University of New South Wales Law Research Series

AUSTLII'S DATALEX DEVELOPER'S MANUAL (1ST EDITION, JUNE 2019)

ANDREW MOWBRAY, GRAHAM GREENLEAF AND PHILIP CHUNG

[2019] *UNSWLRS* 65

UNSW Law
UNSW Sydney NSW 2052 Australia

E: unswlrs@unsw.edu.au
W: http://www.law.unsw.edu.au/research/faculty-publications
AustLII: http://www.austlii.edu.au/au/journals/UNSWLRS/
SSRN: http://www.ssrn.com/link/UNSW-LEG.html

AustLII's DataLex Developer's Manual

Australasian Legal Information Institute (AustLII)

First Edition
JUNE 2019

Andrew Mowbray
AustLII
University of Technology Sydney
andrew@austlii.edu.au

Graham Greenleaf
AustLII
UNSW Sydney
graham@austlii.edu.au

Philip Chung
AustLII
UNSW Sydney
philip@austlii.edu.au

Andrew Mowbray *is Professor Law and Information Technology, University of Technology Sydney and Co-Director, AustLII.*

Graham Greenleaf AM *is Professor of Law & Information Systems, UNSW Sydney and Senior Researcher, AustLII.*

Philip Chung *is Associate Professor of Law, UNSW Sydney and Executive Director, AustLII.*

AustLII, Sydney, Australia

http://www.austlii.edu.au/

AustLII is a joint facility of UTS and UNSW Faculties of Law.

First Edition, June 2019

Contents

1 DataLex legal knowledge-base systems

1.1 Components of AustLII's DataLex legal knowledge-base systems

AustLII's DataLex inferencing software allows the development of Internet-based applications combining knowledge-based inferencing, a limited form of example-based inferencing, and automated document assembly. Applications are in the form of a plain-text knowledge-base (KB), which is written in a quasi-natural language knowledge representation.

The DataLex legal knowledge-based system has the following principal components:

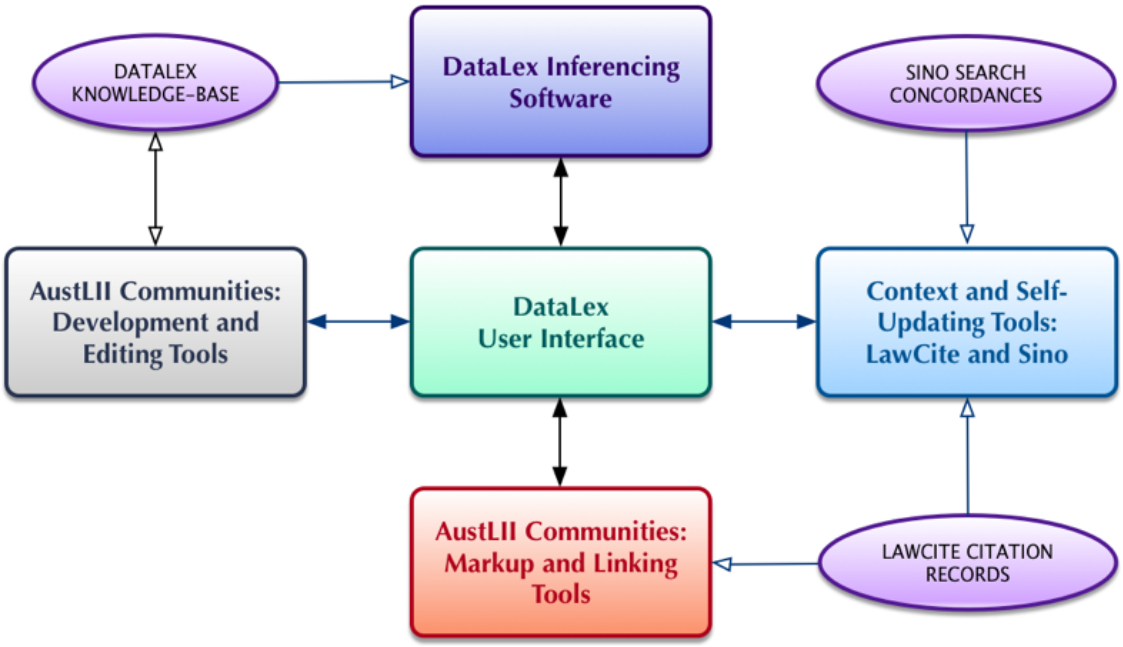


Figure 1.1: Components of the DataLex system.

This Manual describes the development and use of AustLII's DataLex legal knowledge-base systems as at June 2019.[1]

[1] Parts are derived from A Mowbray, G Greenleaf, G King & S Cant *Wysh Developer's Manuals*, AustLII, June 1997.

1.1.1 The DataLex inferencing software - YSH

The DataLex inferencing software (inference engine) is also known as *YSH* (pronounced 'why-shell'), and is referred to as such in

previous documentation. It uses a quasi-natural language knowledge representation. Knowledge may be represented as backward or forward chaining rules, or procedures, or any combinations of these, for the purposes of rule-based inferencing. Knowledge may also be represented as clusters of fact descriptions, 'cases', for the purposes of example-based inferencing using nearest-neighbour discriminant analysis. Facts are shared by the rule-based and example-based inferencing mechanisms, and by the document generator. YSH was developed with the creation of applications to law in mind, but is equally applicable to the creation of other automated interview style applications.

Andrew Mowbray is the author of the DataLex inferencing software / *YSH*. The version covered in this Manual is DataLex/*YSH* Ver. 1.2.1.

1.1.2 The DataLex knowledge-base development tools

Development of DataLex applications primarily takes place within the DataLex section of the AustLII Communities environment.[2] The development environment is that of a closed wiki which preserves the correct formatting of the knowledge-base, and provides a number of tools for checking and correcting syntax. Automated hypertext links between knowledge-base texts and the source texts on AustLII and other LIIs also make it easier to check rules etc against the sources on which they are based, during development. There is also a development environment located outside AustLII Communities which is used for teaching and development of test knowledge-bases.[3] Their use is covered later in this Chapter.

[2] DataLex section of the AustLII Communities <http://austlii.community/wiki/DataLex/>

[3] DataLex knowledge-base Development Tools <http://www.datalex.org/dev/import/>

1.1.3 The DataLex user interface

The user interface to the DataLex inferencing software provides an easy-to-use environment in which end-users conduct a question-and-answer dialogue with the application in order to provide information ('facts') to it in order for the system to draw conclusions, and to conclude a user session by producing a report (and in some cases a document). The interface allows users to ask Why questions are being asked and How conclusions have been reaching, as well as to Forget facts previously provided, and to test hypothetical facts through a What-if facility. The user interface also uses AustLII's automated hypertext markup and free text retrieval facilities to provide automated hypertext links from dialogues, conclusions and reports to the relevant legal sources on AustLII or on other LIIs. This integration between the inferencing components and the legal sources located on LIIs is one of the principal distinctive features of the DataLex approach.

The User Manual for the current[4] DataLex user interface is in

[4] A previous user manual, referred to in some documentation, was A Mowbray, G Greenleaf, G King & S Cant *Wysh User's Manual - 1997*.

Chapter 8 of this Manual. The purposes of features of the software, as described in this Developer's Manual, are sometimes best understood by reference to the User's Manual. The current interface[5] to the DataLex inferencing software (2019) has been developed by Philip Chung, Andrew Mowbray and AustLII consultants.

[5] The earliest web interface to YSH (S Cant and G King, 1997) was called *Wysh* ('web-ysh'), a Common Gateway Interface (CGI) to YSH written in Perl and C. It associated each user session with a session identifier and connects subsequent requests to the appropriate session, or process, by means of UNIX sockets. It is referred to in previous documentation.

1.2 Conventions used in this Manual

The following conventions are used in this Manual to explain commands or file names:

string Words or symbols in bold indicate the actual words or symbols used;

string Words or symbols in italics indicate that their content is variable;

| A vertical bar is used to divide a range of options - don't type it.

1.3 Theoretical foundations of the DataLex approach

There are a number of articles explaining and justifying the approach taken by the DataLex project. The main articles, and bibliography, are as follows:

- A Mowbray, P Chung and G Greenleaf, 'Utilising AI in the Legal Assistance Sector – Testing a Role for Legal Information Institutes' (29 April 2019), presented at *LegalAIIA 2019*, co-located with ICAIL 2019, 17 June 2019, Montréal, Québec, Canada <https://ssrn.com/abstract=3379441>
- G Greenleaf, A Mowbray, and P Chung, 'Building Sustainable Free Legal Advisory Systems: Experiences from the History of AI & Law' (2018) 34(1) *Computer Law & Security Review* 314 <https://ssrn.com/abstract=3021452>
- G Greenleaf, A Mowbray, and P Chung, 'The Datalex Project: History and Bibliography' (3 January 2018). [2018] UNSWLRS 4 or <https://ssrn.com/abstract=3095897>
- G Greenleaf, A Mowbray, and P van Dijk, 'Representing and using legal knowledge in integrated decision support systems - DataLex WorkStations' Artificial Intelligence & Law, Vol 3, 1995, Kluwer, 97-142 <https://papers.ssrn.com/abstract_id=2183481>

1.4 Creating a new knowledge-base in the DataLex Community

The DataLex Community (part of the AustLII Communities) is a collaborative closed wiki-like platform used in the DataLex system

for creating and editing knowledge-bases or rule-bases.

Log into the DataLex Community site <http://austlii.community/wiki/DataLex/> by clicking on the 'Log in' button on the top right hand corner.

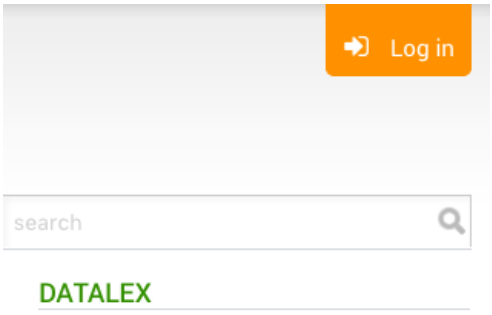


Figure 1.2: DataLex Community login process.

Once the login process is verified, an extra row of buttons for editing and creating new rule-bases will appear on the page.

Figure 1.3: Editing existing rule-base in DataLex Community.

Click on 'Edit' to edit existing rule-bases.

To start a new rule-base (or topic), click on the 'New' button. Do so from the DataLex page, or it will be a sub-page from wherever you start. The following window will appear:

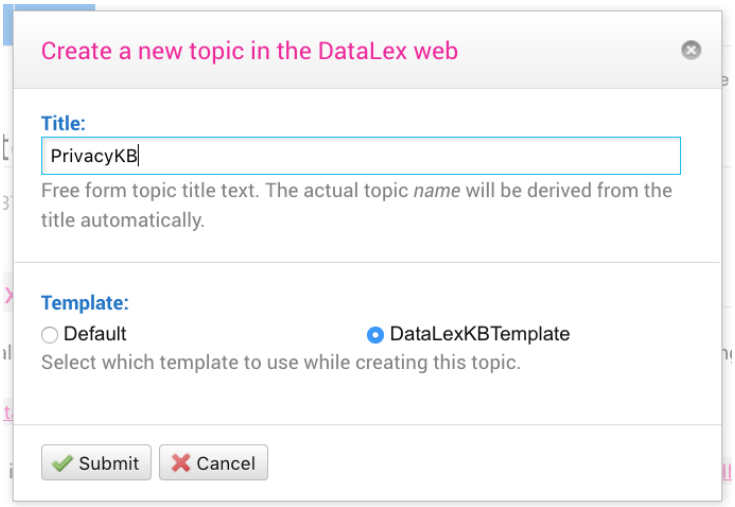


Figure 1.4: Creating new rule-base (or topic) in DataLex Community.

Enter the 'Title' of the rule-base to be constructed. The example here is 'PrivacyKB' for a privacy law knowledge-base. (For easy identifiability, it is useful to give knowledge-bases the suffix 'KB'.) In the 'Template' section, select 'DataLexKBTemplate' (not 'Default'). Then, click on 'Submit'.

In the editing screen for the rule-base (with heading 'Title of knowledge-base'), start editing your rule-base by deleting 'ADD

RULES HERE'. One way to start a rule base is to paste in a legislative section, and start editing it to create rules.

To find your rule-base after you have logged back in, search for the first few letters of the name of your rule-base (eg search for 'FOI' to find FOIDocumentOfAnAgencyKB).

1.5 *Creating a new knowledge-base using the Development Tools*

If you do not yet have an account in AustLII Communities, go to <http://www.datalex.org/dev/import/> (AustLII's *DataLex Knowledge-base Development Tools* page – 'KB Tools page'). No login is necessary. This page enables creation and test-runs of small applications using the DataLex software. However, it does not allow test apps to be saved.

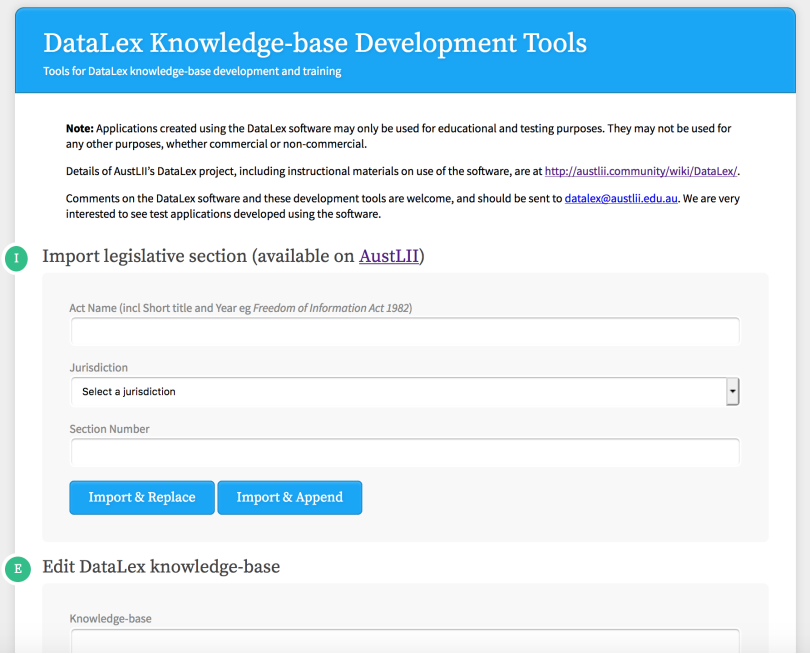


Figure 1.5: DataLex knowledge-base development tools: Importing and Editing rules.

