Localisation of TEX documents: tracklang

Nicola L. C. Talbot

Abstract

TEX is an excellent typesetting system, but its ancient (in computing terms) origin means that it lags behind modern competition in terms of localisation.

Word processors and spreadsheet applications can query the operating system's localisation-related environment variables to determine how to format information, such as dates, times or currency. If the user is writing a single-language document in their own native language, there's no need to keep stating their language and region every time they create a new spreadsheet or word processor document. Whereas with TEX (in its various formats), users may find themselves having to provide this information repeatedly within a single document.

This article describes the development of the tracklang [8] package, which can be used in LATEX or input as a generic TEX file. It attempts to keep track of the localisation setting so that the user doesn't have to redundantly supply information.

1 Introduction

Let's consider two hypothetical people, Alice and Bob. Alice lives in the United Kingdom (UK) and speaks English. Bob lives in Canada and speaks French as his primary language, but is also fluent in English. Alice has her computer set up so that the operating system environment variables include:

```
LANG=en_GB.utf8
LC_ALL=POSIX
```

Bob has something similar, but for some reason he likes to have his messages in English:

```
LANG=fr_CA.utf8
LC_MESSAGES=en_CA.utf8
```

Both Alice and Bob have to send out invoices from time to time. They could just use a spreadsheet which will conveniently look up the localisation variables and format the date using their own regional format (British for Alice and French Canadian for Bob) and will format the currency column according to their region (GBP £ for Alice and CAD $ for Bob). However, Alice and Bob both want to use LATEX, and they've discovered a package called, say, easyinvoice that looks promising.

Alice wants to invoice someone for a DVD costing £5. Rather bizarrely, and with no regard for exchange rates, Bob is coincidentally invoicing someone for a DVD costing C$5. Both start with the same document:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

In both cases this produces the same result:

Invoice Date: June 14, 2016.

Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

This isn't suitable for either Alice or Bob. It's closer to Alice's requirements as it's in English, but the currency is incorrect and the date uses the American style. Alice and Bob both remember about the babel package [1] and decide to load it before easyinvoice. In Alice's case, she does:

```
\usepackage[british]{babel}
```

and Bob does:

```
\usepackage[canadien]{babel}
```

Unfortunately for both of them, this only has a minor change. For Alice, the result is now:

Invoice Date: 14th June 2016.

Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

For Bob, the result is now:

Invoice Date: 14 juin 2016.

Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

So in both cases, the only thing that has changed is the date format. The code for `easyinvoice.sty` is as follows:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{easyinvoice}

\providecommand*{\@date}{\today}
\newcommand{\invoicedatename}{Invoice Date}
\newcommand{\invoiceitemname}{Item}
\newcommand{\invoicepricename}{Price}
\newcommand{\invoicecurrencyname}{EUR}
\newcommand{\invoicepaymentblurb}{Please
pay within 28 days of invoice date.}
```

```
\newcommand*{\itemrow}[2]{\\#1&#2}

\newenvironment{invoice}%
{%
  \par\hfill\invoicedatename: \@date.\par
  \begin{center}%
  \begin{tabular}{lr}
  \invoiceitemname &
  \invoicepricename\
  (\invoicecurrencyname)%
}%
{%
  \end{tabular}%
  \end{center}%
  \invoicepaymentblurb\par
  \medskip\par
}
\endinput
```

The package author has provided a way of altering the fixed names (by providing commands like `\invoicedatename`) but babel can't alter them (since it's unaware of them) and the easyinvoice package author hasn't provided translations. It's therefore necessary for both Alice and Bob to make the necessary changes by redefining the relevant commands. In Alice's case this is just the currency unit:

```
\renewcommand{\invoicecurrencyname}{GBP}
```

However Bob needs to redefine all region-sensitive commands.

This is a nuisance for Alice and Bob (especially Bob) and while they can create a template `.tex` file to copy every time they want to create an invoice, there may be other packages they might want to use that likewise need modifications. It's not the best example for Alice and Bob to present to their spreadsheet-using colleagues in a bid to encourage them to switch to LaTeX.

2 Adding multi-lingual support to packages

Let's suppose now that the author of the easyinvoice package decides to provide some regional support in response to feedback from Alice and Bob. The next version now has some additional lines of code:

```
\newcommand{\invoicebritish}{%
 \renewcommand{\invoicedatename}{Invoice Date}%
 \renewcommand{\invoiceitemname}{Item}%
 \renewcommand{\invoicepricename}{Price}%
 \renewcommand{\invoicecurrencyname}{GBP}%
 \renewcommand{\invoicepaymentblurb}{Please
 pay within 28 days of invoice date.}%
}

\newcommand{\invoicecanadien}{%
 \renewcommand{\invoicedatename}{Date
 de la Facture}%
 \renewcommand{\invoiceitemname}{Article}%
 \renewcommand{\invoicepricename}{Prix}%
 \renewcommand{\invoicecurrencyname}{CAD}%
 \renewcommand{\invoicepaymentblurb}{S'il
 vous pla\^{\i}t payer dans les 28 jours
 suivant la date de facturation.}%
}
```

Now Bob can simply do `\invoicecanadien`, which saves him a few lines of code, but there's not a great deal of difference to Alice who now simply replaces:

```
\renewcommand{\invoicecurrencyname}{GBP}
```

with

```
\invoicebritish
```

Alice may be wondering at this point why the package author has set the defaults to English text with European currency. Many of the packages on CTAN are written by a single author, and the original package was simply to help the author perform some task. The author then decided that the package might be useful to others and made it publicly available. It's therefore not too surprising to find that the package defaults match the requirements of the package author. In this case, it might be that the package author is, say, an English speaker living in the Republic of Ireland (RoI).

How can the easyinvoice package author be more helpful to Alice and Bob? The package could define options that select the appropriate `\invoice`⟨*lang*⟩ command. For example:

```
\DeclareOption{british}{\invoicebritish}
\DeclareOption{canadien}{\invoicecanadien}
\ProcessOptions
```

Now Bob can do:

```
\documentclass{article}
\usepackage[T1]{fontenc}
\usepackage[canadien]{babel}
\usepackage[canadien]{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This is still a bit of a nuisance as Bob has to tell both babel and easyinvoice to use French Canadian. In this example, the document has a single language and is for a single region. The localisation is essentially a document-wide setting here, and therefore this seems a valid instance of making it a document class option:

```
\documentclass[canadien]{article}
\usepackage[T1]{fontenc}
\usepackage{babel}
```

```
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

Now Bob only needs to set this information once per document. Of course, his spreadsheet-using colleagues might point out that they don't need to do it at all, but Bob decides to put up with that.

Bob now remembers that the recipient is in an English-speaking part of Canada, and he decides that perhaps he'd better produce a dual-language invoice, so he tries:

```
\documentclass[canadien,canadian]{article}
\usepackage[T1]{fontenc}
\usepackage{babel}
\usepackage{easyinvoice}

\begin{document}
\selectlanguage{canadien}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}

\selectlanguage{canadian}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This produces:

Date de la Facture: 14 juin 2016.

