AustLII's DataLex
Developers' Manual

Andrew Mowbray • Graham Greenleaf • Philip Chung

April 2021

AustLII's DataLex – Developers' Manual

Version: April 2021

Authors: Andrew Mowbray, Graham Greenleaf and Philip Chung

Andrew Mowbray *is Professor of 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.*

Australasian Legal Information Institute (AustLII)
Level 14, 61 Broadway
Ultimo NSW 2007 Australia

Tel: +61 2 9514 4921
Fax: +61 2 9514 4908
Email: feedback@austlii.edu.au
Web: http://www.austlii.edu.au/

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

Contents

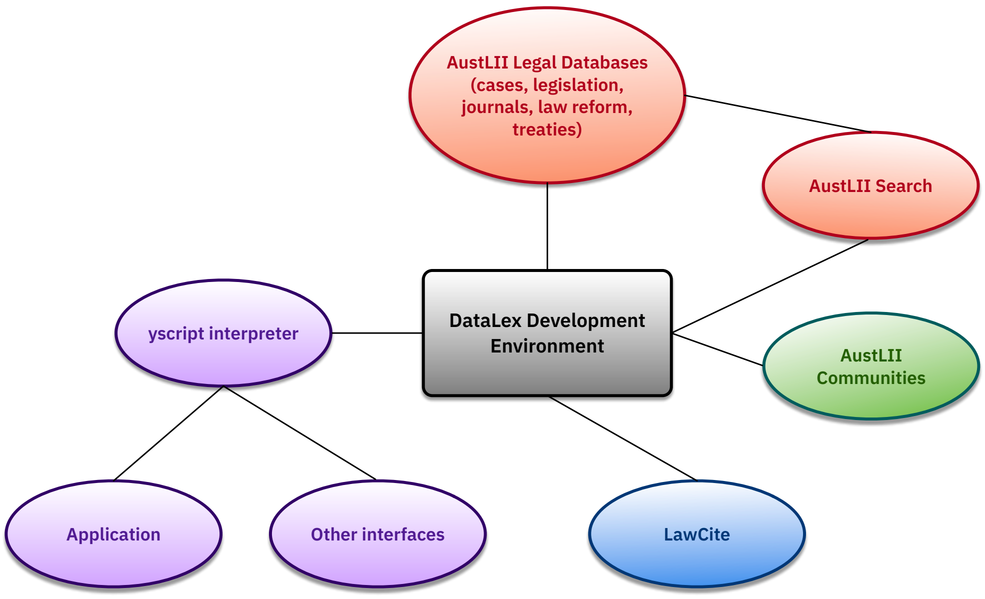
1 DataLex Intelligent Applications Development Environment

1.1 Components of AustLII's DataLex Intelligent Applications Development Environment

AustLII's DataLex is a development environment for building intelligent web-based applications with automated integration of AustLII's legal content. The system is particularly suited to the development of systems that are based on legislation or other rules (Rules as Code) but can be used to construct a range of applications incorporating sophisticated legal reasoning supported by a comprehensive set of legal materials.

DataLex is available via a dedicated development environment as well as being part of the AustLII Communities facility which allows for collaborative and distributed applications development.

The system has the following principal components:


AustLII Legal Databases (cases, legislation, journals, law reform, treaties)
AustLII Search
DataLex Development Environment
yscript interpreter
AustLII Communities
Application
Other interfaces
LawCite

This Manual describes the development and use of DataLex as at April 2021.

1.1.1 The DataLex coding language

DataLex applications are written using a coding language called *yscript* (pronounced “why-script”). yscript is a computer language developed by Andrew Mowbray which allows you to quickly build sophisticated interactive systems that can model legislation, rules and case law. The language supports a declarative rule-based paradigm as well as imperative coding. It also supports analogous reasoning and document assembly. Applications written in yscript can be incorporated as part of systems in various environments using a flexible applications programming interface (API) which can interface with most modern programming languages.

A complete introduction and description of the yscript language is available in the document: *Coding in yscript – a Description of the yscript language* (the “yscript Manual”). This document also includes a very easy to read short tutorial introduction. The coverage of yscript provided below, is a summary only and does not include all features. Experienced developers may just wish to read the yscript Manual. Where there are any differences between this document and *Coding in yscript*, the latter is correct.

1.1.2 The DataLex application development tools

DataLex applications can be developed via a freestanding web-based integrated development environment[1] or, where collaboration is required, within the DataLex section of the AustLII Communities environment.[2]

The development environment provides a number of tools for checking and correcting code. Automated links between application 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.

1.1.3 The DataLex run-time interface

The default DataLex run-time interface 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

[1] DataLex *Development Tools* <http://www.datalex.org/dev/tools/>

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

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 markup and legal search facilities to provide automated links from dialogues, conclusions and reports to the relevant legal sources on AustLII or on other LIIs. This integration between the application code and the legal sources located on LIIs is one of the principal distinctive features of the DataLex approach.

The run-time interface is intended to be used during application development and to demonstrate application functionality. Other interfaces for application deployment can be developed using DataLex or via the yscript API.

The User Manual for the DataLex user interface is in Chapter 8 of this Manual. The DataLex interface has been developed by Philip Chung, Andrew Mowbray and AustLII consultants.

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.
{ *string* }	Curly brackets indicate that contents may be repeated zero or more times. Again, don't type it.