There are two ways to start writing an app:

(i) Simply start writing in the 'Edit DataLex knowledge-base' editing screen, following the instructions in this Manual.

(ii) If you know what section of an Australian Act you would like to start with, go to the 'Import legislative section (available on AustLII)' and enter the Act name, jurisdiction and section, then start editing as in (i) above.

Select the 'Run Consultation' button when you are ready to test your KB. Please note:

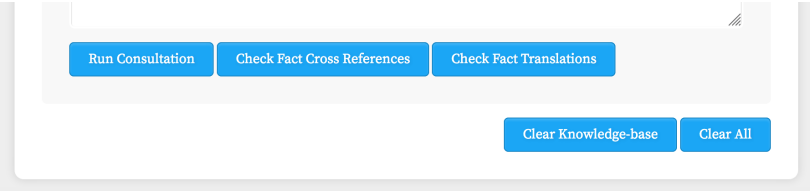


Figure 1.6: DataLex knowledge-base development tools: Running Consultation and Checking rule-base.

- After running your app (successfully or unsuccessfully) you can use your browser to come back to the KB Tools page to make further edits to your KB.
- If your app does not run as intended, use the 'Check Fact Cross References' and 'Check Fact Translations' buttons to run diagnostics (see 2.5 below) to identify problems. Edit and run again.
- If you have spent any significant time developing a test KB, you might want to save a copy in a word processor or other file, in case the browser malfunctions, or if you want to use the KB again after you have quit the browser.

Refinements:

- To add another section of an Act from AustLII to an existing KB, use the 'Import and Append' button after specifying the additional section.
- To over-write and erase an existing KB with a new section from AustLII, use the 'Import and Replace' button.
- To clear an existing KB and start again, use 'Clear knowledge-base'.

1.6 Updates to this Manual

This PDF version of the Manual will be updated periodically. In between PDF editions, incremental updates may be made to the wiki version of the Manual located on AustLII Communities at <http://austlii.community/wiki/DataLex/DataLexDeveloperSManual>.

2 Rule-based inferencing (I): Knowledge-bases and rules

2.1 Introduction

The DataLex inferencing software is an Internet-based expert system shell for the development of inferencing systems (sometimes called 'expert systems') in the legal domain. It may be used to develop systems incorporating rule-based inferencing (discussed in this Chapter and the two following chapters), example or case-based inferencing (Chapter 7), and automated document assembly (Chapter 6). The User Interface Manual is in Chapter 8.

2.1.1 Levels of complexity with DataLex

DataLex is very simple to use to create small practice expert systems, at least for most types of statute-based applications. This is because all you have to do, to get a small system up and running, is to paraphrase a section or two of an Act into a somewhat strict logical form, using logical connectors such as IF, THEN, AND and OR. The result is a knowledge-base, expressed in DataLex's 'English like' knowledge representation language. The DataLex inference engine then does the rest, running your knowledge-base to generate a dialogue with the user, asking questions and giving answers. You don't write any of the questions or answers – DataLex generates them automatically from your knowledge-base.

However, while DataLex can be used easily by relying only on a small number of its features, the DataLex engine has a very powerful and complex range of features which can be used as you proceed to develop more sophisticated applications.

The easiest way to understand how a DataLex knowledge-base is written is to study the examples given in this and the following Chapters, and then to use this Manual to explain features that you don't understand fully.

2.1.2 Main features

The main features of DataLex are:

- a 'quasi-natural-language' or English-like syntax, which encourages isomorphism (similarity between the structure of a knowledge-base and source legal documents), transparency (purpose of rules is relatively obvious) and rapid prototyping (easy to get small systems running);

- rules of any degree of complexity may be written, using propositional logic;
- backward and forward chaining rule-based inferencing;
- conventional procedural code including mathematical calculations;
- a form of reasoning by analogy, or example-based reasoning; and
- a document generation facility.

DataLex is therefore a fairly versatile tool with which a variety of inferencing applications may be created.

2.1.3 *User commands and the DataLex User Interface Manual*

The *DataLex User Interface Manual*, in Chapter 8 of this Manual, explains the interface to DataLex applications when they are running, from a user perspective. It should be read either before or in conjunction with this Chapter.

2.1.4 *Developing a DataLex application – Where is the developer's interface?*

See Chapter 1, sections 1.4 and 1.5 for the two ways to do this.

The developer's interface for DataLex applications is primarily within the DataLex section of the AustLII Communities environment <http://austlii.community/wiki/DataLex/>. There is also a development environment located outside AustLII Communities which is used for teaching and development of test knowledge-bases, the 'DataLex Knowledge-base Development Tools' <http://www.datalex.org/dev/import/> .

2.2 *Knowledge-bases and rules*

A knowledge-base is a set of declarations, so called because they 'declare' items of knowledge about a subject area. This type of programming is therefore called 'declarative' programming, in contrast to 'procedural' programming, which is of the form 'first do this step; then do this step ....'. The author of a DataLex application therefore creates a 'knowledge-base' rather than a 'program'.

2.2.1 *Knowledge-bases as sets of rules*

The most important category of declarations in DataLex is rules (so knowledge-bases are often called rule-bases). A knowledge-base, at its simplest, is therefore a set of rules. When the rule-base is 'run' by DataLex it attempts to find the truth of a specified fact.

It does this by going to a rule which has that fact as its conclusion and examining each of the premises of that rule to determine whether the conclusion of the rule is true. In evaluating the premises of a rule DataLex uses any other rules which have any of the premises as their conclusion. DataLex repeats this process along each branch of reasoning until it reaches a premise for which there is no rule to derive a conclusion. At this point, DataLex interrogates the user about the truth of the premise. It does not generally matter, therefore, in which order the rules occur. Rather, DataLex searches the knowledge-base for relevant rules in relation to each fact or premise which it is evaluating.

2.2.2 Form of a simple rule

In its simplest form, a rule contains four elements:

(i) **the keyword 'RULE'**, indicating the start of a new rule (in the absence of any specification otherwise, the rule will be both backward chaining and forward chaining);

(ii) **the name of the rule** (usually just the name of the Act and section that it paraphrases); The name of a rule should differ from that of any other rule in the rule-base;

(iii) **the keyword 'PROVIDES'**, indicating the start of the body of the rule; and

(iv) **the statement(s)** which make up the inferencing content of the rule (one or more statements). Statements consist of declarations. One of the simplest forms of a statement is 'IF condition THEN conclusion'.

The simplest syntax for a rule is therefore as follows:

RULE name **PROVIDES** *statements*

The example below shows a rule with one moderately complex set of statements:

```
RULE Freedom of Information Act 1982 (Cth) s11 PROVIDES
a person has a legally enforceable right under s11 to obtain
  access to a document ONLY IF
        s11(a) applies OR
        s11(b) applies
```

2.3 Content of rules - keywords and descriptors

Knowledge-base rules consist of keywords and descriptors. *Keywords* are used to join together, in a logical form, a number of *descriptors*, which are simply terms or phrases used to describe some object, event etc.

2.3.1 *Keywords*

Keywords give rules the logical structure used by DataLex to draw inferences. They are written in FULL UPPER CASE so DataLex can distinguish them from their equivalents in ordinary words (which may occur in descriptors).

Some examples of important keywords, or sets of keywords are: ONLY IF; IF .... THEN; IF ... THEN .... ELSE; IS; AND; OR; PLUS; MINUS ; PERSON; THING.

These and other keywords have functions in a DataLex knowledge-base which is very similar to their normal linguistic function as words. This correspondence is a large part of what gives DataLex a 'quasi natural language' or 'English like' syntax.

DataLex is very case-sensitive. It expects keywords to be in FULL UPPER CASE.

There is a list of keywords which may be used with DataLex at the end of Chapter 4.

2.3.2 *Descriptors*

Descriptors may be any sequence of words or symbols but must not contain keywords (although they can contain the lower case versions of them). Descriptors are generally written in lower case, with normal capitalisation. See Chapter 3 for details of how descriptors should be written in order to work best in DataLex.

In the example below, some descriptors used are 'a person has a legally enforceable right under s11 to obtain access to a document', 's11(a) applies' and 'the document is not an exempt document'. These are all attributes.

There are a number of varieties of descriptors, of which the most important are (i) constants, (ii) facts or attributes, (iii) named subjects (a special type of attribute) and (iv) rule names. Each is discussed in detail in the following chapter. A type of attribute used only in documents (see Chapter 6) is called 'text'.

First, however, a simple example of a rule, and how to make it run, is given.

2.4 *Example of a rule – FOI Act s11*

2.4.1 *The section*

The *Freedom of Information Act 1982* (Cth) s11 reads:

11. Subject to this Act, every person has a legally enforceable right to obtain access in accordance with this Act to –

(a) a document of an agency, other than an exempt document; or

(b) an official document of a Minister, other than an exempt document.

A rule-base of 3 rules consisting solely of this section could read as follows. The rules have been (over-)simplified, for demonstration purposes, by ignoring the words 'subject to this Act' in s11.

2.4.2 *A rule-base of 3 rules*

```
RULE Freedom of Information Act 1982 (Cth) s11 PROVIDES
a person has a legally enforceable right under s11 to obtain
access to a document ONLY IF
        s11(a) applies OR
        s11(b) applies

RULE Freedom of Information Act 1982 (Cth) s11(a) PROVIDES
s11(a) applies ONLY IF
        the document is a document of an agency AND
        the document is not an exempt document

RULE Freedom of Information Act 1982 (Cth) s11(b) PROVIDES
s11(b) applies ONLY IF
        the document is an official document of a Minister AND
        the document is not an exempt document
```

2.5 *Running and de-bugging a DataLex application*

Text, such as that above, is all that is needed for a valid knowledge-base. The knowledge-base can be invoked as a DataLex session by selecting the 'Run Consultation' button.

If a knowledge-base does not behave as intended, go back to the editing page, edit the rule, and run it again. The main purpose of the type/paste window on the manual start page is to allow the developer to test minor changes to rules without having to create a new web page each time in order to do so.

2.5.1 *Debugging*

In addition to the 'Run Consultation' button, there are two additional buttons which allow you to check for some types of errors in your knowledge-base, either before you try to run it, or after you so, and it does not perform quite as expected. They are 'Check Fact Cross References' and 'Check Fact Translations'.

There is also another debugging tool that can be used while the application is running, Verbose Mode (see 8.10).

2.5.2 *Check Fact Cross References*

Use of similarly named but not identically named attributes is one of the main causes of errors in YSH knowledge-bases, particularly where rules which are supposed to chain do not do so. The ' Check Fact Cross References' button allows you to check for such errors.

The 'Check Fact Cross References' button causes each fact/attribute to be printed (in alphabetical order - except where it begins with an hypertext link) showing the names of rules which set (*) and rules which use (-) the attribute. Named subjects are also listed.

Use of similar but not identical attributes is one of the main causes of errors in DataLex. The 'Check Fact Cross References' button allows you to check for such errors.

2.5.3 Check Fact Translations

Use of the 'Check Fact Translations' button enables you to check that your attributes are expressed correctly.

For each attribute in the knowledge-base, in the order in which they occur, it shows: (i) prompts (questions); (ii) a translation in positive form; and (iii) a translation in negative form. For example, the interrogative, positive and negative translations of the attribute 's11(a) applies' are as follows:

```
-Does s11(a) apply?
-S11(a) applies.
-S11(a) does not apply.
```

Use the 'Check Fact Translations' button to check that your attributes are expressed correctly.

2.6 Some style guidelines for DataLex applications

Although DataLex is designed to be fairly flexible, it is worth bearing in mind the following guide-lines for developing rule-bases:

2.6.1 Simplicity

Try to aim for simplicity wherever possible. Complicated kludges and workarounds detract from the readability of the code and can have unexpected repercussions, particularly when the knowledge-base is later expanded or changed. Don't use facilities simply because they are available.

2.6.2 Isomorphism

Where the knowledge-base represents rules from a legal source document such as a piece of legislation, try to directly translate the statutory rules into DataLex rules, observing as far as possible the order and grouping of the legislative rules, and adding as little interpretation as possible. Keep other rules, such as interpretation or 'common sense' rules which do not derive directly from the legislation, in a separate part of your rule-base.

2.6.3 Small rules

Avoid large and complicated rules. Small rules are easier to understand and will assist with automatic explanations.

2.6.4 Attribute Names

Include the legal basis for attributes in their descriptors, as in *the layout is in "material form" as defined in s.5*. This will make for more meaningful explanations. Avoid using unnecessarily long descriptors. These make for convoluted questions and explanations. Do not use the translation and prompt options unnecessarily. Try

changing the attribute name to get DataLex to handle it properly, *first*. Avoid use of embedded attributes.

2.6.5 Rule Types

Use only the default rule type unless you have a good reason for doing otherwise. Forward chaining rules and daemons should generally only be used to alter the operation of rules encompassing knowledge rather than to embody knowledge themselves.

2.6.6 Declarative Representation

Do not represent knowledge procedurally using DETERMINE and CALL statements except where unavoidable. Avoid being concerned about the actual operation of knowledge-rich rules and instead concentrate on *describing* the item of knowledge with which you are dealing.

2.6.7 Comments

Avoid relying on comments to understand your code. The code should largely be transparent. However, you can use comments to indicate what legislative provisions you have omitted from your DataLex representation.

3 Rule-based inferencing (II): Descriptors, keywords and expressions

Rules in knowledge-bases consist of keywords and descriptors. *Keywords* are used to join together, in a logical form, a number of *descriptors*, which are simply terms or phrases used to describe some object, event etc. *Descriptors* may be any sequence of words or symbols but must not contain keywords (although they can contain the lower case versions of them). Descriptors are generally written in lower case, with normal capitalisation. The most important types of descriptors, discussed in this Chapter, are (i) constants, (ii) facts or attributes, (iii) named subjects (a special type of attribute) and (iv) rule names.

