tracklang.sty v1.6.2: tracking language options

Nicola L.C. Talbot

Dickimaw Books
dickimaw-books.com

2025-01-14

This document is also available as HTML (tracklang-manual.html).

Abstract

The tracklang package is provided for package developers who want a simple interface to find out which languages the user has requested through packages such as babel and polyglossia. *This package doesn't provide any translations.* Its purpose is simply to track which languages have been requested by the user. Generic TEX code is in tracklang.tex for non-LATEX users.

If the shell escape is enabled or \directlua is available, this package may also be used to query the LC_ALL or LANG environment variable (see §6). Windows users, who don't have the locale stored in environment variables, can use texosquery in combination with tracklang. (Similarly if LC_ALL or LANG don't contain sufficient information.) In order to use texosquery through the restricted shell escape, you must have at least Java 8 and set up texosquery.cfg appropriately. (See the texosquery manual for further details.)

The fundamental aim of this generic package is to be able to effectively say:

> The user (that is, the *document* author) wants to use dialects xx-XX, yy-YY-Scrp, etc in their document. Any packages used by their document that provide multilingual or region-dependent support should do whatever is required to activate the settings for those languages and regions (or warn the user that there's no support).

Naturally, this is only of use if the locale-sensitive packages use tracklang to pick up this information, which is entirely up to the package authors, but at the moment there's no standard

method for packages to detect the required language and region. The aim of tracklang is to provide that method. In particular, the emphasis is on using ISO language and region codes rather than hard-coding the various language labels used by different language packages.

Related articles: "Localisation of TEX documents: tracklang." *TUGboat*, Volume 37 (2016), No. 3, Localisation with `tracklang.tex`,[1] and tracklang FAQ.[2]

[1] `dickimaw-books.com/latex/tracklang`
[2] `dickimaw-books.com/faq.php?category=tracklang`

Contents

List of Tables

Part I.

User Guide

1. Introduction

When I'm developing a package that provides multilingual support (for example, glossaries) it's cumbersome trying to work out if the user has requested translations for fixed text. This usually involves checking if babel or translator or polyglossia has been loaded and, if so, what language settings have been used. The result can be a tangled mass of conditional code. The alternative is to tell users to add the language as a document class option, which they may or may not want to do, or to tell them to supply the language settings to every package they load that provides multilingual support, which users are even less likely to want to do.

The tracklang package tries to neaten this up by working out as much of this information as possible for you and providing a command that iterates through the loaded languages. This way, you can just iterate through the list of tracked languages and, for each language, either define the translations or warn the user that there's no translation for that language.

This package works best with ngerman and german (since it's a simple test to determine if they have been loaded) and recent versions of polyglossia (which conveniently provides `\xpg@bcp@loaded`) or when the language options are specified in the document class option list. It works fairly well with translator but will additionally assume the root language was also requested when a dialect is specified. So, for example,

```
\usepackage[british]{translator}
\usepackage{tracklang}
```

is equivalent to

```
\usepackage[british]{translator}
\usepackage[english,british]{tracklang}
```

This means that `\ForEachTrackedDialect` will iterate through the list "`english,british`" instead of just "`british`", which can result in some redundancy.

Unfortunately I can't find any way of detecting a list of languages loaded through babel's new `\babelprovide` command. As far as I can tell, the only stored list is in `\bbl@loaded` which only contains the languages loaded through package options.

If the ngerman package has been loaded, tracklang effectively does:

```
\TrackPredefinedDialect{ngerman}
```

Similarly, if the german package has been loaded, tracklang effectively does

```
\TrackPredefinedDialect{german}
```

If any document class or package options are passed to tracklang, then tracklang won't bother checking for babel, translator, ngerman, german or polyglossia. So, if the above example is changed to:

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

then the dialect list will just consist of "`british`" rather than "`english,british`". This does, however, mean that if the user mixes class and package options, only the class options will be detected. For example:

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

In this case, only the `british` option will be detected. The user can therefore use the document class option (or tracklang package option) to override the dialect and set the country code (where provided). For example:

```
\documentclass[es-MX]{article}
\usepackage[spanish]{babel}
\usepackage{tracklang}
```

This sets the dialect to `mexicanspanish` and the root language to `spanish`.

Predefined dialects are listed in tables 1.1, 1.2 & 1.3. These may be passed in the document class options or used in `\TrackPredefinedDialect`, as illustrated above.

§2 provides brief examples of use for those who want a general overview before reading the more detailed sections. §3 describes generic commands for identifying the document languages. §5 is for package writers who want to add multilingual support to their package and need to know which settings the user has requested through language packages like babel. §6 is for developers of language definition packages who want to help other package writers to detect what languages have been requested.

Table 1.1.: Predefined ISO Language-Region Dialects. (ISO tag or dialect label may be used as a package option or with `\TrackPredefinedDialect`)

ISO Tag	Dialect Label	ISO Tag	Dialect Label
cy-GB	GBwelsh	de-AT	austrian
de-AT-1996	naustrian	de-BE	belgiangerman
de-CH	swissgerman	de-CH-1996	nswissgerman
de-DE	germanDE	de-DE-1996	ngermanDE
en-AU	australian	en-CA	canadian
en-GB	british	en-GG	guernseyenglish
en-IE	IEenglish	en-IM	isleofmanenglish
en-JE	jerseyenglish	en-MT	maltaenglish
en-NZ	newzealand	en-US	american
es-AR	argentinespanish	es-BO	bolivianspanish
es-CL	chilianspanish	es-CO	columbianspanish
es-CR	costaricanspanish	es-CU	cubanspanish
es-DO	dominicanspanish	es-EC	ecudorianspanish
es-ES	spainspanish	es-GT	guatemalanspanish
es-HN	honduranspanish	es-MX	mexicanspanish
es-NI	nicaraguanspanish	es-PA	panamaspanish
es-PE	peruvianspanish	es-PR	puertoricospanish
es-PY	paraguayspanish	es-SV	elsalvadorspanish
es-UY	uruguayspanish	es-VE	venezuelanspanish
fr-BE	belgique	fr-CA	canadien
fr-CH	swissfrench	fr-FR	france
fr-GG	guernseyfrench	fr-JE	jerseyfrench
ga-GB	GBirish	ga-IE	IEirish
gd-GB	GBscottish	hr-HR	croatia
hu-HU	hungarian	id-IN	bahasa
it-CH	swissitalian	it-HR	istriacountyitalian
it-IT	italy	it-SI	sloveneistriaitalian
it-SM	sanmarino	it-VA	vatican
ms-MY	malay	mt-MT	maltamaltese
nl-BE	flemish	nl-NL	netherlands
pt-BR	brazilian	pt-PT	portugal
rm-CH	swissromansh	sl-SI	slovenia

Other combinations need to be set with `\TrackLocale` or `\TrackLanguageTag`

Table 1.2.: Predefined Root Languages. (†Has an associated territory.) The corresponding tag obtained with \GetTrackedLanguageTag{⟨*dialect*⟩} is shown in parentheses

abkhaz (ab)	afar (aa)	afrikaans (af)
akan (ak)	albanian (sq)	amharic† (am-ET)
anglosaxon (ang)	apache (apa)	arabic (ar)
aragonese† (an-ES)	armenian (hy)	assamese (as)
asturian (ast)	avaric (av)	avestan (ae)
aymara (ay)	azerbaijani (az)	bahasai† (id-IN)
bahasam† (ms-MY)	bambara† (bm-ML)	bashkir (ba)
basque (eu)	belarusian (be)	bengali (bn)
berber (ber)	bihari (bh)	bislama† (bi-VU)
bokmal† (nb-NO)	bosnian (bs)	breton† (br-FR)
bulgarian (bg)	burmese (my)	catalan (ca)
chamorro (ch)	chechen (ce)	chichewa (ny)
chinese (zh)	churchslavonic (cu)	chuvash† (cv-RU)
coptic (cop)	cornish† (kw-GB)	corsican (co)
cree (cr)	croatian (hr)	czech (cs)
danish (da)	divehi† (dv-MV)	dutch (nl)
dzongkha† (dz-BT)	easternpunjabi† (pa-IN)	english (en)
esperanto (eo)	estonian (et)	ewe (ee)
faroese (fo)	farsi (fa)	fijian† (fj-FJ)
finnish (fi)	french (fr)	friulan† (fur-IT)
fula (ff)	galician (gl)	ganda† (lg-UG)
georgian (ka)	german (de)	greek (el)
guarani (gn)	gujarati (gu)	haitian† (ht-HT)
hausa (ha)	hebrew (he)	herero (hz)
hindi (hi)	hirimotu† (ho-PG)	icelandic† (is-IS)
ido (io)	igbo (ig)	interlingua (ia)
interlingue (ie)	inuktitut (iu)	inupiaq (ik)
irish (ga)	italian (it)	japanese (ja)
javanese (jv)	kalaallisut (kl)	kannada† (kn-IN)
kanuri (kr)	kashmiri† (ks-IN)	kazakh (kk)
khmer (km)	kikuyu (ki)	kinyarwanda (rw)
kirundi (rn)	komi† (kv-RU)	kongo (kg)
korean (ko)	kurdish (ku)	kwanyama (kj)
kyrgyz (ky)	lao (lo)	latin (la)
latvian (lv)	limburgish (li)	lingala (ln)
lithuanian (lt)	lsorbian† (dsb-DE)	lubakatanga† (lu-CD)
luxembourgish (lb)	macedonian (mk)	magyar (hu)
malagasy (mg)	malayalam† (ml-IN)	maltese (mt)
manx† (gv-IM)	maori† (mi-NZ)	marathi† (mr-IN)
marshallese† (mh-MH)	mongolian (mn)	nauruan† (na-NR)

Table 1.2.: Predefined Root Languages (Continued)

navajo† (nv-US)	ndonga (ng)	nepali (ne)
nko (nqo)	norsk (no)	northernndebele (nd)
northernsotho (nso)	nuosu† (ii-CN)	nynorsk† (nn-NO)
occitan (oc)	ojibwe (oj)	oriya (or)
oromo (om)	ossetian (os)	pali (pi)
pashto (ps)	piedmontese† (pms-IT)	polish (pl)
portuges (pt)	quechua (qu)	romanian (ro)
romansh† (rm-CH)	russian (ru)	samin (se)
samoan (sm)	sango (sg)	sanskrit (sa)
sardinian† (sc-IT)	scottish (gd)	serbian (sr)
shona (sn)	sindhi (sd)	sinhalese† (si-LK)
slovak (sk)	slovene (sl)	somali (so)
southernndebele† (nr-ZA)	southernsotho (st)	spanish (es)
sudanese (su)	swahili (sw)	swati (ss)
swedish (sv)	syriac (syr)	tagalog† (tl-PH)
tahitian† (ty-PF)	tai (tai)	tajik (tg)
tamil (ta)	tatar (tt)	telugu† (te-IN)
thai† (th-TH)	tibetan (bo)	tigrinya (ti)
tonga† (to-TO)	tsonga (ts)	tswana (tn)
turkish (tr)	turkmen (tk)	twi† (tw-GH)
ukrainian† (uk-UA)	undetermined (und)	urdu (ur)
usorbian† (hsb-DE)	uyghur† (ug-CN)	uzbek (uz)
venda† (ve-ZA)	vietnamese (vi)	volapuk (vo)
walloon (wa)	welsh (cy)	westernfrisian† (fy-N
wolof (wo)	xhosa (xh)	yiddish (yi)
yoruba (yo)	zhuang† (za-CN)	zulu (zu)

Table 1.3.: Predefined Non-ISO Dialects. (†Has an associated territory.) The corresponding language tag obtained with \GetTrackedLanguageTag{⟨*dialect*⟩} is shown in parentheses. If the dialect has a corresponding mapping for the closest matching non-root language \captions⟨*dialect*⟩ or \date⟨*dialect*⟩, this is also included after the tag following a slash.