1.3 Theoretical foundations of the DataLex approach

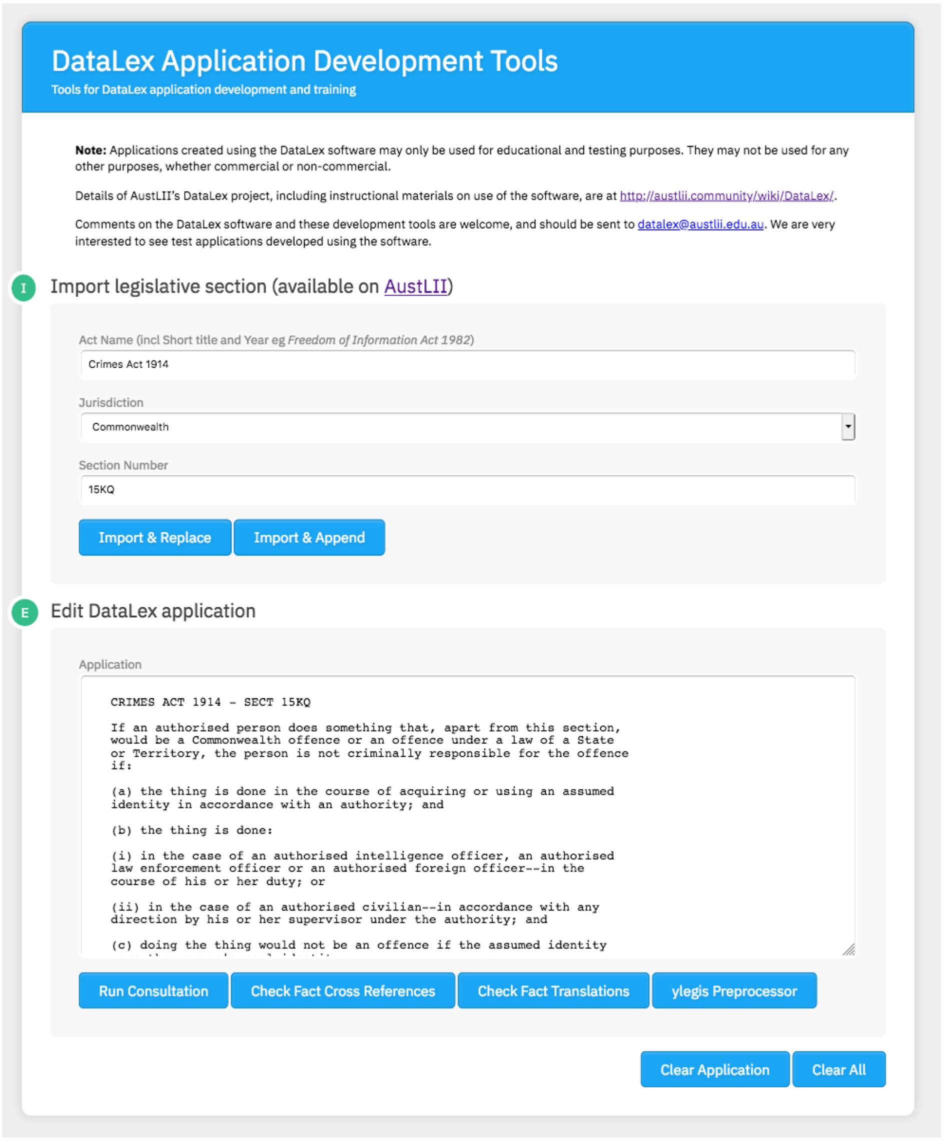
There are a number of articles explaining and justifying the approach taken by the DataLex project. The main articles, and bibliography, are as follows:

- A Mowbray, P Chung and G Greenleaf, 'Utilising AI in the Legal Assistance Sector – Testing a Role for Legal Information Institutes' (April 29, 2019), presented at *LegalAIIA '19*, June 17, 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 324 <https://ssrn.com/abstract=3021452>
- G Greenleaf, A Mowbray, and P.Chung 'The Datalex Project: History and Bibliography' (January 3, 2018). [2018] UNSWLRS 4 <https://ssrn.com/abstract=3095897>

1.4 Creating a new application using the Development Tools

Go to <http://www.datalex.org/dev/tools/>. No login is necessary. This page enables creation and testing of applications using the DataLex software. It does not allow apps to be saved and you will need to keep a separate copy on your own computer.


DataLex Application Development Tools

Tools for DataLex application development and training

Note: Applications created using the DataLex software may only be used for educational and testing purposes. They may not be used for any other purposes, whether commercial or non-commercial.

Details of AustLII's DataLex project, including instructional materials on use of the software, are at http://austlii.community/wiki/DataLex/.

Comments on the DataLex software and these development tools are welcome, and should be sent to datalex@austlii.edu.au. We are very interested to see test applications developed using the software.

I Import legislative section (available on AustLII)

Act Name (incl Short title and Year eg *Freedom of Information Act 1982*): Crimes Act 1914

Jurisdiction: Commonwealth

Section Number: 15KQ

E Edit DataLex application

Application

```
CRIMES ACT 1914 - SECT 15KQ

If an authorised person does something that, apart from this section,
would be a Commonwealth offence or an offence under a law of a State
or Territory, the person is not criminally responsible for the offence
if:

(a) the thing is done in the course of acquiring or using an assumed
identity in accordance with an authority; and

(b) the thing is done:

(i) in the case of an authorised intelligence officer, an authorised
law enforcement officer or an authorised foreign officer--in the
course of his or her duty; or

(ii) in the case of an authorised civilian--in accordance with any
direction by his or her supervisor under the authority; and

(c) doing the thing would not be an offence if the assumed identity
```

There are two ways to start writing an app:

(i) Simply start writing in the 'Edit DataLex application' editing screen, following the instructions about yscript coding 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. You might also want to click on the 'ylegis Preprocessor' button which produces a 'rough cut' of some yscript rules that reflect the section's structure. You can then start editing as in (i) above.

Select the 'Run Consultation' button to test your code, when ready. Please note:

- After running your app (successfully or unsuccessfully) you can use your browser to come back to the KB Tools page to make further edits.
- 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 your code, you might want to save a copy in a word processor or other file, in case the browser malfunctions, or if you want to reuse the code after quitting the browser.