Article	Prix (CAD)
DVD	5

S'il vous plaît payer dans les 28 jours suivant la date de facturation.

Date de la Facture: 14th June 2016.

Article	Prix (CAD)
DVD	5

S'il vous plaît payer dans les 28 jours suivant la date de facturation.

This hasn't worked for two reasons. The first one being that easyinvoice doesn't provide a `canadian` option. This can be added to the package:

```
\newcommand{\invoicecanadian}{%
 \renewcommand{\invoicedatename}{Invoice Date}%
 \renewcommand{\invoiceitemname}{Item}%
 \renewcommand{\invoicepricename}{Price}%
 \renewcommand{\invoicecurrencyname}{CAD}%
 \renewcommand{\invoicepaymentblurb}{Please
 pay within 28 days of invoice date.}%
}
\DeclareOption{canadian}{\invoicecanadian}
```

Bob's document now produces:

Invoice Date: 14 juin 2016.

Item	Price (CAD)
DVD	5

Please pay within 28 days of invoice date.

Invoice Date: 14th June 2016.

Item	Price (CAD)
DVD	5

Please pay within 28 days of invoice date.

This is because the easyinvoice package isn't aware of the language changes. Only the date has changed because that's controlled by babel. The language in effect is the last one in the easyinvoice options list, as that was the one most recently set.

The babel package has hooks that are used when the language is set, such as `\captions`⟨*lang*⟩ which redefines all the kernel fixed-text commands. To be more generally helpful, the easyinvoice package could test for the existence of `\captionsbritish`, `\captionscanadien` and `\captionscanadian` and add to them. For example, the following code could be added to the easyinvoice package:

```
\@ifundefined{captionsbritish}{}
  {\addto\captionsbritish{\invoicebritish}}
\@ifundefined{captionscanadien}{}
  {\addto\captionscanadien{\invoicecanadien}}
\@ifundefined{captionscanadian}{}
  {\addto\captionscanadian{\invoicecanadian}}
```

With this modification, Bob's document now produces:

Date de la Facture: 14 juin 2016.

Article	Prix (CAD)
DVD	5

S'il vous plaît payer dans les 28 jours suivant la date de facturation.

Invoice Date: 14th June 2016.

Item	Price (CAD)
DVD	5

Please pay within 28 days of invoice date.

Alice and Bob are now both happy, but the package author might be feeling somewhat less so. What started out as a simple, short package has bloated. Each supported language and region combination requires a block of code in the form:

```
\newcommand{\invoicebritish}{%
 \renewcommand{\invoicedatename}{Invoice Date}%
```

```
 \renewcommand{\invoiceitemname}{Item}%
 \renewcommand{\invoicepricename}{Price}%
 \renewcommand{\invoicecurrencyname}{GBP}%
 \renewcommand{\invoicepaymentblurb}{Please
 pay within 28 days of invoice date.}%
}
\DeclareOption{british}{\invoicebritish}
\@ifundefined{captionsbritish}{}
  {\addto\captionsbritish{\invoicebritish}}
```

So far the easyinvoice package only supports three language and region combinations. The more options that are added, the more bloated the package becomes and the harder it is to manage it. Another method is needed to trim down this code. The babel package stores the names of all loaded languages in `\bbl@loaded`. It's a bit risky using an internal command defined by another package, especially if it's not documented in the user guide. Internal commands are the closest packages can get to declaring private variables. There's no guarantee that they won't change or disappear in future versions, but let's suppose the easyinvoice package author decides to take a gamble on it. The three `\@ifundefined` blocks can now be changed from

```
\@ifundefined{captionsbritish}{}
  {\addto\captionsbritish{\invoicebritish}}
\@ifundefined{captionscanadien}{}
  {\addto\captionscanadien{\invoicecanadien}}
\@ifundefined{captionscanadian}{}
  {\addto\captionscanadian{\invoicecanadian}}
```

to:

```
\ifdef\bbl@loaded
{%
  \@for\@this@lang:=\bbl@loaded\do{%
    \ifcsdef{invoice\@this@lang}%
    {%
      \cseappto{captions\@this@lang}{%
        \expandonce
         {\csname invoice\@this@lang
          \endcsname}}%
    }%
    {%
      \PackageWarning{easyinvoice}{Sorry,
       no support for language `\@this@lang'}%
    }%
  }%
}%
{}%
```

(The easyinvoice author has wisely decided to use the etoolbox package [3] to help here.) This method has the added advantage of warning the user if their chosen language isn't supported.

The package options are still declared

```
\DeclareOption{british}{\invoicebritish}
\DeclareOption{canadien}{\invoicecanadien}
\DeclareOption{canadian}{\invoicecanadian}
```

in case the user has decided not to use babel. For example, Alice might decide that she ought to explicitly set the date (using `\date`) so she has a record of it if she needs to recheck it. (She might have removed the PDF after posting it to tidy up her file system.)

```
\documentclass{article}
\usepackage[british]{easyinvoice}
\date{14th June 2016}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This correctly displays the GBP currency.

Betty, from elsewhere in the UK, has discovered the easyinvoice package and decides to use it. Unlike Alice, Betty is in the habit of using babel with the `UKenglish` option, so she tries it out:

```
\documentclass{article}
\usepackage[UKenglish]{babel}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

This produces the error message:

```
Package easyinvoice Warning: Sorry, no
support for language `UKenglish'
```

and displays the following in the output:

Invoice Date: 14th June 2016.

Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

This is because the easyinvoice package doesn't recognise `UKenglish` as a synonym for `british`. A simple fix is to add

```
\newcommand{\invoiceUKenglish}{\invoicebritish}
```

to the package code.

