Manual for the

Dialogue Annotation & Research Tool (DART)

Version 3.0

Author:
Martin Weisser

May 2019

Contents

1 Introduction

The Dialogue Annotation and Research Tool (DART, for short) is a research environment designed to allow you to annotate and analyse single or multiple dialogues in batch mode, with the ultimate aim to identify speech acts automatically, and thereby create pragmatically annotated corpora that can then be investigated from a variety of perspectives, all within the same environment. Once a set of dialogues has been analysed, the annotations can be verified and, if necessary, post-edited directly. In addition to the basic annotation and analysis functionality, DART also provides a convenient way for editing and manipulating many of the linguistic resources required for such analyses in order to improve the annotation results and test relevant hypotheses. This includes the ability to keep project notes, and to extract meta information about speakers and the files they feature in.

The individual features and options will be discussed and illustrated in the following sections, beginning with a brief overview of the DART environment.

2 Overview of the DART functionality

We'll begin our brief overview with an 'interface tour'. Figure 1 below provides a quick overview of the DART interface, with a corpus of already annotated files loaded into the workspace, the middle section displaying an analysis of the speech-act statistics for the whole corpus, and the right-hand section displaying an open lexicon (for a different corpus).

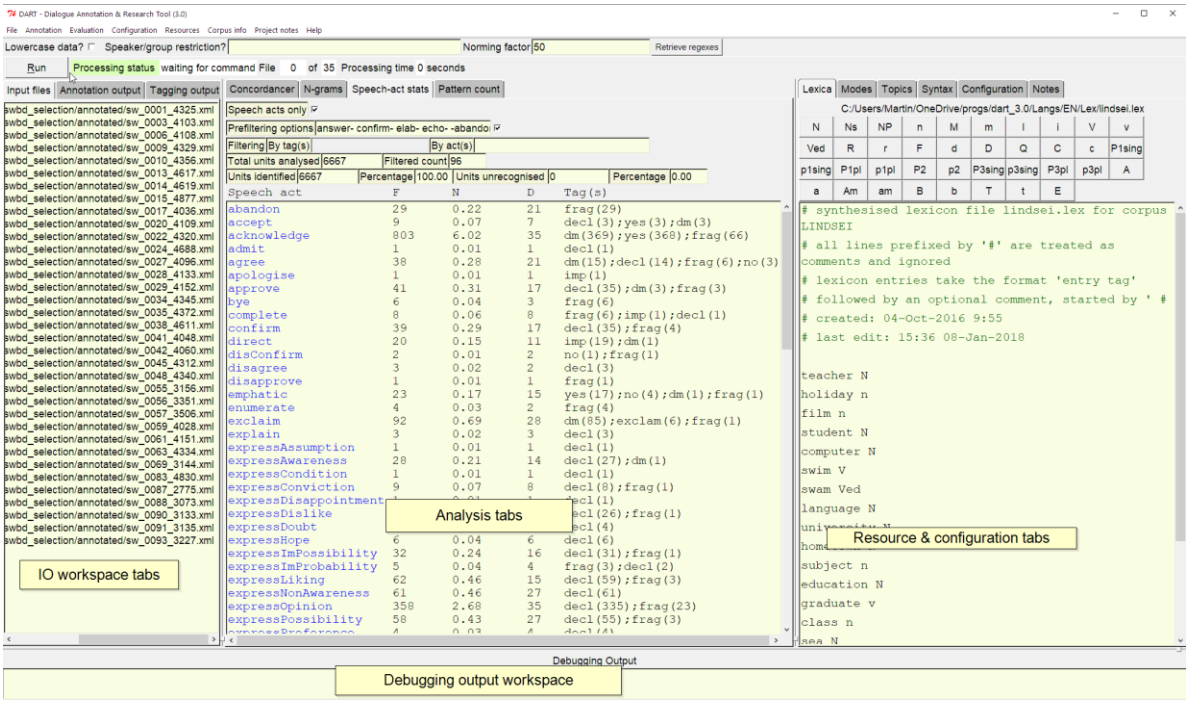


Figure 1 The DART Interface

DART provides facilities for loading a single file or selection of XML files (from a folder or file list; see section 4.2 below) into the 'Input files' workspace, seen in the left-hand window area, and running different actions on these. These actions may consist in pre-processing files after loading them in to a built-in XML editor, triggered by double-clicking an entry in the one of the lists in the IO workspace, splitting longer turns, automatically (re-)numbering turns, inserting punctuation tags, etc., or either annotating these files pragmatically or tagging them morpho-syntactically in batch-mode. The latter function, however, should not be compared to that of dedicated taggers, such as e.g. CLAWS, as it a) does not come anywhere near the same accuracy as such taggers, and b) uses a tagset that is optimised for use within the pragmatic annotation and other DART-specific routines, such as computer-assisted turn-splitting.

Files loaded either into the 'Input files' workspace, or the corresponding 'Annotation output' or 'Tagging output' workspaces (seen on the right-hand side of the IO workspace), can either be analysed using various different analysis routines, such as concordancing, n-gram generation, the extraction of speech-act or (syntactic) tag + speech-act frequencies for annotated data, or the counting of specific annotation patterns, or post-edited in a similar fashion to the pre-processing provided for the input mode, again by opening the relevant file by using hyperlinks to opening it in the built-in XML editor.

DART also, as much as possible, provides options for editing many of the resources required to perform the analyses/annotation, such as e.g. creating configurations for new domains/corpora, editing domain-specific lexica or keyword 'thesauri', syntax definitions, etc. Even though the interface has been developed to be able to control many of these resources easily, even for linguists not versed in programming, some of them currently still require an in-depth knowledge of the system, as well as the Perl programming language DART is written in, so that it's probably best to leave those alone until such time that better and more intuitive interfaces can be developed in future.

At the moment, DART only works for English, but the overall design should eventually make it possible to use one-and-the-same system to analyse different languages.

3 The DART input format

The DART input format consists of a very simple form of XML, as depicted in Figure 2 below.

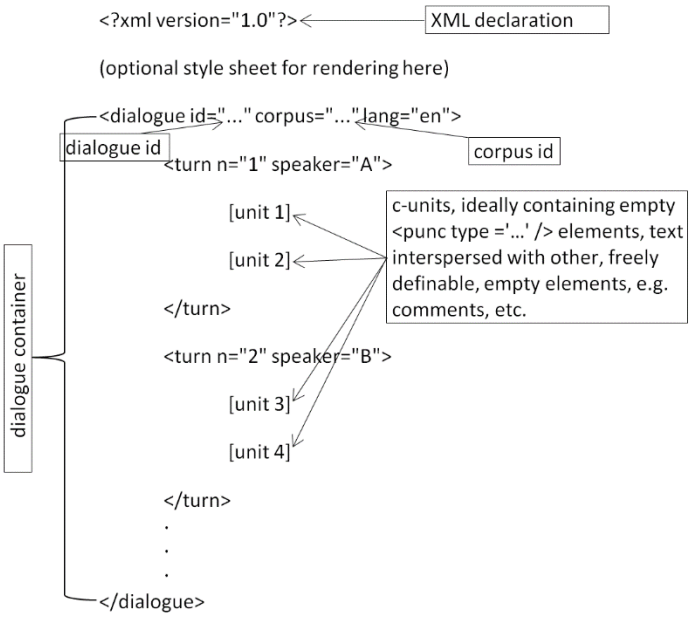


Figure 2 The basic DART XML format

The first part of the file consists of the obligatory XML declaration, followed by the <dialogue>…</dialogue> container element, although an optional style sheet definition may be added in between them. As DART XML files don't have a separate header, relevant meta-information can be stored in the form of attribute–value–pairs in the <dialogue> start element. The ones that are absolutely required for the analysis to run are:

- *id*: a unique identifier for the file within the corpus
- *corpus*: the name of a corpus, which makes it possible to load domain-specific resources, such as e.g. lexica, dynamically
- *lang*: the language of the dialogue, defaulting to *en* for English.

Additional attributes can be freely defined and will simply be copied to the output file, along with the required ones, when the annotated or tagged file is created.

Inside the *dialogue* container, the file needs to be separated into individual <turn>…</turn> containers, each containing two attributes, *n* (for the turn number) and *speaker* (current speaker id). Additional attribute–value pairs may again be added freely by you. Within the turn, the relevant sentence/clause-like units (c-units) should each appear on a separate line, i.e. separated by a newline from one another and the XML tags. XML purists may frown upon this, but this simplifies the processing in DART, and as XML is only used as a widely accepted output format, and any XML parser or rendering browser can easily ignore the extra whitespace, this design

option was chosen for the sake of expedience and to make the output more easily readable for pre- or post-processing without the need for any additional rendering engine.

All words, apart from proper names should be lowercased in order to simplify the processing.

After pragmatic annotation, each line representing a relevant syntactic and pragmatic unit will end up being wrapped in another XML tag whose label depends on the syntactic category identified by DART, as well as a number of attributes. The syntactic categories currently defined are:

- *decl*: declaratives
- *q-yn*: yes/no questions
- *q-wh*: wh-questions
- *imp*: imperatives
- *address*: terms of address
- *dm*: discourse markers (henceforth DMs)
- *exclam*: exclamations
- *yes*: yes responses
- *no*: no responses
- *frag*: fragments, i.e. syntactically ungrammatical or incomplete/elliptical syntactic units

The attributes for each syntactic element, where not all need to be present in the output, are:

- n: the number of the relevant unit, numbered consecutively and independently of turns
- sp-act: the speech act or speech-act combination identified by DART
- polarity: surface polarity
- mode: semantico-pragmatic (interpersonal) information indicating the form of interaction
- topic: semantic information pertaining to what is being talked about
- status: an indication as to whether the unit was abandoned, interrupted, or completed
- disflu: an indication of initial disfluency (not fillers), i.e. repetitions/restarts

Please note that the order of the attributes has changed from previous versions here in order to reflect their relative importance – apart from *n* – in the inferencing process that identifies speech acts better.

DART was originally designed for the analysis of unpunctuated text, but using sensible punctuation markers for major syntactic or prosodic units helps to disambiguate the options for speech acts, so empty XML elements indicating punctuation (e.g. <punct type='stop' />) can now be included at the end of a c-unit to facilitate the analysis. The *type* attribute value options recognised by DART routines within <punct /> elements are:

- stop: indicates finality, generally in declaratives
- query: indicates interrogative nature, usually occurring with either syntactically marked questions or prosodically marked declarative questions
- level: indicates non-finality, possibly in lists
- incomplete: indicates that the speaker has not completed the unit
- exclam: indicates exclamations or imperatives

Other empty XML elements that are currently defined are:

- <pause />: attribute *length* or none, if length unknown
- <backchannel>: attribute *content*
- <overlap>: attributes *n* and *type* (options: *start* & *end*)
- <unclear />: attribute *length* or none for single words
- <comment />: attributes *type* (e.g. *phon*(etic) or *syntax*) and *content*; also *sounds_like* for phonetic transcriptions
- <vocal />: for vocal 'noises', such as coughs, etc.
- <event />: any non-linguistic event, such as background noises

Additional categories can be freely added, either by typing them in or by adding additional buttons or values to the pre- or post-processing options in the built-in XML editor (see section 7.1 below for details). All empty elements, apart from <pause /> and <punc /> are simply ignored in the processing and written to the output file unchanged.