acadian (fr)	american† (en-US)
argentinespanish† (es-AR)	australian† (en-AU)
austrian† (de-AT)	bahasa† (id-IN)
belgiangerman† (de-BE)	belgique† (fr-BE)
bolivianspanish† (es-BO)	brazil† (pt-BR)
brazilian† (pt-BR)	british† (en-GB)
canadian† (en-CA)	canadien† (fr-CA)
chilianspanish† (es-CL)	columbianspanish† (es-CO)
costaricanspanish† (es-CR)	croatia† (hr-HR)
cubanspanish† (es-CU)	cymraeg (cy)
deutsch (de)	dominicanspanish† (es-DO)
ecudorianspanish† (es-EC)	elsalvadorspanish† (es-SV)
flemish† (nl-BE)	francais (fr)
france† (fr-FR)	frenchb (fr)
friulano† (fur-IT)	friulian† (fur-IT)
furlan† (fur-IT)	gaeilge (ga)
gaelic (gd)	galicien (gl)
GBirish† (ga-GB)	GBscottish† (gd-GB)
GBwelsh† (cy-GB)	germanb (de)
germanDE† (de-DE)	guatemalanspanish† (es-GT)
guernseyenglish† (en-GG / british)	guernseyfrench† (fr-GG)
honduranspanish† (es-HN)	hungarian† (hu-HU)
IEenglish† (en-IE / british)	IEirish† (ga-IE)
indon† (id-IN)	indonesian† (id-IN)
isleofmanenglish† (en-IM / british)	istriacountycroatian† (hr-HR)
istriacountyitalian† (it-HR)	italy† (it-IT)
jerseyenglish† (en-JE / british)	jerseyfrench† (fr-JE)
kurmanji (ku)	latein (la)
lowersorbian† (dsb-DE)	malay† (ms-MY)
maltaenglish† (en-MT / british)	maltamaltese† (mt-MT)
mexicanspanish† (es-MX)	meyalu† (ms-MY)
naustrian† (de-AT-1996)	nbelgiangerman† (de-BE-1996 / ngerman)
netherlands† (nl-NL)	newzealand† (en-NZ)

Table 1.3.: Predefined Non-ISO Dialects (Continued)

ngerman (de-1996)	ngermanb (de-1996 / ngerman)
ngermanDE† (de-DE-1996 / ngerman)	nicaraguanspanish† (es-NI)
nil (und)	norwegian† (no-NO)
nswissgerman† (de-CH-1996 / ngerman)	panamaspanish† (es-PA)
paraguayspanish† (es-PY)	persian (fa)
peruvianspanish† (es-PE)	piemonteis† (pms-IT)
polutoniko (el)	polutonikogreek (el)
portugal† (pt-PT)	portuguese (pt)
puertoricospanish† (es-PR)	romanche (rm-CH)
romansch (rm-CH)	rumantsch (rm-CH)
russianb (ru)	sanmarino† (it-SM)
serbianc (sr-Cyrl)	serbianl (sr-Latn)
sloveneistriaitalian† (it-SI)	sloveneistriaslovenian† (sl-SI / slovenian)
slovenia† (sl-SI / slovenian)	slovenian (sl)
spainspanish† (es-ES)	swissfrench† (fr-CH)
swissgerman† (de-CH)	swissitalian† (it-CH)
swissromansh† (rm-CH)	UKenglish† (en-GB)
ukraine† (uk-UA)	ukraineb† (uk-UA)
uppersorbian† (hsb-DE)	uruguayspanish† (es-UY)
USenglish† (en-US)	valencian (ca)
valencien (ca)	vatican† (it-VA)
venezuelanspanish† (es-VE)	

2. Summary of Use

There are three levels of use:

1. document level (code used by document authors);
2. locale-sensitive package level (code for package authors who need to know what languages or locale the document is using, such as glossaries to translate commands like `\descriptionname` or datetime2 to provide localised formats or time zone information);
3. language set-up level (code for packages that set up the document languages, such as babel or polyglossia).

2.1. Document Level

Document level use can be divided into generic TeX use (§2.1.1) and LaTeX-specific use (§2.1.2).

2.1.1. Generic TeX

This section is for generic TeX use. The tracklang files are loaded with `\input`. See §2.1.2 for LaTeX use.

A Unix-like user wants the locale information picked up from the locale environment variable (the `tex` extension may be omitted):

```
\input tracklang.tex % v1.3
\TrackLangFromEnv
% load packages that use tracklang for localisation
```

A Windows user wants the locale information picked up from the operating system (again the `tex` extension may be omitted):

```
\input texosquery.tex
\input tracklang.tex % v1.3
\TrackLangFromEnv
% load packages that use tracklang for localisation
```

Or (texosquery v1.2)

```
\input texosquery.tex % v1.2
\input tracklang.tex % v1.3

\TeXOSQueryLangTag{\langtag}
\TrackLanguageTag{\langtag}
% load packages that use tracklang for localisation
```

A Unix-like user who may or may not have `texosquery` setup to run in the shell escape:

```
\input texosquery.tex
\input tracklang.tex % v1.3

\ifx\TeXOSQueryLangTag\undefined
 \TrackLangFromEnv
\else
 \TeXOSQueryLangTag{\langtag}
 \TrackLanguageTag{\langtag}
\fi
% load packages that use tracklang for localisation
```

A user is writing in Italy in Armenian with a Latin script (`Latn`) and the arevela variant:

```
\input tracklang.tex % v1.3
\TrackLanguageTag{hy-Latn-IT-arevela}
% load packages that use tracklang for localisation
```

A user is writing in English in the UK:

```
\input tracklang.tex
\TrackPredefinedDialect{british}
% load packages that use tracklang for localisation
```

Find out information about the current language (supplied in `\languagename`):

```
\SetCurrentTrackedDialect{\languagename}
Dialect: \CurrentTrackedDialect.
```

```
Language: \CurrentTrackedLanguage.
ISO Code: \CurrentTrackedIsoCode.
Region: \CurrentTrackedRegion.
Modifier: \CurrentTrackedDialectModifier.
Variant: \CurrentTrackedDialectVariant.
Script: \CurrentTrackedDialectScript.
Sub-Lang: \CurrentTrackedDialectSubLang.
Additional: \CurrentTrackedDialectAdditional.
Language Tag: \CurrentTrackedLanguageTag.
```

Additional information about the script can be obtained by also loading tracklang−scripts:

```
\input tracklang-scripts.tex
```

The name, numeric code and direction can now be obtained:

```
Name: \TrackLangScriptAlphaToName{\CurrentTrackedDi-
alectScript}.
Numeric:
\TrackLangScriptAlphaToNumeric{\CurrentTrackedDi-
alectScript}.
Direction:
\TrackLangScriptAlphaToDir{\CurrentTrackedDialect-
Script}.
```

Test for a specific script (in this case `Latn`):

```
Latin?
\ifx\CurrentTrackedDialectScript\TrackLangScriptLatn
 Yes
\else
 No
\fi
```

2.1.2. LaTeX

This section is for LaTeX use. See §2.1.1 for generic TeX use.

With newer versions of polyglossia, where `\xpg@bcp@loaded` is defined, you just need to make sure the languages are set before tracklang is loaded:

```
\documentclass{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
% load packages that use tracklang for localisation
```

For older versions of polyglossia where the regional information is required, use recognised class options:

```
\documentclass[en-GB]{article}
\usepackage{polyglossia}
\setmainlanguage[variant=uk]{english}
% load packages that use tracklang for localisation
```

For babel users where the supplied babel dialect label is sufficient, and is passed either through the document class or package options, there's no need to do anything special:

```
\documentclass[british,canadien]{article}
\usepackage[T1]{fontenc}
\usepackage{babel}
% load packages that use tracklang for localisation
```

If the region is important but there's no babel dialect that represents it, there are several options. The first method is to use the class options recognised by tracklang and the root language labels when loading babel:

```
\documentclass[en-IE,ga-IE]{article}
\usepackage[english,irish]{babel}
% load packages that use tracklang for localisation
```

Another method with babel is to use `\TrackLanguageTag` and map the new dialect label to the nearest matching `\captions`⟨*dialect*⟩:

```
\documentclass{article}

\usepackage{tracklang}% v1.3
\TrackLanguageTag{en-MT}
\SetTrackedDialectLabelMap{\TrackLangLastTracked-
```

```
Dialect}{UKenglish}

\usepackage[UKenglish]{babel}
% load packages that use tracklang for localisation
```

This ensures that the `\captionsUKenglish` hook is detected by the localisation packages. This mapping isn't needed for polyglossia as the caption hooks use the root language label. This mapping also isn't needed if `british` is used instead of `UKenglish` since the `en-MT` (`maltaenglish`) predefined dialect automatically sets up a mapping to `british`. (The default mappings are shown in Table 1.3 on page 7.)

There's no support for `\babelprovide`. If you are using `\babelprovide`, you will need to use the class option or `\TrackLanguageTag` as above.

2.2. Locale-Sensitive Packages

Let's suppose you are developing a package called `mypackage.sty` or `mypackage.tex` and you want to find out what languages the document author has requested. (See also: Using `tracklang.tex` in Packages with Localisation Features.[1])

Generic TEX use (the `tex` extension may be omitted):

```
\input tracklang.tex
```

(Most of the commands used in this section require at least tracklang version 1.3 but 1.4 is better if you want to include the script tag in the `ldf` files.) Note that `tracklang.tex` has a check to determine if it's already been loaded, so you don't need to worry about that.

LATEX use:

```
\RequirePackage{tracklang}
[2019/11/30]% at least v1.4
```

This will picked up any language options supplied in the document class options and will also detect if babel or polyglossia have been loaded.

(LATEX) If you want to allow the user to set the locale in the package options:

```
\DeclareOption*{\TrackLanguageTag{\CurrentOption}}
```

This means the user can do, say,

[1] `dickimaw-books.com/latex/tracklang/otherpkg.shtml`

```
\usepackage[hy-Latn-IT-arevela]{mypackage}
```

With at least version 1.4, it's better to use `\TrackIfKnownLanguage`:

```
\DeclareOption*{%
 \TrackIfKnownLanguage{\CurrentOption}%
 {% successful
  \PackageInfo{mypackage}
{Tracking language `\CurrentOption'}%
 }%
 {% failed
  \PackageError{mypackage}%
  {Unknown language specification `\CurrentOption'}
%
  {You need to supply either a known dialect label
   or a valid language tag}%
 }%
}
```

The rest of the example package in this section uses generic code. If you are using LaTeX, it's better to replace `\def` and `\ifx` with more appropriate LaTeX commands.

If you want to fetch the locale information from the operating system when the user hasn't requested a language:

```
\AnyTrackedLanguages
{}

{% fetch locale information from the operating system
  \ifx\TeXOSQueryLangTag\undefined
    % texosquery v1.2 not available
    \TrackLangFromEnv
  \else
    % texosquery v1.2 available
    \TeXOSQueryLangTag{\langtag}
    \TrackLanguageTag{\langtag}
  \fi
```

```
}
```

Set up the defaults if necessary:

```
\def\fooname{Foo}
\def\barname{Bar}
```

Now load the resource files:

```
\AnyTrackedLanguages
{%
  \ForEachTrackedDialect{\thisdialect}{%
    \TrackLangRequireDialect{mypackage}
{\thisdialect}%
  }%
}
{}% no tracked languages, default already set up
```

Each resource file has the naming scheme ⟨*prefix*⟩-⟨*localeid*⟩`.ldf`. In this example, the ⟨*prefix*⟩ is `mypackage`. The ⟨*localeid*⟩ part may be the language or dialect label (for example, `english` or `british`) or a combination of the ISO language and region codes (for example, `en-GB` or `en` or `GB`). As from version 1.4, ⟨*localeid*⟩ may also include the script or variant. (See the definition of `\IfTrackedLanguageFileExists` on page 44 for further details.)

The simplest scheme is to use the root language label (not the dialect label) for the base language settings and use the ISO codes for regional support.

For example, the file `mypackage-english.ldf`:

```
% identify this file:
\TrackLangProvidesResource{english}[2016/10/06 v1.0]

\TrackLangAddToCaptions{%
  \def\fooname{Foo}%
  \def\barname{Bar}%
}
```

This sets up appropriate the `\captions`⟨*dialect*⟩ hook (if it's found). For other hooks, such as `\date`⟨*dialect*⟩, use `\TrackLangAddToHook` or `\TrackLangRedefHook` instead.