Betty now decides that actually she's going to switch to XeLaTeX and start using the polyglossia package [2] instead. Her document is now:

```
\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage{easyinvoice}
```

```
\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

The invoice goes back to looking like:

Invoice Date: 14th June 2016.

Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

This time easyinvoice gives no warning message. Since babel hasn't been loaded, `\bbl@loaded` is no longer defined, so can't be iterated over.

The earlier method of testing for the existence of commands like `\captionsbritish` no longer works here, as polyglossia only uses the root language name. Thus, although the document has requested the UK variant of English, only `\captionsenglish` is defined.

What should the easyinvoice package do in this situation? Testing if `\captionsenglish` is defined doesn't identify the region. The best that can be done is to modify the package option declarations:

```
\DeclareOption{british}{\invoicebritish
  \ifdef\captionsenglish
  {\appto\captionsenglish{\invoicebritish}}%
  {}%
}
\DeclareOption{canadien}{\invoicecanadien
  \ifdef\captionsfrench
  {\appto\captionsfrench{\invoicecanadien}}%
  {}%
}
\DeclareOption{canadian}{\invoicecanadian
  \ifdef\captionsenglish
  {\appto\captionsenglish{\invoicecanadian}}%
  {}%
}
```

This means that Betty now has to do:

```
\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage[british]{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

That is, Betty has to specify her language and region twice in the document. This shouldn't be necessary.

I mentioned earlier the possibility that the fictional author of such an easyinvoice package might be an English speaker in the RoI. What localisation setting is available for users there who need to write a document in English? The babel package provides the following English options: `english`, `USenglish` (or `american`), `UKenglish` (or `british`), `canadian`, `australian` and `newzealand`; while polyglossia provides: `us` (or `american`), `usmax`, `uk` (or `british`), `australian` and `newzealand`.

Thus, neither babel nor polyglossia provides a way of identifying the English language used in the RoI. The sensible solution would appear to be to use the closest matching alternative. In this case it's `british` (or the UK synonym) to match the date. Aside from political sensitivities, this doesn't help the easyinvoice package because it will assume that the currency should be GBP. There may be other packages the user requires as well that are sensitive to the territory. For example, UTC+1 is generally denoted BST (British Summer Time) in the UK but IST (Irish Summer Time) in the RoI, CET in Jersey or Guernsey, etc.

3 The tracklang package

I have a number of packages for which I want to provide regional support, but they can become so bogged down with the code to determine the document language and region settings that they can end up being too high-maintenance to support. Any development around the basic task of the package becomes sidelined in an attempt to support all the various ways in which a user might want to identify their preferences. Are they using babel or polyglossia or translator (provided with beamer [4]) or ngerman [5] or some other language package that I don't know about?

The aim of the tracklang package is to simplify this. It tries to determine what language and regional settings the user has requested, so that it can provide the information to interested packages in a more accessible manner. It doesn't provide translations. It's not an alternative to babel or polyglossia. It doesn't switch any document settings on. It just attempts to keep track of the user's settings.

3.1 Informing tracklang of the document languages

The LaTeX file `tracklang.sty` inputs the generic TeX code file `tracklang.tex`. LaTeX users can load the package in the usual way:

```
\usepackage{tracklang}
```

However, there's little need to load it directly in the document preamble as it's intended as a resource for package writers, so it's more likely to be loaded in a package:

```
\RequirePackage{tracklang}
```

The only options it has are language or dialect names or regional identifiers to allow them to be picked up from the document class options.

Non-LaTeX users may load the `tracklang.tex` file in the usual way. Pre-version 1.3 required a category code change for the `@` character. Version 1.3 added code to automatically set and restore the catcode for the benefit of non-LaTeX users. Version 1.3 also introduced some new commands to make it easier to query and parse the system environment variables `LC_ALL` and `LANG`.

Since generic code has no concept of document class or package options, generic use requires that the document dialects be identified using

`\TrackPredefinedDialect{`⟨*name*⟩`}`

where ⟨*name*⟩ is a dialect label which is recognised by tracklang.

For example, here's the start of a LaTeX document:

```
\documentclass[british]{article}
\usepackage{tracklang}
```

The analogous plain TeX is:

```
\input tracklang
\TrackPredefinedDialect{british}
```

There are some synonyms available so, for example, instead of `british` I can use `UKenglish` or `en-GB`. The advantage of `british` and `UKenglish` in the document class options list is that they're also recognised by packages such as babel. However, if those packages aren't in use, the ISO form fits in better with global standards.

With version 1.3, you can instead look up your system's language environment variable using

`\TrackLangFromEnv`

This first queries `LC_ALL`. If that's unavailable, it then queries `LANG`. Unfortunately Windows stores the locale information in the registry rather than in environment variables. In this case, if texosquery [7] has also been loaded (either through `\usepackage` for LaTeX users or `\input` for generic use) then tracklang will use texosquery as a fallback if it fails to get a result with the environment variables. (This will also be used as a fallback for LuaTeX if the locale is simply identified as the C or POSIX locale.)

Alice has `LANG` set to `en_GB.utf8`, so instead of

`\TrackPredefinedDialect{british}`

she can just do

`\TrackLangFromEnv`

(Provided either `\directlua` is defined or the shell escape is available.)

The first environment variable to be queried is `LC_ALL`, which Alice has set to `POSIX`. This is not useful for tracklang, (similarly if it had been set to `C`), so `\TrackLangFromEnv` tries again with `LANG`.

As a by-product, the component parts of the localisation identifier are available in the following commands:

`\TrackLangEnvLang`

This contains the language code. For Alice: `en`.

`\TrackLangEnvTerritory`

This contains the territory code. For Alice: `GB`.

`\TrackLangEnvCodeSet`

This contains the code set. For Alice: `utf8`.

`\TrackLangEnvModifier`

This contains the modifier. In Alice's case, this is empty as the modifier isn't present.

The entire value is stored in

`\TrackLangEnv`

If this command has already been defined, then `\TrackLangFromEnv` will skip the environment variable query step. For example, no shell escape (or `\directlua`) is performed with:

```
\def\TrackLangEnv{en_GB}
\TrackLangFromEnv
```

The underscore character here has its usual subscript category code. This is the first choice by tracklang's internal parser when trying to split the language code from the region code. It will also allow a hyphen (with category code 12) as the separator:

```
\def\TrackLangEnv{en-GB}
\TrackLangFromEnv
```

and will finally try the underscore character with category code 12.

```
\edef\TrackLangEnv{en\string_GB}
\TrackLangFromEnv
```

Also, case matters: the language code must be in lower case and the territory code in capitals.

Here's an example plain TeX document:

```
\input tracklang
\TrackLangFromEnv
Language: \TrackLangEnvLang.
Territory: \TrackLangEnvTerritory.
Codeset: \TrackLangEnvCodeSet.
Modifier: \TrackLangEnvModifier.
\bye
```

If this file is called, say, `myDoc.tex`, then if Alice does:

```
pdftex myDoc
```

Then the resulting PDF contains:

Language en. Territory: GB. Codeset: utf8. Modifier: .

