStatements will contain any of these statetements that contain the search term(s).

Take five

```
foreach Statement in TermStatements
do ( statements_convert(numbered,_,Statement,Translation),
    show(Translation),show(nl)
  ).
```

This last step lists all the statements containing the specified formal term. It translates them from the formal syntax into natural language and then displays them on the screen prefixed by the statement number.
For this final step, the control structure **foreach…in…do..** has been combined with two primitives, one from the **Statements** category, **statements_convert(Mode,Kb,Original,Converted)** and one from the **Display** category **show(**Term)

Note: if the 'do' part of the control structure contains more than one primitive or tool then they need to be enclosed within parenthesis

This sequence of instructions is

a) **statements_convert(numbered,_,Statement,Translation)**. The first input parameter 'Mode' is set to 'numbered', one of the mode options[2] . The second input parameter is the anonymous variable againand the third input parameter 'Statement' Contains the formal statement to be translated and numbered.

b) **show(Translation)** This primitive simply shows the numbered translation on the screen.

Show(nl). This last part simply instructs the tool to output a new line (nl) and carriage return after the translated statement.The control structure foreach…in…do, insures that the instructions a, b and c are carried out for each statement in the selected topic, containing the specified term until all such statements have appeared on the screen.

Finally, note that the last line contains the closing bracket for the control structure and a full stop. You must always add a carriage return (↵) after the full stop to complete the tool.

[2] See the 'Description' box for the primitive statements_convert(Mode,Kb,Original,Converted) to see the full list of mode options.

The Trace button.
To observe the various functions of the different steps, select the tool **search_Topic_Statements** once more and in the 'Tool Details' dialog box, press the **Trace** button. This will take you through the tool step by step. This function is particularly useful if a tool does not function properly. You can see at which step the error lies.

10.4 CREATING YOUR OWN TOOL

In order to create a new tool in AKT it is necessary for a user file to be opened to contain the tools. Open the main **Tools** menu and select either **New Tool File** or **Open Tool File** (see above, section 10.2) Once a new tool file has been created, or an existing tool file opened, a dialog box will appear showing a list of 'User defined tools' (*Figure 10.10*). If no tools have been previously created by the user, this dialog box will be empty.

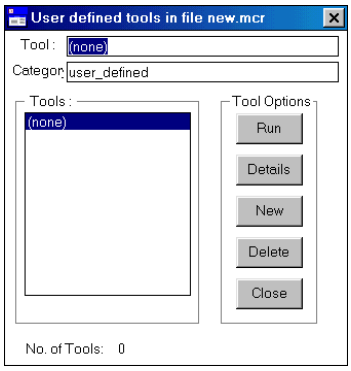


Figure 10.10 *'User defined tools' dialog box*

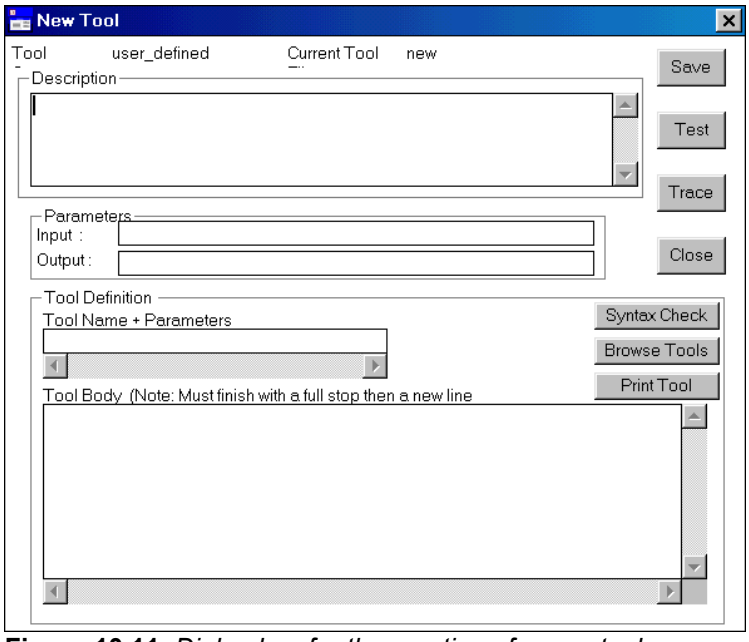


Figure 10.11 *Dialog box for the creation of a new tool*

In order to create a user defined tool, select **New**. A 'New Tool' dialog box appears (*Figure 10.11*)