With pre-v1.4 versions of tracklang, the script isn't included in the file search. If it's needed then either require at least v1.4 or have a base `ldf` file that tries to load a version for the particular script (which can be accessed with `\CurrentTrackedDialectScript`).

Here's an example for a language with different writing systems. The resource file for Serbian `mypackage-serbian.ldf`:

```
% identify file:
\TrackLangProvidesResource{serbian}[2016/10/06 v1.0]

\TrackLangRequestResource{serbian-\CurrentTrackedDi-
alectScript}
{}% file not found, do something sensible here
```

The file `mypackage-serbian-Latn.ldf` sets up the Latin script (`Latn`):

```
\TrackLangProvidesResource{serbian-Latn}
[2016/10/06 v1.0]

\TrackLangAddToCaptions{%
  \def\fooname{...}
% provide appropriate Latin translations
  \def\barname{...}%
}
```

The file `mypackage-serbian-Cyrl.ldf` sets up the Cyrillic script (`Cyrl`):

```
\TrackLangProvidesResource{serbian-Cyrl}
[2016/10/06 v1.0]

\TrackLangAddToCaptions{%
  \def\fooname{...}
% provide appropriate Cyrillic translations
  \def\barname{...}%
}
```

With v1.4+ you just need `mypackage-sr-Latn.ldf` and `mypackage-sr-Cyrl.ldf` for the regionless versions.

2.3. Language Packages

Let's suppose now you're the developer of a package that sets up the language, hyphenation patterns and so on. It would be really helpful to the locale-sensitive packages in §2.2 to know what languages the document author has requested. You can use the tracklang package to identify this

information by tracking the requested localisation, so that other packages can have a consistent way of querying it. (See also: Integrating `tracklang.tex` into Language Packages.[2])

Generic use:

```
\input tracklang
```

Alternative LaTeX use:

```
\RequirePackage{tracklang}[2019/11/30]% v1.4
```

Unlike `\input`, `\RequirePackage` will allow tracklang to pick up the document class options, but using `\RequirePackage` will also trigger the tests for known language packages. (If you want to find out if tracklang has already been loaded and locales have already been tracked, you can use the same code as in the previous section.)

When a user requests a particular language through your package, the simplest way of letting tracklang know about it is to use `\TrackPredefinedDialect` or `\TrackLanguageTag`. For example, if the user requests `british`, that's a predefined dialect so you can just do:

```
\TrackPredefinedDialect{british}
```

Alternatively

```
\TrackLanguageTag{en-GB}
```

If your package uses caption hooks, then you can set up a mapping between tracklang's internal dialect label and your caption label. For example, let's suppose the closest match to English used in Malta (`en-MT`) is the dialect `UKenglish` (for example, the date format is similar between GB and MT):

```
\TrackLanguageTag{en-MT}
\SetTrackedDialectLabelMap{\TrackLangLastTracked-
Dialect}{UKenglish}
\def\captionsUKenglish{%
  \def\contentsname{Contents}%
  % ...
}
```

[2] `dickimaw-books.com/latex/tracklang/langpkg.shtml`

(The predefined `maltaenglish` option provided by tracklang automatically sets the mapping to `british`, but the above method will change that mapping to `UKenglish`.)

This now means that `\TrackLangAddToHook` and `\TrackLangRedefHook` commands can find your language hooks. You don't need the map if your dialect label is the same as tracklang's root language label for that locale. For example:

```
\TrackLanguageTag{en-MT}
\def\captionsenglish{%
  \def\contentsname{Contents}%
  % ...
}
```

When the user switches language through commands like `\selectlanguage` it would be useful to also use `\SetCurrentTrackedDialect{`⟨*dialect*⟩`}` to make it easier for the document author or locale-sensitive packages to pick up the current locale. The ⟨*dialect*⟩ argument may be tracklang's internal dialect label or the dialect label you assigned with `\SetTrackedDialectLabelMap`. It may also be the root language label, in which case tracklang will search for the last dialect to be tracked with that language. For example:

```
\def\selectlanguage#1{%
  \SetCurrentTrackedDialect{#1}%
  % set up hyphenation patterns etc
}
```

See the example in §2.1 or the example in Integrating `tracklang.tex` into Language Packages.[3]

[3] `dickimaw-books.com/latex/tracklang/langpkg.shtml`

3. Generic Use

For plain TEX you can input `tracklang.tex`:

```
\input tracklang
```

or for TEX formats that have an argument form for `\input`:

```
\input{tracklang}
```

As from version 1.3, you don't need to change the category code of `@` before loading `tracklang.tex` as it will automatically be changed to 11 and switched back at the end (if required).

The LATEX package `tracklang.sty` inputs the generic TEX code in `tracklang.tex`, but before it does so it defines

```
\@tracklang@declareoption{⟨dialect⟩}
```

to

```
\DeclareOption{⟨dialect⟩}{\TrackPredefinedDialect{⟨dialect⟩}
}
```

If `\@tracklang@declareoption` isn't defined when `tracklang.tex` is input, it will be defined to ignore its argument.

This means that all the predefined languages and dialects (tables 1.1, 1.2 & 1.3) automatically become package options, so the `tracklang.sty` package can pick up document class options and add them to tracklang's internal list of tracked document languages.

If you're not using LATEX, this option isn't available although you can redefine `\@tracklang@declareoption` to use something analogous to `\DeclareOption`, if appropriate. Otherwise, the document languages need to be explicitly identified (using any of the following commands) so that tracklang knows about them.

```
\TrackPredefinedDialect{⟨dialect label⟩}
```

This will add the predefined dialect and its associated ISO codes to the list of tracked document languages. The ⟨*dialect label*⟩ may be any of those listed in tables 1.1, 1.2 & 1.3.

For example:

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

is the Plain TEX alternative to:

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

Note that it's impractical to define every possible language and region combination as it would significantly slow the time taken to load tracklang so, after version 1.3, I don't intend adding any new predefined dialects. As from version 1.3, if you want to track a dialect that's not predefined by tracklang, then you can use:

```
\TrackLocale{⟨locale⟩}
```

If ⟨*locale*⟩ is a recognised dialect, this is equivalent to using `\TrackPredefinedDialect`, otherwise ⟨*locale*⟩ needs to be in one the following formats:

- ⟨*ISO lang*⟩
- ⟨*ISO lang*⟩`@`⟨*modifier*⟩
- ⟨*ISO lang*⟩`-`⟨*ISO country*⟩
- ⟨*ISO lang*⟩`-`⟨*ISO country*⟩`@`⟨*modifier*⟩

where ⟨*ISO lang*⟩ is the ISO 639-1 or 639-2 code identifying the language (lower case), ⟨*ISO country*⟩ is the 3166-1 ISO code identifying the territory (upper case) and ⟨*modifier*⟩ is the modifier or variant. The hyphen (`-`) may be replaced by an underscore character (`_`). Codeset information in the form `.`⟨*codeset*⟩ may optionally appear before the modifier. For example, `de-DE.utf8@new` (modifier is `new`) or `en-GB.utf8` (modifier is missing). The codeset will be ignored if present, but it won't interfere with the parsing.

For example:

```
\TrackLocale{de-NA@new}
```

indicates German in Namibia using the new spelling.

If a language has different ISO 639-2 (T) and 639-2 (B) codes, then the "T" form should be used. (So for the above example, `deu` may be used instead of `de`, but `ger` won't be recognised.)

Alternatively, you can use

```
\TrackLanguageTag{⟨tag⟩}
```

where ⟨*tag*⟩ is a regular, well-formed language tag or a recognised dialect label. (Irregular grandfather tags aren't recognised.) Note that the tag must start with a language identifier and can't simply be a region code (use "und" if the language is unknown). This command will fully expand ⟨*tag*⟩. A warning is issued if the tag is empty.

If you want to first check that ⟨*tag*⟩ includes a valid language code, then you can instead use:

```
\TrackIfKnownLanguage{⟨tag⟩}{⟨success code⟩}{⟨fail code⟩}
```

This will only track ⟨*tag*⟩ (and then do ⟨*success code*⟩) if ⟨*tag*⟩ starts with a valid language code (or is a predefined dialect) otherwise it will do ⟨*fail code*⟩. Both `\TrackLanguageTag` and `\TrackIfKnownLanguage` will check if ⟨*tag*⟩ is a predefined option. (This saves parsing the tag if it's recognised.)

For example:

```
\TrackLanguageTag{hy-Latn-IT-arevela}
Latn-ME: \TrackIfKnownLanguage{Latn-ME}{success}
{fail}.
brazilian: \TrackIfKnownLanguage{brazilian}{success}
{fail}.
```

This will track `hy-Latn-IT-arevela` and `brazilian` (`pt-BR`) but not `Latn-ME` (because it doesn't contain a valid language code) even though it's a valid script and country code. The above is just for illustrative purposes. Typically the language tracking isn't performed within the document text.

The datetime2 package assumes that any unknown package option is a language identifier. It could simply do:

```
\TrackLanguageTag{\CurrentOption}
```

but users can make mistakes sometimes and this won't provide any helpful information if they, for example, misspelt a package option or forgot the "⟨*key*⟩=" part of a ⟨*key*⟩=⟨*value*⟩ setting. Instead (as from v1.5.5) datetime2 now does:

```
\TrackIfKnownLanguage{\CurrentOption}
{...}% known language
{\PackageError{...}{...}{...}}
```

This will now give the user some guidance.

If ⟨*tag*⟩ contains a sub-language tag, this will be set as the 639-3 code for the *dialect* label. Note that this is different to the root language codes which are set using the language label. For example:

```
\TrackLanguageTag{zh-cmn-Hans-CN}
```

creates a new dialect with the label `zhcmnHansCN`. The root language `chinese` has the 639-1 code `zh` and the dialect `zhcmnHansCN` has the ISO 639-3 code `cmn`.

```
ISO 639-1: \TrackedIsoCodeFromLanguage{639-1}
{chinese}.
ISO 639-3: \TrackedIsoCodeFromLanguage{639-3}
{zhcmnHansCN}.
```

Version 1.2 of texosquery provides the command `\TeXOSQueryLangTag`, which may be used to fetch the operating system's regional information as a language tag. These commands can be used as follows:

```
\input tracklang % v1.3
\input texosquery % v1.2

\TeXOSQueryLangTag{\langtag}
\TrackLanguageTag{\langtag}
```

(If the shell escape is disabled, `\langtag` will be empty, which will trigger a warning but no errors.)

Some of the predefined root language options listed in Table 1.2 on page 5 have an associated region (denoted by †). If `\TrackLocale` is used with just the language ISO code, no region is tracked for that language. For example

```
\TrackLocale{manx}
```

will track the `IM` (Isle of Man) ISO 3166-1 code but

```
\TrackLocale{gv}
```

won't track the region. Similarly for `\TrackLanguageTag`.

(New to version 1.3.) There's a similar command to `\TrackLocale` that doesn't take an argument:

```
\TrackLangFromEnv
```

If the shell escape has been enabled or `\directlua` is available, this will try to get the language information from the system environment variables `LC_ALL` or `LANG` and, if successful, track that.

Since tracklang is neither able to look up the POSIX locale tables nor interpret file locales, if the result is `C` or `POSIX` or starts with a forward slash `/` then the locale value is treated as empty.

> Not all operating systems use environment variables for the system locale information. For example, Windows stores the locale information in the registry. In which case, consider using texosquery.

If the operating system locale can't be obtained from environment variables, then tracklang will use `\TeXOSQueryLocale` as a fallback if texosquery has been loaded. Since texosquery requires both the shell escape and the Java runtime environment, tracklang doesn't automatically load it.