3.1 Attributes (or 'Facts')

An attribute is any descriptor (a sequence of words or symbols which does not contain a keyword) which is not a constant (see below). The purpose of attributes is to hold values which are determined during the evaluation of a knowledge-base. Every sequence of text in a knowledge-base which is not a keyword or a constant must therefore make sense as something which has a value. (Chapter 6 explains an exception where text follows the keyword TEXT)

Attributes are also referred to as 'facts' in the DataLex error messages.

Attributes and other descriptors have a maximum length of 256 characters.

3.1.1 Consistent naming of attributes

Consistent naming of attributes, including consistency in capitalisation and punctuation, is vital to DataLex's operation. DataLex does not forgive inconsistency.

Lack of consistency is the principal cause of DataLex applications running other than as expected. Use the 'Check Fact Cross References' button to check for possible inconsistencies in naming of attributes.

DataLex cannot tolerate inconsistencies in either capitalisation or punctuation.

3.1.2 Boolean (true/false) attributes and their names

The default attribute type is boolean (that is, true/false). When naming boolean attributes, you should choose a name starting with a

subject, then a verb (expressed in the positive or negative) and, optionally, an object.

For example, each of the following is a boolean attribute, correctly expressed:

Subject	*Verb*	*Object*
the claimant	satisfies	s23(1)
the circuit layout	is in	material form
section 9	applies	
section 9	does not apply	to bills of exchange

Table 3.1: Examples of recommended subject/verb/object form

The purpose of the recommended subject/verb/object form is explained below in relation to the generation of questions and explanations (see 3.3 Generating questions and explanations).

3.1.3 Non-boolean attributes - types

DataLex recognises the following attribute *types:*

Type	***Values***	***Example***
BOOLEAN	T/F/U (default type)	See above
INTEGER	whole numbers only	the number of applicants
REAL	fractions accepted	the number of degrees tolerance
STRING	a string of text	the alleged defamatory statement
SEX	M or F	the sex of the claimant
DOLLAR	dollars and cents	the value of the estat
DATE	a date	the date of the intestate's death

Table 3.2: Attribute types recognised by DataLex

Non-boolean attributes are introduced in one of two ways: (i) automatically by use; or (ii) formally by a declaration.

3.1.4 Automatic type recognition of non-boolean attributes

If the first use of an attribute in a knowledge-base requires DataLex to recognise it as something other than boolean, that type is automatically associated with it. From then on, you must use the attribute consistently or an error message will result. In other words, DataLex is able to make an 'intelligent guess' about the type of non-boolean attribute that is intended, based on other aspects of the expression it is first found in. For example, in the expression 'IF the date of arrival IS GREATER THAN 1 May 1977', DataLex is able to work out that the attribute 'the date of arrival' is probably a non-boolean attribute of type DATE, because another date (1 May 1977) appears in conjunction with a relational operator.

3.1.5 Formal attribute type declarations for non-boolean attributes

While DataLex is generally accurate in recognising non-boolean attributes, it sometimes makes an error. This may be avoided or corrected by an explicit declaration of the type of the attribute.

The syntax for formal declarations is:

TYPE *attribute-name*

optionally followed by a list of *translations* and *valid ranges* (discussed below).

For example, to declare attributes to be of the types 'DATE' and 'DOLLAR':

```
DATE the date of the intestate's death
DOLLAR the value of the estate
```

Because of these declarations, or because of automatic recognition, DataLex would only accept responses from a user that were of the specified types.

Attribute declarations should appear outside of rules and procedures. Otherwise, they can appear anywhere in a knowledge-base, provided they appear somewhere in the knowledge-base prior to where the attribute is first used. It is often convenient to group them all at the start.

3.1.6 Range limitation of attribute values [advanced]

If there was a need to further limit the range of acceptable responses from the user (eg to dates only within a specified period, or to amounts less than a certain maximum), then a RANGE statement is available

The syntax is:

RANGE *expression* [**TO** *expression*]

The statement should appear immediately after a fact declaration. It may be used multiple times if there are many valid ranges. Where the optional TO *expression* is used it indicates that the value for the fact should be between the result of the first expression and the result of the second expression. However, in this case the expressions must produce numeric results.

Some examples of RANGE statements are:

```
STRING the name of the intelligence agency
        RANGE "ASIO"
        RANGE "ASIS"
        RANGE "DSD"

DOLLAR the value of the household chattels
        RANGE Ø TO the value of the estate
```

3.1.7 Attribute names for non-boolean attributes

You must choose an attribute name which can be followed by an **'is'** then a value so that DataLex can correctly provide prompts and translations. For example, the non-boolean attribute declarations

given above will correctly result in the following prompts and (when answered) translations:

```
DATE the date of the intestate's death
What is the date of the intestate's death ?
The date of the intestate's death is 1st January 1991.

DOLLAR the value of the estate
What is the value of the estate ?
The value of the estate is $250,000.
```

DataLex cannot tolerate inconsistencies in either capitalisation or punctuation.

3.2 Constants

Whereas attributes have a variable value which is determined during the evaluation of a knowledge-base, a constant has a fixed value. DataLex recognises any of the following descriptors as constants: an **integer** (eg 1000), a **real number** (eg 7.15), a **dollar amount** (eg $950 or $950.00), the words '**true**' and '**false**' (boolean constant) and the words '**male**' and '**female**' (sex constant), a **date** (in any sensible format), and any descriptor placed in *double* quotes (a **string constant**).

DataLex is generally able to automatically recognise constants, and to give them the correct type. If DataLex does not recognise a descriptor as being in any of these categories of constant, it assumes that the descriptor is an attribute.

Constants are used primarily in expressions which use binary operators (eg PLUS; EQUALS; IS LESS THAN; IN) and in assignment statements (see below).

3.3 Generating questions and explanations

One of DataLex's main features is its capacity to automatically generate questions (prompts) by re-parsing the attribute that it is attempting to find a value for, into an interrogative form (ie by re-parsing the part of the rule it is at present evaluating). Similarly, it can provide explanations by re-parsing rules that it has previously evaluated, substituting the values that it has established for those rules.

3.3.1 Automatic generation of questions (prompts) and explanations

Provided that boolean attribute names appear in the subject/verb/object form explained above (see 3.1.2 Boolean (true/false) attributes and their names), or non-boolean attribute names appear in the 'is' form explained above (see 3.1.3 Attribute names for non-boolean attributes), DataLex will normally be able to affect sensible translations automatically, for use during problem

sessions. For the above examples, the following automatic prompts and translations would be generated by DataLex:

```
Does the claimant satisfy s23(1) ?
The claimant satisfies s23(1).
The claimant does not satisfy s23(1).

Is the circuit layout in material form ?
The circuit layout is in material form.
The circuit layout is not in material form.

Does section 9 apply ?
Section 9 applies.
Section 9 does not apply.

What is the date of the intestate's death ?
The date of the intestate's death is 1st January 1991.
```

Use the 'Check Fact Translations' button to check whether sensible prompts and translations are being generated.

3.3.2 Automatic recognition of different forms of the same attribute

DataLex re-parses all boolean attribute names into a consistent positive form for storage purposes, and so recognises different grammatical forms of the same attribute. For example, the following statements all refer to the same attribute:

```
the Act applies
the Act does not apply
the Act does apply
```

It therefore does not matter which form you use in a rule, as DataLex will normally understand that you are referring to the same attribute. In other words, different forms of the attribute can be used in different rules.

3.3.3 Verbs declarations - correcting DataLex's grammar

While DataLex's ability to 'understand' and generate different grammatical forms of the same attribute is reasonably sophisticated, it sometimes makes errors in translating verbs into different tenses.

For boolean attribute names, the most important component of the describing sentence is the verb. DataLex only knows about a small list of very common verbs. For the remainder it simply makes an educated guess (ie it uses a fairly simple set of heuristic rules concerning the behaviour of verbs). DataLex has to be able to locate the verb and transform its plurality and tense.

Where DataLex makes a mistake, its behaviour can be altered by a declaration specifying that a word is a verb and giving the appropriate forms. The syntax for this is:

VERBS { *base~first_person~third_person_singular~past* }

as in:

```
VERBS
        ma~ke~kes~de
        s~ell~ells~old
```

This declares that the forms of 'make' (which has the base 'ma') are 'make' (first person), 'makes' (third person singular), and 'made' (past).

Where another form is the same as the base, just leave that form blank and put two tildes consecutively ie ~~ .

This *verbs* declaration should appear outside of other declarations such as rules and procedures. It is sensible to put a list of all verbs at the start of a knowledge-base. However, a verbs declaration may occur more than once.

Here is a list of common verbs known to cause problems. If you use any of them, you should declare that verb. It would be useful to include this list in any application, in case any of these verbs are used.

```
b~ear~ears~ore
se~nd~nds~nt
t~ell~ells~old
s~ell~ells~old
g~ive~ives~ave
t~ake~akes~ook
ma~ke~kes~de
c~ome~omes~ame
f~ind~inds~ound
b~uy~uys~ought
beg~in~ins~an
br~eak~eaks~oke
br~ing~ings~ought
s~ay~ays~aid
```

Where DataLex cannot recognise a verb, this can sometimes be remedied by putting the word 'will' in front of the verb in the attribute, because DataLex recognises 'will .....' as a compound verb.

3.3.4 *Adding your own attribute translations - PROMPT and TRANSLATE [advanced]*

One of the main purposes of DataLex's automatic re-parsing of rules to produce prompts and explanations is so that there is normally no need to maintain separate bodies of text for each attribute, with all the complications this implies for development and maintenance.

However, if the automatic parsing performed by DataLex is inadequate for some reason, it is possible to 'override' it and to declare what the prompt and translation should be for a particular attribute.

For example, the attribute 'the date of death of the intestate' would normally generate the prompt 'What is the date of death of the intestate?' and the translation would be 'The date of death of the intestate is ....'. This can be altered by adding PROMPT and TRANSLATE statements after an attribute type declaration for the attribute. For example:

```
DATE the date of death of the intestate
        PROMPT when did the intestate die
        TRANSLATE AS the intestate died on <>
```

The use of angle brackets (ie <>) without an attribute name causes the value of the attribute being evaluated to be substituted.

Where an attribute has more than one possible value, different translations for each value may be provided. For example:

```
INTEGER the number of surviving children
        PROMPT how many children survived the intestate
        TRANSLATE 0 AS no children survived the intestate
        TRANSLATE 1 AS one child survived the intestate
        TRANSLATE AS <> children survived the intestate
```

Where no value appears (as in the last TRANSLATE statement above) this is used as the default translation for values which do not match any of the other TRANSLATE statements.

Avoid using your own attribute prompts or translations if possible. DataLex knowledge-bases are easier to maintain if translations are automatic.

3.4 *Named subjects - names of people and things*

Attribute descriptors often contain references to persons and things as their subjects (eg 'the intestate', 'the property'). By default, the generated prompts and translations just use these embedded subject descriptions literally. If you wish, you can have these automatically replaced with names, pronouns and possessives. Subjects which are to be treated in this way are referred to as *named subjects*.

The use of named subjects allows you to instantiate the dialogues that DataLex generates, making them appear much more responsive to the answers you have already given.

Use named subjects wherever possible, as they improve communication.

3.4.1 *Named subject declarations*

Named subjects are a set of special attributes. They are declared in the same way as attributes, but are given the types **PERSON**, **THING** or **PERSONTHING**. When an attribute containing a defined subject is first evaluated, automatic prompts for a subject name and, in the case of persons, the subjects' sex, will be issued. Where the type is *PERSONTHING*, the subject may be either a person or a thing (eg where either a natural person or a company may be a subject). A prompt (*Is x a natural person ?*) will be issued to determine this.

Examples:

```
PERSON the claimant
THING the agreement
PERSONTHING the first party
PERSON the intestate
```

Once a named subject is declared, DataLex will recognise it as a named subject in any subsequent part of the rule-base, without need for any further identification of it as such. Named subjects referred to in other attributes are recognised automatically, and their values are substituted in the other attributes.

For example, where there have been named subject declarations such as the ones above, an attribute in a rule such as 'the claimant has made a statutory declaration concerning the agreement' would generate a prompt such as 'Has John Smith made a statutory declaration concerning the Contract of Insurance?'.

3.4.2 The automatic attribute declarations [advanced]

When a named subject is declared, it results in up to another three automatic attribute declarations. These take the following forms:

the name of subject	(set for all types)
the sex of subject	(set for PERSONS and PERSONTHINGs)
subject **is a natural person**	(set only for PERSONTHINGs)

Table 3.3: Examples of automatic attribute declarations.

These automatically declared attributes can be manipulated just like normal ones. The *types* are STRING, SEX and BOOLEAN respectively. This allows you to work out whether or not a *PERSONTHING* is a natural person, or to force gender as in:

```
RULE the definition of "defacto" PROVIDES
        IF the sex of the intestate EQUALS male THEN
                the sex of the de facto IS female
        ELSE
                the sex of the de facto IS male
```

It also allows you to change the default prompts and translations, as in:

```
PERSONTHING the claimant

STRING the name of the claimant
        PROMPT please enter the claimants' name
        TRANSLATE AS the claimants' name is

BOOLEAN the claimant is a natural person
        TRANSLATE true AS the claimant is a natural person
        TRANSLATE false AS the claimant is a company

SEX the sex of the claimant
        TRANSLATE male AS the claimant is a man
        TRANSLATE female AS the claimant is a woman
```

3.5 Variable attributes [advanced]

An important aspect of DataLex is that it allows legal knowledge to be represented in something approaching English ('quasi natural language' knowledge representation). This is one reason why propositional logic is used as the form of representation, as opposed to predicate calculus. Predicate logic is, however, more powerful. One of its advantages is that it allows rules where there may be a number of instances of an attribute which need to be considered in the one problem session (eg the attribute 'is a child of the intestate' may be satisfied by three children, all of whom may need to be considered).

Variable attributes have been introduced into DataLex as an experimental way of dealing with such problems. However, they detract from the 'English-like' nature of the syntax and should only be used sparingly.

A variable attribute is allowed to contain one (only) variable element, which element is represented as <>. Whenever DataLex encounters this <> symbol in a rule, it looks for instance of the attribute in other rules which are identical except that they have the variable element 'filled in'. These instances of the variability are then 'read into' the rule under consideration. In effect, DataLex creates multiple versions of the rule under consideration, one for each instance of the variable element being satisfied. In any expression containing the <> variable, each instance of the <> variable will be given the same value. DataLex then proceeds to process whichever version of the rule is satisfied on the facts given. A variable attribute is therefore a shorthand way of writing multiple rules with slightly different wordings.