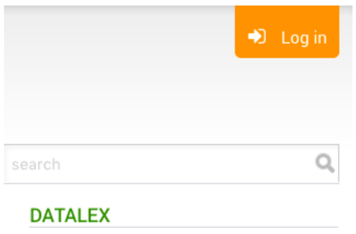
Refinements:

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

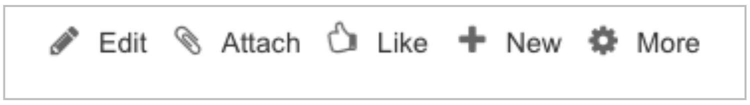
1.5 Creating a new application in the DataLex Community

The DataLex Community (part of the AustLII Communities) is a collaborative closed wiki-like platform used in the DataLex system for creating and editing applications 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.

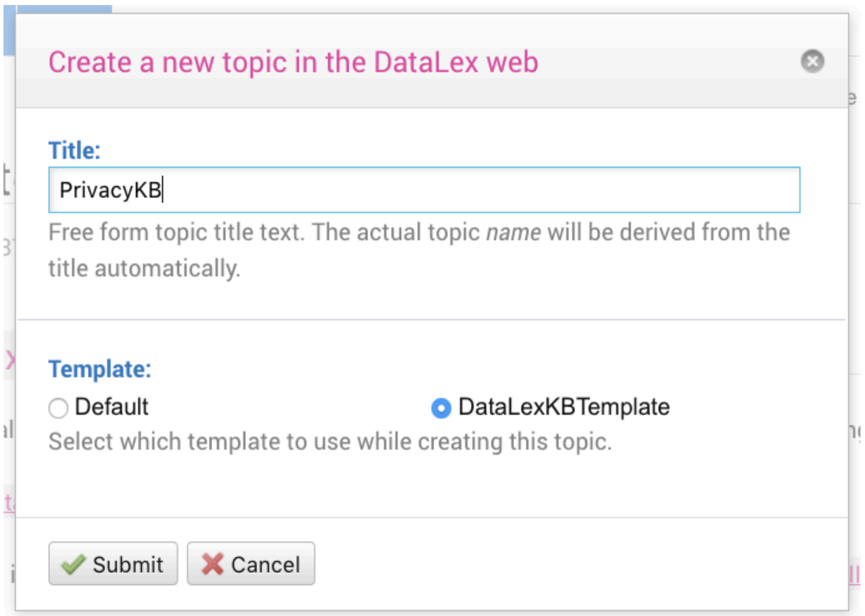


Once the login process is verified, an extra row of buttons for editing and creating new rule-bases will appear on the page.



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:



Create a new topic in the DataLex web

Title:

PrivacyKB

Free form topic title text. The actual topic *name* will be derived from the title automatically.

Template:

Default [✓ DataLexKBTemplate]

Select which template to use while creating this topic.

Enter the 'Title' of the rule-base to be constructed. The example here is 'PrivacyKB' for a privacy law application. (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).

2 Introduction to DataLex Coding

2.1 Introduction

DataLex uses yscript code to build applications. yscript is a language for representing propositional and other types of knowledge and data in a quasi-natural language format[3].

2.1.1 Levels of complexity

yscript is very simple to use to create small interactive applications, particularly those that are based around legislation. 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 executable code that is expressed in an ‘English like’ knowledge representation language. The yscript interpreter then does the rest, running your knowledge base to generate a dialogue with the user, asking questions and giving answers. You do not need to write any of the questions or answers – these are generated automatically from your code.

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

You may wish at this point to refer to Chapter 2 of *Coding with yscript – A description of the yscript language* which provides a shorty and easy to follow introduction to the basics of yscript coding.

2.1.2 Main features

The main features of yscript are:

- a ‘quasi-natural-language’ or English-like syntax, which encourages isomorphism (similarity between the structure of a rule-base and source legal documents), transparency (purpose of rules is relatively obvious) and rapid prototyping (easy to get small systems running);
- code of any degree of complexity may be written, using propositional logic;

[3] DataLex does not implement all of the yscript language. In particular, explanations and multiple-choice answers are not currently supported.

- code is divided into rules which are automatically invoked to derive conclusions;
- conventional procedural code including mathematical calculations is possible;
- a form of reasoning by analogy, or example-based reasoning; and
- a document generation facility.

2.1.3 The DataLex User Interface Manual

The DataLex *User Manual, in* Chapter 6, 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 Where is the developer's interface?

The basic development environment for DataLex applications can be found at: <http://www.datalex.org/dev/tools/>. The developer's interface for DataLex applications within the AustLII Communities environment is at <http://austlii.community/wiki/DataLex/>. This also includes documentation, papers and examples.

2.2 Declarative Coding

A DataLex application is generally a set of declarations, so called because they 'declare' what a (real life) rule says rather than trying to specify a procedure for applying it. This type of programming is therefore called 'declarative' programming, in contrast to 'procedural' (or imperative) programming, which is of the form 'first do this step; then do this step ....'.

2.2.1 Rules

The most important type of declarations are *rules* (so sometimes a piece of yscript code is referred to as a rule-base). Rules contain statements that use and determine the value of *facts*. When rules are executed, the system attempts to find the value of required facts by using other rules or if necessary, by asking the user.

It does this by going to a rule which has a 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, the system uses any other rules which have any of the premises as their conclusion. This process is then repeated along each branch of reasoning until it reaches a premise for which there is no rule to derive a conclusion. At this point, the DataLex interface interrogates the user about the truth of the premise.

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;
(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 content of the rule. 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

yscript code 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 code the logical structure. They are written in FULL UPPER CASE so yscript 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.

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

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

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 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 facts.

There are a number of varieties of descriptors, of which the most important are (i) constants, (ii) facts (iii) named subjects (a special type of fact) and (iv) rule names. Each is discussed in detail in the following chapter.

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.