Plain TEX example:

```
\input texosquery
\input tracklang
\TrackLangFromEnv
```

Document build:

```
etex --shell-escape ⟨filename⟩
```

LATEX example:

```
\usepackage{texosquery}
\usepackage{tracklang}
```

```
\TrackLangFromEnv
```

Document build:

```
pdflatex --shell-escape ⟨filename⟩
```

If the locale can't be determined, there will be warning messages. These can be suppressed using

```
\TrackLangShowWarningsfalse
```

or switched back on again using

```
\TrackLangShowWarningstrue
```

For example, I have the environment variable `LANG` set to `en_GB.utf8` on my Linux system so instead of

```
\TrackPredefinedDialect{british}
```

I can use

```
\TrackLangFromEnv
```

With LaTeX documents I can do

```
\documentclass{article}
\usepackage{tracklang}
\TrackLangFromEnv
```

However, this only helps subsequently loaded packages that use tracklang to determine the required regional settings. For example:

```
\documentclass{article}
\usepackage{tracklang}
\TrackLangFromEnv
```

```
\usepackage[useregional]{datetime2}
```

In my case, with the LANG environment variable set to en_GB.utf8 and the shell escape enabled, this automatically switches on the en-GB date style. Naturally this doesn't help locale-sensitive packages that don't use tracklang.

The \TrackLangFromEnv command also incidentally sets \TrackLangEnv to the value of the environment variable or empty if the query was unsuccessful (for example, the shell escape is unavailable).

If the command:

```
\TrackLangEnv
                                                    user defined
```

is already defined before \TrackLangFromEnv is used, then the environment variable won't be queried and the value of \TrackLangEnv will be parsed instead.

> The parser which splits the locale string into its component parts first tries splitting on the underscore _ with its usual category code 8, then tries splitting on a hyphen - with category code 12, and then tries splitting on the underscore _ with category code 12.

For example:

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

This doesn't perform a shell escape since \TrackLangEnv is already defined. In this case, you may just as well use:

```
\TrackLocale{en-GB}
```

(unless you happen to additionally require the component commands that are set by \TrackLangFromEnv, see below.)

If the shell escape is unavailable (for example, your TEX installation prohibits it), you can set this value when you invoke TEX. For example, if the document file is called myDoc.tex (and it's in Plain TEX):

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

The \TrackLangFromEnv command also happens to store the component parts of the environment variable value in the following commands. (These aren't provided by \TrackLocale.) If the information is unavailable, the relevant commands will be set to empty.

The language code is stored in:

```
\TrackLangEnvLang
```

The territory (if present) is stored in:

```
\TrackLangEnvTerritory
```

The code-set (if present) is stored in:

```
\TrackLangEnvCodeSet
```

The modifier (if present) is stored in:

```
\TrackLangEnvModifier
```

If you want to query the language environment, but don't want to track the result, you can just use:

```
\TrackLangQueryEnv
```

This only tries to fetch the value of the language environment variable (and use texosquery as a fallback, if it has been loaded). It doesn't try to parse the result. The result is stored in \TrackLangEnv (empty if unsuccessful). Unlike \TrackLangFromEnv, this doesn't check if \TrackLangEnv already exists. A warning will occur if the shell escape is unavailable. For systems that store the locale information in environment variables, this is more efficient than using texosquery's \TeXOSQueryLocale command (which is what's used as the fallback).

The above queries LC_ALL and, if that is unsuccessful, then queries LANG (before optionally falling back on texosquery). If you want another environment variable tried after LC_ALL and before LANG, you can instead use:

```
\TrackLangQueryOtherEnv{⟨env-name⟩}
```

For example, to also query LC_MONETARY:

```
\TrackLangQueryOtherEnv{LC_MONETARY}
```

Since this sets \TrackLangEnv, you can use it before \TrackLangFromEnv. For example:

```
\TrackLangQueryOtherEnv{LC_MONETARY}
\TrackLangFromEnv
```

Remember that if you only want to do the shell escape if \TrackLangEnv hasn't already been defined, you can test for this first:

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

It's also possible to just parse the value of \TrackLangEnv without tracking the result using:

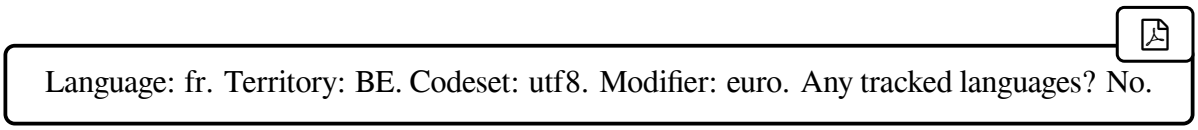
```
\TrackLangParseFromEnv
```

This is like \TrackLangFromEnv but assumes that \TrackLangEnv has already been set and doesn't track the result. The component parts are stored as for \TrackLangFromEnv.

Example (Plain TEX):

```
\input tracklang

\def\TrackLangEnv{fr-BE.utf8@euro}

\TrackLangParseFromEnv

Language: \TrackLangEnvLang.
Territory: \TrackLangEnvTerritory.
Codeset: \TrackLangEnvCodeSet.
Modifier: \TrackLangEnvModifier.
Any tracked languages? \AnyTrackedLanguages{Yes}{No}
.
```

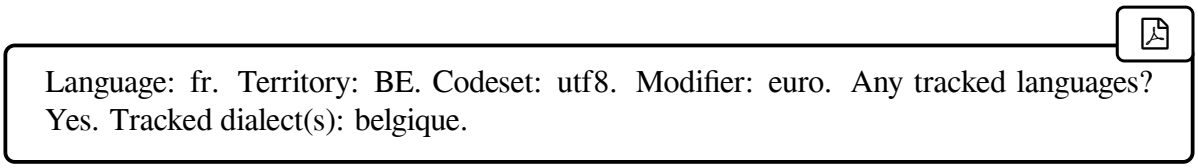
This produces:



Compare this with:

```
\input tracklang

\def\TrackLangEnv{fr-BE.utf8@euro}

\TrackLangFromEnv

Language: \TrackLangEnvLang.
Territory: \TrackLangEnvTerritory.
Codeset: \TrackLangEnvCodeSet.
Modifier: \TrackLangEnvModifier.
Any tracked languages? \AnyTrackedLanguages{Yes}{No}
.
Tracked dialect(s):%
\ForEachTrackedDialect{\thisdialect}
{\space\thisdialect}.
```

This produces:



If `\TrackLangFromEnv` doesn't recognise the given language and territory combination, it will define a new dialect and add that.

For example, tracklang doesn't recognise `en-BE`, so the sample document below defines a new dialect labelled `enBEeuro`:

```
\input tracklang

\def\TrackLangEnv{en-BE.utf8@euro}

\TrackLangFromEnv

Language: \TrackLangEnvLang.
```

```
Territory: \TrackLangEnvTerritory.
Codeset: \TrackLangEnvCodeSet.
Modifier: \TrackLangEnvModifier.
Any tracked languages? \AnyTrackedLanguages{Yes}{No}
.
Tracked dialect(s):%
\ForEachTrackedDialect{\thisdialect}
{\space\thisdialect}.
```

This now produces:

Language: en. Territory: BE. Codeset: utf8. Modifier: euro. Any tracked languages? Yes. Tracked dialect(s): enBEeuro.

4. Supplementary Packages

In addition to the main `tracklang.tex` file and `tracklang.sty` LaTeX wrapper, the tracklang package also provides supplementary files for region and script mappings.

`tracklang-region-codes.tex`

This file is only loaded if a mapping is required between numeric and alphabetic region codes. If `\TrackLanguageTag` encounters a numeric region code, it will automatically input `tracklang-region-codes.tex`, if it hasn't already been input. This file provides the following commands.

`\TrackLangAlphaIIToNumericRegion{`⟨*alpha-2 code*⟩`}`

Expands to the numeric code corresponding to the given alpha-2 code or empty if no mapping has been supplied.

`\TrackLangNumericToAlphaIIRegion{`⟨*numeric code*⟩`}`

Expands to the alpha-2 code corresponding to the given numeric code or empty if no mapping has been supplied.

`\TrackLangIfKnownAlphaIIRegion{`⟨*alpha-2 code*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if there's an alpha-2 to numeric region code mapping, otherwise expands to ⟨*false*⟩.

`\TrackLangIfKnownNumericRegion{`⟨*numeric code*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if there's a numeric to alpha-2 region code mapping, otherwise expands to ⟨*false*⟩.

`\TrackLangAlphaIIIToNumericRegion{`⟨*alpha-3 code*⟩`}`

Expands to the numeric code corresponding to the given alpha-3 code or empty if no mapping has been supplied.

```
\TrackLangNumericToAlphaIIIRegion{⟨numeric code⟩}
```

Expands to the alpha-3 code corresponding to the given numeric code or empty if no mapping has been supplied.

```
\TrackLangIfKnownAlphaIIIRegion{⟨alpha-3 code⟩}{⟨true⟩}{⟨false⟩}
```

Expands to ⟨*true*⟩ if there's an alpha-3 to numeric region code mapping, otherwise expands to ⟨*false*⟩.

Mappings are established with:

```
\TrackLangRegionMap{⟨numeric⟩}{⟨alpha-2⟩}{⟨alpha-3⟩}
```

Predefined mappings are listed in Table A.1 on page 84.

When `tracklang-region-codes.tex` is input, it can load additional files that provide supplementary mappings.

```
\TrackLangAddExtraRegionFile{⟨file⟩}
```

This command adds the supplied ⟨*file*⟩ to the list of extra region code files that should be input by `tracklang-region-codes.tex`, unless `tracklang-region-codes.tex` has already been input, in which case ⟨*file*⟩ will be input straight away.

`tracklang-scripts.tex`

The tracklang−scripts package provides information about ISO 15924 scripts. The file isn't automatically loaded. If you want to use any of the commands provided in it you need to input it.

Plain TEX:

```
\input tracklang-scripts
```

There's a simple wrapper package `tracklang-scripts.sty` for LATEX users:

```
\usepackage{tracklang-scripts}
```

\TrackLangScriptMap{⟨*letter code*⟩}{⟨*numeric code*⟩}{⟨*script name*⟩}{⟨*direction*⟩}{⟨*parent script*⟩}

Defines a mapping. The first argument is the four letter alpha code, such as `Latn` or `Cyrl`. The second argument is the numeric code. The third argument is the script's name, for example "Imperial Aramaic". The fourth argument is the direction, which may be one of: `LR` (left-to-right), `RL` (right-to-left), `TB` (top-to-bottom), `varies` or `inherited`. The ⟨*parent*⟩ argument is for the parent writing system, which may be left blank (currently unsupported).

This command defines:

\TrackLangScript⟨*Code*⟩

which expands to ⟨*Code*⟩ for use with `\IfTrackedDialectIsScriptCs`.

See Table A.2 on page 88 for a summary of all the mappings that are provided by the file `tracklang-scripts.tex`.

\TrackLangScriptAlphaToNumeric{⟨*alpha code*⟩}

Expands to the numeric code corresponding to the given alpha code or empty if no mapping.

\TrackLangScriptIfKnownAlpha{⟨*alpha code*⟩}{⟨*true*⟩}{⟨*false*⟩}

Expands to ⟨*true*⟩ if there is a known alpha to numeric mapping or ⟨*false*⟩ otherwise.

\TrackLangScriptNumericToAlpha{⟨*numeric code*⟩}

Expands to the alpha code corresponding to the given numeric code or empty if no mapping.

\TrackLangScriptIfKnownNumeric{⟨*numeric code*⟩}{⟨*true*⟩}{⟨*false*⟩}

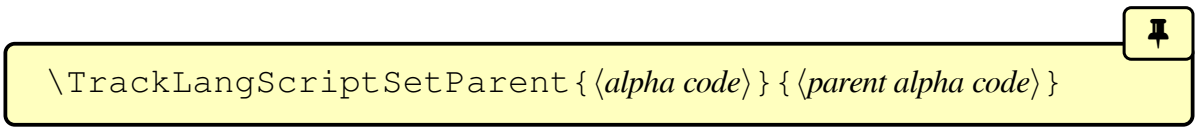
Expands to ⟨*true*⟩ if there is a known numeric to alpha mapping or ⟨*false*⟩ otherwise.

\TrackLangScriptAlphaToName{⟨*alpha code*⟩}

Expands to the name corresponding to the given alpha code or empty if no mapping.