The information that must be specified about a tool includes:

- Description of the tool's function

  The 'Description', written in plain text should describe the tool's function and give instructions for its use. It is worth remembering that if the knowledge base is to be used by others then the description should be as informative as possible.

- Input and output parameters

  The input and output parameters (arguments) required by the tool. Parameters are the means of passing information into or out of the tool. Each tool can have any number/combinations of input and output parameters. A tool does not necessarily need input parameters nor output parameters – it depends on its function. You only need any parameters when you are passing information into or extracting information from a tool.

- Tool Definition –a) Tool name and parameters
  b) tool body

a) The 'toolname' must start with a lower case letter e.g. ***find_trees*** or if it is required to start with a capital letter then the whole toolname must be enclosed in single quotes e.g. ***'Find_trees'***. The arguments or parameters contained in parenthesis after the toolname generally begin with a capital letter indicating a variable but they can be a fixed value if required. For example *Kb* and *Date* are variables but *attributes* and '*John Smith'* are fixed values. The toolname should express succinctly the function carried out by the tool.

b) The 'tool body' contains all the functions, control structures and appropriate punctuation that allows the tool to perform the required process. If the tool body does not have the correct form or syntax then AKT will report a 'Syntax Error' whenever an attempt is made to save the tool or the syntax check button selected.

10.4.1 INCORPORATING EXISTING TOOLS / PRIMITIVES WITHIN A NEW TOOL DEFINITION

There are various ways of adding a primitive, systems tool, control structure or previously defined user tool to the definition of the new tool; two possible ways using cut and paste functions are described below.

Method 1: Select the **Browse Tools** box from the 'New Tools' dialog box (see above, *Figure 10.11*). This will display a list of all the tools. Highlight the primitive you think you need and then select **Details.** The 'Description' within the 'Tool Details' will enable you to decide whether or not it is indeed the primitive you want to use. Once you have the desired primitive highlight the 'Tool Name + Parameters' box and press **Ctrl c** to copy it. Return to the 'New Tool' dialog box and taking the cursor to the 'tool body' box, press **Ctrl v** to paste it into the tool.

Method 2: Another method of copying a primitive, once you are familiar with the program, is to select **Help** from the main menu, and from the dropdown menu that appears, select **Tool List**. The following screen will then appear (*Figure 10.12*):