2.4.2 The corresponding code

This section can be represented as three rules. Note that in capturing what the section says, we model not only its effect but also the structure and justification for any outcomes that might follow from applying it:

```
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 a DataLex application

Text, such as that above, is all that is needed for a valid piece of code. The code can be executed as a DataLex session by selecting the ‘Run Consultation’ button.

If your code does not behave as intended, go back to the editing page, edit, 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 code 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 code, 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 facts is one of the main causes of errors, particularly where rules which are supposed to invoke each other 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 to be printed (in alphabetical order) showing the names of rules which set (*) and rules which use (-) the fact (including named subjects).

Use of similar but not identical fact names 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 facts are expressed correctly.

For each fact in the code, 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 fact ‘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 facts 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 guidelines 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 code is later expanded or changed. Don't use facilities simply because they are available.

2.6.2 Isomorphism

Where the code 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 Fact Names

Include the legal basis for facts 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 fact name to get DataLex to handle it properly, first. Avoid use of embedded facts.

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.

3 Descriptors, Keywords and Expressions

yscript code consists 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, (iii) named subjects (a special type of fact) and (iv) rule names.

3.1 Facts

A fact 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 facts is to hold values which are determined during the evaluation of the code.

3.1.1 Consistent naming of facts

Consistent naming of facts, including consistency in capitalisation and punctuation, is vital.

Lack of consistency (including capitalisation and punctuation) is the principal cause of applications running other than as expected. Use the 'Check Fact Cross References' button to check for possible inconsistencies in naming of facts.

3.1.2 Boolean (true/false) facts and their names

The default fact type is boolean (that is, it is a proposition that can be true or false). When naming boolean facts, 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 fact, 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

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 facts - types

yscript recognises the following fact *types:*

Type	***Values***	***Example***
BOOLEAN	true or false	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
GENDER	male, female or unspecified	the preferred gender of the claimant
DOLLAR	dollars and cents	the value of the estate
DATE	a date	the date of the intestate's death

Non-boolean facts 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 facts

If the first use of a fact requires that it is something other than boolean, that type is automatically associated with it. From then on, you must use the fact consistently or an error message will result. In other words, the language interpreter is able to make an ‘intelligent guess’ about the type of non-boolean fact 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’, the fact ‘the date of arrival’ will of type DATE, because another date (1 May 1977) appears in conjunction with a relational operator.

3.1.5 Formal fact type declarations

While non-boolean facts are generally identified automatically, sometimes it is necessary to make an explicit declaration of the type of the fact.

The basic syntax for formal declarations is:

TYPE fact-name

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

For example, to declare facts 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, during execution DataLex would only accept responses from a user that were of the specified types.

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

3.1.6 Range limitation of fact 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]

This 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 or date 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 0 TO the value of the estate
```

3.1.7 Naming non-boolean facts

You must choose a fact name which can be followed by an **‘is’** or an **‘are’** then a value so that prompts and translations can be provided. For example, the non-boolean fact 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.
```

3.2 Constants

Whereas facts have a value which is determined during the evaluation, a constant has a fixed value. yscript 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 '**femal**e' (sex constant), a **date** (in any sensible format), and any descriptor placed in double quotes (a **string constant**).

Constants are recognised automatically. If a descriptor is not any of these categories of constant, it assumes that the descriptor is a fact.

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 yscript's main features is its capacity to automatically generate questions (prompts) by re-parsing the fact name 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 prompts and explanations

Provided that boolean fact names appear in the subject/verb/object form explained above (see 3.1.2 Boolean (true/false) facts and their names), or non-boolean fact names appear in the 'is' form explained above (see 3.1.3 Fact names for non-boolean facts), yscript 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:

```
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 Different forms of the same fact

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

```
the Act applies
the Act does not apply
the Act does apply
the Act doesn’t apply
```

It therefore does not matter which form you use in a rule.

3.3.3 Correcting grammatical errors

yscript uses a light weight natural language parser to divide a proposition into a subject, verb and object. Based on this, different grammatical forms of the same fact can be generated. Sometimes, however, it will make errors.

Where an error occurs, the first thing to consider is to change the fact name to something simpler or to make the presence of the verb more obvious. yscript will generally prefer *auxiliary* verbs (such as “is”, “was”, “has”, “will”, “can”, “shall” and so forth) over anything else. Inserting such a verb (to form a compound verb) will often fix the problem.

If this doesn’t work, you can force a proposition to divide at a verb by preceding this with a tilde (ie ~) character. For example:

```
the well-being of all children~matter
```

If you do this, you will need to consistently include the tilde character whenever you refer to the fact.

3.3.4 Adding fact translations [advanced]

One of the main purposes of yscript'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 fact, with all the complications this implies for development and maintenance.

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

For example, the fact '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 a fact type declaration for the fact. 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 a fact name causes the value of the fact being evaluated to be substituted.

Where a fact 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 fact prompts or translations if possible. Code is easier to maintain if translations are automatic.

3.4 Named subjects - names of people and things

Fact 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 facts. They are declared in the same way as facts, but are given the types **PERSON**, **THING** or **PERSONTHING**. When a fact containing a defined subject is first evaluated, automatic prompts for a subject name and, in the case of persons, the subjects' preferred gender and preferred form of address, 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 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, it will be recognised as a named subject in any subsequent reference to facts, without need for any further identification of it as such. Named subjects referred to in other facts are recognised automatically, and their values are substituted in the other facts.

For example, where there have been named subject declarations such as the ones above, a fact 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 fact declarations [advanced]

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

the name of subject (set for all types)
the gender of subject (set for PERSONS and PERSONTHINGs)
the preferred form of address for subject (PERSONS/PERSONTHINGs)
subject **is a person** (set only for PERSONTHINGs)

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

```
PERSONTHING the client
RULE common sense about companies PROVIDES
IF the client is a person THEN
      the client is not a company
```