4 Processing options for annotation

4.1 Adding a new file to a corpus

A (new) sample file can be created through the 'File → New' menu option or pressing 'Ctrl + Alt+n'.

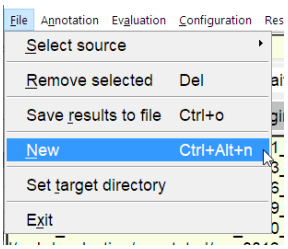


Figure 3 Dialog for creating a new, 'blank', annotation file

You will then be prompted to provide a filename, and DART will create a new skeleton XML dialogue in the DART format, containing the XML declaration and a dialogue container tag, with all required attributes, as well as the filename as an ID, and open this in a new XML editor window. If you don't specify a folder, this file will automatically be created in the 'test' folder within the DART 'data' folder. Dialogue content can then be inserted from a text file, split into turns, and pre-processed to conform to the expected format using the editor's features in the pre-processing phase (see 4.3 below).

4.2 Selecting file(s) for processing

Corpus data in the form of single or multiple files for pre-processing – and also for annotation, or analysis – can be loaded via the 'File → Select Source' menu.

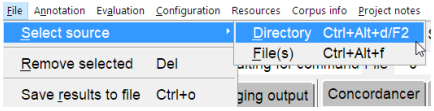


Figure 4 Source selection options

As Figure 4 above shows, selecting the 'File → Select source → Directory' option or pressing the keyboard shortcut 'Ctrl + Alt+d' or 'F2' allows for a whole directory to be selected (see Figure 5), while choosing 'File()' ('Ctrl + Alt+f') will open a file selection dialogue (see Figure 6), where individual or multiple files can be chosen.

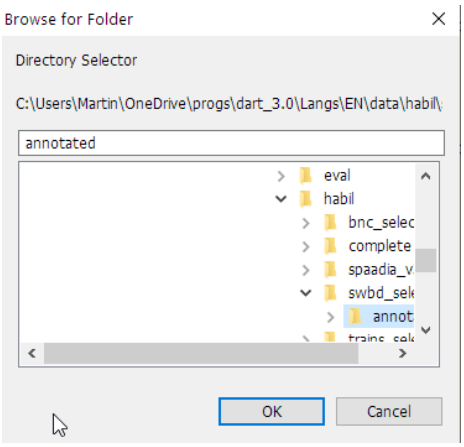


Figure 5 Directory selector

If no directory is chosen, and the dialogue cancelled, or you press the 'Esc' key, by default, files in the 'test' directory will be selected.

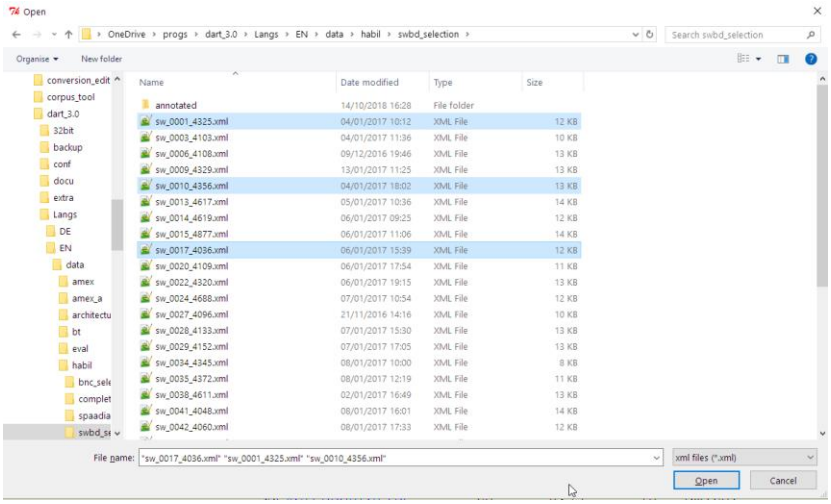


Figure 6 File selector (multiple files selected)

Cancelling the individual file selector dialogue will simply return you to the main DART interface. Once a file or directory has been selected, the relevant file(s) are added to the 'Input files' workspace and double-clicking on a filename will open the relevant file in the built-in editor. The default target directory will be set to a sub-directory of the input directory named 'annotated' (or 'tagged' in tagging mode). If the output target directory doesn't exist, it will be created once an annotation or tagging process is started. The processing status indicator towards the top right-hand corner will also reflect the number of files selected, as illustrated in Figure 7.

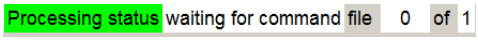


Figure 7 Processing status indicator (prior to processing)

This processing status indicator will change to reflect the number of files processed once a processing operation has been triggered.

IO lists can now also be ‘pruned’ easily, simply by (multi-)selecting the files to be excluded and pressing ‘Del’ or using the ‘File → Remove selected’ menu option. Any ensuing annotation, tagging, or analysis operation will then only be carried out on the remaining files.

4.3 Pre-processing

Any file that appears in the input files workspace can be opened in the built-in DART editor by double-clicking one of the list entries below an IO workspace tab. Before any annotation has been carried out, this will usually be done to pre-process the file to break long turns or to fix issues that may have arisen from an automatic conversion process to the DART format.

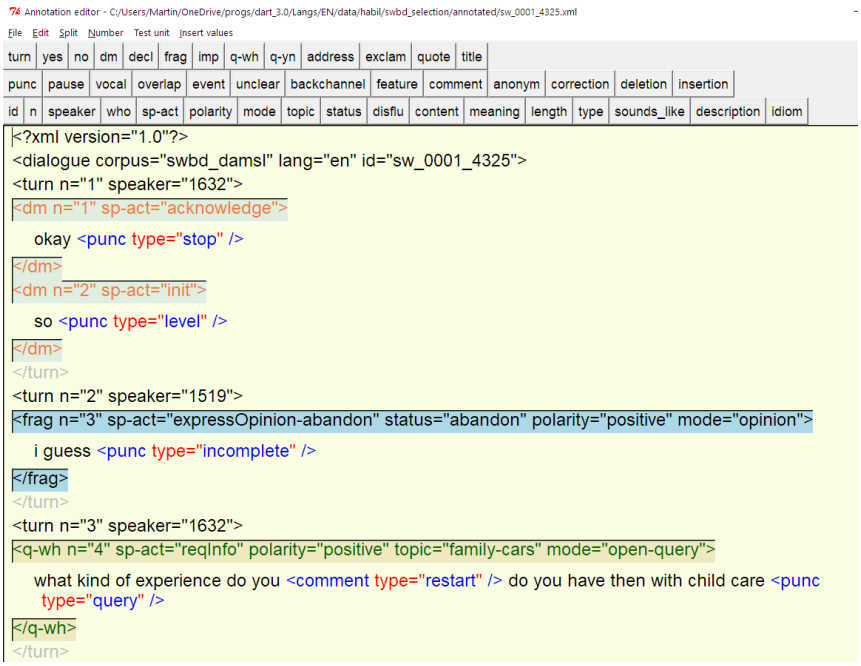


Figure 8 The built-in editor showing a file opened for pre-editing

The editor has standard key bindings for saving (‘Ctrl + s’), copying (‘Ctrl + c’), cutting (‘Ctrl + x’), pasting (‘Ctrl + v’), as well as undo (‘Ctrl + z’) and redo (‘Ctrl + y’) operations. To close the window, the binding ‘Ctrl + w’ can be used, and ‘Ctrl + S’ is a shortcut for saving and closing the window in one go, while ‘Ctrl + f’ triggers the find popup box for you to type in a search string. To repeat a search for the next occurrence once an item has been found and is still highlighted, you can press’F3’. To trigger the replace popup, press ‘Ctrl + h’.

Note: Regex replace operations unfortunately don’t work in the current implementation for the editor widgets in Perl/Tk, so that only the regular replacement options are possible, while the regex option does work for normal searches without replacements. In addition, the search interface is a bit quirky and sometimes remembers things that were put on the clipboard, so that the

search box may be prefilled. The best way to deal with this is to press ‘Ctrl + /’ to select the everything in the search box and then simply type in your search string.

For ease of recognition, and to be able to distinguish the main text from them, empty elements are colour-coded in blue, while their attribute names are highlighted in red.

The ‘Edit’ menu, shown in Figure 9, makes it possible to insert the contents of a text file into a newly created corpus file or to add timestamps (‘Shift + F5’) in the initial stages of corpus creation, while the remaining functions become more relevant once annotated files exist.

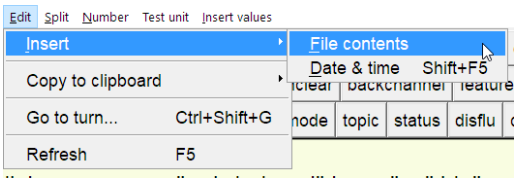


Figure 9 The ‘Edit’ menu of the built-in XML editor

The ‘Copy to clipboard’-submenu, as the name implies, provides the option to either copy a ‘cleaned-up’ piece of annotated text (‘Ctrl + Shift + c’) or the name of the currently open file (‘Ctrl + b’) to the clipboard, so that they can e.g. be pasted into an article as examples. ‘Go to turn…’ (‘Ctrl + Shift + g’) pops up a ‘GoTo’ dialog, where inputting the number of a specific turn and pressing ‘Enter’ will automatically jump to that turn, while ‘Refresh’ (‘F5’) will trigger the colour coding of the whole file after edit operations where this doesn’t happen automatically.

As generally longer turns in fact do constitute multiple pragmatic units, DART provides supporting routines for splitting them, located on the ‘Split’ menu.

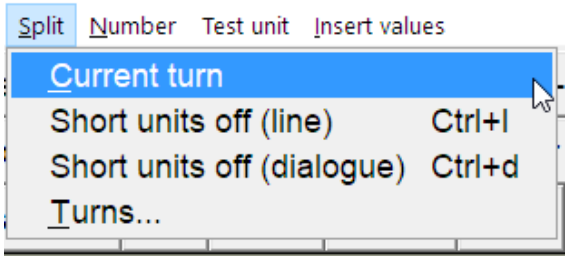


Figure 10 the ‘Split’ menu

The ‘Current turn’ entry essentially breaks the turn into two at the cursor position, automatically inserting a turn-end tag, a new empty turn, and a turn-start tag at the point where the cursor is located. This is useful when one realises that a turn has been incorrectly transcribed (or converted), and in fact the units belong to two different speakers.