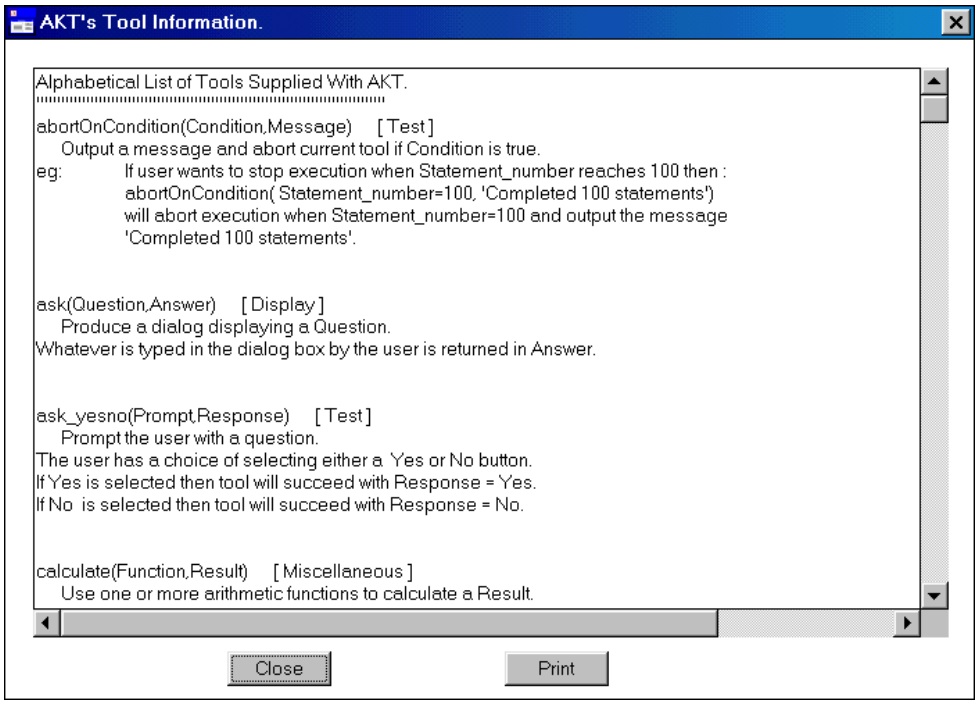


Figure 10.12 *List of Tools available in **Help** on the main menu*

The list is alphabetical and gives the tool name and the category to which it belongs and the description of what it does. Using the scroll bar enables you to select the tool you require.

Highlight the tool name and parameters (but not the category, nor the definition) and press **Ctrl c**. Return to the 'New Tool' dialog box and setting the cursor in the 'tool body' box, press **Ctrl v** to paste it into the tool.

10.4.2 TESTING THE SYNTAX OF A NEW TOOL

In order to check whether or not the syntax of the tool is correct, select **Syntax Check**. If there is an error in the syntax anywhere within the tool one of the following messages will appear (*Figures 10.13 and 10.14*):

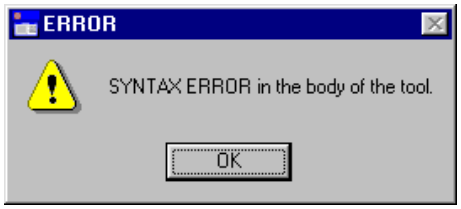


Figure 10.13 *Error message which appears if the format of the control structure is wrong, or the punctuation incorrect*

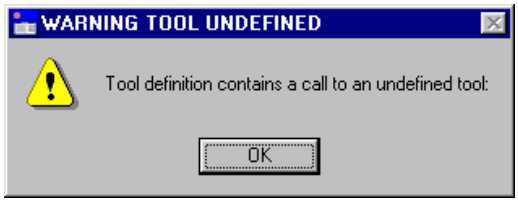


Figure 10.14 *Message received if a non-existent tool has been used*

A **syntax error** occurs when the format of the tool head or body does not correspond to the Prolog syntax which is used by the code underlying the AKT application. A **call to an undefined tool**, on the other hand, means that the name of a tool used in the function does not exist. It may have been incorrectly spelt or the number of parameters may be wrong.

As the message does not tell you specifically where the error lies within the tool, it is useful to carry out a syntax check after each new line is added. In order to do this you can temporarily put a full stop at the end of the new line and follow it with a carriage return (↵); then press **Syntax Check**. If a message 'Tool syntax is ok' appears then you can continue building the tool, first removing the full stop and replacing it with a comma before starting a new line. If a message appears as in *Figure 10.13* or *10.14* above, then you should not proceed to a new line until the current line of instructions has been corrected.

10.4.3 AN EXAMPLE OF CREATING A TOOL

The best method of explaining how to create a tool is by demonstration.

In this example we are creating a tool which will list all the statements related to a specified term (or terms) and then saving these statements in the form of a new knowledge base.

Select **New** from the 'user defined tools' Tool dialog box (as above, *Figure 10.10*) and the 'New Tool' dialog box appears (as above, *Figure 10.11*).