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

SEX the gender of the claimant
      TRANSLATE male AS the claimant identifies as male
      TRANSLATE female AS the claimant identifies as female
      TRANSLATE unspecified AS the claimant does not identify
      as being male or female
```

3.5 Expressions - the use of operators

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

3.5.1 (Pre) Unary Operators

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

3.5.2 (Post) Unary Operators

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

YEARS	date years multiplier

3.5.3 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; evaluation of expressions ceases 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 evaluation continues the other arguments in the expression even though an 'AND' condition fails or an 'OR' condition is satisfied; Used to force evaluation of all alternatives.)*

3.5.4 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 Rules and Statements

4.1 Order of evaluation of rules

Rules are executed (or *evaluated*) as follows:

(i) A session commences by executing a goal rule - see 4.2 below concerning 'GOAL' rules for how such goal rules are specified.

(ii) As statements in the rule are executed, where the value of a fact is unknown the system will execute other rules that potentially may derive a value for the fact. Rules are invoked on a backward and forward chaining basis, in that rules are first invoked in a backward-chaining fashion whenever a fact needs to be evaluated in order to determine whether a rule will 'fire'. However, whenever a new fact value becomes known, all rules using that fact 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.* During the course of executing a rule, it can explicitly CALL another rule.

(iii) Once the original goal rule has completed, execution finishes. A report explaining each of the conclusions reached from the goal rule is displayed.

4.2 Goal rules

Goal rules are indicated by inserting the keyword GOAL before the RULE declaration. For example:

```
        GOAL RULE Copyright Act 1968 s32(4) PROVIDES
```

Where no goal rules are declared, then the first rule will be regarded as the goal. If this is the case or there is only one goal rule then the user session will evaluate this rule.

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 attempting to derive a fact value using backward and/or forward chaining rules, it will evaluate rules in the order in which they appear in the code. The order of appearance will not normally have any effect on the outcome 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-name

The statements for the named rule or procedure will be executed and control will be returned to the next statement after the CALL.

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. In practice, this feature is generally never used.

4.4 Types of rules

4.4.1 Default rule type

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

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

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

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

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. A FORWARD rule is evaluated when the first fact needed to execute the rule becomes known. Where necessary, *FORWARD* rules will ask the user for any other fact 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 fire 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 4.6.8 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 Generic rules [advanced]

Sometimes you may have to write rules which relate to different subjects, but which are otherwise identical. Rather than having to rewrite the rules, rules can be written that include facts with one (only) variable element, which element is represented as <>. A *generic rule* is therefore a shorthand way of writing multiple rules with slightly different wordings.

Whenever the rule parser encounters this <> symbol in a rule, it looks for instance of the fact 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, multiple versions of the rule are automatically created, 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.

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 fact “the author was a ‘qualified person’ <u>at the time the work was made</u> 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 fact “the author was a ‘qualified person’ <u>at the time the work was made</u> under s32(4)” will obtain a ‘true’ value.

Similarly, if the system needs to know a value for the fact, “the author was a ‘qualified person’ <u>for a substantial part of the period during which the work was made</u> under s32(4)” (emphasis added), the rule will ask the appropriate questions to obtain a value for this fact.

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

Generic rules should only be used sparingly and with considerable care.

4.6 Statements

Rules are comprised of *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.

4.6.1 IF-THEN-ELSE statements

IF-THEN-ELSE statements provide for conditional evaluation of facts. 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
```

4.6.2 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.

4.6.3 Assignments and Assertions

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

4.6.4 Assertions

An assertion is used to state that a proposition (fact) 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) facts. An assertion statement simply consists of a boolean fact name expressed in the positive or negative form, optionally preceded by the keyword **ASSERT** (where necessary to separate the assertion from a previous statement). Multiple assertions can be separated by an AND operator (which is sometimes more natural). 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** 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:

```
IF the circuit layout is in writing THEN
      the circuit layout is in material form
ASSERT the circuit layout is an eligible layout
```

4.6.5 Assignments

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

fact IS constant

fact1 (unknown) IS fact2 (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 facts (dates, numbers etc) where **ONLY IF** is more appropriate in the case of booleans (true/false).

4.6.6 Syntax for assignments and assertions

The syntax for assertions is:

[**ASSERT**] *fact* **{ AND** *fact* }

The syntax for assignments is:

[**ASSERT**] *fact* **IS** *expression*

or [**ASSERT**] *fact* **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
```

4.6.7 DETERMINE Statement

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

DETERMINE [**IF**] *fact*

The effect is to cause the value of fact to be determined by first evaluating any relevant backward chaining rules (commencing with any which have *fact* 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. 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.

4.6.8 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*. 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).

4.6.9 WHILE-DO and REPEAT-UNTIL Statements

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

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

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

4.6.10 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).

5 DataLex Integration with AustLII

5.1 Overview - Integration 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 rule-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 rule-bases, and the inclusion of rule-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 knowledgebase, 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 6, 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 rule-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 rule-base (the LINK ... TO ... keywords)

In addition to automatic links to AustLII, specific links can be specified in the rule-base. The keywords LINK and TO are used to specify in a rule-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 rule-base to anywhere on the World-Wide-Web, not just to AustLII.

5.4.1 Example

```
LINK document of an agency TO
http://www2.austlii.edu.au/au/legis/cth/consol_act/foia1982222/s4.h
tml#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 rule-bases

It is also possible to use LINK ...TO ... to create links from a rule-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+Minist
er%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’

‘Co-operative inferencing’, as we are tentatively calling it, is an innovative aspect of DataLex. It allows different rule-base developers to place rule-bases on any web page anywhere in other rule-bases located elsewhere on the web which they specify are to be ‘included’. In this sense, rule-base development becomes a ‘co-operative’ activity where developers can contribute their small (or not so small) rule-bases to a larger enterprise.

5.6.1 The INCLUDE keyword

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

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

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

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

5.6.2 Example

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

```
INCLUDE
http://austlii.community/foswiki/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 Specifying a goal

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 rule-base.

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

6 Document Assembly using DataLex

DataLex 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 dispositions 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 facts

The *text* argument is a piece of text to be generated as part of the document being assembled if the conditions of the rule are satisfied. A *text* argument may include embedded facts, but is not in itself a fact.

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 facts (the facts 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 dispositions 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 testamentary 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 starts a new paragraph and *TEXT* just inserts a space (ie no new line).

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.

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.
```

6.3 'Personalising' documents - embedded facts

Where a document contains variable information (eg the name of the testator, the value of property, the date of death), this variable information (a fact) can be included in the *text* of a document statement by embedding the fact in the text. In the example above, the embedded fact '<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 fact '<the date of the old will>' will cause the user to be prompted for the value of that fact.