Similar results are obtained with ε-TEX, XETEX and LuaTEX. Unfortunately it doesn't work with the non-extended TEX:

```
tex myDoc
```

This produces the following warnings:

```
tracklang Warning: \TrackLangQueryEnv
non-operational as can't determine if the
shell escape has been enabled. (Consider
using eTeX or pdfTeX.)
tracklang Warning: \TrackLangFromEnv
non-operational as \TrackLangEnv is empty
```

Neither `\shellescape` nor `\pdfshellescape` are defined, so tracklang can't determine if the shell escape is available, and therefore it won't make the attempt. This avoids the possibility of triggering the error:

```
! I can't find file `"|kpsewhich --var-value=LC_ALL"'.
l.2 \input |"kpsewhich --var-value=LC_ALL"
(Press Enter to retry, or Control-D to exit)
Please type another input file name:
```

If `\shellescape`/`\pdfshellescape` is defined but is zero (disabled), the first warning changes to:

```
tracklang Warning: \TrackLangQueryEnv
non-operational as shell escape has been
disabled
```

If the shell escape is disabled, Alice can instead define `\TrackLangEnv` from the command line:

```
tex "\\def\TrackLangEnv{$LANG}\\input myDoc"
```

Alternatively, she can use LuaTEX:

```
luatex --no-shell-escape myDoc
```

This now uses `\directlua` to obtain the environment variable value.

Bob doesn't have `LC_ALL` set, but he does have `LC_MESSAGES`. If he wants to query this, he can use:

```
\TrackLangQueryOtherEnv{LC_MESSAGES}
\TrackLangFromEnv
```

This first tries `LC_ALL`, but if that doesn't yield a result, it then tries the variable name provided in the argument (`LC_MESSAGES` in this example). If that also doesn't provide a value, it falls back on `LANG`. The result is again stored in `\TrackLangEnv` so `\TrackLangFromEnv` doesn't repeat the environment variable query. The code can be slightly modified to only perform `\TrackLangQueryOtherEnv` if `\TrackLangEnv` hasn't already been defined:

```
\ifx\TrackLangEnv\undefined
  \TrackLangQueryOtherEnv{LC_MESSAGES}
\fi
\TrackLangFromEnv
```

There's a significant difference between directly setting a dialect using `\TrackPredefinedDialect` (including implicitly through the document class options) and using `\TrackLangFromEnv`.

With `\TrackPredefinedDialect`, an error will occur if an explicit label isn't recognised (or in the case of a document class option, it will be ignored). Whereas with `\TrackLangFromEnv`, if the language and territory combination is unrecognised, tracklang will define a new dialect to represent it.

For example, Jacques from Brussels can use:

```
\TrackPredefinedDialect{fr-BE}
```

since `fr-BE` is recognised by tracklang, but he can't replace `fr-BE` with `en-BE`, since that's not a predefined dialect.

However, Jacques can do:

```
\input tracklang
\def\TrackLangEnv{en-BE}
\TrackLangFromEnv
Language: \TrackLangEnvLang.
Territory: \TrackLangEnvTerritory.
Codeset: \TrackLangEnvCodeSet.
Modifier: \TrackLangEnvModifier.
\bye
```

The resulting PDF now shows:

Language: en. Territory: BE. Codeset: .
Modifier: .

The emphasis here is on reading the locale environment variables such as `LANG` because it's easy to call `kpsewhich` from TEX and capture the output. However, version 1.3 of tracklang also introduces a command for parsing a regular language tag. For example:

```
\TrackLanguageTag{hy-Latn-IT-arevela}
```

The next version of texosquery (1.2) will include a new option which can be used to access the locale information in this format:

```
\input texosquery
\input tracklang
\TeXOSQueryLangTag{\langtag}
\TrackLanguageTag{\langtag}
```

3.2 Support for known language packages

The LATEX file `tracklang.sty` has some awareness of babel, translator, polyglossia and ngerman. After it has input `tracklang.tex` and processed any options, it then tests if any of the declared options have actually been used. For example:

```
\documentclass{article}
\usepackage[british]{babel}
\usepackage{tracklang}
```

Here `british` has been passed to babel, not the document class. This means that it's not detected when tracklang's options are processed.

When this occurs, tracklang has to go through the pesky process of trying to work out if any of the language packages that it knows about have been loaded. If any have, then tracklang needs to work out the language settings. The simplest of these is ngerman. If it's been loaded, that just means doing

```
\TrackPredefinedDialect{ngerman}
```

(Similarly for german.)

The hardest of these is polyglossia, as it currently doesn't keep a list of all the user's selected languages. Instead, tracklang needs to iterate through all known languages and check each one to determine if it has been loaded by testing the existence of \⟨*lang*⟩`@loaded`. (For versions of tracklang before 1.3, the iteration was over a hard-coded list of known polyglossia languages, but this could miss any new languages that might later be supported, so as of v1.3 the iteration is over all tracklang's declared options, which is a longer list and therefore slower.) There's also no way of determining if a language was loaded with a particular variant, so the regional information can't be determined. These limitations may be addressed in the future, which would make integration with polyglossia much easier.

If babel has been loaded, then `\bbl@loaded` should be defined, in which case tracklang can iterate through that list and add each loaded language to the list of tracked dialects. In the event that `\bbl@loaded` isn't defined but babel is loaded, tracklang will iterate through a list of its own predefined dialects that are available as package options and test if the captions hook exists for that option. As with the above case of polyglossia, this is a much longer list.

If translator has been loaded, tracklang iterates over the internal language list `\trans@languages`.

This is a bit clumsy, but it tidies the mess away from other packages so they don't have to do it.

3.3 Querying tracklang for the document languages

All the dialects tracked using the commands in the previous sections are stored by tracklang in an internal list. The *root* languages are stored in another list, and any provided ISO codes are also stored.

This section looks at how a package can query this information to determine which localisation settings need to be applied. You can test if any languages are being tracked using:

```
\AnyTrackedLanguages{⟨true⟩}{⟨false⟩}
```

For example:

```
\input tracklang
\TrackPredefinedDialect{en-GB}
\AnyTrackedLanguages{Yes}{No}.
```

produces: Yes.

You can iterate over all known dialects using

```
\ForEachTrackedDialect{⟨cs⟩}{⟨body⟩}
```

This sets the control sequence given by ⟨*cs*⟩ at the start of each iteration and does ⟨*body*⟩. For example:

```
\input tracklang
\TrackPredefinedDialect{en-CA}
\TrackPredefinedDialect{fr-CA}
Dialects:
\ForEachTrackedDialect
  {\thisdialect}{\thisdialect. }
\bye
```

This produces:

Dialects: canadian. canadien.

If only the root language name is given, that will appear in the dialect list. For example:

```
\input tracklang
\TrackPredefinedDialect{english}
Dialects:
\ForEachTrackedDialect
  {\thisdialect}{\thisdialect. }
\bye
```

This produces:

Dialects: english.