For example, s32(4) of the Copyright Act 1968 (Cth) specifies whether a person is a 'qualified person' in determining whether a work is protected by copyright. Various different timing and other conditions can satisfy the requirements for a 'qualified person'. The rule below shows that only one rule need be written to capture this.

```
RULE Copyright Act 1968 s32(4) PROVIDES
        the author was a 'qualified person' <> under s32(4) ONLY IF
        the author was an Australian citizen <> OR
        the author was an Australian protected person <> OR
        the author was a person resident in Australia <>
```

If the system needs to determine at any time a value for the attribute "the author was a 'qualified person' *at the time the work was made* under s32(4)" (emphasis added), in order to process another rule, the above rule will cause the following questions to be asked:

```
Was the author a 'qualified person'
    at the time the work was made under s32(4)?    [emphasis added]
```

```
Was the author an Australian protected person
    at the time the work was made under s32(4)?    [emphasis added]

Was the author a person resident in Australia
    at the time the work was made under s32(4)?    [emphasis added]
```

If the answer to any of these is 'yes', the rule will fire and the attribute "the author was a 'qualified person' *at the time the work was made* under s32(4)" (emphasis added) will obtain a 'true' value.

Similarly, if the system needs to know a value for the attribute, "the author was a 'qualified person' *for a substantial part of the period during which the work was made* under s32(4)" (emphasis added), the rule will ask the appropriate questions to obtain a value for this attribute.

In other words, one variable rule can be used to obtain values for numerous similar but not identical attributes which have similar conditions for their satisfaction.

Variable attributes should only be used sparingly and with considerable care.

3.6 Expressions - the use of operators

An expression consists of attribute and constant references, connected by operators (types of keywords). Expressions are used to build more complex statements. Attribute names and constants have already been discussed. Operators therefore describe relationships between two attributes (in the case of binary operators), or (in the case of a Unary operator) transform an existing attribute. The available operators (in order of precedence) are:

3.6.1 (Pre) Unary Operators

NOT	boolean NOT
DAY	extract day from date
MONTH	extract month from date
YEAR	extract year from date

3.6.2 (Post) Unary Operators

DAYS	date days multiplier
WEEKS	date weeks multiplier
MONTHS	date months multiplier
YEARS	date years multiplier

3.6.3 Binary Operators

DIVIDED BY	arithmetic division
TIMES	arithmetic multiplication
PLUS	arithmetic addition
MINUS	arithmetic subtraction
IN	relation in (substring)
EQUALS	relational equality

NOT EQUALS	relational inequality
IS GREATER THAN	relational greater than
IS LESS THAN	relational less than
IS GREATEREQUAL THAN	relational greater equals
IS LESSEQUAL THAN	relation less or equal
AND	boolean conditional AND
OR	boolean conditional OR
	(The normal AND and OR; AND has higher binding strength than OR; DataLex ceases evaluation of expressions where an 'AND' condition fails or an 'OR' condition is satisfied, and does not evaluate the other arguments in the expression)
AND/OR	boolean conditional OR (high binding)
	(A special OR with a higher binding strength than AND; use instead of BEGIN-END pairs to ensure the order of evaluation)
AND/WITH	boolean non-conditional AND
OR/WITH	boolean non-conditional OR
AND/OR/WITH	boolean non-conditional OR (high binding)
	(Special AND and OR operators where DataLex continues to evaluate the other arguments in the expression even though an 'AND' condition fails or an 'OR' condition is satisfied; Used to force DataLex to evaluate all alternatives.)

3.6.4 Examples of the use of expressions

```
the year in which the layout was made PLUS 10

the date of death PLUS 50 YEARS

YEAR the date of death

the value of the estate IS GREATER THAN 0
```

4 Rule-based inferencing (III): Rule types, statements and evaluation order

Rule types and statements control the order in which the evaluation of rules, and attributes within rules, take place in a DataLex inferencing session.

4.1 Order of evaluation of rules, and within rules, in inferencing

The main elements of the process whereby DataLex uses a rule-base to draw inferences are as follows:

(i) The evaluation of a particular attribute (eg determining the truth of a fact) is set as DataLex's current 'goal' - see 4.2 below concerning 'LISTED' and 'GOAL' rules for how such goals are determined.

(ii) DataLex uses the rules in its rule-base to infer a value for this attribute, principally through the use of backward chaining and forward chaining reasoning. DataLex uses both backward and forward chaining reasoning, in that rules are first invoked in a backward-chaining fashion whenever an attribute needs to be evaluated in order to determine whether a rule will 'fire'. However, whenever a new attribute value becomes known, all rules using that attribute are silently evaluated (a forward chaining daemon).

However, which rules participate in the backward chaining process and which in the forward chaining process, and how they do so, is determined to some extent by what types of rules they are declared to be - see below concerning *Types of rules.*

(iii) Once a rule is being evaluated for purposes of either backward or forward chaining, the order in which the attributes and statements which make up the rule's content are evaluated depends largely on the order in which they occur.

4.2 Goals - LISTED rules and GOAL rules

DataLex must start its inferencing process by determining what its current goal is. DataLex will attempt to evaluate the first rule in the rule-base, if no rule is declared to be either a GOAL or LISTED. If only one rule other than the first is a GOAL RULE or LISTED RULE, DataLex will start to evaluate that rule. Otherwise, DataLex will start by giving the user a choice between all rules in the rule-base which are declared to be either GOAL or LISTED rules. The chosen rule is then treated as the current goal.

For present purposes, there is no difference between LISTED rules and GOAL rules.

4.2.1 Multiple GOALS

More than one rule may be declared to be a GOAL. When DataLex is invoked it will automatically present the user with a list of the names of all rules specified as GOALS, and ask the user which one is to be evaluated. Names of rules which are GOALS must therefore be sensible enough to appear in a menu of goals.

4.3 Order of evaluation of rules

When the system is inferring a value using backward and/or forward chaining rules, it will evaluate rules in the order in which they appear in the knowledge-base. The order of appearance will not normally have any effect on the outcome of a consultation, but can affect whether questions of the user are asked in a sensible order. More general rules should be declared before more specific ones, where they relate to the same subject matter. Procedures may be declared in any order.

4.3.1 Calling rules [advanced]

All types of rules can be specifically *CALLed*. The syntax is:

CALL *rule/procedure name*

The statements for the named rule or procedure will be executed and control will be returned to the next statement after the CALL. This statement should only be used to control knowledge-rich rules. It should not be part of any rule which itself contains knowledge of any sort.

4.3.2 Rule names

The rule *name* is used to document what the rule is about and to give a point of reference for *calls*. Each rule name should be different. Rule names are essential if a rule is to be a GOAL RULE, because the user

must know which rule they are choosing to evaluate. Rule names are optional but should be used.

Examples of some ways of naming rules:

```
RULE subsistence of copyright PROVIDES ....

RULE Copyright Act s36(1) PROVIDES ....

RULE Copyright Act s36(2) PROVIDES ....

RULE Copyright Act s36(2) [continuation 1] PROVIDES ....
```

4.3.3 *The ORDER declaration [advanced]*

The order of rule evaluation can be controlled by specifying the rule order in an ORDER block, with the syntax:

ORDER *rule-name* {**THEN** *rule-name*}

The main purpose of this is to allow rules to be written in the order in which they appear in legislation, without this necessarily determining the order in which they might fire. An order declaration must appear before the rules named.

4.4 *Types of rules*

4.4.1 *The default type - backward and forward chaining*

The default rule type is both backward chaining and a forward-chaining daemon. So, a rule that starts

```
RULE name of the rule PROVIDES ...
```

Use the default form unless there is good reason not to.

will be both backward and forward chaining, in default of any other specification.

4.4.2 *Declaring other types of rules*

You can alter this rule behaviour by declaring the type of the rule. The possible types are *BACKWARD*, *DAEMON*, *DOCUMENT*, *FORWARD* and *PROCEDURE*. Each is explained below.

To declare that a rule is a particular type, you put the type of the rule before the keyword RULE at the start of the rule. Examples:

```
BACKWARD RULE the name of the rule PROVIDES ...
```

This rule will only be backward chaining.

```
FORWARD RULE the name of the rule PROVIDES ...
```

This rule will only be forward chaining.

4.4.3 *Syntax for rule types*

The rule declaration syntax is:

[GOAL]
PROCEDURE | DAEMON | BACKWARD | FORWARD | RULE
[RULE] [*name*] **PROVIDES** *statements*

4.4.4 *Backward rules*

If a rule is declared to be a *BACKWARD RULE* it is only ever used for backward chaining.

4.4.5 *Forward rules*

FORWARD RULES are only used for forward-chaining. DataLex attempts to evaluate a FORWARD rule when the first attribute needed to execute the rule becomes known. Where necessary, *FORWARD* rules will ask the user for any other attribute value necessary to evaluate the rule (ie they do not operate 'silently' - they ask questions where necessary).

4.4.6 *Daemons*

DAEMONS are like *FORWARD* rules but operate silently (ie they never ask the user for information and will silently fail to file if they need to do so).

4.4.7 *Procedures*

PROCEDURES are not invoked by either forward or backward chaining. Evaluation of a procedure must be invoked explicitly, either by the procedure being called (see 4.3.1 below concerning calls), or by the procedure being declared to be a goal and invoked as a goal.

4.4.8 *Documents*

DOCUMENTS are like procedures but are used to generate documents (see later Chapter 6 concerning *Documents*).

4.5 *Statements*

There are several different types of statements. These include: assignments and assertions (using ONLY IF, IS and ASSERT), conditional evaluation of facts (using IF-THEN and IF-THEN-ELSE statements), conditional looping (using WHILE-DO and REPEAT-UNTIL statements), DETERMINE statements and CALL statements.

For most purposes, conditional evaluation of facts (IF-THEN-ELSE) and assignments and assertions (using ONLY IF, IS and ASSERT) are the only types that need to be used.

4.5.1 Conditional evaluation of facts (IF-THEN-ELSE statements)

IF-THEN-ELSE statements provide for conditional evaluation of attributes. The syntax is:

IF *expression* **THEN** *statement* [**ELSE** *statement*]

expression is evaluated and if true, the *statement* following the **THEN** is executed. If an **ELSE** *statement* is provided and *expression* evaluates false, then the statement following ELSE will be executed.

The ELSE part of the statement is optional.

Examples:

```
IF         it is raining
THEN   you should take an umbrella
ELSE   you should go out
```

4.5.2 Inclusive definitions

Where a statutory definition is only inclusive (ie not exhaustive), the IF-THEN form is appropriate. For example, the definition of 'dramatic work' in the Copyright Act 1968 (Cth) can be represented in part as

```
IF the work is a choreographic  work or other dumb show OR the work
  is a scenario for a script for a cinematograph film
THEN the work is a dramatic work
```

There is no ELSE because many other undefined types of drama may qualify as dramatic works.

One rule can include a number of IF-THEN statements in succession.

4.5.3 Assigning values - Assignments and Assertions (IS, ONLY IF and ASSERT)

Values may be assigned to attributes by use of the **IS** operator (or the equivalent **ONLY IF** operator) or (in the case of boolean attributes) by assertion.

4.5.4 Assertions

An assertion is used to state that an attribute has a true or false value (ie to assert that it is true or false). Assertions can therefore only be used with boolean (true or false) attributes. An assertion statement simply consists of a boolean attribute name expressed in the positive or negative form, *optionally* preceded by the keyword **ASSERT** or **AND**.

For example:

```
the Act applies
```

is the same as

```
ASSERT the Act applies
```

The following are also the same:

```
the corporation is an overseas corporation AND the Act does not apply

ASSERT the corporation is an overseas corporation AND the Act does not apply
```

The **ASSERT** or **AND** keyword should only be used where it is necessary to separate multiple assignments and assertions, or to separate an assignment or assertion from a previous expression. For example:

```
the circuit layout is in material form AND
the circuit layout is an eligible layout
```

4.5.5 Assignments

IS and **ONLY IF** are used to assert that two attributes have identical values (but not that either are true/false), or that an attribute is identical to a constant. They can therefore be used in either of two ways:

attribute **IS** *constant*
attribute1 (unknown) **IS** *attribute2 (known)*

There is no difference between the **IS** and **ONLY IF** operators, but normally the use of **IS** will yield more natural English statements in relation to valued attributes (dates, numbers etc) where **ONLY IF** is more appropriate in the case of booleans (true/false).

4.5.6 Syntax for assignments and assertions

The syntax for assignments and assertions is:

[**AND** | **ASSERT**] *attribute* **IS** *expression*

or

[**AND** | **ASSERT**] *attribute* **ONLY IF** *expression*

Where an ELSE statement is merely the negation of a THEN statement, this is exactly the same as an ONLY IF statement (which is preferable as it is more understandable). For example,

```
IF         it is raining
THEN   you should take an umbrella
ELSE   you should not take an umbrella
```

would be better expressed as

```
you should take an umbrella ONLY IF it is raining
```

4.5.7 *DETERMINE Statement*

The DETERMINE statement allows for control over attribute evaluation. The syntax is:

DETERMINE [IF] *attribute*

The effect is to cause DataLex to determine a value for the attribute by first evaluating any relevant backward chaining rules (commencing with any which have *attribute* as a conclusion), and then, if necessary, prompt the end-user for a value.

The DETERMINE statement is sometimes useful as part of a GOAL RULE. It allows the user to simply specify that DataLex should attempt to evaluate a particular attribute. For example, the FOI example given earlier could commence with a rule including the statement:

```
DETERMINE IF a person has a legally enforceable right under
  s11 to obtain access to a document
```

However, this procedural approach will defeat the purpose of a declarative rule base if mis-used. In the above example, it would provide no advantages.

Avoid the use of DETERMINE statements.

4.5.8 *CALL Statement [advanced]*

The CALL statement allows *rules* and *procedures* to be invoked explicitly. The syntax is:

CALL *procedure-name*

The statements for the named *rule* or *procedure* will be executed and control will be returned to the next statement after the *CALL*. This statement should only be used to control knowledge-rich rules. It should not be part of any rule encompassing knowledge of any sort. CALLs are a procedural device which detracts from the declarative nature of the knowledge-base. They are valuable mainly for document generation, which is inherently procedural (see Chapter 6).