6.3.1 Named Subjects in documents

Named subjects will not be replaced in text. 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 the user to be asked 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.
```

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

However, merely putting a fact 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:

```
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 facts 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).

6.4 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 dispositions 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 fact ‘all former testamentary dispositions are to be revoked’. The user will be prompted for a value for this fact by being asked ‘Are all former testamentary dispositions 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 facts 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 any reasoning associated with rules or otherwise. 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 fact a condition in the DOCUMENT rule.

For example, a statement in a DOCUMENT rule such as

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

will cause the system 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 a fact in a RULE.

Conclusions from rules are generated during a document generation session, 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 fact '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 facts 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 testamentary dispositions 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.

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 fiancée>.
      ELSE
              NUMBERED PARAGRAPH This will is made in contemplation of
              my marriage with <the testator/testatrix's fiancée> 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 fiancée>
              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 fiancée>
              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) Reasoning using DataLex

7.1 Example-based reasoning – overview

In addition to rule-based reasoning, 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 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 Interaction between examples and rules

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 a fact which is the conclusion of each EXAMPLE. After the facts of a particular problem are obtained from the user, the system compares the facts of the problem to the facts in the EXAMPLEs, and tries to find which is the 'nearest case'.

When no further rules can be found to assist in determining the value for a fact, the system will look to see if the fact 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.

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 fact preceding ONLY IF), or its negation. In the 'finder's cases' example above and below, the common conclusion is the fact '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 facts 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 facts 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 facts 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 a fact descriptor) should represent one significant facet of the example.

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

The *IF-THEN* form should only be used where the fact 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 fact is weighted on the basis of how poorly it *divides the example set*, as measured by its inverse variance.

7.6.1 PANNDA

When the system is about to attempt to determine a value for a fact using an example set, it first finds all examples which relate to it (that is, all examples where the fact appears as the target of an assignment). It then determines (or asks the user for) a value for all facts used in the examples. Finally, it compares each example with the situation described by these fact 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 fact is set to the same value as the nearest example. The *similar* or *all-fours* fact for the nearest example is set to true. If the example is not on all fours, the *distinguished* fact is also set for the furthest case. All of these facts (including the target fact itself) receive sensible explanatory associations (for *how/reporting*). Not all possible supporting facts are used for explanations. Rather, only *significant* ones are reported (*significant* facts 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 facts used in other examples. This feature can be used to generalise the effect of an

example. The missing facts 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 a fact 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 fact as their conclusion. Treat that fact as the conclusion fact 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 fact). This is where legal expertise is involved. Define each of these aspects as a DataLex fact.
3. Analyse all of the cases to establish (if possible) the value of each of the facts identified for any of the cases.
4. Represent each case as an EXAMPLE, with values for as many of the facts as are known for that case. It does not matter that values for some facts are not known.

7.7.1 RULEs, EXAMPLEs and DOCUMENTS interact

The values of facts 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 fact as conclusion. Where a RULE uses a fact, backward chaining will invoke an EXAMPLE with that conclusion once all RULEs have been exhausted. The same applies where a fact is used in a DOCUMENT rule.

If a fact 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):

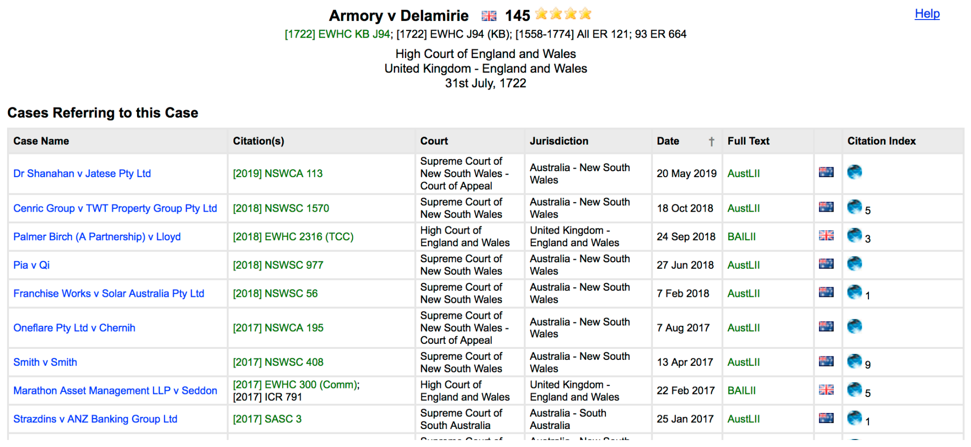
- Automatic links will be made to any properly described Australian legislation;
- Explicit links may be made from any other text using the LINK .... TO .... keywords;

- 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 example 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 rule-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.