First give a description of what the tool does in the 'Description' box. In this case a sentence such as 'This tool collects all statements relating to a user specified term/terms and then saves them as a new knowledge base' will express its function adequately.

The toolname 'saveStatementsToKb' acts as a sufficient 'aide memoire' for its function. Note that the first letter in the tool name is in lower case. In this instance there are two externally supplied parameters needed to make the tool run; the knowledge base you are interrogating, the term or terms you are looking for. Thus the full name of the tool will be saveStatementsToKb(Kb,Term).

Then move the cursor down to the tool body. The tool we wish to create should do the following things;

a) search for all the statements about a given term (or terms) in a specified knowledge base
b) save the formal statements as a separate knowledge base
c) convert the formal statements into natural language
d) display the statements in natural language.

BEFORE YOU START - SYNTAX CHECK

As you build a new tool you should carry out a syntax check after *each new line*, so that any source of error can easily be traced. See above, 10.4.2

For a) we need the primitive:
statements_search(Kb,Term,Hierarchy,SearchOption,StatementsIn,ListFormalStatements).

We access this by pressing **Browse Tools** in the 'New Tool' dialog box (as above, Figure 10.11), and selecting **Primitives** then **Statement**. We then highlight the primitive required and press **Details**. From the 'Details' dialog box we highlight the toolname plus the arguments and press **Ctrl c**. Returning to the 'New Tool' dialog box we move the cursor to 'tool body' and press **Ctrl v**[3].

Looking at the primitive, we need to tailor it to our needs.

The description of the primitive is:

> This tool enables the user to search the complete knowledge base or a subset of it, and identify statements that contain any specified formal term, synonym, source, topic or a combination of these items. The tool has the same capability as the interactive Kb / Boolean Search menu option. It collects together in a list any formal statements from the StatementsIn list that contain Term or its superobject or subobject depending on SearchOption.
>
> The user can restrict the effect of the SearchOption to a particular hierarchy by specifying the Hierarchy parameter. However this parameter can be ignored for more general searches (set Hierarchy = _). If user is not interested in the effect of object hierarchies then SearchOption should be set to object.
>
> If StatementsIn = all then search will be carried out on all statements in Kb knowledge base.

This description box allows us to see how we can implement the primitive. Thus, as we are not interested in hierarchies, we will replace the Hierarchy variable with the anonymous variable ('_') and the SearchOption is set to 'object'. Finally, as we want the search to be carried out on all statements in the knowledge base, the StatementsIn variable is replaced by 'all'.

The tailored primitive will therefore look like this:

statements_search(Kb,Term,_ ,object ,all,ListFormalStatements),
Carry out a syntax check (as recommended in the box above).

For b) we need the **Statement** primitive, **statements_save(Kb,Statements)**.

Enter the primitive details via the **Browse Tools**. Highlight the tool name plus parameters and, using **Ctrl c** and **Ctrl v**, place the primitive as the second line in the new tool, add a comma and press carriage return.

[3]An alternative, quicker route is Method 2 as described above, 10.2.1.

The growing tool now looks like this:

statements_search(Kb,Term,_ ,object ,all,ListFormalStatements),
statements_save(Kb,Statements),

When you are combining primitives together to create a new tool there are two points to remember

- The parameter names in the tool body must be identical to the parameter names in the tool head when referencing the input or output parameters.
- When a primitive in the new definition is using the output parameter of a primitive used earlier on in the tool body as an input parameter, the two parameter names must be identical.