Use of CALLs should generally be avoided (except in DOCUMENT rules).

4.5.9 *WHILE-DO and REPEAT-UNTIL Statements [advanced]*

The WHILE-DO and REPEAT-UNTIL statement pairs, provide for conditional looping. The syntax is:

WHILE *expression* **DO** *statement*

and

REPEAT *statements* **UNTIL** *expression*

4.5.10 Use of BEGIN - END pairs

Multiple statements can be grouped by use of a **BEGIN-END** pair. This is the same as using parentheses to group statements.

Example:

```
IF the Act does not apply THEN BEGIN
        the claimant fails AND
        there is nothing more to do
END
```

The use of BEGIN-END pairs is largely unnecessary due to the AND/OR operator (see below).

4.6 Appendix – List of Main Keywords (all types) used by DataLex

4.6.1 Rule types

```
RULE
GOAL
PROCEDURE
BACKWARD
FORWARD
DAEMON
DOCUMENT
EXAMPLE
```

4.6.2 Attribute types

```
BOOLEAN
INTEGER
REAL
DOLLAR
SEX
STRING
DATE
```

4.6.3 Document types

(Documents only)

```
PARAGRAPH
LINE
TEXT
```

4.6.4 Named subject types

```
THING
PERSONTHING
```

4.6.5 Translation operators

```
TRANSLATE - AS
PROMPT
```

4.6.6 Statement operators

```
ASSERT
DETERMINE
IF - THEN - ELSE
ONLY IF
IS
WHILE - DO
REPEAT - UNTIL
CALL
BEGIN - END
```

4.6.7 Expression operators

(Pre) Unary Operators

```
NOT
MINUS
PLUS
DAY
MONTH
YEAR
```

(Post) Unary Operators

```
DAYS
WEEKS
MONTHS
YEARS
```

Binary Operators

```
DIVIDED BY
TIMES
PLUS
MINUS

IN
EQUALS
NOT EQUALS
IS GREATER THAN
IS LESS THAN
IS GREATEREQUAL THAN
IS LESSEQUAL THAN

AND/OR
AND/OR/WITH
AND
AND/WITH
OR
OR/WITH
```

4.6.8 Layout operators

(Documents only)

```
NUMBERED
LEVEL
```

4.6.9 Miscellaneous keywords

```
RANGE - TO
VERBS
PROVIDES
ORDER
```

5 Integration of DataLex knowledge-bases with AustLII and other LIIs

5.1 Overview - Integration of DataLex knowledge-bases with their sources

DataLex has five principle features which enable it to be integrated into the web context, and, in particular, into AustLII and AustLII Communities:

1. Automated addition of links to AustLII legislation;
2. Automated addition of links to case law on AustLII or any collaborating legal information institute (LII), or with a citation table in the LawCite citator;
3. Explicit links to any other web resources;
4. Explicit links to searches over AustLII (or other search engine); and
5. Cooperative inferencing using knowledge-bases from multiple pages or sites.

Further forms of integration which are not yet available are the inclusion of links from AustLII primary materials to DataLex knowledge-bases, and the inclusion of knowledge-bases in AustLII search results.

See the articles listed in Chapter 1 for the theoretical advantages of various types of integration discussed in this chapter.

5.2 Automatic links to AustLII legislation

Links to names of Acts (and sections within Acts) that are located on AustLII can be added automatically to your knowledge-base, without the need to create explicit links to those Acts or sections.

To effectively create links to AustLII legislation, observe the following guidelines:

- Each time an Act or section is referred to in the body of a rule, put the full name of the Act and section (for example 'Privacy Act 1988

section 6D'). If the Act name is not included, the mark-up software might not be able to determine in which Act the section is to be found.

- Reference to 'section 5' or 's.5' or 's5' or 's5(3) or 'subsection 5(3)' are effective, but 'paragraph 5(3)' is not – change 'paragraph' to 'section'.
- Automatic links are not created to words defined in Acts. However, as shown below, explicit links can be created to such definitions.
- Automatic links are not (as yet) provided to legislation in jurisdictions outside Australia, but explicit links may be created to such legislation (see below).

5.3 Automatic links to case law

Where a decision in a case is properly cited (either by a neutral citation or proprietary citation) in the name of a rule, or in the body of the rule, this will result in the automatic creation of a hypertext link to either (i) the text of the decision, if the decision is included in AustLII or another collaborating LII (eg NZLII, BAILII, HKLII, PacLII, SafLII, CanLII), or (ii) the LawCite citator, if the decision has a citation table there. The LawCite record for a decision can also be accessed from that decision.

Links to these cases are available in relevant reports and explanations, and to provide assistance when the user is answering questions relevant to a case. For example, in the Finder KB application, when the user is asked about the finder of a chattel 'Was he the occupier of the premises ?', and responds 'Why?', the system replies 'This will help determine whether or not the situation is similar to Armory v Delamirie [1722] EWHC KB J94.', with a link to the LawCite citator entry.

As discussed in Chapter 7, with EXAMPLE rules based on decisions in particular cases, it is particularly important that a full title and citation for the decision be included in the title of the EXAMPLE. Automatic links to cases in Reports means that the user can go to the cases cited in the Report, in order to assess whether they agree with the suggestions for following and distinguishing particular cases given in the Report. In making such a decision they can inspect not only the text of the suggested cases, but also the LawCite record for each of the suggested cases in order to determine whether there are subsequent cases that have a bearing on the suggested cases (and may have been decided after the knowledge-base was written). For discussion of the value of such facilities, see the article 'Utilising AI in the Legal Assistance Sector – Testing a Role for Legal Information Institutes' cited in Chapter 1.

5.4 Explicit links in a knowledge-base (the LINK ... TO ... keywords)

In addition to automatic links to AustLII, specific links can be specified in the knowledge-base. The keywords LINK and TO are used to specify in a knowledge-base that a particular word or phrase is always to appear as a hypertext link to a particular URL. This is very useful for creating links to definitions or cases.

LINK ...TO ... can be used to create links from a knowledge-base to anywhere on the world-wide-web, not just to AustLII.

5.4.1 Example

```
LINK document of an agency TO
  http://www.austlii.edu.au/au/legis/cth/consol_act/foia1982222/
    s4.html#document_of_an_agency

RULE Freedom of Information Act 1982 (Cth) s11(a) PROVIDES
s11(a) applies ONLY IF
        the document is a document of an agency AND
        the document is not an exempt document
```

5.5 Stored searches from DataLex knowledge-bases

It is also possible to use LINK ...TO ... to create links from a knowledge-base to a stored search over AustLII, or over any other web-based search engine.

5.5.1 Example

To link to a search over AustLII for the phrase 'official document of a Minister':

```
LINK official document of a Minister TO
  http://www.austlii.edu.au/cgi-bin/sinosrch.cgi?
    method=auto&query=%22official+document+of+a+Minister%22

RULE Freedom of Information Act 1982 (Cth) s11(b) PROVIDES
s11(b) applies ONLY IF
        the document is an official document of a Minister AND
        the document is not an exempt document
```

5.6 'Co-operative inferencing' – knowledge-bases in multiple locations

'Co-operative inferencing', as we are tentatively calling it, is an innovative aspect of DataLex. It allows different knowledge-base developers to place knowledge-bases on any web page anywhere in cccse other knowledge-bases located elsewhere on the web which they specify are to be 'included'. In this sense, knowledge-base

development becomes a 'co-operative' activity where developers can contribute their small (or not so small) knowledge-bases to a larger enterprise.

5.6.1 The INCLUDE keyword

The use of the keyword INCLUDE in a knowledge-base, followed by the URL of another page containing a DataLex knowledge-base, will cause the second knowledge-base to be loaded with the first knowledge-base, and the two run together.

More than two knowledge-bases can be declared to be INCLUDEd. There is no limit on the number.

It does not matter if an INCLUDEd knowledge-base INCLUDEs the knowledge-base that INCLUDEd it - ie DataLex does not go into an endless loop loading the same knowledge-bases.

It is useful to make the URLs of INCLUDEd knowledge-bases live links, so that users of a knowledge-base can conveniently view all knowledge-bases which are to be included in a consultation. See the 'FOI s11 (start here)' knowledge-base for examples.

5.6.2 Example

To include a KB 'DefinitionOfDocument' in the evaluation of this freedom of information KB, so that the attribute 'the item requested is a document' will be evaluated:

```
INCLUDE http://austlii.community/wiki/DataLex/DefinitionOfDocument
GOAL RULE Access to documents under Freedom of Information Act 1982 (Cth)
   s11 PROVIDES
        the person applying does have a legally enforceable right under
             s11 of the Freedom of Information Act 1982 to obtain access
                  to the document requested ONLY IF
                           the item requested is a document AND
Freedom of Information Act 1982 s11 (1)(a) applies AND/OR
Freedom of Information Act 1982 s11 (1)(b) applies
```

5.6.3 At least one GOAL rule must be specified if INCLUDE is used

As in the example above, you must specify which rule is the GOAL RULE that is to start the consultation, because the operation of INCLUDE means that you cannot be certain which rule DataLex will consider is the first one appearing in your knowledge-base.

If more than one GOAL RULE is specified in a set of 'co-operative' knowledge-bases, the user will be given a choice of which rule is to start the consultation. GOAL RULEs may be declared in any knowledge-base.

6 Document assembly using DataLex

DataLex also includes an automated document generation (or 'assembly') component. This aspect is not yet developed fully. The features described below are sufficient to generate simple documents.

6.1 DOCUMENT rules

Documents may be generated by declaring rules of type *DOCUMENT*. Normally, one DOCUMENT rule will generate one paragraph of a document, and a group of rules can be used to generate all the clauses of a legal document. DOCUMENT rules differ from other types of rules only in that the statements *PARAGRAPH* and *TEXT* are available to *write* paragraphs to documents. The syntax is discussed below under the heading *Statements*.

Document rules are only ever effective if they are declared a *GOAL* rule or if explicitly *called* (via the *CALL* statement) from other rules.

6.1.1 DOCUMENT rules as goals

If the GOAL rule is a DOCUMENT rule, the usual report generated by a consultation is replaced by the generated document (ie no report is generated).

6.1.2 Example - a clause of a will

The following example is a DOCUMENT rule for one clause of a will, with two alternative conditional forms of the clause. The elements of the example are explained below.

```
DOCUMENT Revocation PROVIDES
IF all former testamentary disposition are to be revoked THEN
        NUMBERED PARAGRAPH I revoke all former testamentary dispositions.
ELSE
        NUMBERED PARAGRAPH I revoke all former testamentary dispositions
        except clause(s) <list of clauses from the old will which are to be
        saved> of my testamentary disposition dated <the date of the old will>
        which clause(s) I hereby confirm.
```

6.2 Text generation statement types - PARAGRAPH and TEXT

The two statement types PARAGRAPH and TEXT allow text to be added to documents from rules of type *DOCUMENT*. They have no effect in non-document rules, and should not be used in such rules. The syntax for these special types of document statements is:

[NUMBERED] [LEVEL *number*] [**PARAGRAPH** | **TEXT**] *text*

6.2.1 The text argument and embedded attributes

The *text* argument is a piece of text (of any length) to be generated as part of the document being assembled if the conditions of the rule are satisfied. A *text* argument may include embedded attributes, but is not in itself an attribute, so DataLex does not attempt to obtain a value (true/false) for it. DataLex recognises that a piece of text is *text*, not an attribute, because it is preceded by the statement type PARAGRAPH or TEXT.

However, DataLex does attempt to obtain a value for any attributes embedded within the text, provided that those attributes are enclosed in angle brackets (ie <*attribute*>).

For example, the following statement would cause all of the text after 'PARAGRAPH' to be printed in a new paragraph. The values of the embedded attributes (the attributes within angle brackets ie <>) will be obtained from the user in a dialogue (see below).

```
PARAGRAPH I revoke all former testamentary dispositions
except clause(s) <list of clauses from the old will which are to be
saved> of my testamentary disposition dated <the date of the old will>
which clause(s) I hereby confirm.
```

The example given above and on the previous page will generate the following dialogue:

```
1) Are all former testamentary disposition to be revoked ?
   ** n

2) What is list of clauses from the old will which are to be saved?
   ** 1, 5 and 17

3) What is the date of the old will ?
   ** 1 May 1993

   REPORT

   1. I revoke all former testatmentary dispositions except clause(s)
       1, 5 And 17 of my testamentary disposition dated 1 May 1993 which
       clause(s) I hereby confirm.
```

6.2.2 *Differences between PARAGRAPH and TEXT*

The difference between the two types of statements is simply one of layout: the *PARAGRAPH* statement places an HTML paragraph marker before the text (ie a carriage return and a blank line) and *TEXT* just inserts a space (ie no new line).

For example, the statements:

```
PARAGRAPH I revoke all former testamentary dispositions.

I give all my property to my husband.
```

will be generated as:

```
I revoke all former testamentary dispositions.  I give all my property to my husband.
```

The correct code to cause the second sentence to be a new paragraph is:

```
PARAGRAPH I revoke all former testamentary dispositions.
PARAGRAPH I give all my property to my husband.
```

PARAGRAPH must be used to cause a new paragraph of text to be included in a document. It is insufficient to simply place new paragraphs or lines in the text argument, as DataLex will ignore these when it generates the document.

6.3 *'Personalising' documents - embedded attributes*

Where a document contains variable information (eg the name of the testator, the value of property, the date of death), this variable information (an attribute) can be included in the *text* of a document statement by embedding the attribute in the text. In the example above, the embedded attribute '<list of clauses from the old will which are to be saved>' will cause the user to be prompted to list those clause numbers, and the numbers will then be included in the generated document. The embedded attribute '<the date of the old will>' will cause the user to be prompted for the value of that attribute.