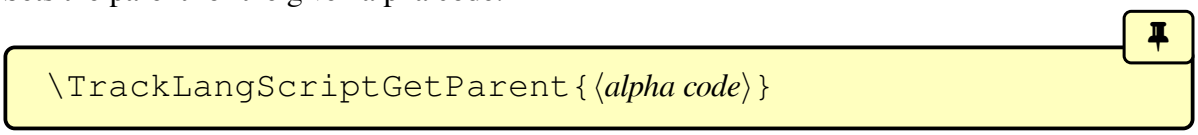
\TrackLangScriptAlphaToDir{⟨*alpha code*⟩}

Expands to the direction corresponding to the given alpha code or empty if no mapping.



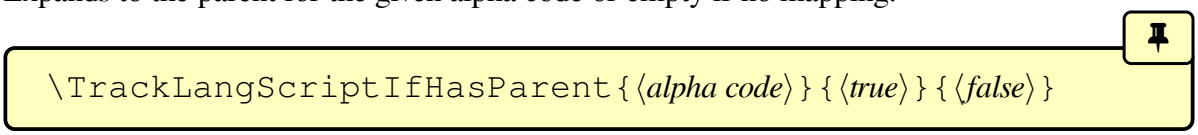
`\TrackLangScriptSetParent{`⟨*alpha code*⟩`}{`⟨*parent alpha code*⟩`}`

Sets the parent for the given alpha code.



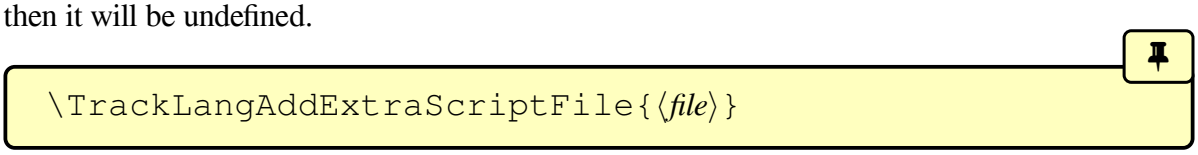
`\TrackLangScriptGetParent{`⟨*alpha code*⟩`}`

Expands to the parent for the given alpha code or empty if no mapping.



`\TrackLangScriptIfHasParent{`⟨*alpha code*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`

Expands to ⟨*true*⟩ if the given alpha code has a parent or to ⟨*false*⟩ otherwise. Note that if a parent is explicitly set to empty with `\TrackLangScriptSetParent` then it will be considered defined, but if the ⟨*parent*⟩ argument was empty in `\TrackLangScriptMap`, then it will be undefined.



`\TrackLangAddExtraScriptFile{`⟨*file*⟩`}`

This command adds ⟨*file*⟩ to the list of extra script files that should be input by `tracklang-scripts.tex`, unless `tracklang-scripts.tex` has already been input, in which case ⟨*file*⟩ will be input straight away.

5. Detecting the User's Requested Languages

The tracklang package tries to track the loaded languages and the option names used to identify those languages. For want of a better term, the language option names are referred to as dialects even if they're only a synonym for the language rather than an actual dialect. For example, if the user has requested `british`, the *root language* label is `english` and the dialect is `british`, whereas if the user requested `UKenglish`, the root language label is `english` and the dialect is `UKenglish`. The exceptions to this are the tracklang package options that have been specified in the form ⟨*iso lang*⟩-⟨*iso country*⟩ (listed in Table 1.2 on page 5). For example, the package option `en-GB` behaves as though the user requested the package option `british`.

If `\TrackLocale` or `\TrackLangFromEnv` are used and the locale isn't recognised a new dialect is created with the label formed from the ISO codes (and modifier, if present). Similarly for `\TrackLanguageTag` a new dialect is created with a label that's essentially the language tag without the hyphen separators. For example,

```
\TrackLocale{xx-YY}
```

will add a new dialect with the label `xxYY`,

```
\TrackLocale{xx-YY@mod}
```

will add a new dialect with the label `xxYYmod` and

```
\TrackLanguageTag{xx-Latn-YY}
```

will add a new dialect with the label `xxLatnYY`.

> If `\TrackLocale` or `\TrackLangFromEnv` find a modifier, the value will be sanitized to allow it to be used as a label. If the modifier is set explicitly using `\SetTrackedDialectModifier`, no sanitization is performed.

In addition to the root language label and the dialect identifier, many of the language options

also have corresponding ISO codes. In most cases there is an ISO 639-1 or an ISO 639-2 code (or both), and in some cases there is an ISO 3166-1 code identifying the dialect region. Where a language has different ISO 639-2 (T) and 639-2 (B) codes, the "T" version is assumed.

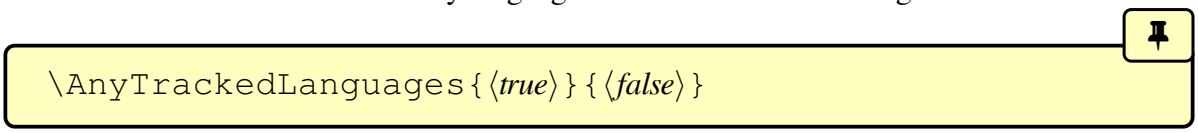
When the `tracklang.sty` LaTeX package is loaded, it first attempts to find the language options through the package options supplied to tracklang. This means that any languages that have been supplied in the document class options should get identified (provided that the document class has used the standard option declaration mechanism). If no languages have been supplied in this way, `tracklang.sty` then attempts to identify language settings in the following order:

1. if `\bbl@loaded` is defined (babel), tracklang will iterate over each label in that command definition;
2. if `\trans@languages` is defined (translator), tracklang will iterate over each label in that command definition;
3. if ngerman has been loaded, the `ngerman` dialect will be tracked;
4. if german has been loaded, the `german` root language will be tracked;
5. if polyglossia has been loaded:
   a) if `\xpg@bcp@loaded` has been defined, tracklang will iterate over the BCP 47 tags in that command definition;
   b) if `\xpg@loaded` has been defined, tracklang will iterate over each language label in that command definition;
   c) tracklang will iterate over all tracklang options and test if the root language has been loaded.

Note that this references internal commands provided by other packages. Of these, only the polyglossia commands are documented in the package manual, and so are the only ones that can be relied on.

Each identified language and dialect is added to the *tracked language* and *tracked dialect* lists. Note that the tracked language and tracked dialect are labels rather than proper nouns. If a dialect label is identical to its root language label, the label will appear in both lists.

You can check whether or not any languages have been detected using:



This will do ⟨*true*⟩ if one or more languages have been tracked otherwise it will do ⟨*false*⟩. (Each detected dialect will automatically have the root language label added to the tracked language list, if it's not already present.)

If you want to find out if any of the tracked dialects matches a particular language tag, you can use:

```
\GetTrackedDialectFromLanguageTag{⟨tag⟩}{⟨cs⟩}
```

If successful, the supplied control sequence ⟨*cs*⟩ is set to the dialect label, otherwise ⟨*cs*⟩ is set to empty. The test is for an exact match on the root language, script, sub-language, variant and region. The control sequence ⟨*cs*⟩ will be empty if none of the tracked dialects matches all five of those elements. (If the script isn't given explicitly, the default for that language is assumed.) In the event that ⟨*cs*⟩ is empty, you can now (as from v1.3.6) get the closest match with:

```
\TrackedDialectClosestSubMatch
```

(which is set by `\GetTrackedDialectFromLanguageTag`). This will be empty if no tracked dialects match on the root language or if there's a tracked dialect label that exactly matches the label formed by concatenating the language code, sub-language, script, region, modifier and variant.

For example (Plain TEX):

```
\input tracklang
\TrackLanguageTag{en-826}
Has en-Latn-GB been tracked?
\GetTrackedDialectFromLanguageTag{en-Latn-GB}
{\thisdialect}%
\ifx\thisdialect\empty
 No!
\else
 Yes! Dialect label: \thisdialect.
\fi
\bye
```

This matches because the territory code 826 is recognised as equivalent to the code `GB`, and the default script for `english` is `Latn`. In this case, the dialect label is `british`. Note that this doesn't require the use of `\TrackLanguageTag` to track the dialect. It also works if the dialect has been tracked using other commands, such as `\TrackLocale`.

Here's an example that doesn't have an exact match, but does have a partial match:

```
\input tracklang
\TrackLanguageTag{de-CH-1996}
Has de-DE-1996 been tracked?
\GetTrackedDialectFromLanguageTag{de-DE-1996}
{\thisdialect}%
```

```
\ifx\thisdialect\empty
 No!
  \ifx\TrackedDialectClosestSubMatch\empty
    No match on root language.
  \else
    Closest match: \TrackedDialectClosestSubMatch.
  \fi
\else
 Yes! Dialect label: \thisdialect.
\fi
\bye
```

In this case the result is:

Has de-DE-1996 been tracked? No! Closest match: nswissgerman.

You can iterate through each tracked dialect using:

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

At the start of each iteration, this sets the control sequence ⟨*cs*⟩ to the tracked dialect and does ⟨*body*⟩.

You can iterate through each tracked language using:

`\ForEachTrackedLanguage{`⟨*cs*⟩`}{`⟨*body*⟩`}`

At the start of each iteration, this sets the control sequence ⟨*cs*⟩ to the tracked language and does ⟨*body*⟩.

The above for-loops use the same internal mechanism as LaTeX's `\@for` loop. Since this isn't defined by TeX, a similar command (`\@tracklang@for`) will be defined that works in the same way.

The provided control sequence ⟨*cs*⟩ is updated at the start of each iteration to the current element. The loop is terminated when this control sequence is set to `\@nil`. This special control sequence should never been used as it's just a marker and isn't actually defined. If you get an error message stating that `\@nil` is undefined, then it's most likely due to a loop control sequence being used outside the loop. This can occur if the loop contains code that isn't expanded until later. For example, if the loop code includes `\AtBeginDocument`, you need to ensure that the loop control sequence is expanded before being added to the hook.

You can test if a root language has been detected using:

```
\IfTrackedLanguage{⟨language-label⟩}{⟨true⟩}{⟨false⟩}
```

where ⟨*language-label*⟩ is the language label. If true, this does ⟨*true*⟩ otherwise it does ⟨*false*⟩.

You can test if a particular dialect has been detected using:

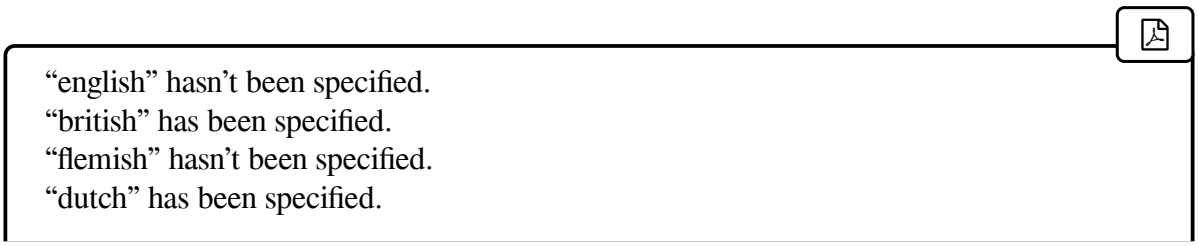
```
\IfTrackedDialect{⟨dialect-label⟩}{⟨true⟩}{⟨false⟩}
```

where ⟨*dialect-label*⟩ is the dialect label. If the root language was explicitly specified, then it will also be detected as a dialect.

For example:

```
\documentclass[british,dutch]{article}

\usepackage{tracklang}

\begin{document}
``english'' \IfTrackedDialect{english}{has}{hasn't}
been specified.

``british'' \IfTrackedDialect{british}{has}{hasn't}
been specified.

``flemish'' \IfTrackedDialect{flemish}{has}{hasn't}
been specified.

``dutch'' \IfTrackedDialect{dutch}{has}{hasn't}
been specified.

``english'' or an English variant
\IfTrackedLanguage{english}{has}{hasn't}
been specified.
\end{document}
```

This produces:



"english" or an English variant has been specified.

You can find the root language label for a given tracked dialect using:

```
\TrackedLanguageFromDialect{⟨dialect⟩}
```

If ⟨*dialect*⟩ hasn't been defined this does nothing otherwise it expands to the root language label.