Thus, the input parameter 'Statements' in this second primitive is replaced by 'ListFormalStatements', the output parameter from the primitive above. When amended, the second line of the new tool looks like this:

statements_save(Kb,ListFormalStatements),

Do a syntax check and then go on to c)

For c) we need the **Statement** primitive **statements_convert(Mode,Kb,Original,Converted)** Enter the primtive details via the **Browse Tools** and copy and paste the tool name and parameters as above in a). Then add a comma and press carriage return. The growing primitive should now look like this:

statements_search(Kb,Term,_ ,object ,all,ListFormalStatements),
statements_save(Kb,ListFormalStatements),
statements_convert(Mode,Kb,Original,Converted),

The description box below suggests ways in which the primitive **statements_convert(Mode,Kb,Original,Converted)** might be modified.

Allows user to change the format of either a single or a list of statements. The choices are :
(a) Mode = formal Convert list of statement identifiers to a list of formal statements.
(b) Mode = identifier Convert list of formal statements to a list of statement identifiers.
(c) Mode = translate Translate a list of formal statements to their natural language equivalents.
(d) Mode = numbered Convert list of statement identifiers or formal statements to numbered list of natural language equivalents.
eg: statements_convert(formal, soil,[1,25,73],FormalStatements)
statements_convert(identifier,soil,att_value(black_soil,fertility,high),Number)
statements_convert(translate, soil,[att_value(black_soil,fertility,high)],Translation)
statements_convert(numbered, soil,att_value(black_soil,fertility,high), Numbered)
NOTE: when converting formal statements to their equivalent statement number it is possible to have more than one solution if the knowledge base contains duplicate statements from different sources or more than one knowledge base is loaded.

As the statements should retain their identifying number, the Mode variable is replaced by the option 'numbered'. The second variable Kb remains the same. The variable 'Original' is replaced by the input parameter from the previous primitive 'ListFormalStatements'. The variable 'Converted' we can give whatever name we choose – in this instance 'TranslatedStatements' would seem appropriate.

Thus, the third line, when amended, should look like this:

statements_convert(numbered,Kb, ListFormalStatements, TranslatedStatements),

Do a syntax check and then go on to d)

Finally to d). The tool output will not automatically be displayed in the tool output menu. To display the tool output, you must incorporate instructions into the tool itself. To do this we add the Display primitive **show(Item)**. Whatever appears in place of the Item variable will appear on the screen. We need to show two things, (i) a message saying what the output is and (ii) the output itself. Therefore we will need the primitive **show(Item)** twice.

The tool would now look like this:
statements_search(Kb,Term,_ ,object ,all,ListFormalStatements),
statements_save(Kb,ListFormalStatements),
statements_convert(numbered,Kb, ListFormalStatements, TranslatedStatements),
show(Item),
show(Item)

The variable *Item*, in the first **show(Item)** primitive would be a message. This message must be in inverted commas. 'This is a numbered natural language version of the statements' would be an appropriate message.

The variable *Item*, in the second **show(Item)** primitive should be the statements themselves. Thus Item should be replaced by the output parameter of the third primitive TranslatedStatements.

Thus the final tool will look like this:

statements_search(Kb,Term,_ ,object ,all,ListFormalStatements),
statements_save(Kb,ListFormalStatements),
statements_convert(numbered,Kb, ListFormalStatements, TranslatedStatements),
show('This is a numbered natural language version of the statements'), show(nl),
show(TranslatedStatements).

You will note that show(nl) has been added to the fourth line, in order for the statements to appear on a new line below the message, rather than directly appended to it. The new line is not necessary for the tool to work, it simply makes the output easier to read. Finally, the tool is finished with a full stop and carriage return (↵).

A tool can *only* be completed by putting a full stop at the end of the last line and entering a carriage return (↵).

Carry out **Syntax Check** on last time. Your screen should look like *Figure 10.15* below. If the message 'Tool syntax is ok' appears, you can proceed to the next step.