In the case of Jacques' unknown combination:

```
\input tracklang
\def\TrackLangEnv{en-BE}
\TrackLangFromEnv
Dialects:
\ForEachTrackedDialect
  {\thisdialect}{\thisdialect. }
\bye
```

The result is now:

Dialects: enBE.

A useful command that can be used within the body of `\ForEachTrackedDialect` is

```
\IfTrackedLanguageFileExists
  {⟨dialect⟩}{⟨prefix⟩}{⟨suffix⟩}
  {⟨found code⟩}{⟨not found code⟩}
```

This tests the existence of a file whose name is in the form ⟨*prefix*⟩⟨*tag*⟩⟨*suffix*⟩ where the ⟨*tag*⟩ part is determined by the ⟨*dialect*⟩. If a match is found, ⟨*found code*⟩ is performed, otherwise ⟨*not found code*⟩. The ⟨*not found code*⟩ part is also done if ⟨*dialect*⟩ hasn't been added to the list of tracked dialects, or if ⟨*dialect*⟩ is empty, but this situation won't occur when used within the ⟨*body*⟩ argument of `\ForEachTrackedDialect`.

In the ⟨*found code*⟩ part, you can obtain the value of ⟨*tag*⟩ from

`\CurrentTrackedTag`

This means that within ⟨*found code*⟩ you can do

`\input{`⟨*prefix*⟩`\CurrentTrackedTag` ⟨*suffix*⟩`}`

Other convenient commands available for use within ⟨*found code*⟩ are as follows:

`\CurrentTrackedLanguage`

This is set to the root language label (for example, `english` if the dialect is `british`).

`\CurrentTrackedDialect`

This is set to the dialect label (for example, `british`). This is the same value as ⟨*dialect*⟩.

`\CurrentTrackedRegion`

This is set to the region code (ISO 3166-1), if known for this dialect (empty otherwise).

`\CurrentTrackedIsoCode`

This is set to the ISO code (either 639-1 or 639-2) for the root language, if known (empty otherwise).

`\IfTrackedLanguageFileExists` guesses what the ⟨*tag*⟩ should be based on whether or not the dialect has an ISO 3166-1 country code, and if the root language has an ISO 639-1 or 639-2 language code. The first guess that matches a file name on TEX's path will provide the value of ⟨*tag*⟩.

For example, for the `british` dialect, the tries will be in the order: `british` (dialect label), `en-GB` (ISO 639-1 and ISO 3166-1), `eng-GB` (ISO 639-2 and ISO 3166-1), `en` (ISO 639-1), `eng` (ISO 639-2), `GB` (ISO 3166-1), `english` (language label). Whereas, for the `UKenglish` dialect, the tries will be in the order: `UKenglish` (dialect label), `en-GB`, `eng-GB`, `en`, `eng`, `GB`, `english` (language label). It's therefore best not to use a file naming scheme that has dialect labels as the ⟨*tag*⟩ part unless there's a particular reason to treat synonymous dialect labels differently.

Synonyms for the root language are treated as regionless dialects; so, for example, with `francais` the order is just: `francais` (dialect label), `fr`, `fra`, `french` (language label). Compare this with the regionless `french` language where the order is: `french` (dialect label), `fr`, `fra`, `french` (language label). Here the dialect label is identical to the language label so the fourth guess either won't be tried (because a match has already been found) or will fail (because if it did match, it would've been picked up on the first guess).

3.4 Example package using tracklang

Returning to the example easyinvoice package, it no longer needs to define all the language options, as they'll be picked up by tracklang. The code that changes the commands that produce the fixed text (such as `\invoicedatename`) will go in separate files, which will use the naming scheme

`easyinvoice-`⟨*tag*⟩`.ldf`

This fits in with `\IfTrackedLanguageFileExists`, where ⟨*prefix*⟩ is `easyinvoice-` and ⟨*suffix*⟩ is `.ldf`.

These files can simply be input using `\input`, but it's useful to provide an equivalent to commands like `\RequirePackage` and `\ProvidesPackage`. The new improved version of easyinvoice is now:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{easyinvoice}
\RequirePackage{etoolbox}
\RequirePackage{tracklang}

% If user hasn't requested a language
% try LC_ALL or LANG environment variable
\AnyTrackedLanguages{}{\TrackLangFromEnv}

% Set defaults
\newcommand{\invoicedatename}{Invoice Date}
\newcommand{\invoiceitemname}{Item}
\newcommand{\invoicepricename}{Price}
\newcommand{\invoicecurrencyname}{EUR}
\newcommand{\invoicepaymentblurb}{Please
pay within 28 days of invoice date.}

\providecommand*{\@date}{\today}
\newcommand*{\ProvidesInvoiceResource}[1]{%
  \ProvidesFile{easyinvoice-#1.ldf}%
}
\newcommand*{\RequireInvoiceResource}[1]{%
  \ifcsundef{ver@easyinvoice-#1.ldf}%
  {%
    \input{easyinvoice-#1.ldf}%
  }%
  {}%
}
\newcommand*{\RequireInvoiceDialect}[1]{%
  \IfTrackedLanguageFileExists{#1}%
  {easyinvoice-}% prefix
  {.ldf}% suffix
  {%
    \RequireInvoiceResource\CurrentTrackedTag
  }%
  {%
     \PackageWarning{easyinvoice}%
     {No support for dialect `#1'}%
  }%
}
\ForEachTrackedDialect{\this@dialect}{%
  \RequireInvoiceDialect\this@dialect
}

% Main package code:
\newcommand*{\itemrow}[2]{\\#1&#2}
```

```
\newenvironment{invoice}%
{%
  \par\hfill\invoicedatename: \@date.\par
  \begin{center}%
  \begin{tabular}{lr}
  \invoiceitemname &
  \invoicepricename\
  (\invoicecurrencyname)%
}%
{%
  \end{tabular}%
  \end{center}%
  \invoicepaymentblurb\par
  \medskip\par
}
\endinput
```

At the start, this loads tracklang. If it hasn't picked up any localisation, an attempt is made to query the environment variables `LC_ALL` or `LANG`.

Now for the LDF files. The language settings are provided in a file that uses the root language label in the ⟨*tag*⟩ part. The territory settings are provided in a file that uses the ISO country code in the ⟨*tag*⟩ part.

For example, `easyinvoice-english.ldf`:

```
\ProvidesInvoiceResource{english}
\providecommand*{\englishinvoice}{%
  \renewcommand{\invoicedatename}{Invoice Date}%
  \renewcommand{\invoiceitemname}{Item}%
  \renewcommand{\invoicepricename}{Price}%
  \renewcommand{\invoicepaymentblurb}{Please
  pay within 28 days of invoice date.}%
}
\englishinvoice