6.3.1 *Relationship between named subjects and embedded attributes*

DataLex will not recognise that an attribute is embedded in text just because it has been declared to be a named subject. For example, the declarations

```
DATE the date of the old will
STRING list of clauses from the old will which are to be saved
```

will *not* cause DataLex to ask the user for values in a rule where angle brackets have been omitted, such as

```
PARAGRAPH I revoke all former testamentary dispositions
        except clause(s) list of clauses from the old will which are to be
        saved of my testamentary disposition dated the date of the old will
        which clause(s) I hereby confirm.
```

However, merely putting an attribute in angle brackets does not give it a type - to do so it is necessary to declare it as a named subject as well. For example, in the dialogue above, an answer 'a few weeks ago' to the question 'What is the date of the old will ?' will be accepted. In contrast, if the declaration 'DATE the date of the old will' had been made, the following dialogue will occur:

It is necessary to put attributes in angle brackets (< >); merely making them named subjects is insufficient.

```
3) What is the date of the old will ?
   ** a few weeks ago

Please respond with a date.
```

It is preferable to declare all embedded attributes as named subjects, as well as embedding them in angle brackets, so as to ensure that the user always gives the correct type of answer (eg a date).

6.3.2 *Prompts for embedded attributes*

The prompt for an embedded attribute is always 'what is .....?', even if the embedded attribute is a named subject of type PERSON. Therefore, it is necessary to either tolerate questions such as 'What is the testator's spouse?' , or to embed attributes in the form '<the name of the testator's spouse>' so as to get a question 'What is the name of the testator's spouse?'. However, if this second approach is taken, the embedded attribute will not match the named subject 'the testator's spouse' and this may cause other problems when that attribute is used elsewhere. In some cases it may be better to tolerate the awkwardly phrased question.

6.4 *Alternative clauses in a document*

An important element in document assembly is to allow alternative versions of a clause or paragraph or sentence to be generated, depending on the user's circumstances. For example, the structure of the example given above for a clause of a will is as follows:

```
DOCUMENT Revocation PROVIDES
IF all former testamentary disposition are to be revoked THEN
        NUMBERED PARAGRAPH .......Alternative text (1).......
ELSE
        NUMBERED PARAGRAPH ........Alternative text (2)........
```

Because of the use of the IF-THEN-ELSE statement, which version of the clause is generated depends upon the value of the attribute 'all former testamentary disposition are to be revoked'. The user will be prompted for a value for this attribute by being asked 'Are all former testamentary disposition to be revoked?'. If the user answers 'yes', then text (1) will be generated, but otherwise (ELSE) text (2) will be generated.

By the use of IF-THEN-ELSE statements, and any other conditional statements used in DataLex, templates for complex documents may be created.

6.5 *Generating successive paragraphs of a document - use of CALL statements*

The discussion above concentrates on the generation of single paragraphs of documents. To assemble a whole document it is usually necessary to create a GOAL rule which provides an overall procedural order for the creation of the document. For example, in the Will Generator example below, the following GOAL rule is used:

```
GOAL DOCUMENT Last Will & Testament PROVIDES
        the date of execution of the will IS today
        CALL Preamble
        CALL Revocation
        CALL Contemplation of Marriage
```

By use of the CALL statement, this rule calls three other rules in succession, those with the names 'Preamble', 'Revocation' and 'Contemplation of Marriage'. In effect, it provides that this is the correct order of assembly of the clause of this document. The names following CALL must match the names of DOCUMENT rules.

The use of CALL statements may also be made conditional. For example, where a clause generated by a rule named 'Revocation' can only be used if a particular section of an Act applies (eg the Contracts Act s17), then the following CALL statement could be used:

```
        IF s17 Contracts Act applies THEN CALL Revocation
```

6.5.1 *Document generation is essentially procedural*

This use of CALL statements as the basic method of assembling documents means that document assembly with DataLex is essentially procedural rather than declarative. Backward and forward chaining rules will rarely be useful to control the order of assembly of a document, because their normal usage is as rules which fire when needed, rather than in a controlled order (such as occurs with CALL statements). However, as discussed below, the evaluation of attributes used in DOCUMENT rules may trigger the operation of backward and forward chaining rules.

6.6 *Numbering paragraphs*

6.6.1 *The NUMBERED keyword*

If a statement is prefixed with the *NUMBERED* keyword, the paragraph will be numbered automatically.

6.6.2 *The LEVEL keyword*

The optional *LEVEL* keyword is used to control the type of numbering to be employed. *number* must be between 1 and 7 (inclusive). The numbering style at each level is:

1. Level One
(1) Level Two
(a) Level Three
(i) Level Four
(A) Level Five
(I) Level Six
- Level Seven

Levels can be skipped (ie it is possible to go directly, say, from Level One to Level Three).

6.7 *Integration of inferencing and document generation*

One of the main strengths of DataLex as a document generator is that the document generation is fully integrated with forward and backward chaining inferencing. Therefore, where the evaluation of any statutory provision or other legal condition is a precondition for the generation of part of a document, it is only necessary to make the appropriate attribute a condition in the DOCUMENT rule, and DataLex will automatically use backward and forward chaining to interact with other parts of the rule-base.

For example, a statement in a DOCUMENT rule such as

```
IF the Act applies THEN PARAGRAPH ....(text follows)...
```

will cause DataLex to backward chain to evaluate a rule that has 'the Act applies' as a conclusion.

Note that the first DOCUMENT rule must be a GOAL rule or else DataLex will not produce a document.

6.8 *Use of other DataLex features with document assembly*

Some normal DataLex commands do not have any meaningful use when a DOCUMENT rule is being evaluated. The 'Why' command will only result in sensible answers when DataLex is evaluating an attribute in a RULE.

Conclusions from rules are generated during a document generation consultation, and are shown as numbered blue buttons. Explanations (How?) can be shown by selecting a conclusion. If a document is generated by a consultation, no separate Report is also generated – the Document replaces the Report.

The following DataLex functions do operate with document assembly: Facts ('What' command) appear as numbered green

buttons; 'Forget' and 'Forget All' will forget facts and generate alternative documents.

Hypertext links to legislation (automatic links) or to defined terms or other text (explicit links) can be used with document generation in the same fashion as with other DataLex inferencing.

6.9 *Example - a will generator*

See <http://austlii.community/wiki/DataLex/WillGeneratorKB> for the simple will generator reproduced below. Note the following aspects:

- The GOAL Document is largely comprised of procedural steps.
- The attribute 'the person making the Will is legally capable of making a Will' causes the evaluation of the 'Capability' rule, by backward chaining. This rule could be expanded much further.
- The use of embedded attributes such as <list of clauses from the old will which are to be saved>, <the testator/testatrix's fiancee> and <the joint beneficiaries>.

```
DATE the date of execution of the Will
DATE the date of the old Will
INTEGER the maximum number of months within which the wedding must take place
PERSON the person making the Will
PERSONTHING the sole beneficiary
PERSON the sole executor
PERSON the testator/testatrix's fiancee
PERSON the joint beneficiaries

GOAL DOCUMENT Last Will & Testament PROVIDES
IF the person making the Will is legally capable of making a Will THEN BEGIN
        CALL Disclaimer
        CALL Preamble
        CALL Revocation
        CALL Contemplation of Marriage
        CALL Sole Beneficiary
        CALL Attestation END
ELSE the person making the Will should not make a Will

RULE Capability PROVIDES
the person making the Will is not legally capable of making a Will ONLY IF
    the person making the Will is not of sound mind OR
    s6 of the Wills, Probate and Administration Act 1898 applies OR
    the person making the Will is subject to some other form of incapacity

DOCUMENT Disclaimer PROVIDES
PARAGRAPH Disclaimer: This is not a real Will and must not be used as such.
This will does not purport to accurately represent the law of any jurisdictions.

DOCUMENT Preamble PROVIDES
PARAGRAPH This will dated <the date of execution of the Will> is
```

```
made by me <the person making the Will>, of
<the testator/testatrix's address>, <the testator/testatrix's occupation>.

DOCUMENT Revocation PROVIDES
IF all former testatmentary disposition are to be revoked THEN
        NUMBERED PARAGRAPH I revoke all former testatmentary dispositions.
ELSE
        NUMBERED PARAGRAPH I revoke all former testamentary dispositions
        except clause(s) <list of clauses from the old will which are to be
        saved> of my testamentary disposition dated <the date of the old Will>
        which clause(s) I hereby confirm.

DOCUMENT Contemplation of Marriage PROVIDES
IF this Will is to be made in contemplation of marriage THEN
        IF the Will is to be conditional on the marriage actually
        taking place THEN
                IF the person making the Will is domiciled in Western Australia AND
                the person making the Will does not own immovables in other States
                THEN
                        NUMBERED PARAGRAPH This will is made in
                        contemplation of my marriage with
                        <the testator/testatrix's fiancee>.
                ELSE
                        NUMBERED PARAGRAPH This will is made in
                        contemplation of my marriage with
                        <the testator/testatrix's fiancee>
                        and is conditional on the marriage taking place
                        within <the maximum number of months within which the
                        wedding must take place> months.
        ELSE IF the testator/testatrix is domiciled in Western Australia THEN
                NUMBERED PARAGRAPH This will is made in contemplation of my
                marriage with <the testator/testatrix's fiancee>
                but shall not be void if the marriage does not take place.
        ELSE
                NUMBERED PARAGRAPH This will is made in contemplation of my
                marriage with <the testator/testatrix's fiancee>
                but is not conditional on the marriage taking place.

DOCUMENT Sole Beneficiary PROVIDES
IF everything disposed of under the Will is to be left one person THEN BEGIN
              IF the sole beneficiary is over 18 THEN
              NUMBERED PARAGRAPH I give the whole of my estate to
              <the sole beneficiary>  whom  I appoint my sole executor.
              ELSE BEGIN
              NUMBERED PARAGRAPH I give the whole of my estate to
              <the sole beneficiary>
              NUMBERED PARAGRAPH I appoint the <the sole executor>
              as my sole executor. END
END ELSE BEGIN NUMBERED PARAGRAPH I give the whole of my estate in equal
     shares to <the joint beneficiaries>
     NUMBERED PARAGRAPH I appoint the <the sole executor> as my
     sole executor. END

DOCUMENT Attestation PROVIDES
```

```
PARAGRAPH Signed by the testator in our presence and attested by us in the
presence of him and each other.
```

7 Case-based (example-based) inferencing using DataLex

7.1 Example-based reasoning – overview

In addition to rule-based inferencing, DataLex also supports one very limited form of analogous reasoning (also known as 'example-based' or 'case-based' reasoning). This form of analogous reasoning is based on a method of measuring similarity of examples (and drawing conclusions from this) called PANNDA (Precedent Analysis by Nearest Neighbour Discriminant Analysis), developed by Alan Tyree and described in his book *Expert Systems in Law*, Prentice Hall, 1990. See also further explanation below.

The PANNDA inferencing component is included in this version of DataLex primarily to allow experiments to be carried out in (i) quasi-natural language representations of examples; (ii) the integration of rule-based and case-based reasoning, and (iii) the integration of case-based reasoning with hypertext and text retrieval. The inferencing methods used by PANNDA are, in this context, of secondary interest and not the main point of the exercise, although they are of interest in their own right.

7.2 Relationship between examples and rules in DataLex's inferencing

PANNDA is implemented in DataLex as types of rules which are called EXAMPLEs. A set of EXAMPLEs is used, for example, to represent all of the cases on a particular legal question. This legal question will be represented as an attribute which is the conclusion of each EXAMPLE. When DataLex obtains the facts of a problem from the user, it compares the facts of the problem to the facts in the EXAMPLEs, and tries to find which is the 'nearest case'.

When DataLex is trying to infer a value for an attribute and no further rules can be found to assist, it will look to see if the attribute is the subject of an *example set*. Example-based reasoning is therefore only used 'when the rules run out' (to use one well-known formulation).

This type of reasoning is most usefully used when there are a set of cases (or other types of examples) which do not seem to conform

to any obviously discernible rule, but have various factors which recur from case to case (although with different values), and where no single case provides any binding authority.

7.3 Knowledge representation – EXAMPLEs

A set of cases is represented as a set of EXAMPLEs, where an EXAMPLE is a particular type of rule declaration.

An EXAMPLE commences with the keyword EXAMPLE, followed by the name of the EXAMPLE and the keyword PROVIDES. The first EXAMPLE in a set would normally be declared to be a GOAL, but there would normally be little point in declaring other EXAMPLEs to be GOALs. A knowledge-base containing only an EXAMPLE set would treat the first EXAMPLE as a GOAL.

The content of an EXAMPLE is normally an assignment (an expression which uses the ONLY IF keyword), such as:

EXAMPLE Armory v Delamirie [1722] EWHC KB J94 PROVIDES

```
the finder wins ONLY IF
    the finder was not the occupier of the premises AND
    the chattel was not attached AND  ..... [etc]
```

Each EXAMPLE in the set must have the same conclusion (the attribute preceding ONLY IF), or its negation. In the 'finder's cases' example above and below, the common conclusion is the attribute 'the finder wins' (or its negative form 'the finder does not win'). The keywords ONLY IF therefore function in a rather different way in EXAMPLEs than in RULEs. An EXAMPLE could be considered as meaning something like 'An EXAMPLE where the finder wins, Armory v Delamirie, IS the finder was not the occupier of the premises AND the chattel was not attached AND ..... [etc]',

7.3.1 Automatic attributes and example names

It is important that each EXAMPLE be named sensibly. In most instances, the name of a case will be the name of an EXAMPLE (eg Armory v Delamirie [1722] EWHC KB J94).

This name is used to construct three automatic attributes of the form:

```
the situation is similar to example-name;
the situation is on all fours with example-name; and
example-name can be distinguished
```

These automatic attributes are used by PANNDA to generate reports.

7.3.2 *Formal syntax for an EXAMPLE*

The syntax for defining examples which form part of an *example set* is a restricted version of that used for rules:

[GOAL] EXAMPLE [RULE] *name* **PROVIDES**
[IF *expression* **THEN]** *assignment*

The *expression* component of either the *IF* guard or the *assignment* itself, should consist of a number of *relative expressions* separated by an *AND* operator. Each *relative expression* (normally just an attribute descriptor) should represent one significant facet of the example.

The *OR* connector should *not* be used – if you really have to, use *AND/OR* instead.

The *IF-THEN* form should only be used where the attribute about which the example relates is non-boolean.