The next two entries, ‘Short units off (line)’ (‘Ctrl + l’) and ‘Short units off (dialogue)’ (‘Ctrl + d’) respectively try to detect and split off shorter initial units, such as DMs, for either the current line only or the whole dialogue. Although it’s useful to do this in the pre-processing

phase, it's not absolutely necessary to do so, as the splitting routines will automatically be carried out during the automated annotation process, too. However, performing the splitting during the pre-processing phase, combined with manual correction, is more accurate, especially as some shorter initial units, such as the ones containing *so*, may be ambiguous, where the same word may either represent an initiating DM that indicates the beginning of a new phase in the dialogue or a logical connector that indicates a conclusion. In the latter case, it would be wrong to split this off, and this may in fact be prevented by adding an underscore character to the end of the word, i.e. *so_*, thereby masking it. All such underscore characters will automatically be removed from the final annotation.

The final menu entry, 'Turns', will trigger an interactive turn splitting process that suggests potential splitting points to you, as depicted in Figure 11.

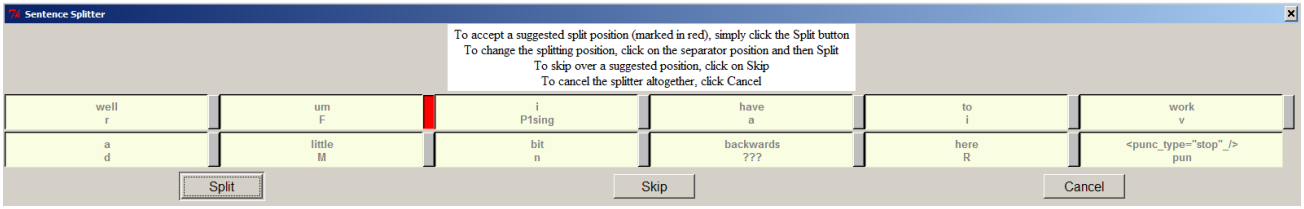


Figure 11 The sentence splitter dialogue

As the above figure shows, you can either accept a suggested splitting position, skip over it, or change it by clicking on one of the separator position buttons, while clicking 'Cancel' will exit the splitting routine altogether. However, as the split positions suggested by this routine may not be very sophisticated, it's probably better to break turns manually, which will also lead to a better understanding of the contents of the dialogue that may help with later interpretations.

Another useful option is hidden behind the 'Number' menu, with its associated sub-menus 'Turns' ('Ctrl + t') and 'Units' ('Ctrl + u'). In a pre-editing context, the 'Turns' sub-menu is the most relevant, as it may often be necessary to split turns where an automated conversion program has incorrectly assigned units within the same turn to one speaker, or merge them, e.g. where backchannels have erroneously been handled as separate turns, thereby artificially breaking a speaker's turn. As it would be very cumbersome to manually re-number all the turns in such cases, especially in a rather lengthy dialogue, the editor will take care of this re-numbering automatically for you via this menu option, returning you to the last edit position once the process has been completed. Re-numbering units is only relevant after the pragmatic annotation process has been carried out, and post-processing of units may be necessary.

For manual editing inside the editor, there is a set of user-configurable/-editable toolbars (see Figure 12) and (context-)menu items (see Figure 13).

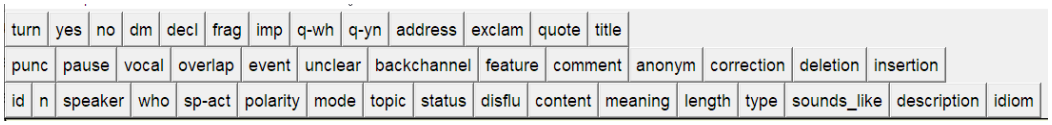


Figure 12 The toolbars in the built-in XML editor

The top toolbar contains a set of dialogue-relevant XML container elements that can either just be inserted or wrapped around selected textual content. Most of these, though, apart from <turn>, relate to dialogue elements that will only be relevant for post-processing, in order to revise parts of the automated annotation. The second row contains a set of empty XML elements that may be used predominantly in pre-editing mode, usually in combination with one or more of the attributes inserted by clicking buttons found in the third row.

To prevent you from having to type in potential attribute values, thereby potentially also producing typos, as well as to some extent constraining pre- or post-editors to only use a set of fixed options to ensure consistency across a project, attribute values can be inserted from both the 'Insert values' menu and also via the context (right-click) menu. The options for the former are shown in Figure 13.

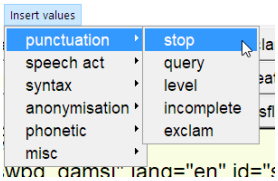


Figure 13 The 'Insert values' menu

The sets of values that are essentially relevant in a pre-processing context are all, apart from 'speech act' and 'syntax'. For instance, 'phonetic' can be used to add values to transcribe phonetic phenomena when using the 'sounds_like' attribute. 'punctuation' can be used to insert attributes for the 'punc' empty element, but in fact it's generally easier to use keyboard shortcuts to insert the whole element. See section 8 below for details.

The remaining option, 'speech act' only becomes relevant in a post-processing context, once pragmatic annotation has been carried out.

The 'Test unit' option, depicted in Figure 14, allows you to get a 'sneak preview' of how the unit would be annotated.

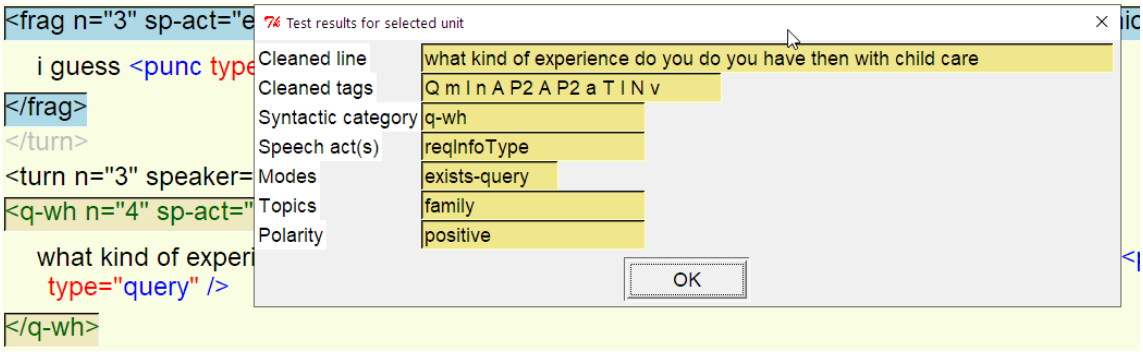


Figure 14 example of the 'Test unit' function

Although this feature is mainly intended to make it possible to verify why a particular unit may not have been annotated with an expected speech act after the pragmatic annotation process has been carried out, it can also be used to identify gaps in the lexicon, weaknesses in the tagset or syntax rules, as well as to test whether the text-cleanup routines successfully remove all unwanted material, such as backchannels or comments from the unit to be analysed. As the analysis is only conducted for a single unit, however, the inferencing process that is normally conducted to identify the final speech-act label cannot be carried out fully to determine contextual features, such as identifying whether a unit functions as a response, etc., and thus such 'secondary' speech acts are not listed.

4.4 Corpus annotation

DART provides two options for annotating files loaded into the input files workspace. The main one is the one labelled 'Pragmatic', and is triggered by selecting the relevant option from the 'Annotation' menu or pressing 'Ctrl + Alt + a'.

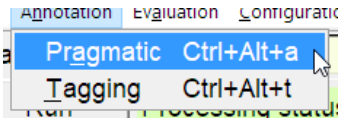


Figure 15 The annotation options menu

The second option, 'Tagging' ('Ctrl + Alt + t'), can be used for both un-annotated and annotated files. This tagging, however, as pointed out above, is fairly rudimentary, and its accuracy can and should in no way be compared to that of a dedicated tagger such as CLAWS. Its main function is to verify issues pertaining to the pragmatic annotation process, such as finding incorrectly identified syntactic units, due to errors in the syntax rules or out-of-lexicon words.

All files selected for processing are annotated in batch mode without any further user input, unless of course an error occurs in the processing, in which case one or more error messages will be written to the 'Debugging Output' window at the bottom of the DART main window, and the annotation process may be interrupted altogether, depending on the seriousness of the

error, or whether some form of error handling has successfully been implemented. Most frequently, such errors relate to ill-formed XML, or incorrectly/incompletely specified resources, such as e.g. a *lang* or *corpus* attribute missing from the 'header' of a file. To avoid any well-formedness related errors, you should ideally check corpus data loaded into the workspace before running any annotations by selecting 'Evaluation → Check wellformedness'. Missing or incorrectly specified *lang* or *corpus* attributes will be reported during the annotation process itself, though.

Having the appropriate attributes for *lang* and *corpus* present in the container element is essential because only this way all the necessary resource files can be dynamically loaded while reading in dialogue files for annotation.

During the annotation process, the processing status indicator is constantly updated upon completion of the annotation of each file, and the name of the result file is inserted in the relevant output files workspace. Even annotating relatively long files of more than 100 turns should generally take no longer than about a maximum of 20-30 seconds, so if the processing indicator has not changed for a longer period of time, it's likely that an error may have occurred, in which case you should first check the debugging output for errors to fix, and, in the worst case, close and restart DART.

4.5 Post-processing

Just as for pre-editing, files for post-processing are opened in the built-in XML editor window by double-clicking one of items on an IO list tab. If the post-processing is to be done immediately after the annotation process, the list will be on one of the output tabs, but if the post-processing is started at a later time, the files will need to be loaded into the list on the 'Input files' tab again.

A pragmatically annotated file will look very different from an un-annotated one because additional XML elements will have been wrapped around the individual syntactic units, and also because these elements are colour-coded in order to reflect at least part of their (semanti-co-)pragmatic potential, which can be seen in Figure 16.