% polyglossia check: \captions<root language>
\ifundef\captionsenglish
{% babel check: \captions<dialect>
  \ifcsundef{captions\CurrentTrackedDialect}{}%
  {%
    \csgappto{captions\CurrentTrackedDialect}%
      {\englishinvoice}
  }%
}%
{\gappto\captionsenglish{\englishinvoice}}%
\endinput
```

The territory file `easyinvoice-GB.ldf`:

```
\ProvidesInvoiceResource{GB}
\providecommand*\GBinvoice{%
 \renewcommand{\invoicecurrencyname}{GBP}%
}
\GBinvoice
\endinput
```

The dialect settings are stored in a file where the ⟨*tag*⟩ part is formed from the ISO language code and country code. This file needs to load the root language LDF file and the territory LDF file.

For example, `easyinvoice-en-GB.ldf` can look like this:

```
\ProvidesInvoiceResource{en-GB}
\RequireInvoiceResource{english}
\RequireInvoiceResource{GB}

\ifundef\captionsenglish
{%
  \ifcsundef{captions\CurrentTrackedDialect}%
  {}%
  {%
    \csgappto{captions\CurrentTrackedDialect}{%
      \GBinvoice
    }%
  }%
}%
{\gappto\captionsenglish{\GBinvoice}}
```

If, for example, babel has been loaded with the `british` option, this means the `\captionsbritish` hook now includes

```
\englishinvoice
\GBinvoice
```

With polyglossia, these are in `\captionsenglish` (but tracklang must be informed that the `en-GB` dialect is required).

The LDF files rely on `\CurrentTrackedDialect` being set, which it will be when the file is loaded within `\IfTrackedLanguageFileExists`. If an attempt is made to use `\RequireInvoiceResource` when this command hasn't been set, there'll be a problem with the caption hooks.

Although `\RequireInvoiceResource` could include a check for this, `\RequireInvoiceDialect` is a more general purpose command, so it's better to restrict `\RequireInvoiceResource` to use within the resource files:

```
% Default behaviour outside of resource files:
% generate an error and ignore the argument.
\newcommand*{\noop@RequireInvoiceResource}[1]{%
  \PackageError{easyinvoice}
  {\string\RequireInvoiceResource\space only
   permitted within invoice resource files.}
  {}%
}
\let\RequireInvoiceResource
  \noop@RequireInvoiceResource

% Actual behaviour of \RequireInvoiceResource
% used within resource files.
\newcommand*{\@RequireInvoiceResource}[1]{%
  \ifcsundef{ver@easyinvoice-#1.ldf}%
  {%
    \input{easyinvoice-#1.ldf}%
  }%
```

```
  {}%
}

% General use command.
\newcommand*{\RequireInvoiceDialect}[1]{%
  \IfTrackedLanguageFileExists{#1}%
  {easyinvoice-}% prefix
  {.ldf}% suffix
  {%
% Enable \RequireInvoiceResource so that it can
% be used in resource files.
    \let\RequireInvoiceResource
       \@RequireInvoiceResource
% Load resource file.
    \RequireInvoiceResource\CurrentTrackedTag
% Disable \RequireInvoiceResource.
    \let\RequireInvoiceResource
      \noop@RequireInvoiceResource
  }%
  {%
     \PackageWarning{easyinvoice}%
     {No support for dialect `#1'}%
  }%
}
```

(This could be extended to add code prohibiting `\RequireInvoiceDialect` within resource files.)

3.5 Using a tracklang-enabled package

With this new arrangement, Alice can do:

```
\documentclass{article}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

As long as she has the shell escape enabled or she's using LuaLaTeX, the result is:

Invoice Date: June 14, 2016.

Item	Price (GBP)
DVD	5

Please pay within 28 days of invoice date.

This still uses the default US date style because easyinvoice doesn't make any changes to `\today`. Alice could load datetime2 [6] as well, but it might be helpful for easyinvoice to do this automatically.

The datetime2 package also uses tracklang, so it seems the best solution would be to just load it with

```
\RequirePackage{datetime2}
```

However, datetime2 defaults to numeric ISO date style. The `useregional` option is required to switch on the regional support. However, it's best not to use the optional argument of `\RequirePackage` as it can result in a package option clash error if it has already been loaded. It's possible that the user has already loaded datetime2 with their own preferred style, and easyinvoice shouldn't interfere with this.

Thus, a better approach is to use:

```
\PassOptionsToPackage
  {useregional=text}{datetime2}
\RequirePackage{datetime2}
```

Maybe easyinvoice should also allow the user to pass options to datetime2 within easyinvoice's option list:

```
\PassOptionsToPackage
  {useregional=text}{datetime2}
\DeclareOption*{%
 \PassOptionsToPackage
   {\CurrentOption}{datetime2}
}
\ProcessOptions
\RequirePackage{datetime2}
```

So the new improved easyinvoice package now works just fine for Alice; however, Betty, who's using polyglossia, still needs to explicitly indicate her region:

```
\documentclass[en-GB]{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

Since tracklang has detected polyglossia's `english` setting, `\TrackLangFromEnv` isn't used. To help here, the easyinvoice package could provide an option to insist on querying the environment variable even if there are other languages present. For example:

```
\DeclareOption{env}{\TrackLangFromEnv}
```

This means that Betty can now do:

```
\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
\usepackage[env]{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

which saves her the redundant document option.

Another possibility is to add a test for the existence of `\TrackLangEnv`, regardless of whether or not any languages have been detected:

```
\ifdef\TrackLangEnv
  {\TrackLangFromEnv}
  {\AnyTrackedLanguages{}{\TrackLangFromEnv}}
```

This will add to the document's dialect list if it's not already present. In the case of `en-GB`, the dialect is considered a synonym for `british` but not a synonym of `UKenglish`, even though both dialects have the same language and country codes.

If neither babel nor polyglossia are loaded, the last dialect in the list will be the one in effect. For example, the following document adds `fr-CA` to the list of tracked dialects:

```
\documentclass[fr-CA]{article}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

However, if the document is compiled using

```
pdflatex '\def\TrackLangEnv{en-GB}\input{myDoc}'
```

Then the `en-GB` setting will override `fr-CA`.

Seán from the RoI also decides to use easyinvoice but he prefers to have the date include the time and zone information:

```
\documentclass{article}
\usepackage{easyinvoice}
\date{\DTMnow}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

He has `LC_ALL` set to `en_IE` and as he has shell escape enabled (or is using LuaLaTeX) this is added to the list of tracked dialects. In this case, only the `english` LDF file is loaded, not the `GB` or `en-GB` files. This means that the currency is unchanged, which is fine for Seán.