7.4 *An example of a case representation by EXAMPLEs*

The following is the knowledge representation for one case on the finding of chattels.

```
EXAMPLE Armory v Delamirie [1722] EWHC KB J94 PROVIDES
    the finder wins ONLY IF
        the finder was not the occupier of the premises AND
        the chattel was not attached AND
        the non-finder was not the owner of the real estate AND
        the non-finder was not the owner of the chattel AND
        there was a bailment of the chattel AND
        there was not a term in a lease which mentioned  found items AND
        there was not a master-servant relationship between the parties AND
        the chattel was not hidden AND
        there was not an attempt to find the true owner of the chattel AND
        there was prior knowledge of the existence of the chattel
```

7.5 *Reports generated by DataLex EXAMPLE reasoning*

An example of a simple Report generated by the FINDER KB follows.

```
Mr Sweep wins because the situation is similar to Hannah v Peel
[1945] KB 509 and South Staffordshire Water Co v Sharman
[1896] 2 QB 44 can be distinguished.

The situation is similar to Hannah v Peel [1945] KB 509 because:
Mr Sweep was not the occupier of the premises; Mr Lud was the
owner of the real estate; and there was not a bailment of the chattel.

South Staffordshire Water Co v Sharman [1896] 2 QB 44 can be
distinguished because there was not a master-servant relationship
between the parties.
```

7.6 Principles behind the case-based inferencing component

The underlying mechanism used to handle analogous reasoning is based on Alan Tyree's PANNDA (Precedent Analysis by Nearest Neighbour Discriminant Analysis) algorithm. The theory behind PANNDA is described in A Tyree *Expert Systems in Law*, Prentice-Hall, 1990. For further details of this approach see articles co-authored by Alan Tyree, and references to PANNDA and 'the Finders' cases', in 'The Datalex Project: History and Bibliography' cited in Chapter 1. A key aspect of PANNDA is that each matching attribute is weighted on the basis of how poorly it *divides the example set*, as measured by its inverse variance.

7.6.1 PANNDA inferencing

When the system is about to attempt to infer a value for an attribute using an example set, it first finds all examples which relate to it (that is, all examples where the attribute appears as the target of an assignment). It then infers (or asks the user for) a value for all attributes used in the examples. Finally, it compares each example with the situation described by these attribute values and finds the *nearest* and *furthest* example. The furthest example is the one with the closest facts but giving a different result to the nearest one.

The target attribute is set to the same value as the nearest example. The *similar* or *all-fours* attribute for the nearest example is set to true. If the example is not on all fours, the *distinguished* attribute is also set for the furthest case. All of these attributes (including the target attribute itself) receive sensible explanatory associations (for *how/reporting*). Not all possible supporting attributes are used for explanations. Rather, only *significant* ones are reported (*significant* attributes are those which tend to, in themselves, divide the example set or in this instance have unusual values).

7.6.2 PANNDA's DataLex implementation

The main difference between earlier versions of PANNDA and this one is the use of the quasi-natural language knowledge representation.

The original PANNDA approach has also been extended in several minor respects:

- The original PANNDA algorithm dealt only in boolean facts and outcomes. There has never really been any good reason why the outcomes had to be boolean (they are not used in determining which case to follow or distinguish). Accordingly, this restriction has been dropped in the DataLex implementation.
- DataLex also supports non-boolean facts. The variance for each of these is calculated in the context of the present fact value.

Accordingly, care should be taken with use of equality operators. These should only be used where the fact can only take one of a discrete number of values.

- It is not necessary that each example contains all of the attributes used in other examples. This feature can be used to generalise the effect of an example. The missing attributes become, in effect, *wild*. Such examples, are, of course, much easier to match. Again caution is called for.

7.7 Steps in developing an EXAMPLE set

1. Identify an attribute which cannot be determined in any obvious rule-based way, but for which there are a set of cases or other examples which give a value for that attribute as their conclusion. Treat that attribute as the conclusion attribute of the example set.

2. For each case, identify all the aspects of the case which appear to have some bearing on the outcome of the case (ie the value of the conclusion attribute). This is where legal expertise is involved. Define each of these aspects as a DataLex attribute.

3. Analyse all of the cases to establish (if possible) the value of each of the attributes identified for any of the cases.

4. Represent each case as an EXAMPLE, with values for as many of the attributes as are known for that case. It does not matter that values for some attributes are not known.

7.7.1 RULEs, EXAMPLEs and DOCUMENTS can interact

The values of attributes used in EXAMPLEs (eg 'the finder was not the occupier of the premises', as used in the example below), will be determined (in the first instance) by backward chaining to determine if there is a RULE with that attribute as conclusion. Where a RULE uses an attribute, backward chaining will invoke an EXAMPLE with that conclusion once all RULEs have been exhausted. The same applies where an attribute is used in a DOCUMENT rule.

If an attribute which is evaluated by an example set is intended to be a GOAL, it may be necessary to create a rule along the following lines.

```
GOAL RULE Determine whether the finder wins PROVIDES
        DETERMINE the finder wins
```

7.7.2 Use of hypertext links with EXAMPLE rules

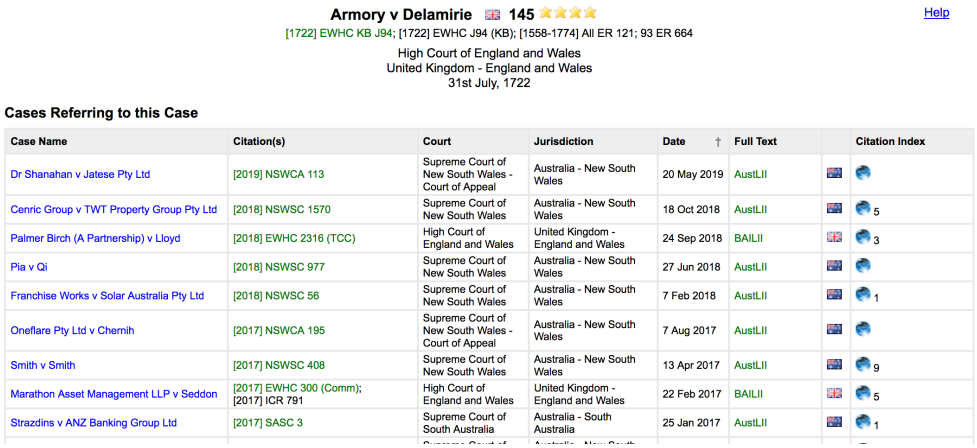
Hypertext links to sources can be used with EXAMPLE rules as with any other rules (as detailed in Chapter 5):

- Automatic links will be made to any properly described Australian legislation;
- Explicit links may be made from any other text using the LINK .... TO .... keywords;
- Embedded searches may be linked to any terms using the LINK .... TO .... keywords, such as to terms in keywords which have been interpreted by case law, like 'bailment' or 'occupier' (as in the Finder KB below).
- Automatic links to properly cited cases, as discussed below.

With EXAMPLE rules based on decisions in particular cases, it is particularly important that a full title and citation for the decision be included in the title of the EXAMPLE. This will then result in the automatic creation of a hypertext link to either (i) the text of the decision, if the decision is included in AustLII or another collaborating LII (eg NZLII, BAILII, HKLII, PacLII, SafLII, CanLII), or (ii) the LawCite citator, if the decision has a citation table there. The LawCite record for a decision can also be accessed from that decision.

Automatic links to cases means, as in the example Report given in 7.5 above, that the user can go to the cases cited in the Report, in order to assess whether they agree with the suggestions for following and distinguishing particular cases given in the Report. In making such a decision they can inspect not only the text of the suggested cases, but also the LawCite record for each of the suggested cases in order to determine whether there are subsequent cases that have a bearing on the suggested cases (and may have been decided after the knowledge-base was written). For discussion of the value of such facilities, see the article 'Utilising AI in the Legal Assistance Sector – Testing a Role for Legal Information Institutes' cited in Chapter 1.

Furthermore, links to these cases are available to provide assistance when the user is answering questions relevant to a case. In the Finder KB example, when the user is asked about the finder 'Was he the occupier of the premises ?', and responds 'Why?', the system replies 'This will help determine whether or not the situation is similar to Armory v Delamirie [1722] EWHC KB J94.', with a link to the LawCite citator entry (Figure 7.1).



Armory v Delamirie 145

[1722] EWHC KB J94; [1722] EWHC J94 (KB); [1558-1774] All ER 121; 93 ER 664

High Court of England and Wales
United Kingdom - England and Wales
31st July, 1722

Help

Cases Referring to this Case

Case Name	Citation(s)	Court	Jurisdiction	Date	Full Text		Citation Index
Dr Shanahan v Jatese Pty Ltd	[2019] NSWCA 113	Supreme Court of New South Wales - Court of Appeal	Australia - New South Wales	20 May 2019	AustLII		
Cenric Group v TWT Property Group Pty Ltd	[2018] NSWSC 1570	Supreme Court of New South Wales	Australia - New South Wales	18 Oct 2018	AustLII		5
Palmer Birch (A Partnership) v Lloyd	[2018] EWHC 2316 (TCC)	High Court of England and Wales	United Kingdom - England and Wales	24 Sep 2018	BAILII		3
Pia v Qi	[2018] NSWSC 977	Supreme Court of New South Wales	Australia - New South Wales	27 Jun 2018	AustLII		
Franchise Works v Solar Australia Pty Ltd	[2018] NSWSC 56	Supreme Court of New South Wales	Australia - New South Wales	7 Feb 2018	AustLII		1
Oneflare Pty Ltd v Chernih	[2017] NSWCA 195	Supreme Court of New South Wales - Court of Appeal	Australia - New South Wales	7 Aug 2017	AustLII		
Smith v Smith	[2017] NSWSC 408	Supreme Court of New South Wales	Australia - New South Wales	13 Apr 2017	AustLII		9
Marathon Asset Management LLP v Seddon	[2017] EWHC 300 (Comm); [2017] ICR 791	High Court of England and Wales	United Kingdom - England and Wales	22 Feb 2017	BAILII		5
Strazdins v ANZ Banking Group Ltd	[2017] SASC 3	Supreme Court of South Australia	Australia - South Australia	25 Jan 2017	AustLII		1

Figure 7.1: LawCite records for *Armory v Delamirie*.

7.7.3 *Example – the 'finder's cases'*

The Finder KB can be accessed from the DataLex Community page on AustLII Communities, or directly to its location at <http://austlii.community/wiki/DataLex/FinderKB>. Note these aspects:

- Each EXAMPLE rule includes in its name a full citation to the case on which it is based.
- The 'trespasser rule' RULE is evaluated before the EXAMLE rules (it should include authority for the proposition it states, but is incomplete).

```
PERSON the finder
PERSON the non-finder

GOAL RULE the finder wins PROVIDES
DETERMINE the finder wins

RULE trespasser rule PROVIDES
IF the finder is a trespasser THEN the finder does not win

EXAMPLE Armory v Delamirie  [1722] EWHC KB J94 PROVIDES
    the finder wins ONLY IF
        the finder was not the occupier of the premises AND
        the chattel was not attached AND
        the non-finder was not the owner of the real estate AND
        the non-finder was not the owner of the chattel AND
        there was a bailment of the chattel AND
        there was not a term in a lease which mentioned  found items AND
        there was not a master-servant relationship between the parties AND
        the chattel was not hidden AND
        there was not an attempt to find the true owner of the chattel AND
        there was prior knowledge of the existence of the chattel

EXAMPLE Bridges v Hawkesworth  (1851) 21 LJQB 75 PROVIDES
    the finder wins ONLY IF
        the finder was not the occupier of the premises AND
        the chattel was not attached AND
        the non-finder was the owner of the real estate AND
```

```
        the non-finder was not the owner of the chattel AND
        there was a bailment of the chattel AND
        there was not a term in a lease which mentioned  found items AND
        there was not a master-servant relationship between the parties AND
        the chattel was not hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was not prior knowledge of the existence of the chattel

EXAMPLE Elwes v Brigg Gas  (1886) 33 Ch D 562 PROVIDES
    the finder does not win ONLY IF
        the finder was the occupier of the premises AND
        the chattel was attached AND
        the non-finder was the owner of the real estate AND
        the non-finder was not the owner of the chattel AND
        there was not a bailment of the chattel AND
        there was a term in a lease which mentioned  found items AND
        there was not a master-servant relationship between the parties AND
        the chattel was hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was not prior knowledge of the existence of the chattel

EXAMPLE Hannah v Peel  [1945] KB 509 PROVIDES
    the finder wins ONLY IF
        the finder was not the occupier of the premises AND
        the chattel was not attached AND
        the non-finder was the owner of the real estate AND
        the non-finder was not the owner of the chattel AND
        there was not a bailment of the chattel AND
        there was not a term in a lease which mentioned  found items AND
        there was not a master-servant relationship between the parties AND
        the chattel was hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was not prior knowledge of the existence of the chattel

EXAMPLE Corporation of London v Yorkwin  [1963] 1 WLR 982 PROVIDES
    the finder does not win ONLY IF
        the finder was the occupier of the premises AND
        the chattel was attached AND
        the non-finder was the owner of the real estate AND
        the non-finder was not the owner of the chattel AND
        there was a bailment of the chattel AND
        there was a term in a lease which mentioned  found items AND
        there was not a master-servant relationship between the parties AND
        the chattel was hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was not prior knowledge of the existence of the chattel

EXAMPLE Moffatt v Kazana  [1969] 2 QB 152 PROVIDES
    the finder does not win ONLY IF
        the finder was the occupier of the premises AND
        the chattel was not attached AND
        the non-finder was not the owner of the real estate AND
        the non-finder was the owner of the chattel AND
        there was not a bailment of the chattel AND
```

```
        there was not a term in a lease which mentioned  found items AND
        there was not a master-servant relationship between the parties AND
        the chattel was hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was prior knowledge of the existence of the chattel

EXAMPLE South Staffordshire Water Co v Sharman  [1896] 2 QB 44 PROVIDES
    the finder does not win ONLY IF
        the finder was not the occupier of the premises AND
        the chattel was attached AND
        the non-finder was the owner of the real estate AND
        the non-finder was not the owner of the chattel AND
        there was not a bailment of the chattel AND
        there was not a term in a lease which mentioned  found items AND
        there was a master-servant relationship between the parties AND
        the chattel was hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was not prior knowledge of the existence of the chattel

EXAMPLE Yorkwin v Appleyard  [1963] 1 WLR 982 PROVIDES
    the finder does not win ONLY IF
        the finder was not the occupier of the premises AND
        the chattel was attached AND
        the non-finder was not the owner of the real estate AND
        the non-finder was not the owner of the chattel AND
        there was not a bailment of the chattel AND
        there was not a term in a lease which mentioned  found items AND
        there was a master-servant relationship between the parties AND
        the chattel was hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was not prior knowledge of the existence of the chattel
```

8 DataLex user interface manual

8.1 Relationship to the previous chapters

Most user interface features are affected by choices by the developer in how the knowledge-base is written. The following extract from a consultation using the ElectKB application on Australian electoral law will be used throughout this chapter to illustrate aspects of the interface.