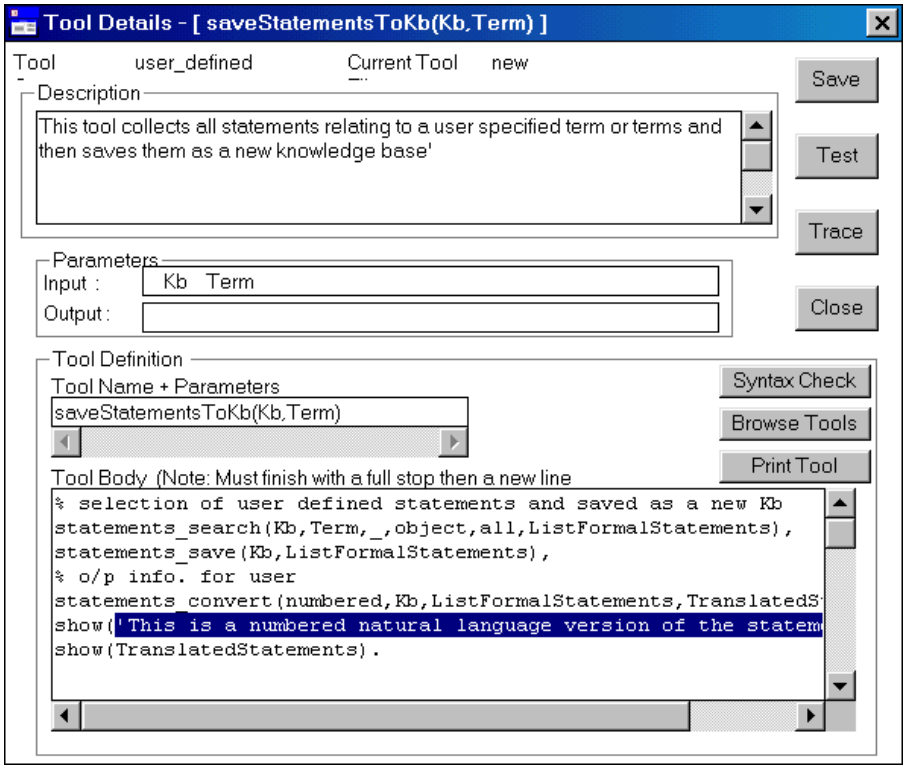


Figure 10.15 *The new tool, saveStatementsToKB(Kb,Term), completed*

Then press **Save**. A dialog box will appear (*Figure 10.16*) with the two parameters in the toolname, requesting you to specify which are the input parameters. In this case both parameters are input parameters, so tick the two and then press **OK**.

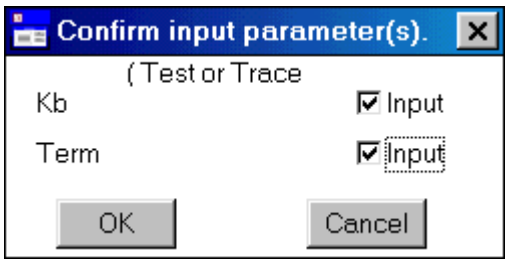


Figure 10.16 *Dialog box requesting the input parameters.*

10.4.4. TRACING AND TESTING A NEW TOOL

Once a new tool has been created it is important to test whether it works properly. This can be done by using either **Trace** to step through the tool one line at a time or **Test** to run the whole tool at once. In this example we will look for all the statements containing 'soil' in the 'treefodd' knowledge base and save them as a separate knowledge base called 'treesoil'.

10.4.4.a The trace key

If, after following the example of the new tool above (10.4.3) you select **Trace**, the same dialog box as *Figure 10.16* above will appear, in which the input parameters are specified. Tick both and press **OK**. Then another dialog box appears requesting you to enter the input parameters (*Figure 10.17*). For this example the parameters are entered as follows:

- Kb: soil
- Term: soil and erosion

Then press **Continue**. The following screen appears (*Figure 10.18*).