Since datetime2 has been loaded with the regional option on, its `en-IE` style is automatically set, so UTC+1 is displayed as IST, as shown below:

Invoice Date: 14th June 2016 1:10pm IST.

Item	Price (EUR)
DVD	5

Please pay within 28 days of invoice date.

This is a simple solution for all the countries that use the Euro currency; however, multilingual documents that switch from one territory to another need help to return to the default. This can be done by defining a command for setting the country defaults (in `easyinvoice.sty`). For example:

```
\newcommand*{\countrydefaultinvoice}{%
  \renewcommand{\invoicecurrencyname}{EUR}%
}
```

Now the root language LDF file needs to add this to the captions hooks:

```
\ifundef\captionsenglish
{%
  \ifcsundef{captions\CurrentTrackedDialect}{}%
  {%
    \csgappto{captions\CurrentTrackedDialect}%
    {%
      \englishinvoice
      \countrydefaultinvoice
    }%
  }%
}%
{%
  \gappto\captionsenglish{%
   \englishinvoice
   \countrydefaultinvoice
  }%
}
```

Remember that `easyinvoice-en-GB.ldf` adds to the hook after this, so `\GBinvoice` will override this default setting if the dialect is `en-GB`.

Let's not forget about Bob in Canada. He needs `easyinvoice-CA.ldf`:

```
\ProvidesInvoiceResource{CA}
\providecommand*\CAinvoice{%
 \renewcommand{\invoicecurrencyname}{CAD}%
}
\CAinvoice
\endinput
```

The Canadian English file `easyinvoice-en-CA.ldf`:

```
\ProvidesInvoiceResource{en-CA}
\RequireInvoiceResource{english}
\RequireInvoiceResource{CA}

\ifundef\captionsenglish
{%
  \ifcsundef{captions\CurrentTrackedDialect}%
  {}%
  {%
    \csgappto{captions\CurrentTrackedDialect}{%
      \CAinvoice
    }%
  }%
}%
{\gappto\captionsenglish{\CAinvoice}}
\endinput
```

The French Canadian file `easyinvoice-fr-CA.ldf` is similar:

```
\ProvidesInvoiceResource{fr-CA}
```

```
\RequireInvoiceResource{french}
\RequireInvoiceResource{CA}

\ifundef\captionsfrench
{%
  \ifcsundef{captions\CurrentTrackedDialect}%
  {}%
  {%
    \csgappto{captions\CurrentTrackedDialect}{%
      \CAinvoice
    }%
  }%
}%
{\gappto\captionsfrench{\CAinvoice}}
\endinput
```

This needs `easyinvoice-french.ldf`:

```
\ProvidesInvoiceResource{french}
\providecommand*{\frenchinvoice}{%
 \renewcommand{\invoicedatename}{Date
 de la Facture}%
 \renewcommand{\invoiceitemname}{Article}%
 \renewcommand{\invoicepricename}{Prix}%
 \renewcommand{\invoicepaymentblurb}{S'il
 vous pla\^{\i}t payer dans les 28 jours
 suivant la date de facturation.}%
}
\frenchinvoice

\ifundef\captionsfrench
{%
  \ifcsundef{captions\CurrentTrackedDialect}{}%
  {%
    \csgappto{captions\CurrentTrackedDialect}%
    {%
      \frenchinvoice
      \countrydefaultinvoice
    }%
  }%
}%
{%
  \gappto\captionsfrench{%
   \frenchinvoice
   \countrydefaultinvoice
  }%
}
\endinput
```

This suits Jacques just fine as, like Seán, he only needs the root language file since he wants the country default.

Meanwhile Hank, over in the USA, only needs `easyinvoice-US.ldf`:

```
\ProvidesInvoiceResource{US}
\providecommand*\USinvoice{%
 \renewcommand{\invoicecurrencyname}{USD}%
}
\USinvoice
\endinput
```

and `easyinvoice-en-US.ldf`:

```
\ProvidesInvoiceResource{en-US}
\RequireInvoiceResource{english}
\RequireInvoiceResource{US}

\ifundef\captionsenglish
{%
  \ifcsundef{captions\CurrentTrackedDialect}%
  {}%
  {%
    \csgappto{captions\CurrentTrackedDialect}{%
      \USinvoice
    }%
  }%
}%
{\gappto\captionsenglish{\USinvoice}}
\endinput
```

Now Seán decides to provide an Irish Gaelic version `easyinvoice-irish.ldf`:[1]

```
\ProvidesInvoiceResource{irish}
\providecommand*{\irishinvoice}{%
  \renewcommand{\invoicedatename}{D\'ata
    Sonraisc}%
  \renewcommand{\invoiceitemname}{M\'{\i}r}%
  \renewcommand{\invoicepricename}{Praghas}%
  \renewcommand{\invoicepaymentblurb}{Tabhair
    \'{\i}oc laistigh de 28 l\'a \'o dh\'ata
    an tsonraisc.}%
}
\irishinvoice

\ifundef\captionsirish
{%
  \ifcsundef{captions\CurrentTrackedDialect}{}%
  {%
    \csgappto{captions\CurrentTrackedDialect}%
    {%
      \irishinvoice
      \countrydefaultinvoice
    }%
  }%
}%
{%
  \gappto\captionsirish{%
   \irishinvoice
   \countrydefaultinvoice
  }%
}
\endinput
```

Again, he doesn't need to worry about providing a `ga-IE` LDF file since he wants the default currency.

Now Ciaran in Northern Ireland discovers this and tries to produce an invoice in Irish Gaelic:

[1] If the Irish and French text here are a bit iffy, it just goes to show how unwise it is to expect someone to provide translations for languages they don't know or aren't fluent in. They tend to cheat and use a popular translation website.

```
\documentclass[ga-GB]{article}
\usepackage{easyinvoice}

\begin{document}
\begin{invoice}
\itemrow{DVD}{5}
\end{invoice}
\end{document}
```

To his surprise, although the date is in Irish and the currency is GBP, the text is in English:

Invoice Date: 14 Meitheamh 2016.

Item	Price (GBP)
DVD	5

Please pay within 28 days of invoice date.

An inspection of the transcript shows that only the `GB` LDF file has been loaded. The problem here is that there's no `ga-GB` file, so the first LDF file to match ⟨*tag*⟩ is the `GB` file.