You can find the tracked dialects from a given root language using:

```
\TrackedDialectsFromLanguage{⟨root language label⟩}
```

This will expand to a comma-separated list of dialect labels if the root language label has been defined, otherwise it does nothing.

You can test if a language or dialect has a corresponding ISO code using:

```
\IfTrackedLanguageHasIsoCode{⟨code
type⟩}{⟨label⟩}{⟨true⟩}{⟨false⟩}
```

where ⟨*code type*⟩ is the type of ISO code (for example, 639-1 for root languages or 3166-1 for regional dialects), and ⟨*label*⟩ is the language or dialect label. Note that the 639-3 may be set for the dialect rather than root language for sub-languages parsed using `\TrackLanguageTag`.

Alternatively, you can test if a particular ISO code has been defined using:

```
\IfTrackedIsoCode{⟨code type⟩}{⟨code⟩}{⟨true⟩}{⟨false⟩}
```

where ⟨*code type*⟩ is again the type of ISO code (for example, 639-1 or 3166-1), and ⟨*code*⟩ is the particular code (for example, `en` for ISO 639-1 or `GB` for ISO 3166-1).

You can fetch the language (or dialect) label associated with a given ISO code using:

```
\TrackedLanguageFromIsoCode{⟨code type⟩}{⟨code⟩}
```

This does nothing if the given ⟨*code*⟩ for the given ISO ⟨*code type*⟩ has not been defined, otherwise it expands a comma-separated list of language or dialect labels.

You can fetch the ISO code for a given code type using:

```
\TrackedIsoCodeFromLanguage{⟨code type⟩}{⟨label⟩}
```

where ⟨*label*⟩ is the language or dialect label and ⟨*code type*⟩ is the ISO code type (for example, 639-1 or 3166-1). Unlike `\TrackedLanguageFromIsoCode`, this command only expands to a single label rather than a comma-separated list.

The above commands do nothing in the event of an unknown code or code type, so if you accidentally get the wrong code type, you won't get an error. If you're unsure of the code type, you can use the following commands:

```
\TwoLetterIsoCountryCode
```

This expands to 3166-1 and is used for the two-letter country codes.

```
\TwoLetterIsoLanguageCode
```

This expands to 639-1 and is used for the two-letter root language codes.

```
\ThreeLetterIsoLanguageCode
```

This expands to 639-2 and is used for the three-letter root language codes.

```
\ThreeLetterExtIsoLanguageCode
```

This expands to 639-3. This code is only used for a root language if there's no 639-1 or 639-2 code. It may also be used for a dialect if a sub-language part has been set in the language tag parsed by `\TrackLanguageTag`.

The `\Get`... commands below are designed to be expandable. If the supplied ⟨*dialect*⟩ is unrecognised they expand to empty. Remember that the dialect must first be identified as a tracked language for it to be recognised.

As from v1.3, the language tag for a given dialect can be obtained using:

```
\GetTrackedLanguageTag{⟨dialect⟩}
```

where ⟨*dialect*⟩ is the label identifying the dialect. Uses the `und` (undetermined) code for unknown languages.

As from v1.3, each tracked dialect may also have an associated modifier, which can be fetched using:

```
\GetTrackedDialectModifier{⟨dialect⟩}
```

where ⟨*dialect*⟩ is the label identifying the dialect. This value is typically obtained by parsing a POSIX locale identifier with `\TrackLocale` or `\TrackLangFromEnv` but may be set explicitly. (See §6 for setting this value. Likewise for the following commands.)

You can test if a dialect has an associated modifier using:

```
\IfHasTrackedDialectModifier{⟨dialect⟩}{⟨true⟩}{⟨false⟩}
```

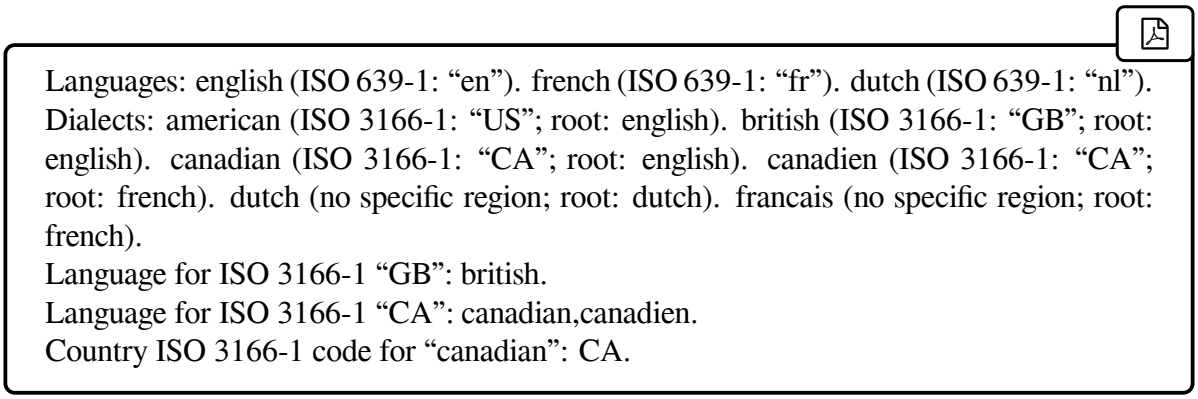
If the dialect has an associated modifier this does ⟨*true*⟩ otherwise it does ⟨*false*⟩.

For example:

```
\documentclass[british,francais,american,canadian,canadien,dutch
{article}

\usepackage{tracklang}

\begin{document}
Languages:
\ForEachTrackedLanguage{\ThisLanguage}
{\ThisLanguage\space
(ISO \TwoLetterIsoLanguageCode:
``\TrackedIsoCodeFromLanguage{\TwoLetterIsoLanguage-
Code}{\ThisLanguage}''). }

Dialects:
\ForEachTrackedDialect{\ThisDialect}
{\ThisDialect\space
(\IfTrackedLanguageHasIsoCode{\TwoLetterIsoCountry-
Code}{\ThisDialect}%
 {ISO \TwoLetterIsoCountryCode:
  ``\TrackedIsoCodeFromLanguage{\TwoLetterIsoCoun-
tryCode}{\ThisDialect}''} {no specific region};
root: \TrackedLanguageFromDialect{\ThisDialect}). }

Language for ISO \TwoLetterIsoCountryCode\ ``GB'':
\TrackedLanguageFromIsoCode{\TwoLetterIsoCountry-
Code}{GB}.

Language for ISO \TwoLetterIsoCountryCode\ ``CA'':
\TrackedLanguageFromIsoCode{\TwoLetterIsoCountry-
Code}{CA}.

Country ISO \TwoLetterIsoCountry-
Code\ code for ``canadian'':
\TrackedIsoCodeFromLanguage{\TwoLetterIsoCountry-
Code}{canadian}.
```

```
\end{document}
```

This produces:



As from v1.3, each tracked dialect may also have an associated variant, which can be fetched using:

```
\GetTrackedDialectVariant{⟨dialect⟩}
```

where ⟨*dialect*⟩ is the label identifying the dialect. This value is typically obtained by parsing a language tag with `\TrackLanguageTag` but may be set explicitly.

You can test if a dialect has an associated variant using:

```
\IfHasTrackedDialectVariant{⟨dialect⟩}{⟨true⟩}{⟨false⟩}
```

As from v1.3, each tracked dialect may also have an associated script, which can be fetched using:

```
\GetTrackedDialectScript{⟨dialect⟩}
```

where ⟨*dialect*⟩ is the label identifying the dialect.

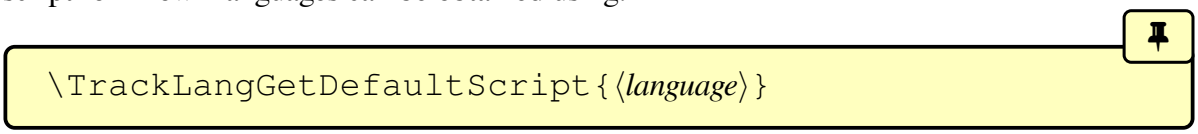
You can test if a dialect has an associated script using:

```
\IfHasTrackedDialectScript{⟨dialect⟩}{⟨true⟩}{⟨false⟩}
```

If the dialect has an associated script this does ⟨*true*⟩ otherwise it does ⟨*false*⟩. This information is provided for language packages that need to know what script is required, but there's no guarantee that the script will actually be set in the document. Similarly for all the other attributes described here.

Note that the script should be a recognised four-letter ISO 15924 code, such as `Latn` or `Cyrl`. If a dialect doesn't have an associated script then the default for the root language should

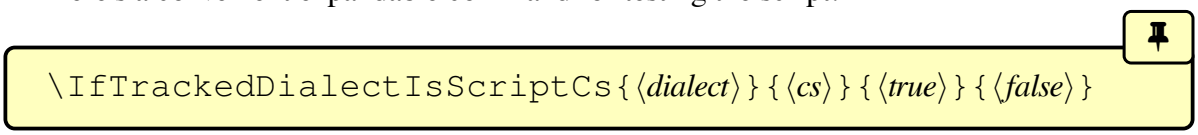
be assumed. For example, `Latn` for English dialects or `Cyrl` for Russian dialects. The default script for known languages can be obtained using:



```
\TrackLangGetDefaultScript{⟨language⟩}
```

Most root languages have a default script, but there are a few without one as it may depend on region, politics or ideology.

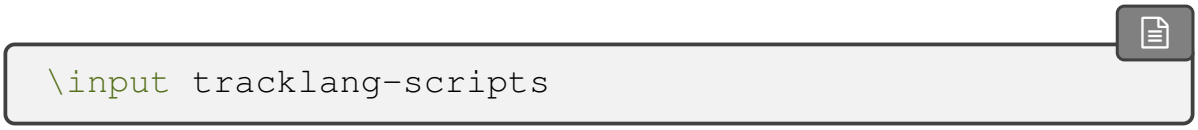
There's a convenient expandable command for testing the script:



```
\IfTrackedDialectIsScriptCs{⟨dialect⟩}{⟨cs⟩}{⟨true⟩}{⟨false⟩}
```

This tests if the given tracked dialect has an associated script and compares the value with the replacement text of ⟨*cs*⟩. If the dialect hasn't been explicitly assigned a script, then test is performed against the default script for the root language.

The supplementary package tracklang−scripts provides some additional commands relating to writing systems, including commands in the form `\TrackLangScript`⟨*Code*⟩ where ⟨*Code*⟩ is the ISO 15924 four-letter code. If the dialect doesn't have an associated script, ⟨*false*⟩ is done. This package isn't loaded automatically, so you'll need to explicitly load it. The generic code is in `tracklang-scripts.tex`:



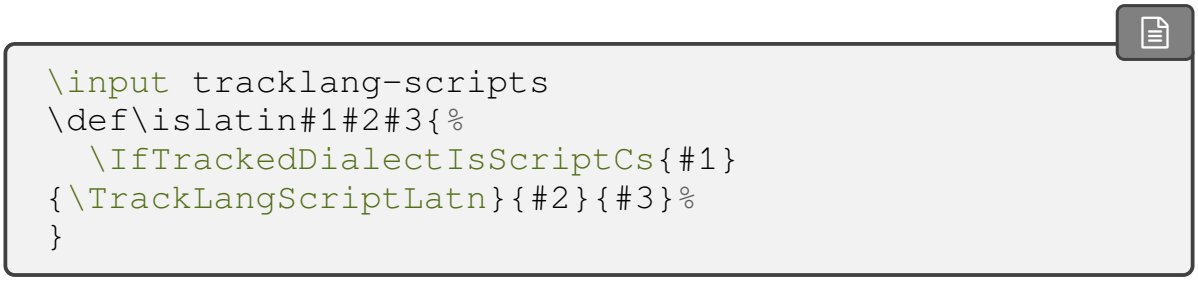
```
\input tracklang-scripts
```

There's a convenient LaTeX wrapper `tracklang-scripts.sty`:

```
\usepackage{tracklang-scripts}
```

See §4 for further details of that package.