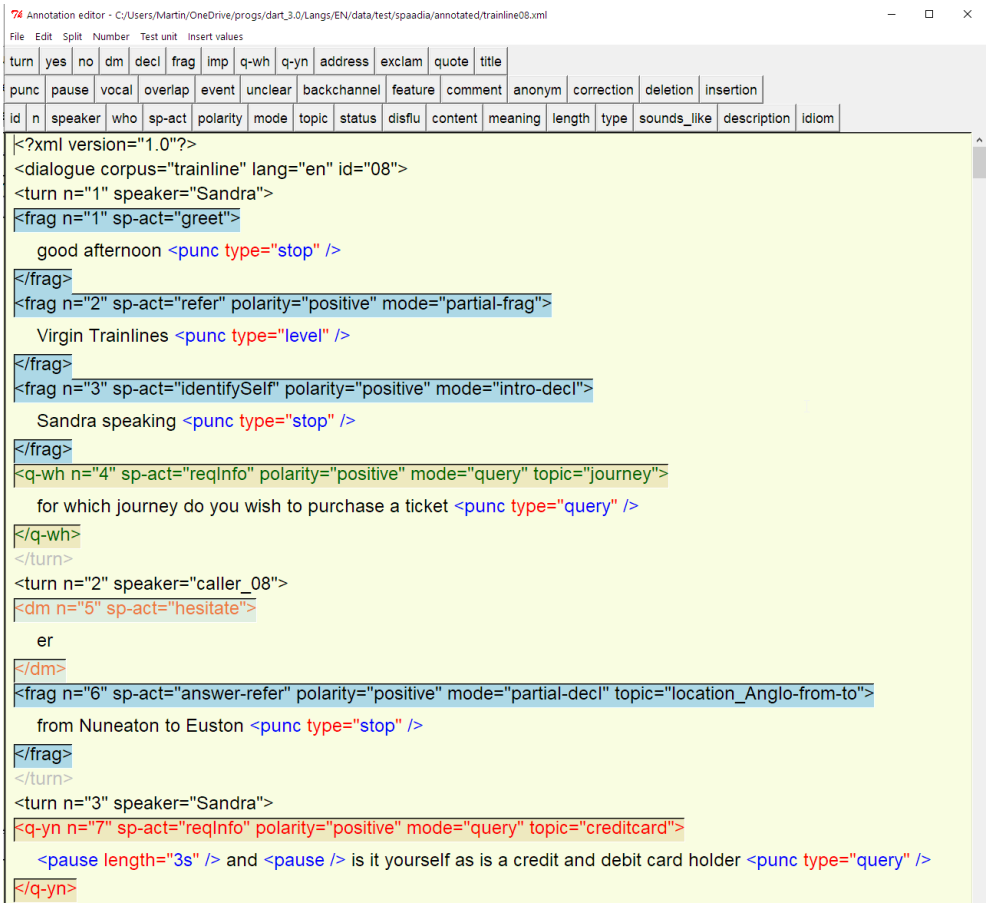


Figure 16 The built-in XML editor showing a file opened for post-processing

The colour coding semantics are as follows:

- *decl*: grey background, blue text
- *q-yn*: ivory background, red text (signifying closed set of options for answers)
- *q-wh*: ivory background, green text (signifying open set of options for answers)
- *imp*: light blue background, red text (signifying imperative force)
- *address*: orange background, dark blue text
- *dm*: greyish blue background, orange text
- *exclam*: light-blue background, orange text
- *yes*: blue background, white text (mnemonic symbolising a ‘forward-moving’ option, as in an ‘entry permitted’ traffic sign)
- *no*: red background, white text (mnemonic symbolising a blocking option, as in an ‘no entry’ traffic sign)
- *frag*: light blue background, black text

Beginnings of turns appear in black text colour, but their ends are 'greyed out', so as to make them less prominent. Any textual material appears in black text colour, without any special background. The tag colours can actually be edited in the 'tag_colours.cnf' configuration file, but in order to change them, it's necessary to know the X11 colour names[1] or provide a hex value prefixed by #.

4.5.1 General editing options

In general, the post-processing will consist in fixing any potential errors the program may have made in the annotation phase. Most of these errors will generally pertain to incorrectly or incompletely identified speech acts, but some of them may also be related to other levels of annotation, such as e.g. errors in the syntactic annotation, in case the built-in rule set has not been able to identify the nature of a unit, e.g. 'mis-interpreting' a declarative as a fragment, or a fragment starting with an infinitive as an imperative, etc.

In case a syntactic element has been misidentified, the corresponding tag name can simply be retyped or selected from the 'syntax' value or context menu (see Figure 17).

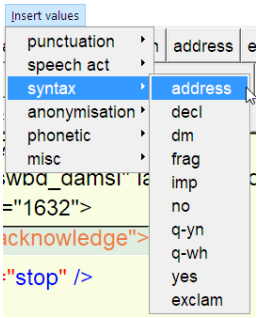


Figure 17 The 'syntax' value menu

To add or replace speech-act values, items can be selected from the relevant context menu item (see Figure 18 below), too.

[1] See https://en.wikipedia.org/wiki/X11_color_names#Color_name_chart.

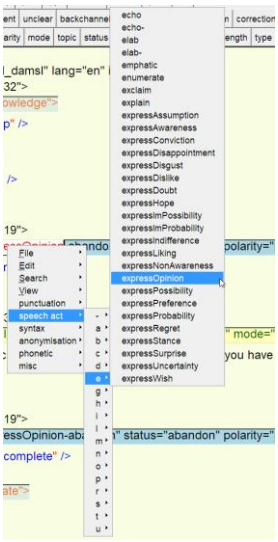


Figure 18 The ‘speech acts’ context menu

As the number of potential speech acts is very high, the menu is sub-categorised alphabetically to make it easier to find a particular speech act.

If, for some reason, you decide that a unit that was either not manually split in the pre-processing or by some of the automatic splitting routines of the annotation process, then you can add a manual line break between the units and move any relevant element items to be retained via copy-and-paste, and then add a new syntactic element plus attributes by using the toolbar options for wrapping text in element tags and creating attributes.

In some cases, for instance if turns have not been correctly marked in an automatic conversion of dialogue data to the DART format, you can also manually break turns, and then insert the appropriate turn tags and attributes, although, ideally, such issues should be identified and fixed in the pre-processing stage of a corpus.

4.5.2 (Re-)Numbering turns or c-units

In both of the last two cases described above, for turns or units, respectively, the numbering would of course then be off and need to be adjusted. As this would again be extremely tedious and error prone if done manually, the XML editor provides functions for updating the numbers automatically from the ‘Number’ item on the ‘Edit’ menu. These functions can be triggered from anywhere within the text, and will automatically position the cursor again at the relevant point after the update operation has been completed, so that post-processing can continue smoothly.

5 Analysis options

As of version 2, DART no longer has an 'Analysis' menu. Instead, all analysis modules – 'Concordancer', 'N-grams', 'Speech-act stats', and 'Pattern count' – are controlled via the relevant tabs in the middle section of the program. The 'Run' button (Ctrl + r') on the main program toolbar is context-sensitive and triggers the respective analysis functions depending on which of the notebook tabs is currently selected. All analysis functions, apart from the speech-act analysis, can generate basic descriptive statistics for either annotated or un-annotated corpora.

The only option that remains from the original menu is a simple *word count* of the basic textual content, i.e. excluding the contents of tags or empty elements. This function has now been integrated into the 'Corpus info' menu.

The output of all modules can also be filtered for particular speakers or speaker groups via a regex in the 'Speaker/group restriction?' box in the program's main toolbar, and the frequencies for the 'N-grams' and 'Speech-act stats' modules can be normed according to an adjustable norming factor, whose default is set to 50, based on the assumption that a) it's more sensible to norm frequencies based on functional units, rather than the number of words, and b) that most interactions/dialogues will contain at least 50 units.

5.1 Concordancing

Perhaps the most essential analysis feature DART incorporates is a (line-based) concordancing facility located on the 'Concordancer' tab of the middle section. For non-annotated data, this concordancer can be used to investigate particular expectations or hypotheses in a way that is essentially rather similar to 'traditional' corpus pragmatics. More importantly, though, it allows the researcher to explore annotated data, partly in order to verify or test the effectiveness of the different processing routines at various levels, but mainly to develop an understanding of the interplay of the different linguistic elements that contribute towards the generation of pragmatic meaning. The concordance module, just like the other analysis modules, automatically takes the currently 'activated' list of files in the IO workspace area as its input.

Figure 19 shows an illustration of a concordance output for the combined search for the <q-wh> (syntactic) tag and the target word *what* on the next line, which only finds occurrences of wh-questions that contain the word *what*.

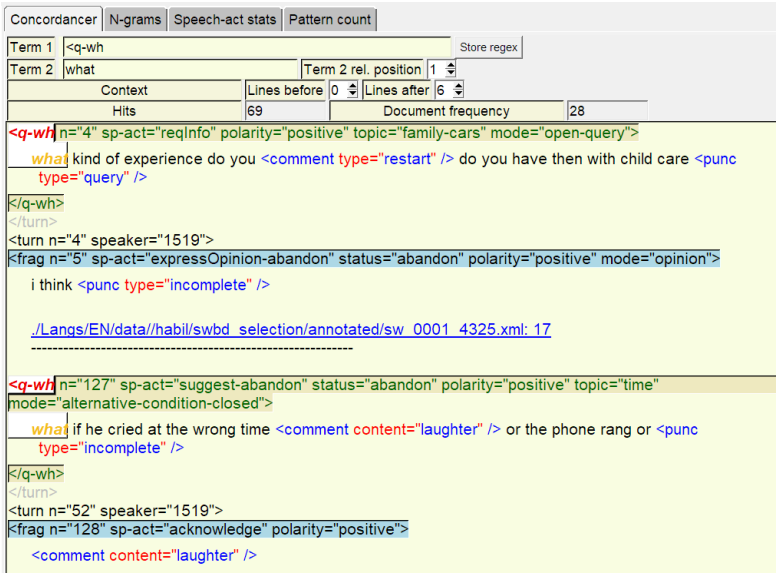


Figure 19. Concordance output for <w-qh and search term ‘what’ on the next line

As the illustration above shows, the concordancer currently provides the option to look for a combination of two *search terms* (patterns), where the terms may appear on different lines, and the relative line number of the second term can be specified as either a positive or negative integer, with a positive value indicating that the 2nd term is to follow, and a negative one that it ought to precede. When searching for only a single term only, this term needs to be specified in the ‘Term 1’ box.

While terms can be specified freely, some of the most common terms of interest for the analysis of annotated dialogues are already predefined via auto-completable drop-down lists. Here, the first box contains the beginnings of all syntactic elements and potential speech acts, while the second holds only the speech-act attributes. Commonly used example search terms can also be saved to a file that can be accessed and edited via the ‘Retrieve regexes’ button on the top toolbar. The expression currently specified in the ‘Term 1’ box can also automatically be appended to the list of stored regexes by clicking on the ‘Store regex’ button on the ‘Concordancer’ tab. As the list is stored in a text file that allows comments to be inserted, explanations pertaining to each entry can also be added, which makes it easier to remember their particular functions, especially if they haven’t been used for some time.