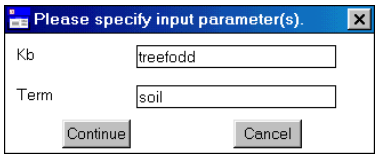


Figure 10.17 *Specifying the input parameters for use of the* ***saveStatementsToKB(Kb,Term)*** *tool*

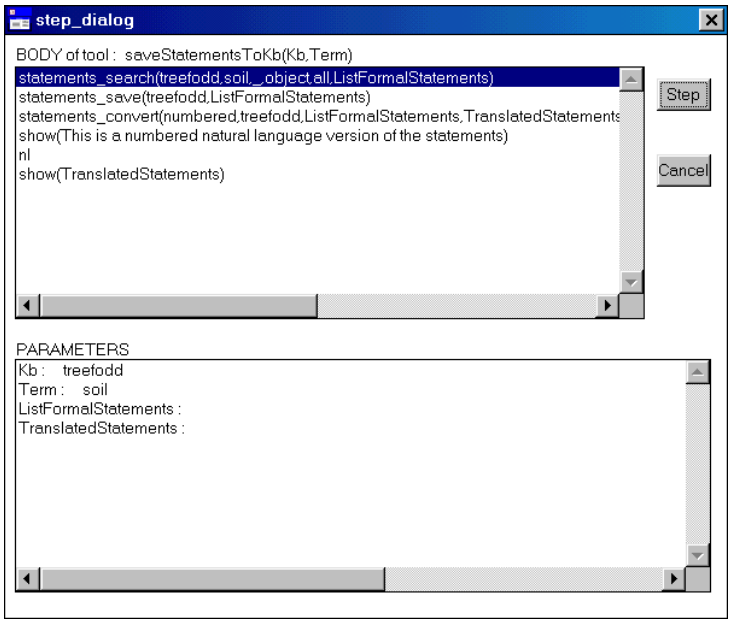


Figure 10.18 *Using the Trace option*

Each time you press **Step**, the highlighted bar will travel down one step in the tool and carry out the instructions on that line, the results appearing in the box beneath marked 'Parameters'. By the time you have reached the last line, the screen will look like this (*Figure 10.19*);

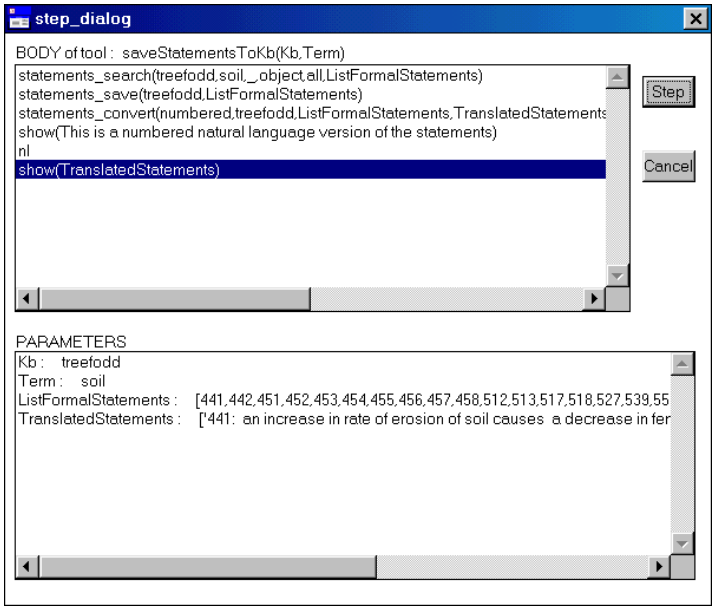


Figure 10.19 *Using the Step button, going though the tool line by line*

The **Trace** function allows you to follow the tool step by step and to monitor the actions of each line of the tool. The **Trace** function is enabled only if the tool is syntactically correct. Therefore, if the tool is not giving the required output it means that one of the individual functions is probably being used incorrectly. The step by step facility allows you to determine where it goes wrong by observing the parameter values in the lower box.