For example, the following defines a command to check if the given dialect should use a Latin script:



```
\input tracklang-scripts
\def\islatin#1#2#3{%
  \IfTrackedDialectIsScriptCs{#1}
{\TrackLangScriptLatn}{#2}{#3}%
}
```

Note that the script value doesn't mean that the document is actually using that script. It means that this is the user's *desired* script, but whether that script is actually set relies on the appropriate settings in the relevant language package (such as polyglossia's `script` key).

As from v1.3, each tracked dialect may also have a sub-language identifier (for example, `arevela`), which can be fetched using:

```
\GetTrackedDialectSubLang{⟨dialect⟩}
```

where ⟨*dialect*⟩ is the label identifying the dialect.

You can test if a dialect has an associated sub-tag using:

```
\IfHasTrackedDialectSubLang{⟨dialect⟩}{⟨true⟩}{⟨false⟩}
```

If the dialect has an associated sub-tag this does ⟨*true*⟩ otherwise it does ⟨*false*⟩.

As from v1.3, each tracked dialect may also have additional information, which can be fetched using:

```
\GetTrackedDialectAdditional{⟨dialect⟩}
```

where ⟨*dialect*⟩ is the label identifying the dialect.

You can test if a dialect has additional information using:

```
\IfHasTrackedDialectAdditional{⟨dialect⟩}{⟨true⟩}{⟨false⟩}
```

If the dialect has additional information this does ⟨*true*⟩ otherwise it does ⟨*false*⟩.

Most packages that implement multilingual support have a set of language definition files for each supported language or dialect. It may be that only the root language is needed, if there are no variations between that language's dialect (for the purposes of that package), or it may be that separate definition files are required for each dialect. However it can be awkward trying to map the requested dialect or language label to the file name. Should, say, the file containing the French code be called ⟨*prefix*⟩`-french-`⟨*suffix*⟩ or ⟨*prefix*⟩`-frenchb-`⟨*suffix*⟩ or ⟨*prefix*⟩`-francais-`⟨*suffix*⟩? Should, say, the file containing the British English code be called ⟨*prefix*⟩`-british-`⟨*suffix*⟩ or ⟨*prefix*⟩`-UKenglish-`⟨*suffix*⟩? If you want to modularise the language support for your package so that each language module has a different maintainer will the maintainers know what tag to use for their language?

To help with this, tracklang provides:

```
\IfTrackedLanguageFileExists{⟨dialect⟩}{⟨prefix⟩}{⟨suffix⟩}{⟨true code⟩}{⟨false code⟩}
```

This attempts to find the file called ⟨*prefix*⟩⟨*localeid*⟩⟨*suffix*⟩ where ⟨*localeid*⟩ is determined from ⟨*dialect*⟩ (see below). If the file is found then

```
\CurrentTrackedTag
```

is set to ⟨*localeid*⟩ and ⟨*true code*⟩ is done, otherwise ⟨*false code*⟩ is done. If this command is empty, then the dialect hasn't been detected. If the dialect has been detected, but no file can be found, then `\CurrentTrackedTag` is set to the final attempt at determining ⟨*localeid*⟩.

There's a convenient shortcut command new to version 1.3:

```
\TrackLangRequireDialect[⟨load code⟩]{⟨pkgname⟩}{⟨dialect⟩}
```

which uses `\IfTrackedLanguageFileExists` to input the resource file if found. The prefix is given by ⟨*pkgname*⟩- and the suffix is `.ldf`. A warning is issued if no resource file is found. Note that while it makes sense for ⟨*pkgname*⟩ to be the same as the base name of the package that uses these resource files, they don't have to be the same. This command additionally defines:

```
\TrackLangRequireDialectPrefix
```

to ⟨*pkgname*⟩, which allows the prefix to be picked up by resource file commands, such as `\TrackLangProvidesResource` and `\TrackLangRequireResource`. (See below.)

The optional argument ⟨*load code*⟩ is the code that actually inputs the required file. This defaults to

```
\TrackLangRequireResource{\CurrentTrackedTag}
```

The `\IfTrackedLanguageFileExists` command sets up the current tracked dialect with:

```
\SetCurrentTrackedDialect{⟨dialect⟩}
```

which enables the following commands that may be used within ⟨*true code*⟩ or ⟨*false code*⟩:

```
\CurrentTrackedDialect
```

Expands to the dialect label.

```
\CurrentTrackedLanguage
```

If the dialect hasn't been detected, this command will be empty, otherwise it will expand to the root language label (which may be the same as the dialect label).

```
\CurrentTrackedRegion
```

If the dialect hasn't been detected, this command will be empty. If the dialect has been assigned an ISO 3166-1 code, `\CurrentTrackedRegion` will expand to that code, otherwise it will be empty.

```
\CurrentTrackedIsoCode
```

If the dialect hasn't been detected, this command will be empty. Otherwise it may be empty or it may expand to the ISO 639-1 or ISO 639-2 or ISO 639-3 code.

```
\CurrentTrackedDialectModifier
```

The dialect's modifier or empty if not set. (This is set but not used in the set of possible ⟨*localeid*⟩ values.)

```
\CurrentTrackedDialectVariant
```

The dialect's variant or empty if not set.

```
\CurrentTrackedDialectSubLang
```

The dialect's sub-language code or empty if not set.

```
\CurrentTrackedDialectAdditional
```

The dialect's additional information or empty if not set.

```
\CurrentTrackedLanguageTag
```

The dialect's language tag. Take care not to confuse this with `\CurrentTrackedTag`.

```
\CurrentTrackedDialectScript
```

The dialect's script. If the dialect doesn't have the script set, the default script for the language is used instead.

`\IfTrackedLanguageFileExists` behaves as follows:

- If no dialect with the given label has been detected, the condition evaluates to *false* and `\CurrentTrackedTag` is empty.
- If a dialect with the given label has been detected, then:
  - For each possible ⟨*localeid*⟩ in an ordered set of tags determined by the dialect label (see below), the first file matching ⟨*prefix*⟩⟨*localeid*⟩⟨*suffix*⟩ that's found on TEX's path results in the condition evaluating to *true* and `\CurrentTrackedTag` is set to the current ⟨*localeid*⟩ in the set. The rest of the set of possible values of ⟨*localeid*⟩ is skipped.
  - If no file matching ⟨*prefix*⟩⟨*localeid*⟩⟨*suffix*⟩ is found on TEX's path, then the condition evaluates to *false* and `\CurrentTrackedTag` is set to the final ⟨*localeid*⟩ in the set (the language label).

The ordered set of possible values of ⟨*localeid*⟩ is determined from the given dialect.

The ordering has changed in version 1.4, which now also includes the script and variant. This new ordering should typically make the more common combinations closer to the start of the search.

The possible values of ⟨*localeid*⟩ are listed below in the order of priority used by `\IfTrackedLanguageFileExists`. Note that the set may contain repetitions (for example, if the dialect label is the same as the root language label). If an item contains an element that hasn't been set (such as the ISO 639-3 code or a sub-language ⟨*sublang*⟩ or variant) then that item is skipped.

1. ⟨*localeid*⟩ is just the value of `\CurrentTrackedLanguageTag`.
2. ⟨*localeid*⟩ is just the dialect label.
3. ⟨*localeid*⟩ is ⟨*ISO 639-1*⟩−⟨*sublang*⟩−⟨*script*⟩−⟨*region*⟩.
4. ⟨*localeid*⟩ is ⟨*ISO 639-1*⟩−⟨*script*⟩−⟨*region*⟩.
5. ⟨*localeid*⟩ is ⟨*ISO 639-1*⟩−⟨*sublang*⟩−⟨*region*⟩ (if there's no script or if the script is the default for the given language). ⟨*ISO 639-1*⟩−⟨*region*⟩ (if there's no script or if the script is the default for the given language).
6. ⟨*localeid*⟩ is ⟨*ISO 639-1*⟩−⟨*sublang*⟩−⟨*script*⟩.
7. ⟨*localeid*⟩ is ⟨*ISO 639-1*⟩−⟨*script*⟩.
8. ⟨*localeid*⟩ is ⟨*ISO 639-1*⟩−⟨*sublang*⟩.

9. ⟨*localeid*⟩ is just ⟨*ISO 639-1*⟩.
10. ⟨*localeid*⟩ is ⟨*ISO 639-2*⟩−⟨*sublang*⟩−⟨*script*⟩−⟨*region*⟩.
11. ⟨*localeid*⟩ is ⟨*ISO 639-2*⟩−⟨*script*⟩−⟨*region*⟩.
12. ⟨*localeid*⟩ is ⟨*ISO 639-2*⟩−⟨*sublang*⟩−`region` (if there's no script or if the script is the default for the given language). ⟨*ISO 639-2*⟩−⟨*region*⟩ (if there's no script or if the script is the default for the given language).
13. ⟨*localeid*⟩ is ⟨*ISO 639-2*⟩−⟨*sublang*⟩−⟨*script*⟩.
14. ⟨*localeid*⟩ is ⟨*ISO 639-2*⟩−⟨*script*⟩.
15. ⟨*localeid*⟩ is ⟨*ISO 639-2*⟩−⟨*sublang*⟩.
16. ⟨*localeid*⟩ is just ⟨*ISO 639-2*⟩.
17. ⟨*localeid*⟩ is ⟨*ISO 639-3*⟩−⟨*sublang*⟩−⟨*script*⟩−⟨*region*⟩.
18. ⟨*localeid*⟩ is ⟨*ISO 639-3*⟩−⟨*script*⟩−⟨*region*⟩.
19. ⟨*localeid*⟩ is ⟨*ISO 639-3*⟩−⟨*sublang*⟩−⟨*region*⟩ (if there's no script or if the script is the default for the given language). ⟨*ISO 639-3*⟩−⟨*region*⟩ (if there's no script or if the script is the default for the given language).
20. ⟨*localeid*⟩ is ⟨*ISO 639-3*⟩−⟨*sublang*⟩−⟨*script*⟩.
21. ⟨*localeid*⟩ is ⟨*ISO 639-3*⟩−⟨*script*⟩.
22. ⟨*localeid*⟩ is ⟨*ISO 639-3*⟩−⟨*sublang*⟩.
23. ⟨*localeid*⟩ is just ⟨*ISO 639-3*⟩.
24. ⟨*localeid*⟩ is just ⟨*region*⟩.
25. ⟨*localeid*⟩ is ⟨*ISO 639-1*⟩−⟨*sublang*⟩−⟨*variant*⟩ or ⟨*ISO 639-1*⟩−⟨*variant*⟩ if ⟨*sublang*⟩ is missing.
26. ⟨*localeid*⟩ is ⟨*ISO 639-2*⟩−⟨*sublang*⟩−⟨*variant*⟩ or ⟨*ISO 639-2*⟩−⟨*variant*⟩ if ⟨*sublang*⟩ is missing.
27. ⟨*localeid*⟩ is ⟨*ISO 639-3*⟩−⟨*sublang*⟩−⟨*variant*⟩ or ⟨*ISO 639-3*⟩−⟨*variant*⟩ if ⟨*sublang*⟩ is missing.
28. ⟨*localeid*⟩ is just the value of `\CurrentTrackedLanguage` (the root language label).

For example (pre v1.3):

```
\AnyTrackedLanguages
{%
  \ForEachTrackedDialect{\ThisDialect}%
  {% try to load the language file for this dialect
    \IfTrackedLanguageFileExists{\ThisDialect}%
    {mypackage-}% file prefix
    {.ldf}% file suffix
    {\input mypackage-\CurrentTrackedTag.ldf}
% file found
    {% file not found
      \PackageWarning{mypackage}
{No support for language
       `\ThisDialect'}%
    }%
  }%
}
{% no languages detected so use defaults
}
```

With version 1.3 onwards, this can be written more concisely as:

```
\AnyTrackedLanguages
{%
  \ForEachTrackedDialect{\ThisDialect}%
  {% try to load the language file for this dialect
    \TrackLangRequireDialect{mypackage}
{\ThisDialect}%
  }%
}
{% no languages detected so use defaults
}
```

which additionally enables the tracklang version 1.3 commands described below, such as `\TrackLangRequireResource`.