If terms are specified freely, rather than picked from the dropdown lists, the full, rich set of *Perl* regular expressions can be used. This makes it possible for advanced users to specify highly complex patterns to search for, but at the same time still enables the ‘average user’ to simply ‘look for words’. Searches can be made case insensitive, something that may e.g. be required for dialogue data that has been transcribed using uppercase characters for stressed syllables.

The output consists of the relevant target line with the search term(s) highlighted, and ‘surrounded’ by a specified variable context of *n* numbers of lines preceding or following the target

line itself. In our example, the following context is set to 6 lines, allowing us to see the immediate responses to the questions. The context is controlled via the spinners next to the 'Lines before' and 'Lines after' labels. This make it possible to easily widen or collapse it in an appropriate manner and simply run the search again. Each hit is followed by a line containing the file name and line number where it was found. The number of hits, as well as their dispersion, i.e. document frequency, is provided above the output window.

Information from the concordancing window can be copied and pasted straight into a word-processor if you want to e.g. extract only a few examples from a corpus for use in a publication. Alternatively, a complete list of all concordance hits, along with information about the search terms, number of hits, and dispersion, can be saved to a text file via 'File → Save results to file' or pressing 'Ctrl + o' (with the cursor inside the results window). In the text output, 'Term 1' will be highlighted through three enclosing angle brackets, e.g. <<< <q-wh >>>, and, 'term 2', if present, through 3 enclosing square ones, e.g. [[[what]]], for the above example.

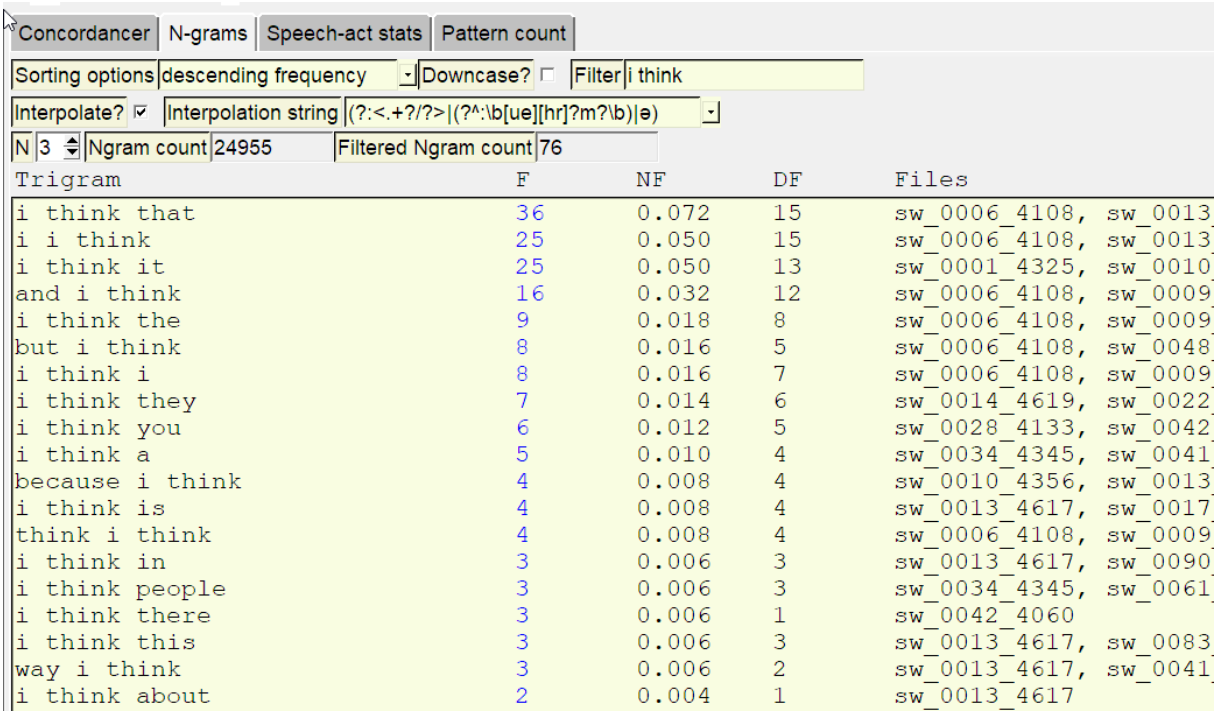
However, what's perhaps even more useful in the annotation context is that the source information for each individual hit again triggers a hyperlink that opens an XML editor window at the relevant line number – with the search term highlighted –, allowing efficient on-the-spot post-processing. This can, for instance, be used to automatically identify and select all occurrences where DART has been unable to assign a speech act to a unit, and thus quickly fill the annotation gaps. Another use for this would be if a user who, due to a lack of programming knowledge, does not have the expertise to fix particular analysis rules that have been identified as misfiring, but has the linguistic understanding to identify these cases in the results, could select all miscategorised instances and correct them in this way.

5.2 Basic frequency analysis (uni- and n-grams)

N-gram analysis is an important means of identifying recurring lexico-grammatical or *phraseological* patterns on different linguistic levels in dialogue corpora. And once these have been discovered, they can be modelled and integrated into the automated annotation routines. The 'N-grams' tab in DART provides access to this functionality to create frequency/n-gram lists for a given corpus. Unlike in other programs, such as AntConc, where unigrams are treated separately as 'Word lists', DART treats word lists and longer n-gram lists in the same way and combines them on the same analysis tab.

Once any recurrent 'themes' have been identified, their general patterns of occurrence, along with any inherent variability, can be deduced, and regular expressions for pattern matching created and grouped under suitable category labels in the different DART resource files (see section 7.2 below). If such a pattern can be matched against a textual unit during the annotation process, the corresponding label can be recorded and later potentially be used as part of the inferencing process in identifying speech acts.

To make it easier for you to verify any hypotheses developed from looking at the frequency data, all entries are again hyperlinked – via the raw frequency (F) –, so that they trigger the built-in concordancer with pre-defined search terms. Figure 20 below illustrates the basic output format and available configuration options in the form of a filtered trigram list.



Trigram	F	NF	DF	Files
i think that	36	0.072	15	sw_0006_4108, sw_0013
i i think	25	0.050	15	sw_0006_4108, sw_0013
i think it	25	0.050	13	sw_0001_4325, sw_0010
and i think	16	0.032	12	sw_0006_4108, sw_0009
i think the	9	0.018	8	sw_0006_4108, sw_0009
but i think	8	0.016	5	sw_0006_4108, sw_0048
i think i	8	0.016	7	sw_0006_4108, sw_0009
i think they	7	0.014	6	sw_0014_4619, sw_0022
i think you	6	0.012	5	sw_0028_4133, sw_0042
i think a	5	0.010	4	sw_0034_4345, sw_0041
because i think	4	0.008	4	sw_0010_4356, sw_0013
i think is	4	0.008	4	sw_0013_4617, sw_0017
think i think	4	0.008	4	sw_0006_4108, sw_0009
i think in	3	0.006	3	sw_0013_4617, sw_0090
i think people	3	0.006	3	sw_0034_4345, sw_0061
i think there	3	0.006	1	sw_0042_4060
i think this	3	0.006	3	sw_0013_4617, sw_0083
way i think	3	0.006	2	sw_0013_4617, sw_0041
i think about	2	0.004	1	sw_0013_4617

Figure 20 A trigram frequency analysis, filtered for 'I think'

A number of different sorting options exist to produce different 'views' of the data. The default option is a frequency list sorted in descending order of frequency (*n-1*), with a secondary *alphabetical* sort order. This can be reversed (*1-n*) to list low-frequency items, such as *hapax legomena*, first, perhaps in order to identify highly domain-specific content. The next two options are for alphabetical sorting (*a-z*) and its counterpart (*z-a*). While the first four options, as already pointed out above, are mainly useful for identifying vocabulary or phraseological patterns, the final sorting option (*rev*) creates a reverse-sorted list that may be used for identifying morphological features that can be modelled in the morpho-syntactic analysis (tagging) routines, or to find n-gram sequences that belong to a common topic.

For genuine n-grams, i.e. more than one word, fillers can be removed based on regex patterns to create cleaned up n-grams. These fillers are then automatically be re-interpolated into the concordance searches if the 'Interpolate' option is checked because otherwise not all patterns would be found again.

As before, the 'Downcase' option should probably only be used to handle data that uses mixed case to represent prosodic features, such as stress, as otherwise the DART format assumes that only proper nouns are capitalised, anyway.

The ‘Filter’ option makes it possible to ‘zoom in’ on particular patterns, again based on regexes. The number of filtered n-grams displayed is reported along the full n-gram count. In the above example, the filter causes only trigrams that contain *i think* to be displayed.

Apart from the raw frequencies, the ‘N-gram’ module also reports the (normed) relative frequency (NF), the document frequency (DF), as well as the files (Files) that the n-gram occurs in.

5.3 Speech-act (frequency) analysis

Producing ‘Speech-act stats’, of course only makes sense to use if a corpus has already been annotated. The default output from this module lists speech acts according to the syntactic tag they occur with, sorted first according to tag categories, then in descending frequency of the speech act itself, plus the raw frequency (F) of occurrence in the corpus, the (normed) relative frequency (N), and the document frequency (D). Some sample output, filtered by a particular speaker, is shown in Figure 21.

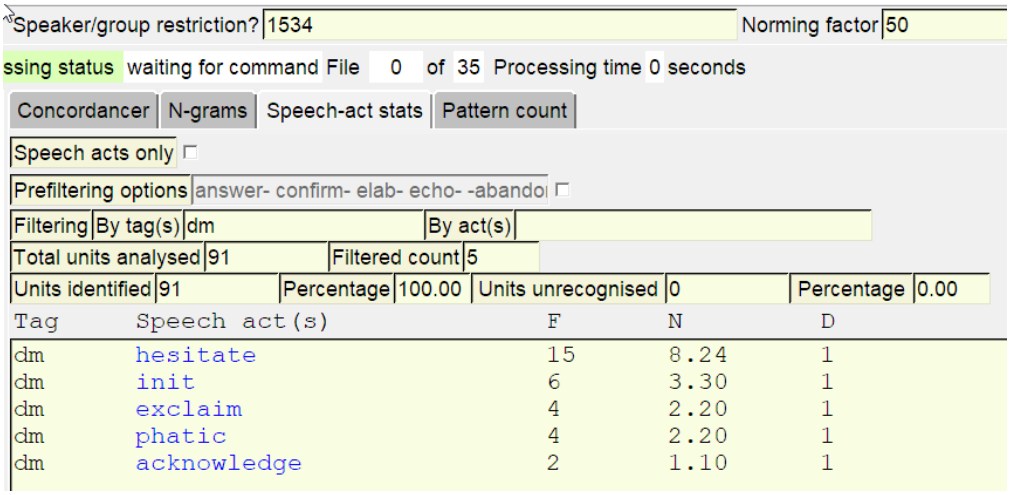


Figure 21 Sample speech-act frequency output, filtered by speaker & tag ‘dm’

As can be seen here, the frequency list also includes information as to how many units were identified or left unlabelled (marked *unrecognised*) on the speech-act level, as well as their percentages. As the list presented above is filtered ‘By tag(s)’ for ‘dm’, the ‘Filtered count’ reported for all units of this particular speaker – reported as ‘Total units analysed’ – contains only 5 tag + act category combinations. In this mode, it is also possible to filter by specific act in addition to tags, or only by acts. The ‘Prefiltering options’ make it possible to exclude ‘secondary’ acts that mark a specific response type or abandonment of the unit from being reported along with the ‘primary’ ones, so that, in effect, all acts that do or don’t occur with excluded secondary ones get reported together. Once the pre-filtering list is activated, it can also be edited, for instance to exclude only markers of abandonment. The list itself is defined

in the DART configuration file and can, if necessary, be adjusted there, too, to create different startup options.