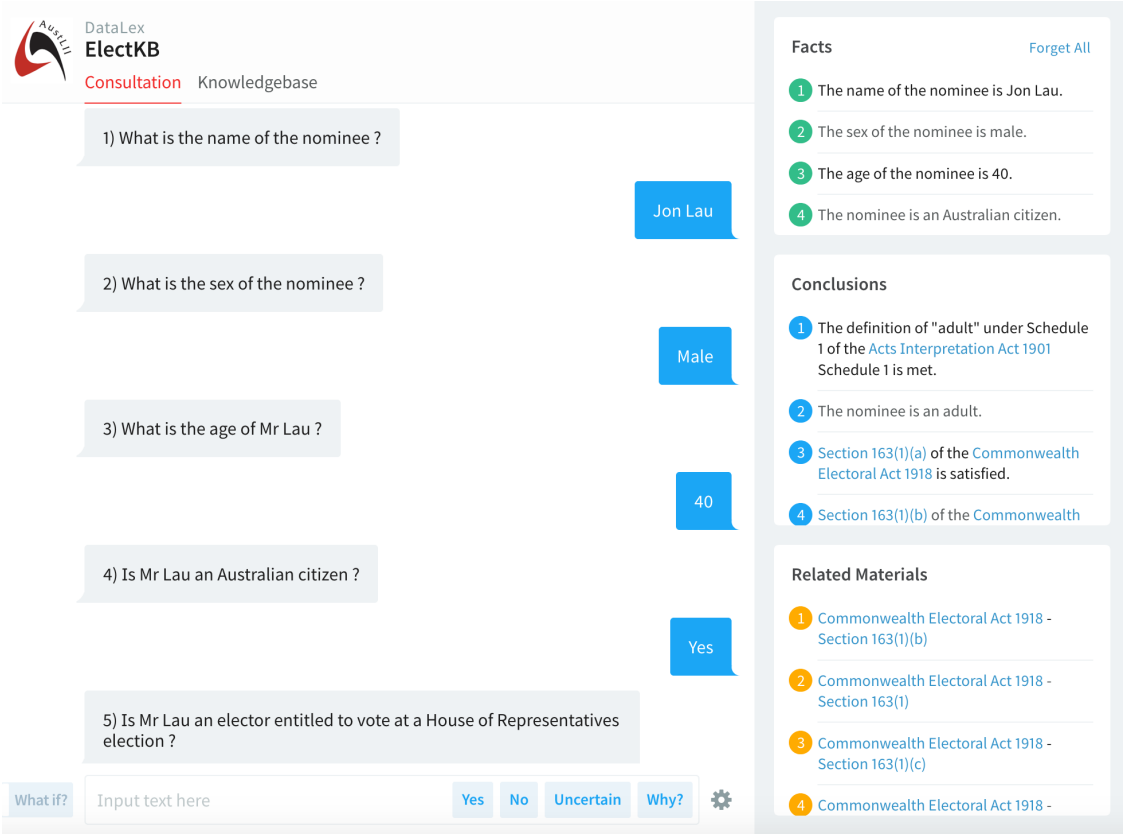


Figure 8.1: DataLex user interface features: *Consultation*, *Facts*, *Conclusions*, *Related Materials*.

8.2 Starting a consultation

A consultation is usually started by the user going to the knowledge-base, and selecting 'Run consultation' from above the KB text. It is possible to create a link to directly invoke the consultation, but this

has the disadvantage (in the normal case) that the user does not see the KB text, including any reservations or caveats that the author may have expressed about it.

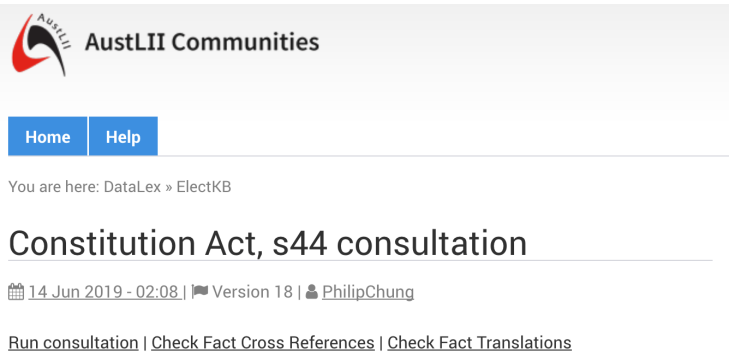


Figure 8.2: Click 'Run Consultation' to start a DataLex Consultation.

8.3 *Choice of goals*

Except in a consultation that has only one goal, it is necessary for the user to select which goal will be evaluated, by selecting the appropriate numbered gray button for the desired goal. A number can be entered instead.

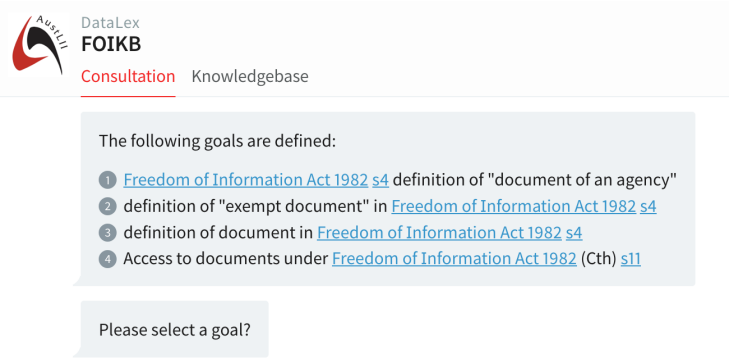


Figure 8.3: Choice of goals at start of Consultation.

8.4 *Answering questions*

Most questions asked by DataLex require a **yes/no/uncertain** response. These can be issued by clicking the relevant button (at the bottom of the screen) or by typing the response into the text field.

Figure 8.4: Answering questions during Consultation.

8.4.1 *Buttons and numbers*

Where a question can be answered by selecting a numbered button, it can also be answered by typing the number and pressing enter.

8.4.2 *Why? – Providing reasons for questions*

The Why? command (at the bottom of the screen) can be given to any question asked when a RULE or EXAMPLE (but not a DOCUMENT) is being evaluated.

In the current user interface, the Why command can be re-issued, in order to show the next attribute on the explanation stack (ie a broader reason why the current question is being asked).

8.4.3 *Hypothetical answers (What-if?)*

The What-if? button (at the bottom of the screen) can be selected in response to any question, in order to test what conclusions or other responses will be generated if the given answer is correct. What-if? must be de-selected in order for the consultation to continue.

8.4.4 *Uncertain answers*

If 'Uncertain' (at the bottom of the screen) is selected in response to any question, the dialogue may continue if a value for that attribute is not essential to a conclusion being reached. If 'Uncertain' is sufficient to require a particular conclusion to be reached, a Report will be generated to that effect.

8.5 *Showing facts (What?)*

All facts known are shown automatically, either as user-provided facts (numbered green buttons) or as inferred facts (numbered blue buttons).

8.6 *Forgetting facts (Forget)*

Selecting a green numbered button will cause that fact to be forgotten, and the consultation to go back to that point in the dialogue.

8.6.1 *Forgetting all facts*

'Forget All' appears at the top right of the list of known facts, and can be selected in order to forget all facts and re-start the consultation. Typing 'forget all' in response to any question will also cause all facts to be forgotten and the consultation to re-start. Also, at the conclusion of the current consultation, the user is asked if (s)he wishes to forget all current facts. The consultation can also be restarted from verbose mode, by selection of the 'Restart consultation' link.

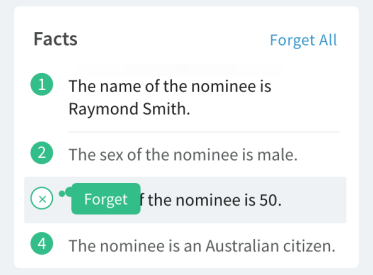


Figure 8.5: Forgetting facts during Consultation.

8.7 Obtaining explanations for conclusions (How?)

Selecting a numbered blue conclusion button will result in an explanation for that conclusion being displayed in a pop-up.

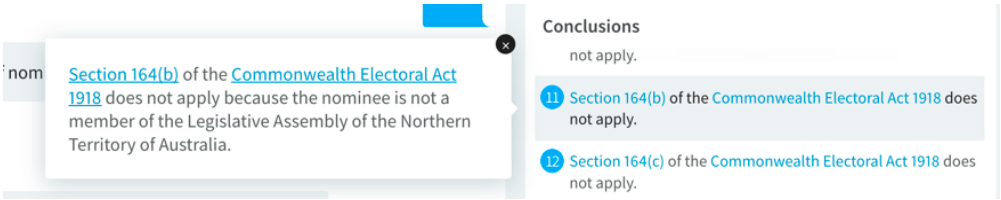


Figure 8.6: DataLex 'How' explanation during consultation.

8.8 Reports

At the conclusion of consultation a Report is generated, setting out all reasoning which is essential to the conclusion which has been reached. Some conclusions reached will not be displayed in the Report, because they were not essential to the final conclusion (these are often conclusions generated by forward-chaining rules). Reports contain such hypertext links as are automatically provided or explicitly linked.

Reports can be downloaded (as RTF, PDF, HTML or TXT), printed, or displayed in a full window.

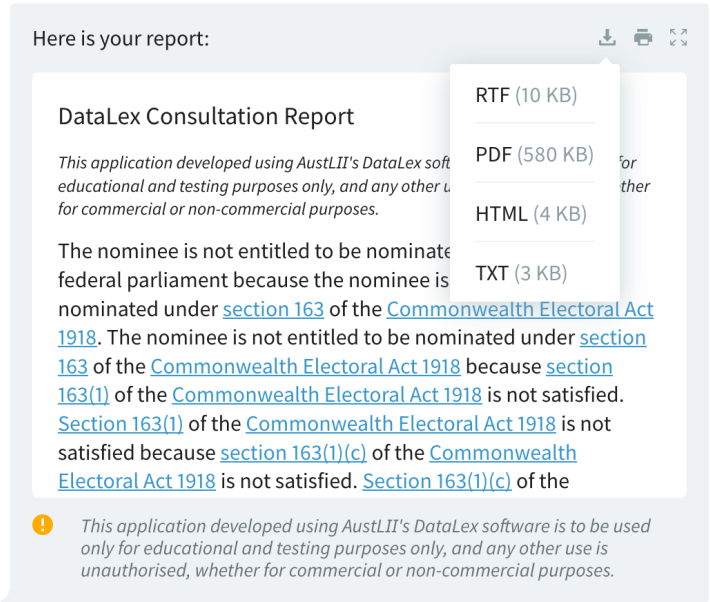


Figure 8.7: DataLex Report generated after a consultation.

8.8.1 Documents generated

Where a document is generated, a Report is not also generated, but conclusions and explanations for them may be viewed (see above).

8.9 Links to sources

Where links are provided in the KB, either automatically or explicitly, to sections of Acts, cases, and other relevant sources of rules, then these links will appear in questions, conclusions, explanations (Why? and How?), Related Materials and Reports.

8.9.1 Returning to the dialogue

Selecting a hypertext link will cause the linked content to appear (i) in the whole window of the consultation, or (ii) alternatively, only in the right-hand panel.

To return to the dialogue either use the back button in situation (i), or the cancel (X) button at the top right of the the right-hand panel in situation (ii).

8.9.2 Related materials

The names of current rules being evaluated (and previous rules evaluated), including any hypertext links to sources contained in those rule names, are shown under 'Related Materials' on the bottom right, and may be selected for display at any time.

8.10 Viewing consultations in verbose mode

If the gear wheel at the bottom right of the consultation interface is selected, the user is given three options for different means of viewing details of the evaluation of the knowledge-base as the consultation progresses.

8.10.1 Viewing the rule being evaluated

In default, the 'Rule' option is displayed, showing the rule(s) currently being evaluated. Selecting 'See more…' under that rule, will display the next rule in the knowledge-base.

8.10.2 Viewing the consultation in Verbose mode

Choosing 'Verbose' mode causes an explanation of why the consultation asks questions, and what it concludes from them, to be displayed. For example:

```
* DETERMINED VALUE FOR the sex of the nominee
* DETERMINED VALUE FOR the age of the nominee
* FORWARD-CHAINING FOR the age of the nominee
* BLOCKED Commonwealth Electoral Act 1918 - Section 163(1)(a)
* FIRING Acts Interpretation Act 1901 Schedule 1
* DETERMINED VALUE FOR the definition of "adult" under Schedule 1 of
  the Acts Interpretation Act 1901 Schedule 1 is met
* FORWARD-CHAINING FOR the definition of "adult" under Schedule 1 of
```

```
  the Acts Interpretation Act 1901 Schedule 1 is met
* FIRING Acts Interpretation Act 1901 Schedule 1
* DETERMINED VALUE FOR the nominee is an adult
* DETERMINED VALUE FOR section 163(1)(a) of the Commonwealth
  Electoral Act 1918 is satisfied
```

8.10.3 Saving the transcript of a consultation

Choosing 'Transcript' gives the user a choice of including in a transcript one or more of 'Conversations', 'Facts', 'Conclusions' and 'Report', and also whether the transcript should imitate the layout of the consultation, or just be in plain text. As yet, the transcript cannot automatically be saved anywhere, and nor can previous transcripts be uploaded in order to save time in entering a long set of facts for testing purposes.

DataLex

Australasian Legal Information Institute (AustLII)
A joint facility of UTS and UNSW Faculties of Law

Level 12, 235 Jones St
Ultimo NSW 2007
Australia

T: +61 2 9514 4921
E: datalex@austlii.edu.au
W: http://www.austlii.edu.au/