10.4.4.b The Test button

If you do not wish to test your new tool line by line you can instead select **Test** for a trial run of the tool. As in **Trace** the same dialog boxes appear, the first one specifying the input parameters (see above *Figure 10.16*) the second one requesting the input parameters to be entered (see above, *Figure 10.17*). After pressing the **Continue** key the 'Save Kb As' dialog box appears, requiring you to enter a name for the new knowledge base.

Once you have chosen and entered a name for the new knowledge base, press **Save**. The next dialog box to appear is then the 'Tool output' screen, giving all the statements containing 'soil'. Table 10.1 lists all the statements that appear in the Tool output screen:

Table 10.1 *Statements containing the formal terms 'soil' in the knowledge base 'treefodd' retrieved by the tool* ***saveStatementsToKB(Kb,Term)***
Results of running the tool :

Call : saveStatementsToKb/2 on 22/5/2001 at 10:40:38

Kb = treefodd

Term = soil
This is a numbered natural language version of the statements
441: an increase in rate of erosion of soil causes a decrease in fertility of soil
442: an increase in rate of movement of soil causes an increase in rate of erosion of soil
451: the soil condition is moisture_stressed causes the crop condition is moisture_stressed
452: an increase in dampness of soil causes a decrease in vigour of crop
453: an increase in dampness of soil causes an increase in rate of infestation of crop_pest
454: an increase in dampness of soil causes a decrease in crude_fibre_content of tree_leaf
455: the dampness of soil is low causes the soil condition is moisture_stressed
456: a decrease in fertility of soil causes a decrease in vigour of crop
457: a decrease in temperature of soil causes a decrease in vigour of crop
458: a decrease in temperature of soil causes a decrease in rate of germination of crop seed
512: shading causes an increase in dampness of soil
513: shading causes a decrease in temperature of soil
517: the land site_quality is malilo causes an increase in fertility of soil
518: a decrease in quantity of manure causes a decrease in fertility of soil
527: an increase in effect of shading causes an increase in dampness of soil
539: a decrease in rukhopan of tree_leaf causes an increase in fertility of soil
556: an increase in competitiveness of fodder_tree causes a decrease in fertility of soil
568: an increase in soil_binding_ability of fodder_tree causes a decrease in rate of movement of soil
573: a decrease in competitiveness of tree causes an increase in fertility of soil
574: a decrease in difficulty of ploughing of crop_land causes an increase in fertility of soil
630: the moisture_content of shaded_cropland soil is greater_than open_cropland soil if the system season is hiudae_crop

End : saveStatementsToKb/2 on 22/5/2001 at 10:41:28

10.4.5 COMPLETING THE TOOL

Either before or after selecting Trace or Test you can save the new tool both locally to memory or to a disk file by pressing **SAVE**. If you have already saved once before the following message will appear (*Figure 10.20*):

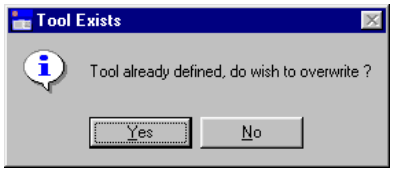


Figure 10.20 *Message that appears when saving a tool for the second time.*

Select **Yes**. Once the tool has been saved, select **Close**. If you now go to **Tools** under the main **Tools** menu, you will find that under 'User defined tools' your new tool is listed as **saveStatementsToKb(Kb,Term)**.

10.5 EDITING TOOLS

Only user defined tools can be edited. Primitives, control structures and systems tools cannot be edited. To edit a user defined tool, simply enter 'User defined tools' in the 'Tools' dialog box and select the tool in question. Select **Details**. You can then freely alter both the tool description and the tool definition. However, remember to press **Save** in the 'Tool Details' box to preserve the changes, otherwise any alterations will be lost.

10.6 DIRECTING TOOL OUTPUT TO A FILE

It is possible to direct any tool output to a file, rather than to a screen window. This is particularly userful for tool outputs larger than 64 Kb which is the limit for screen output. In order to do this go to the main **Tools** menu and select **Select tool output mode**. The program will then allow you to save the tool output in a file of your choice.