The tag + act frequency list in general allows the researcher to gain an overview of the distribution of particular dialogue features, as well as to identifying potential analysis problems or gaps in the system's rules for a given corpus/domain, especially in cases where it has not been possible to assign a relevant speech act.

The second mode for speech-act analysis is used to report speech-act categories, but, this time, not grouped according to tags. An example of this is displayed in Figure 22.

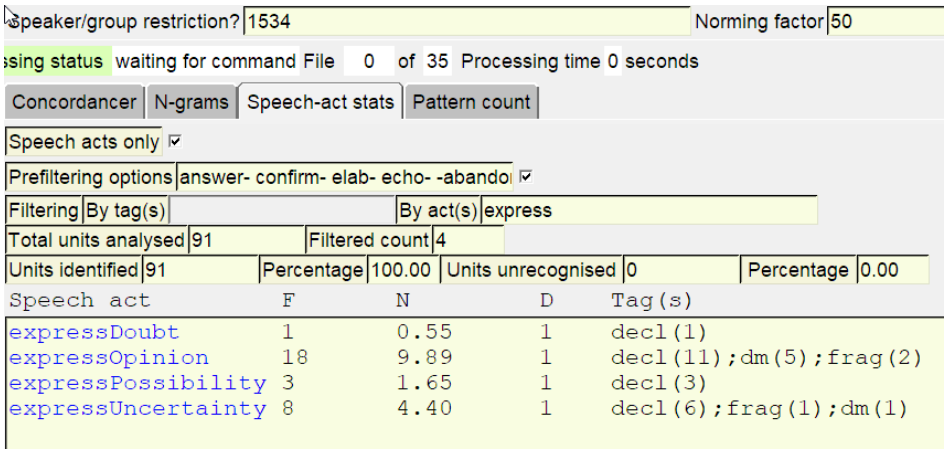
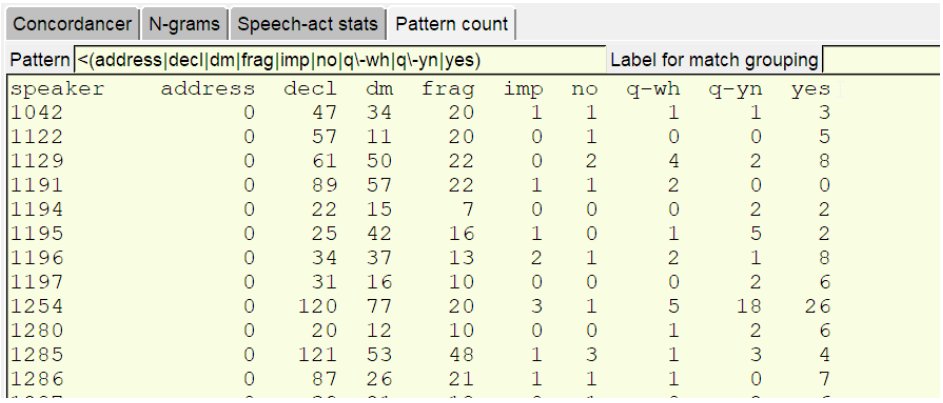


Figure 22 Sample speech-act 'only' frequency output, filtered by act 'express'

Here, all secondary acts are filtered out, and the speech acts reported are filtered by the term 'express', which represents a partial speech act that only occurs as one of its various sub-types, such as *expressOpinion*, which represents a subjective impression or attitude on the part of the speaker. The option for filtering 'By tag(s)' is unchecked, so, even if any expression may still be in the corresponding box, the filter itself will be deactivated. In this mode, the same types of frequencies are reported, but, in addition, a list of all tags the particular speech act occurs with, along with their frequencies, is appended in the 'Tag(s)' column.

5.4 Counting patterns

In addition to the other analysis modes, DART also offers an option for counting (regex) patterns per speaker, and reporting a raw count of the frequencies. In order to do the count, it goes through all selected files, automatically compiles a list of speaker IDs, and counts the specified (sub-)patterns. A listing of such a count for all syntactic units can be seen in Figure 23.


Concordancer | N-grams | Speech-act stats | Pattern count

Pattern: <(address|decl|dm|frag|imp|no|q\-wh|q\-yn|yes) | Label for match grouping:

speaker	address	decl	dm	frag	imp	no	q-wh	q-yn	yes
1042	0	47	34	20	1	1	1	1	3
1122	0	57	11	20	0	1	0	0	5
1129	0	61	50	22	0	2	4	2	8
1191	0	89	57	22	1	1	2	0	0
1194	0	22	15	7	0	0	0	2	2
1195	0	25	42	16	1	0	1	5	2
1196	0	34	37	13	2	1	2	1	8
1197	0	31	16	10	0	0	0	2	6
1254	0	120	77	20	3	1	5	18	26
1280	0	20	12	10	0	0	1	2	6
1285	0	121	53	48	1	3	1	3	4
1286	0	87	26	21	1	1	1	0	7

Figure 23 Pattern count for syntactic unit types (un-grouped)

The above example counts and lists all the syntax categories separately, but as their combination essentially represents the overall number of units per speaker, it would be useful to be able to count and group them together, too. Doing so is possible by providing a grouping label for them in the corresponding box on the right. Thus, for instance using the label ‘units’ in this box, and provided that there is at least one set of grouping brackets present in the ‘Pattern’ box, which is the case in our example, this will allow us to get a joined count for units reported instead of the individual categories.

6 Evaluation options: sample selection & consistency checking

When you have to conduct some form of evaluation on the data that have been analysed, you're often faced with the issue of having to select an unbiased sample. DART facilitates this task by allowing you to specify a fixed number of files to be selected randomly from a given data set loaded into the input files workspace. These files can then be investigated in various ways, either 'manually' or using the DART analysis features described above. This feature is triggered by selecting 'Choose files' from the 'Evaluation' menu depicted below.

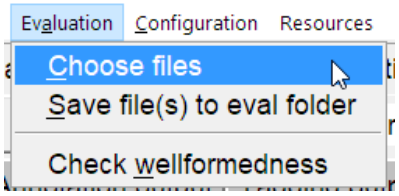


Figure 24 The 'Evaluation' menu

Upon clicking this option, you are prompted to input the number of files to be chosen randomly, and the list of files will then be written to the output files workspace. The next option on the menu, 'Save file(s) to eval folder', will then copy the selected files to a special folder called 'eval' within the 'data' folder for the relevant corpus, also clearing the output files workspace. In the course of this process, a test is also run on the 'eval' folder to determine whether it already contains any files, in which case you can either specify to delete any existing files, add the newly chosen set, or cancel the operation, in case you want to manually back up any existing evaluation files, e.g. to a sub-folder within the 'eval' folder, because DART only deletes files in the main folder, but not (sub-)folders.

The final option, 'Check wellformedness', allows you to test the well-formedness and consistency of the XML annotations to identify errors related to the XML structure of the corpus files. Such errors would otherwise prevent the files from being displayed in a web browser or processed by other applications that 'understand' XML coding if the corpus data is passed on to other researchers who do not have access to DART. Any error messages resulting from this check are written to the Debugging Output window at the bottom of the screen.

The debugging output window can also be used by the expert user to output interim analysis results or other information for debugging purposes. Any error messages produced by the operating system or Perl (which DART is written in) are also redirected there to potentially help the expert user to fix programming errors, and would allow the naïve user to copy and paste them to pass them on to a more experienced one.

Furthermore, at the end of each annotation or tagging operation, an indication of the length of the overall process is also provided in the 'Processing time' indicator near the top of the DART window in order to be able to evaluate the efficiency of the tool in a rather rough manner. It's

important to note, though, that this output only provides a relative measure of efficiency and does not represent a true benchmark, as various factors may influence the behaviour of the tool. Amongst those are the speed of the processor, to some degree the caching mechanisms of the perl interpreter, the amount of debugging information output generated, etc.

7 Creating & editing resources

Being a research environment that tries to incorporate as many of the tasks involved in analysing and annotating dialogues, DART also provides convenient interfaces and mechanisms that enable the researcher to create and update the relevant resources as much as possible. The following sections will describe the different levels on which this is currently feasible – or advisable – in some more detail.

7.1 Adjusting configuration options

One of the central notions in the design of DART is that of *genericity*. This means that many of the resources that are employed in analysing dialogues are initially reduced to a *common core* of elements that represent recurrent features of dialogues on different levels. These resources are always available and already make it possible to analyse dialogues from new domains to a relatively satisfactory extent without even creating or using additional resources. Augmenting the generic core by suitable, more domain-specific resources then generally increases the quality of the analyses/annotations further. Some of the relevant areas where this notion is applicable will be discussed further below.

As DART has thus been designed to maximise the flexibility in adapting its analysis resources to different domains, as well as (at least theoretically) different languages, there are a variety of features that make it possible to create new resources for this purpose, and to later 'pull in' these resources at runtime. The 'Configuration' menu allows you to edit the main configurable properties for DART, such as the startup options or tags, attributes, and values for the XML editor via 'Edit configuration file', as well as to create new domain/corpus folders via 'Edit corpus configuration'.