The solution is to add `easyinvoice-ga-GB.ldf`:

```
\ProvidesInvoiceResource{ga-GB}
\RequireInvoiceResource{irish}
\RequireInvoiceResource{GB}

\ifundef\captionsirish
{%
  \ifcsundef{captions\CurrentTrackedDialect}%
  {}%
  {%
    \csgappto{captions\CurrentTrackedDialect}{%
      \GBinvoice
    }%
  }%
}%
{\gappto\captionsirish{\GBinvoice}}
\endinput
```

For any new LDF file, no change is required to the code in `easyinvoice.sty`. As long as the files are placed on TEX's path, easyinvoice will detect them.

4 Language packages

A *language package* is one that actually sets the document language (hyphenation patterns, redefining fixed name commands such as `\contentsname`, possibly set fonts and so on; e.g., babel). The easyinvoice package is an example of a package that needs to know the document language. How can language package authors help packages like easyinvoice?

Let's suppose I want to write a language package that sets up a document for Ancient Greek. If this is for single language documents (just Ancient Greek and nothing else), all I need to do is add the following lines to my package:

```
\input{tracklang}% v1.3
\TrackPredefinedDialect{greek}
\SetTrackedDialectModifier{greek}{ancient}
```

I've used `\input` rather than `\RequirePackage` here to skip the tests for babel, polyglossia etc. There's no need to test for the possible language packages because this is the language package. (There's a test in `tracklang.tex` to prevent multiple loading.)

In this case the label `greek` is recognised by tracklang, but if it weren't, I could replace the above with:

```
\input{tracklang}
\TrackLocale{el@ancient}
```

This has the ISO 639-1 code (`el`) with a modifier (`ancient`). `\TrackLocale` works in the same way as `\TrackLangFromEnv` but doesn't use any of the `\TrackLangEnv`... commands. If I prefer to use an IETF language tag I can use `\TrackLanguageTag` instead.

As of version 1.3, tracklang recognises nearly 200 languages with ISO 639-1 or 639-2 codes. However, if my root language isn't included in that list, I can add it using:

```
\AddTrackedLanguage{greek}
\AddTrackedIsoLanguage{639-1}{el}{greek}
\AddTrackedIsoLanguage{639-2}{ell}{greek}
```

or for a regional dialect:

```
\AddTrackedDialect{greekCY}{greek}
\AddTrackedIsoLanguage{639-1}{el}{greek}
\AddTrackedIsoLanguage{639-2}{ell}{greek}
\AddTrackedIsoLanguage{3166-1}{CY}{greekCY}
```

If my package is providing support for multiple languages or dialects with caption hooks in the form `\captions`⟨*lang*⟩, then I also need to use `\AddTrackedDialect` if ⟨*lang*⟩ isn't recognised by tracklang.

```
% user has requested "ancientgreek":
\AddTrackedDialect{ancientgreek}{greek}
\AddTrackedIsoLanguage{639-1}{el}{greek}
\AddTrackedIsoLanguage{639-2}{ell}{greek}
% define caption hook:
\def\captionsancientgreek{%
  ...}
```

In this case, tracklang doesn't recognise 'ancientgreek', but since it does recognise 'greek' and knows the ISO codes for it, I can actually just do:

```
% user has requested "ancientgreek":
\AddTrackedDialect{ancientgreek}{greek}
\AddTrackedLanguageIsoCodes{greek}
% define caption hook:
\def\captionsancientgreek{%
  ...}
```

Now if a user wants to use this language package and easyinvoice, then easyinvoice can find out the

document language without having to know anything about my Ancient Greek package.

Note that the above code is all generic with the exception of

```
\input{tracklang}
```

which needs to be replaced with:

```
\input tracklang
```

for plain TEX. (This syntax also works with LATEX.)

5 Summary

5.1 Document authors

Load the language package before any packages that use tracklang. For example:

```
\documentclass{article}
\usepackage[british]{babel}
\usepackage{easyinvoice}
```

If the region is needed but isn't provided by the language package (or no language package required), use the ISO format. For example:

```
\documentclass[en-IE]{article}
\usepackage[english]{babel}
\usepackage{easyinvoice}
```

Generic use (query operating system):

```
\input tracklang
\TrackLangQueryEnv
\input genericinvoice
```

5.2 Package writers

LATEX packages need to use

```
\RequirePackage{tracklang}
```

to pick up babel, etc., options. Generic use:

```
\input tracklang
```

In either case, if no languages found, query OS:

```
\AnyTrackedLanguages{}{\TrackLangFromEnv}
```

For package `foo`, put the language or regional commands in separate `foo-`⟨*tag*⟩`.ldf` files, which are loaded using

```
\def\RequireFooResource#1{\input foo-#1.ldf}
\def\RequireFooDialect#1{%
  \IfTrackedLanguageFileExists{#1}{foo-}{.ldf}%
    {\RequireFooResource\CurrentTrackedTag}%
    {}% no support warning
}
\ForEachTrackedDialect{\thisdialect}{%
  \RequireFooDialect\thisdialect
}%
```

6 Conclusion

The tracklang package provides a way for package authors to conveniently query the document language settings to make it easier to provide multilingual support. The generic code allows it to be used with multiple TEX formats, and the LATEX code additionally detects and supports common language packages.

`\IfTrackedLanguageFileExists` allows a modular approach so that localisation support can be added and maintained independently of the main package code. This shifts the expectation that a single person (the package author) should not only be able to write TEX code but also be fluent in all known languages and dialects, to a community-based approach with the package author maintaining the base package code and any interested volunteers providing the benefit of their own local knowledge.

References

[1] Javier Bezos and Johannes L. Braams. The babel package, 2016. `ctan.org/pkg/babel`.

[2] François Charette and Arthur Reutenauer. polyglossia: an alternative to the babel package, 2016. `ctan.org/pkg/polyglossia`.

[3] Philipp Lehman. The etoolbox package, 2011. `ctan.org/pkg/etoolbox`.

[4] Vedran Miletić, Joseph Wright, and Till Tantau. The beamer class, 2015. `ctan.org/pkg/beamer`.

[5] Bernd Raichle. Kurzbeschreibung `german.sty` und `ngerman.sty`, 1998. `ctan.org/pkg/german`, `ctan.org/pkg/ngerman`.

[6] Nicola Talbot. The datetime2 package, 2016. `ctan.org/pkg/datetime2`.

[7] Nicola Talbot. texosquery: Query OS information from TEX, 2016. `ctan.org/pkg/texosquery`.

[8] Nicola Talbot. The tracklang package, 2016. `ctan.org/pkg/tracklang`.

◇ Nicola L. C. Talbot
School of Computing Sciences
University of East Anglia
Norwich Research Park
Norwich NR4 7TJ
United Kingdom
`N.Talbot (at) uea dot ac dot uk`
`http://www.dickimaw-books.com`