Armory v Delamirie 🇬🇧 **145** ★★★★ Help

[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

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

7.7.3 Example – the 'finder's cases'

The Finder KB can be accessed from the DataLex Communities page, 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
```

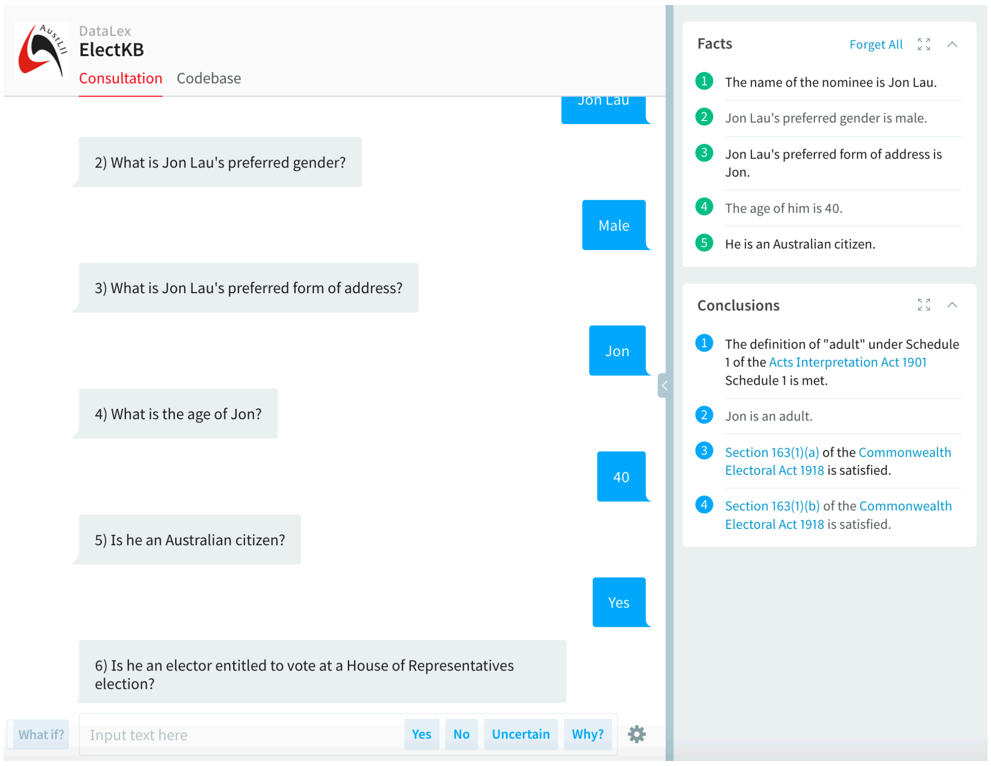
```
        the chattel was hidden AND
        there was an attempt to find the true owner of the chattel AND
        there was not prior knowledge of the existence of the chattel

    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 code is written. The following extract from a session using the ElectKB application on Australian electoral law will be used throughout this chapter to illustrate aspects of the interface.



Jon Lau

2) What is Jon Lau's preferred gender?

Male

3) What is Jon Lau's preferred form of address?

Jon

4) What is the age of Jon?

40

5) Is he an Australian citizen?

Yes

6) Is he an elector entitled to vote at a House of Representatives election?

Facts

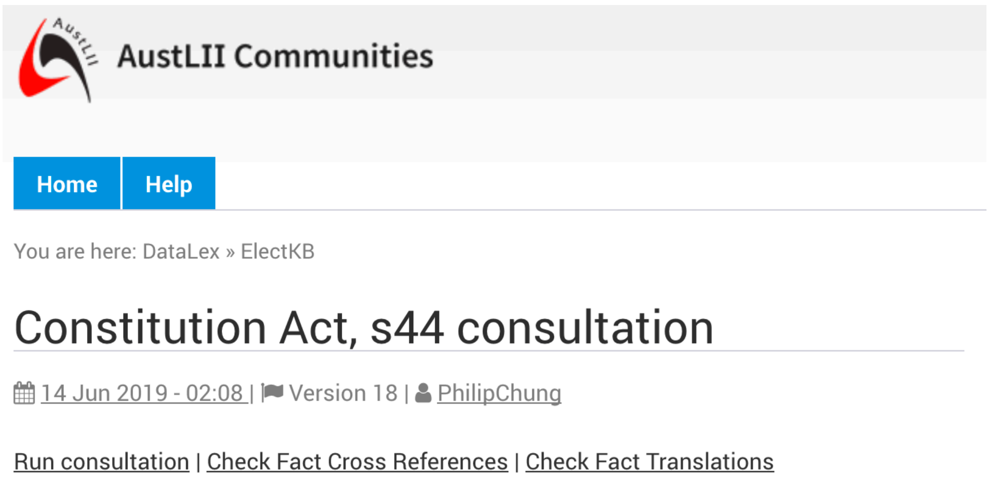
1. The name of the nominee is Jon Lau.
2. Jon Lau's preferred gender is male.
3. Jon Lau's preferred form of address is Jon.
4. The age of him is 40.
5. He is an Australian citizen.

Conclusions

1. The definition of "adult" under Schedule 1 of the Acts Interpretation Act 1901 Schedule 1 is met.
2. Jon is an adult.
3. Section 163(1)(a) of the Commonwealth Electoral Act 1918 is satisfied.
4. Section 163(1)(b) of the Commonwealth Electoral Act 1918 is satisfied.

8.2 Starting a session

A session is usually started by the user going to the rule-base, and selecting ‘Run consultation’ from above the code text. It is possible to create a link to directly invoke the session, but this has the disadvantage (in the normal case) that the user does not see the code, including any reservations or caveats that the author may have expressed about it.



You are here: DataLex » ElectKB

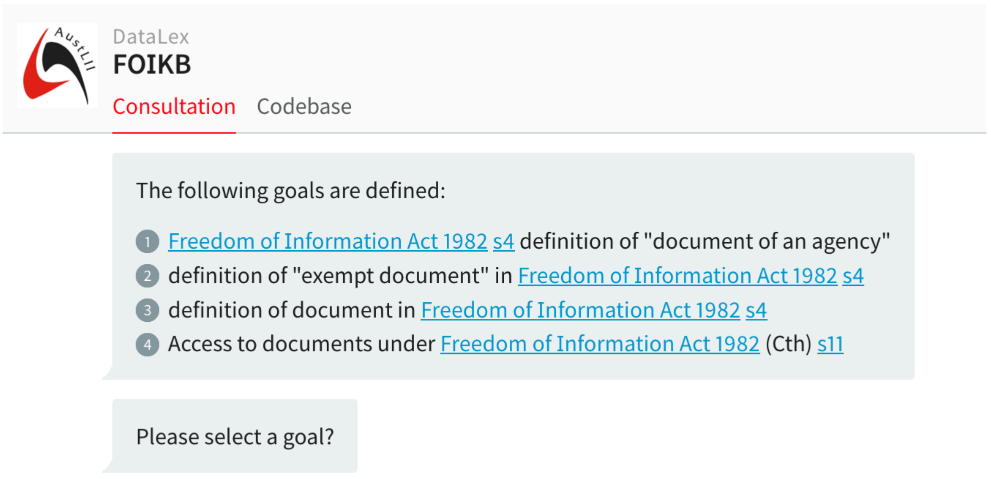
Constitution Act, s44 consultation

14 Jun 2019 - 02:08 | Version 18 | PhilipChung

Run consultation | Check Fact Cross References | Check Fact Translations

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 grey button for the desired goal. A number can be entered instead.