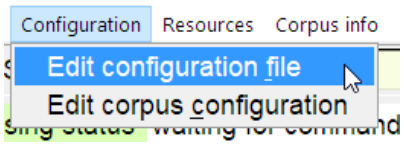
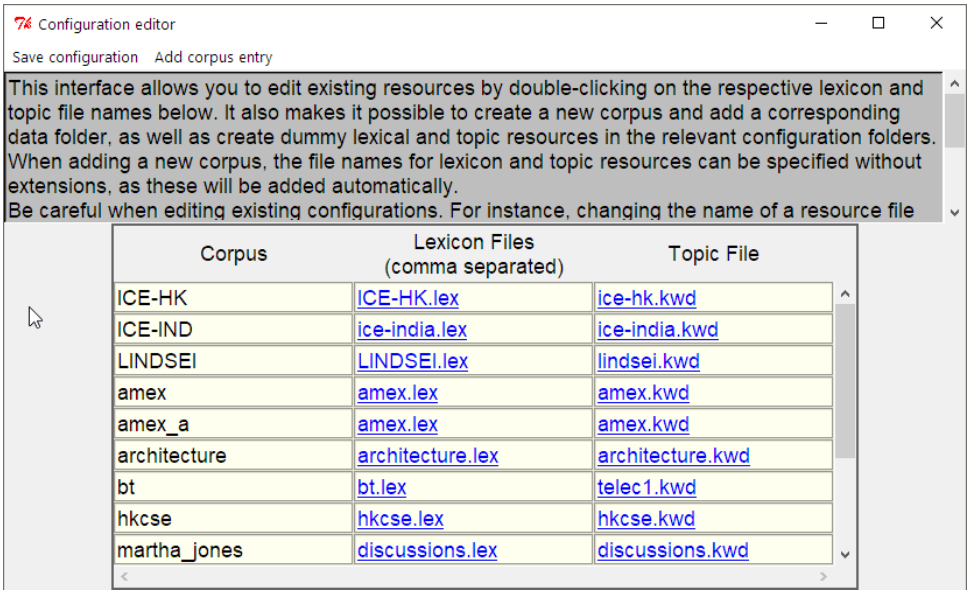


Figure 25 The 'Edit'resources' menu

7.1.1 Working with the corpus configuration editor

Clicking ‘Edit corpus configuration’ will open up the ‘Configuration editor’ window, as illustrated in Figure 26.



Corpus	Lexicon Files (comma separated)	Topic File
ICE-HK	ICE-HK.lex	ice-hk.kwd
ICE-IND	ice-india.lex	ice-india.kwd
LINDSEI	LINDSEI.lex	lindsei.kwd
amex	amex.lex	amex.kwd
amex_a	amex.lex	amex.kwd
architecture	architecture.lex	architecture.kwd
bt	bt.lex	telec1.kwd
hkcse	hkcse.lex	hkcse.kwd
martha_jones	discussions.lex	discussions.kwd

Figure 26 The corpus configuration editor

Through this interface, you can add a new domain/corpus by selecting the ‘Add corpus entry’ option. This will add a new row to the configuration table, where the name of a corpus, as well as (optionally) the names for one or more lexica (comma-separated), as well as a topic file, can be specified. Once you choose the ‘Save configuration’ option, a new folder with the corpus name will be created (if it doesn’t already exist), along with (currently) 3 sub-folders named ‘info’, ‘notes’, and ‘stats’. Please note, though, that files belonging to a corpus need not necessarily reside inside this folder to be able to make use of the associated resources for annotation, etc. as long as the corpus name appears in the header of any dialogue XML file, all the necessary resources associated with it can always be loaded dynamically, provided of course that they don’t inadvertently get deleted.

The first of the sub-folders can be used to store meta information on the corpus and the speakers. Currently, there is only one option on the ‘Corpus info’ menu that make use of this, the ‘Create → Speaker and file info’, which extracts information about the speakers from the files and creates 2 files, one called ‘files_and_speakers.txt’ and the other ‘speakers_and_files.txt’, where the former lists file names and all the speakers that occur in them, while the latter lists speaker names and their associated files. Future versions will probably include facilities for accessing speaker properties, creating group definitions for speakers (e.g. call-centre agents & customers, teachers & students, etc.) to allow you to create comparative descriptive statistics for individual speakers or speaker groups, or potentially define various levels of ‘authority’ for a given speaker

to indicate whether there's a hierarchical structure among/between speakers, which may, for instance, allow DART to 're-define' a speech act *directive* as a command, etc.

The second folder, 'notes', can be used to collect and store textual notes regarding different observations related to the data or keeping records of editing processes, while the 'stats' folder can be used to store the results of any frequency analyses conducted on the corpus.

If filenames have been specified for lexica and topic files, 'dummy' files, containing only information about the entry format and date & time of creation, are created in the 'Lex' and 'Topics' folders for the relevant language[2], respectively, if no file of the same name exists. However, for lexicon files, usually a better option is to 'synthesise' a new lexicon from a corpus loaded into the input files workspace. All lexicon and topic files that already exist when the interface is opened are also hyperlinked to opening the relevant file in the editor.

7.1.2 Customising XML resources

Editing other configuration files is achieved by clicking on 'Edit configuration file'. All these files have the extension ('.cnf'), and, once selected through the file open dialogue, will open in the editor window below the 'Configuration' tab. As the editor windows below tabs have no separate menus or toolbars, saving each open file is achieved by pressing 'Ctrl + s' inside the editor, and closing it via 'Ctrl + w'. If another file is still open in the editor when a new file is opened, and this file has been edited without saving, you will first be prompted to save or discard any changes made to that file before opening the new one. Upon closing DART, it will also check for any unsaved files and prompt you to save them if necessary.

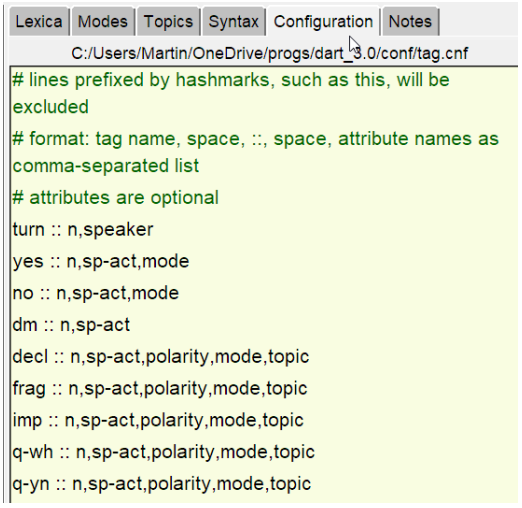


Figure 27 the 'Configuration' tab, showing the XML (container) tag configuration file

[2] As stated before, to date only English is supported.

The format for the different configuration files varies, depending on the function and number of the configuration elements described there, but each format is either easily recognisable or can be identified via a format description at the top. However, what all these files have in common is that they allow the addition of comment lines, which start with a hash mark (#), as can e.g. be seen in Figure 27. Additional items can freely be added by you for all XML-related manual annotation resources, thus allowing different projects to create their own customised annotation options, as well as possibly constraining annotators/editors of the data to use only items that appear on the relevant toolbars or menus. In order to remove existing entries temporarily, the relevant lines can also simply be commented out.

7.1.3 Changing DART startup options or defaults

The main configuration file for DART itself, called 'dart.cnf', currently allows you to configure the following options:

- the default startup folder: this is useful if you work with the same data for a longer period of time and don't want to have to go through the complete routine of choosing a folder all the time in the directory selector dialogue; instead, once the dialogue is open, you can simply press 'Esc' to load the default folder, while pressing 'Enter' without selecting a folder will automatically default to the 'test' folder in the main 'data' folder;
- the norming factor: as stated before, the norming factor is currently set 50, but if you want to norm to higher or lower numbers, perhaps because the dialogues in your corpus are longer, or because you want to norm to the lowest common denominator of units based on particular speaker information from your corpus, you can easily do so;
- the default analysis tab: this is useful for different stages of the analysis process; in the beginning of a new corpus annotation project, it might be most useful to activate the 'Concordancer' or 'N-gram' tabs automatically, while, once a corpus has been fully annotated, switching to either the 'Speech-act stats' or 'Pattern count' may be more useful;
- the background colour: I personally find the default glaring white of most programs difficult on the eyes, especially when working with a program for extended periods of time, but if you don't like my choice for the background, feel free to specify a different hex value ;-)
- the default value for the 'type' attribute of the <punc /> tag: by default, clicking on the button for the empty 'punc' element, the attribute 'stop' will be inserted; if, however, you e.g. find yourself working with data that contains a lot of questions, you can change this to 'query';

- defaults for pre- and post-context: these options allow you to define how many lines before or after a hit in the concordancer should be displayed; this is particularly useful if you want to investigate interaction patterns, such as questions and their responses, in which case you'll probably want to set the default for the lines to follow to something like 7 or even higher, so that you'll be able to see the immediate responses to any queries.
- defaults for height (in lines) and width (in characters) for the analysis and resource editor windows: as the DART layout is fairly complex, it sadly doesn't scale well for different resolutions; I've therefore decided to set some defaults that allow all sub-windows to be displayed, even at lower resolutions; especially if you have a larger monitor, though, you can adjust these settings to be able to arrange the display of the different program sections and the debugging window better;
- the default tagging format: currently, if you use the tagging feature (at all), by default tags will be output on a separate line from the text, hence creating an interlinear version; if you prefer a more common format, such as the traditional underscore one or an XML output, you can change this option;

All of the options above also have explanations preceding them in the configuration file, so you should easily be able to see how to change them. To be on the safe side, in order to be able to revert to older/original settings, you can also copy and comment out the corresponding lines before editing them. In this way, if you inadvertently delete information that might prevent DART from staring properly, despite default also being set inside the program, you can also open the file in any other text editor capable of producing UTF-8 output and change the settings back.

Further default configuration options will probably also be introduced in the future, as well as perhaps a more 'interactive' interface for setting/changing these.

7.2 Editing linguistic resource files

The 'Resources' menu provides access to editing and other functions for the linguistic resources required for the annotation process, namely syntax, mode, topic, and lexicon files, which all open below their respective tabs. Figure 28 shows the relevant menus.

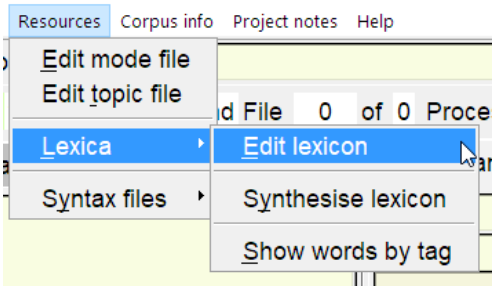


Figure 28 the 'Resources' menu

Out of the sub-menu options depicted above, the 'Lexica' entry provides two additional tasks, one that assists in the creation of new domain-specific lexica, 'Synthesise lexicon', and the other, 'Show words by tag', which makes it possible to check if a word is listed under a specific tag.

'Synthesise lexicon' essentially creates a word list out of the current corpus, then 'subtracts' all items that already exist in the generic lexicon, also carrying out some morphological analysis, and then presents a list of suitable words ('Word'), a dispersion-adjusted frequency value ('AF'), the raw frequency ('F'), and finally a potential tag, in the lexicon editor for further editing (see Figure 29).

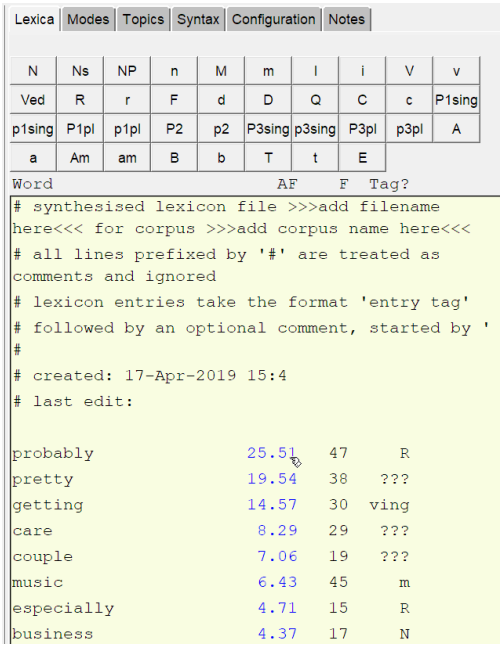


Figure 29 example of a synthesised domain-specific lexicon

The list is sorted according to the adjusted frequency, which represents the importance of the word in the corpus better and should help in making a decision as to whether the word should be included or deleted. The 'AF' value is also hyperlinked to the concordancer in order to be able to verify the use of the word in terms of PoS, especially if it may be grammatically polysemous, i.e. represent multiple word categories. Once a decision has been made to include a word,

rather than delete it, all that remains to be done is to potentially reduce it to its base form, remove the frequency information, and add a/correct the PoS tag by clicking the appropriate button on the tag toolbar above the editor window. All the buttons have tooltips associated with them that explain their meanings, but the most important distinction to remember about the tag set is that case signals whether an item should be considered ambiguous with regard to its potential word class or not, with uppercase indicating non-ambiguity and lowercase ambiguity in most cases, but potentially also a special syntactic role. If a tag that has been suggested through the morphological analysis already seems to be correct, then the word itself need not be included in the lexicon because the same tag, or an even better one, due to additional contextual information, would most likely be assigned during the analysis phase, so keeping it in the lexicon will actually increase redundancy. For instance, nearly all {-ly} adverbs will automatically be tagged correctly as 'R', so usually these don't need to be kept. Instead, more time should be spent trying to understand the function of words that may be incorrect, ambiguous, or have no suggested tag (marked as '???'). In the above example, the word *music*, due to its presumed {-ic} suffix has been marked as being predominantly a modifier, i.e. usually an adjective, but while it may well act as a kind of modifier as the first element in a compound noun, it would be best to change this tag to 'N'.

The remaining menus items, 'Edit modes file', 'Edit topics file', and 'Syntax files', all provide access to facilities for editing or creating particular feature category definitions. *Modes* are semantico-pragmatic markers/key-phrases that indicate specific levels of verbal interaction that, on their own or in combination with syntactic features, assist DART in identifying speech acts. Thus, for instance an expression of a *condition*[3], unless it occurs inside a 'what if' question, signals that the speaker is conveying information that constrains the options provided in the context of the dialogue, in a similar way to how presenting an alternative (see below) 'widens' them. Using the little word *please*, where it is not a verb, clearly indicates that the speaker is making a(n indirect) request, even if the containing syntactic unit takes the form of an interrogative, while a 'discourse marker' like *I see* marks the utterer's *awareness* or understanding of what has been said, etc.

Topics, on the other hand, represent key-phrases that predominantly reflect the true semantic content of a syntactic unit, i.e. what is being talked about. Potential everyday (generic) topics that can be expressed as patterns, for instance, are phrases that indicate dates, times and durations, places, directions, etc., which may also become specifiers for otherwise underspecified referring expressions. More domain-specific ones, e.g. in the Trainline context, would be patterns relating to bookings, particular types of tickets, seat options, journeys, rail cards/passes, etc. The latter are of slightly lesser interest to the pragmatic annotation side, but of course still provide valuable information that is worth investigating, especially if one wants to explore how

[3] Mode labels in the examples below are given in italics.

particular topics (such as making appointments, etc.) are dealt with in a number of different dialogues, and by different speakers or speaker groups.

The modes, topics, and basic syntax files for turn-initial short units (such as discourse markers, yeses, nos, etc.), basically follow the same format. Each line that does not begin with a comment symbol (#) specifies a label associated with a particular category, followed by a space double colon and a space (::). The remaining part of each line then contains one or more regular expression patterns, separated by three underscores (___). These patterns may minimally represent individual words, but unless these unmistakably represent a particular category, this could often be misleading, from either a computational or a conceptual linguistic perspective. For instance, on the computational side, the pattern that represents the mode *alternative* in the current implementation is defined as *either___\bor\b* (*\b* indicating a word boundary), where simply specifying *either___or* would intuitively seem correct to the average linguist who is not accustomed to computational analysis, but lead to fatal errors because not only would this quite rightly find *either*, *neither*, and *or*, but also *floor*, *flour*, *hour*, *poor*, *boring*, *organism*, etc., just because the latter words contain the character sequence <o> + <r>. Similarly, on the linguistic level, if we expected to be able to correctly identify all occurrences of *performatives* by simply specifying the performative marker *hereby* as a pattern, but ignoring that this ought to be preceded by a first-person pronoun, we could potentially end up with a number of false positives where some performative action is only being reported on, etc. Thus, writing appropriate patterns of this kind would minimally require a good understanding of regular expressions and an appropriate amount of linguistic expertise. However, the researcher who expects a particular pattern to be indicative of a specific phenomenon in the data could always at least test it directly on the data by creating the (sub-part of the) expression inside the concordancer and then pasting it into the corresponding resource file. This, in itself could potentially represent a good exercise for researchers who have little experience thinking in terms of linguistic patterns.

Regarding sub-categories for modes and topics, there is again an option to divide these into generic and domain-specific ones, although in general, modes tend to be generic in nature, and there is only one additional modes file so far, which specifies patterns that deal with punctuation. This was actually incorporated into DART only at a relatively late stage because the analysis methodology was originally developed to work with files that excluded all punctuation, and also to some extent why the punctuation is not added directly to the text, but instead indicated by an empty XML element (e.g. *<punc type=”stop” />*). The topic files, however, are clearly divided, and usually it is best to create a new domain-specific one for each corpus that is added, unless the new data is very similar to an earlier corpus. As the labels for modes and topics are also used to create value entries for post-processing tasks, in case it is impossible to define a particular pattern for a feature that is frequently needed, it is also possible to add an ‘empty’ label by simply writing the label name followed by the space, double colon, and space on a separate line in the resource file.

All the resource files discussed immediately above, plus the syntax files for short syntactic units, not only have the same basic syntax for defining patterns, but are also compiled at run-time into more complex regular expression patterns that make counting and adding these features to the annotation output easier. The main difference between modes and topics files, on the one hand, and 'short syntax' files, on the other, is that the latter are not subdivided into generic and domain-specific files, but rather organised into specific categories that are then checked for and removed from the beginnings of longer syntactic unit lines in a specified order and marked up as separate syntactic elements. In contrast, the other features are all added to the respective attribute labels according to their frequency of occurrence, so that the ordering inside the files is in fact irrelevant and can be used more to group related labels for conceptually similar categories together, where it is also advisable to add a comment line before the beginning of each group.

The remaining syntax files represent DART's 'grammars' for the major syntactic categories and, currently, editing the analysis routines unfortunately still requires considerable programming expertise and an in-depth understanding of almost all the processing mechanisms involved, so tinkering with these is not for the faint-hearted and can easily lead to serious annotation errors. Further improvements to produce grammar files that would be easier to understand and manipulate for the 'average linguist' may be developed in future, though, in order to separate the linguistic logic behind the processing even further from that required for the programming part.

8 Keyboard shortcuts

8.1 Corpus File Handling

The following shortcuts are used for loading or removing files from the IO workspace, or creating a new XML corpus file.

Shortcut	Description
Ctrl + Alt + d or F2	load complete corpus folder
Ctrl + Alt + f	load individual file(s)
Ctrl + Alt + n	create new file
Del	remove files selected in workspace

8.2 Editor Shortcuts

The shortcuts described below work in the XML editor and mostly in the resource editor windows, too, depending on the functionality necessary for given tasks. Please note that, unlike in many other editors, the shortcuts for copying, cutting & pasting do not have corresponding entries on the ‘Edit’ menu.

Shortcut	Description
Ctrl + s	save
Ctrl + w	close editor window/file
Ctrl + a	select all
Ctrl + c	copy
Ctrl + x	cut
Ctrl + v	paste
Ctrl + b	copy filename to clipboard
Ctrl + f	find
Ctrl + h	replace
Ctrl + t	renumber turns
Ctrl + u	renumber units
Ctrl + Shift + t	test unit
Ctrl + +	enlarge font size
Ctrl + -	decrease font size
F3	find next selection
Shift + F3	find previous selection
F5	refresh display/colour coding
Shift + F5	insert date and time

Ctrl + q	tag selection as quote

8.3 Punctuation Shortcuts

There are a number of special shortcuts defined inside the annotation editor for quickly inserting different punctuation tags. In general, there are two versions of these, one that only inserts the tag, and the other that inserts the tag and a line break.

Shortcut	Description
Ctrl + .	stop
Ctrl + Alt + .	stop + newline
Ctrl + ,	level
Ctrl + Alt + ,	level + newline
Ctrl + !	exclam
Ctrl + Alt + !	exclam + newline
Ctrl + ?	query
Ctrl + Alt + ?	query + newline
Ctrl + :	incomplete
Ctrl + Alt + :	incomplete + newline

8.4 Annotation & Tagging

The shortcuts below trigger the two different types of annotation.

Shortcut	Description
Ctrl + Alt + a	annotate corpus/file(s)
Ctrl + Alt + t	tag corpus/file(s)

8.5 Analysis Shortcuts

The following shortcuts are relevant in an analysis context, and work for all the tabs.

Shortcut	Description
Ctrl + r	run the analysis (based on selected analysis tab)
Ctrl + o	save analysis output to file (based on selected analysis tab)

9 Known Issues

When ‘installing’ DART into a folder that contains Chinese (or other ‘non-English’) characters, DART is unable to load configuration and/or data files. In this case, please install/copy the distribution into a folder without these characters in the path, e.g. C:\DART or D:\DART.

Avoid installing DART into any of the Windows ‘Programs’ folders, unless you have administrative rights and are prepared to change many permissions for sub-folders and configuration files. It appears that, by default, these folders do not allow you to add or modify files, which is essential for running annotations or customising DART resources.