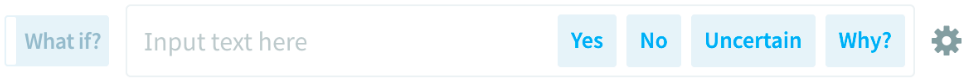
The following goals are defined:

1. Freedom of Information Act 1982 s4 definition of "document of an agency"
2. definition of "exempt document" in Freedom of Information Act 1982 s4
3. definition of document in Freedom of Information Act 1982 s4
4. Access to documents under Freedom of Information Act 1982 (Cth) s11

Please select a goal?

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.



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 fact 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 session to continue.

8.4.4 Uncertain answers

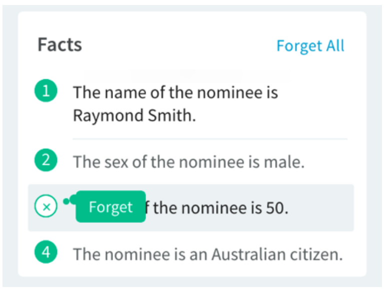
If ‘Uncertain’ (at the bottom of the screen) is selected in response to any question, the dialogue may continue if a value for that fact is not essential to a conclusion being reached. If ‘Uncertain’ is sufficient to require a particular conclusion to be reached, a Report will be generated to that effect.

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.

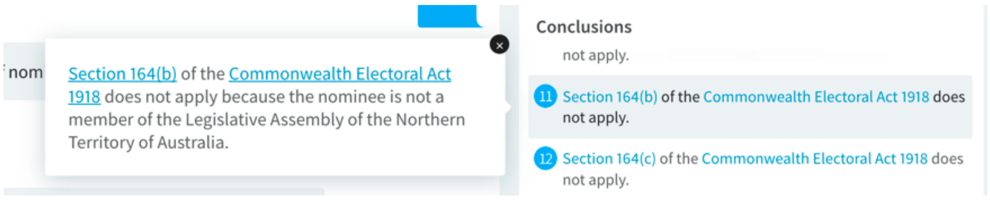

Facts
Forget All
1 The name of the nominee is Raymond Smith.
2 The sex of the nominee is male.
Forget f the nominee is 50.
4 The nominee is an Australian citizen.

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

8.7 Obtaining explanations for conclusions (How?)

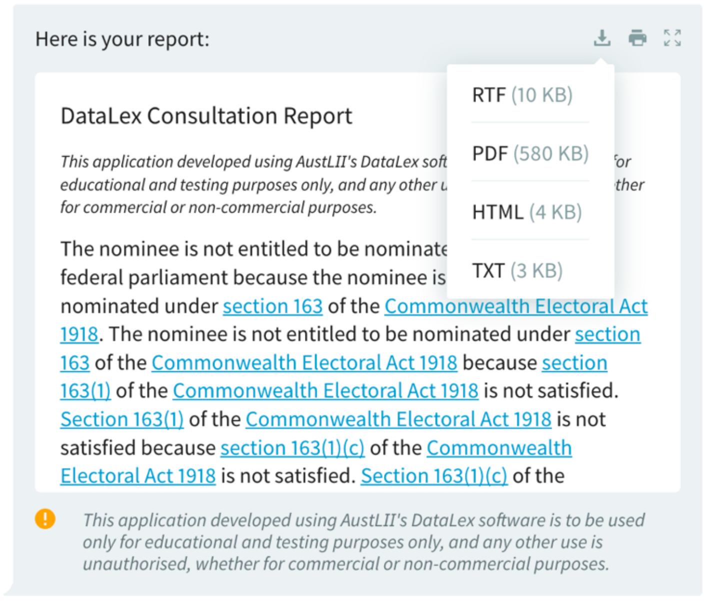
Selecting a numbered blue conclusion button will result in an explanation for that conclusion being displayed in a pop-up.


nom
Section 164(b) of the Commonwealth Electoral Act 1918 does not apply because the nominee is not a member of the Legislative Assembly of the Northern Territory of Australia.
Conclusions
not apply.
11 Section 164(b) of the Commonwealth Electoral Act 1918 does not apply.
12 Section 164(c) of the Commonwealth Electoral Act 1918 does not apply.

8.8 Reports

At the conclusion of the session a Report is generated, setting out all reasoning which is essential to conclusions which have been reached. Some conclusions reached will not be displayed in the Report, because they were not essential to the final conclusions 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.


Here is your report:

RTF (10 KB)
PDF (580 KB)
HTML (4 KB)
TXT (3 KB)

DataLex Consultation Report

This application developed using AustLII's DataLex soft… for educational and testing purposes only, and any other u… ther for commercial or non-commercial purposes.

The nominee is not entitled to be nominate… federal parliament because the nominee is… nominated under section 163 of the Commonwealth Electoral Act 1918. The nominee is not entitled to be nominated under section 163 of the Commonwealth Electoral Act 1918 because section 163(1) of the Commonwealth Electoral Act 1918 is not satisfied. Section 163(1) of the Commonwealth Electoral Act 1918 is not satisfied because section 163(1)(c) of the Commonwealth Electoral Act 1918 is not satisfied. Section 163(1)(c) of the

This application developed using AustLII's DataLex software is to be used only for educational and testing purposes only, and any other use is unauthorised, whether for commercial or non-commercial purposes.

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, 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 session, 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 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 sessions 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 rule-base as the session 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 rule-base.

8.10.2 Viewing the session in Verbose mode

Choosing ‘Verbose’ mode causes an explanation of why questions are asked, 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 session

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 session, 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.