If, for example, `\ThisDialect` is `british`, then the file search will be in the order:

1. `mypackage-en-GB.ldf` (language tag)
2. `mypackage-british.ldf` (dialect label)
3. `mypackage-en-Latn-GB.ldf` (639-1 language code, script, region)
4. `mypackage-en-GB.ldf` (639-1 language code, region)

5. `mypackage-en-Latn.ldf` (639-1 language code, script)
6. `mypackage-en.ldf` (639-1 language code)
7. `mypackage-eng-Latn-GB.ldf` (639-2 language code, script, region)
8. `mypackage-eng-GB.ldf` (639-2 language code, region)
9. `mypackage-eng-Latn.ldf` (639-2 language code, script)
10. `mypackage-eng.ldf` (639-2 language code)
11. `mypackage-GB.ldf` (region)
12. `mypackage-english.ldf` (language label)

If, for example, `\ThisDialect` is `naustrian`, then the file search will be in the order:

1. `mypackage-de-AT-1996.ldf` (language tag)
2. `mypackage-naustrian.ldf` (dialect label)
3. `mypackage-de-Latn-AT.ldf` (639-1 language code, script, region)
4. `mypackage-de-AT.ldf` (639-1 language code, region)
5. `mypackage-de-Latn.ldf` (639-1 language code, script)
6. `mypackage-de.ldf` (639-1 language code)
7. `mypackage-deu-Latn-AT.ldf` (639-2 language code, script, region)
8. `mypackage-deu-AT.ldf` (639-2 language code, region)
9. `mypackage-deu-Latn.ldf` (639-2 language code, script)
10. `mypackage-deu.ldf` (639-2 language code)
11. `mypackage-AT.ldf` (region)
12. `mypackage-de-1996.ldf` (639-1 language code, variant)
13. `mypackage-deu-1996.ldf` (639-2 language code, variant)
14. `mypackage-german.ldf` (language label)

If, for example, `\ThisDialect` is `francais`, then the file search will be in the order:

1. `mypackage-fr.ldf` (language tag)
2. `mypackage-francais.ldf` (dialect label)

3. `mypackage-fr-Latn.ldf` (639-1 language code, script)
4. `mypackage-fr.ldf` (639-1 language code)
5. `mypackage-fra-Latn.ldf` (639-2 language code, script)
6. `mypackage-fra.ldf` (639-2 language code)
7. `mypackage-french.ldf` (language)

This is because the predefined `francais` option has no region assigned to it. Be careful if the dialect label is the actual root language. For example, if `\ThisDialect` is `french`, then the file search will be in the order:

1. `mypackage-fr.ldf` (language tag)
2. `mypackage-french.ldf` (dialect label)
3. `mypackage-fr-Latn.ldf` (639-1 language code, script)
4. `mypackage-fr.ldf` (639-1 language code)
5. `mypackage-fra-Latn.ldf` (639-2 language code, script)
6. `mypackage-fra.ldf` (639-2 language code)
7. `mypackage-french.ldf` (language)

Note that the last try will always fail in this case since if the file exists, it will be found on the second try.

If the dialect label is identical to the root language label then it means that all associated information is the default for that language. For example, in the above case of `french`, the script is `Latn` and the region is unspecified. The root language label can therefore be used as the fallback in the event of no other match but for the specific case where the dialect is identical to the root language then all unnecessary file name checks can be skipped.

If you're only providing support for the root languages (pre v1.3):

```
\AnyTrackedLanguages
{%
  \ForEachTrackedLanguage{\ThisLanguage}%

{% try to load the language file for this root language
    \IfTrackedLanguageFileExists{\ThisLanguage}%
    {mypackage-}% file prefix
    {.ldf}% file suffix
    {\input mypackage-\CurrentTrackedTag.ldf}
% file found
```

```
    {% file not found
      \PackageWarning{mypackage}
{No support for language
       `\ThisLanguage'}%
    }%
  }%
}
{% no languages detected so use defaults
}
```

With version 1.3 onwards, this can be written more concisely as:

```
\AnyTrackedLanguages
{%
  \ForEachTrackedLanguage{\ThisLanguage}%

{% try to load the language file for this root language
    \TrackLangRequireDialect{mypackage}
{\ThisLanguage}%
  }%
}
{% no languages detected so use defaults
}
```

which additionally enables the commands described below. Note that in this case, if more than one dialect for the same language has been tracked, only the hooks for the last dialect for that language will be adjusted, so it's usually best to iterate over the dialects.

The following `\TrackLang...Resource...` commands may only be used in resource files that are loaded using `\TrackLangRequireDialect`. An error will occur if the file is input through some other method.

Within the resource file ⟨*pkgname*⟩`-`⟨*localeid*⟩`.ldf`, you can identify the file using (new to version 1.3):

```
\TrackLangProvidesResource{⟨tag⟩}[⟨version info⟩]
```

where ⟨*tag*⟩ is the locale identifier.

If `\ProvidesFile` is defined (through the LaTeX kernel) this is used, otherwise a simplified generic alternative is used that's suitable for other TeX formats.

The resource file can load another resource file ⟨*pkgname*⟩`-`⟨*tag*⟩`.ldf`, using (new to version 1.3):

```
\TrackLangRequireResource{⟨tag⟩}
```

For example, the dialect file `foo-en-GB.ldf` might need to load the root language resource file `foo-english.ldf`:

```
% (In file foo-en-GB.ldf)
% Declare this regional file:
\TrackLangProvidesResource{en-GB}
% load root language file foo-english.ldf:
\TrackLangRequireResource{english}
```

If `foo-english.ldf` is also identified with `\TrackLangProvidesResource`, this will ensure that it's only loaded once.

It may be that you want to load a file depending on the input encoding. The inputenc package defines `\inputencodingname`, but this is only used with pdfLaTeX. To avoid repeated tests to determine whether or not `\inputencodingname` has been defined, you can use:

```
\TrackLangEncodingName
```

This will expand to `utf8` if `\inputencodingname` hasn't been defined, otherwise it will expand to `\inputencodingname`. For example:

```
\InputIfFileExists{foo-\TrackLangEncodingName.ldf}
{% support available for the document encoding
}
{% no support for the document encoding
}
```

If you require the resource file and want to perform ⟨*code1*⟩ if it's loaded at this point or ⟨*code2*⟩ if it's already been loaded then you can use:

```
\TrackLangRequireResourceOrDo{⟨tag⟩}{⟨code1⟩}{⟨code2⟩}
```

If you want to load a resource file if it exists (without an error if it doesn't exist), then you can use

```
\TrackLangRequestResource{⟨tag⟩}{⟨not found code⟩}
```

If the file doesn't exist, ⟨*not found code*⟩ is done.

Note that these \...Resource... commands are only permitted within the resource files. They are internally enabled through \TrackLangRequireDialect.

The above restriction on the resource files loaded through \TrackLangRequireDialect, and the fact that it internally uses \IfTrackedLanguageFileExists, means that commands like \CurrentTrackedLanguage or \CurrentTrackedDialect may be used in those files. This means that the name of the captions hook can be obtained through them. (Remember that the file foo-en-GB.ldf might have been loaded with, say, the british dialect or with the synonymous UKenglish dialect or with a dialect label that doesn't have a corresponding caption hook, such as enGBLatn.)

The polyglossia package has language caption hooks in the form \captions⟨*language*⟩ (where ⟨*language*⟩ is the root language label) whereas babel has dialect captions hooks in the form \captions⟨*dialect*⟩ (where ⟨*dialect*⟩ is the dialect label). This leads to a rather cumbersome set of conditionals:

```
\ifcsundef{captions\CurrentTrackedLanguage}
{%
  \ifcsundef{captions\CurrentTrackedDialect}%
  {}%
  {%
    \csgappto{captions\CurrentTrackedDialect}{%
      % code to append to hook
    }%
  }%
}%
{%
  \csgappto{captions\CurrentTrackedLanguage}{%
    % code to append to hook
  }%
}
% do code now to initialise
```

Note that the above has been simplified through the use of etoolbox commands, which isn't suitable for generic use. It also doesn't query the mapping from tracklang's dialect label to the closest matching babel dialect label.

Instead, tracklang provides a command to perform this set of conditionals using generic code:

\TrackLangAddToHook{⟨*code*⟩}{⟨*type*⟩}

where ⟨*code*⟩ is the code to append to the ⟨*type*⟩ hook. This always performs ⟨*code*⟩ after testing

for the hook in case the hook is undefined or has already been called (for example, ngerman uses `\captionsngerman` when the package is loaded, not at the start of the document).

Note that this command is enabled through `\TrackLangRequireDialect` so should only be used inside resource files.

Since `captions` is a commonly used hook type, there's a shortcut command provided:

```
\TrackLangAddToCaptions{⟨code⟩}
```

This is equivalent to

```
\TrackLangAddToHook{⟨code⟩}{captions}
```

There may be some hooks, such as `\date`⟨*dialect*⟩, that need redefining rather than appending to, so there's an analogous command:

```
\TrackLangRedefHook{⟨code⟩}{⟨type⟩}
```

which will redefined the hook to do ⟨*code*⟩.

Note that no expansion is performed on ⟨*code*⟩ when appending or redefining a hook.

5.1. Examples

The examples in this section illustrate the above commands.

5.1.1. `animals.sty`

This example is for a trivial package called `animals.sty` that defines three textual commands: `\catname`, `\dogname` and `\ladybirdname`. The default values are: "cat", "dog" and "bishy-barney-bee".[1] The supported languages are defined in files `animals-`⟨*localeid*⟩`.ldf`.

Here's the code for `animals.sty`:

```
% Example package animals.sty
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{animals}

\RequirePackage{tracklang}[2019/11/30]% v1.4

% Any undeclared options are language settings:
```

[1] Thass Broad Norfolk, my bewties :-P

```
\DeclareOption*{%
 \TrackIfKnownLanguage{\CurrentOption}%
 {% successful
  \PackageInfo{animals}
{Tracking language `\CurrentOption'}%
 }%
 {% failed
   \PackageError{animals}%
   {Unknown language specification `\CurrentOption'}
%
   {You need to supply either a known dialect label
    or a valid language tag}%
 }%
}

\ProcessOptions

% Default definitions
\newcommand\catname{cat}
\newcommand\dogname{dog}
\newcommand\ladybirdname{bishy-barney-bee}

\AnyTrackedLanguages
{%
  \ForEachTrackedDialect{\this@dialect}{%
    \TrackLangRequireDialect{animals}{\this@dialect}
%
  }%
}
{% no tracked languages, default already set up
}

\endinput
```

Here's a Plain TEX version that picks up the language from the locale environment variable:

```
\input tracklang

\TrackLangFromEnv

% Default definitions
```

```
\def\catname{cat}
\def\dogname{dog}
\def\ladybirdname{bishy-barney-bee}

\AnyTrackedLanguages
{%
  \ForEachTrackedDialect{\thisdialect}{%
    \TrackLangRequireDialect{animals}{\thisdialect}
%
  }%
}
{% no tracked languages, default already set up
}
```

In the event that a user or supplementary package for some reason wants to load a resource file for a language that hasn't been tracked, it might be worth providing a command for this purpose:

```
\newcommand*{\RequireAnimalsDialect}[1]{%
  \TrackLangRequireDialect{animals}{#1}%
}
```

The loop can then be changed to:

```
\ForEachTrackedDialect{\this@dialect}{%
  \RequireAnimalsDialect\this@dialect
}%
```

The `animals-english.ldf` file valid for both the Plain TEX and LATEX formats contains:

```
\TrackLangProvidesResource{english}

\def\englishanimals{%
  \def\catname{cat}%
  \def\dogname{dog}%
  \def\ladybirdname{bishy-barney-bee}%
}

\TrackLangAddToCaptions{\englishanimals}
```

The `animals-en-GB.ldf` file contains:

```
\TrackLangProvidesResource{en-GB}
\TrackLangRequireResource{english}

\def\enGBanimals{%
  \englishanimals
  \def\ladybirdname{ladybird}%
}
\TrackLangAddToCaptions{\enGBanimals}
```

The `animals-en-US.ldf` file contains:

```
\TrackLangProvidesResource{en-US}
\TrackLangRequireResource{english}

\def\enUSanimals{%
  \englishanimals
  \def\ladybirdname{ladybug}%
}
\TrackLangAddToCaptions{\enUSanimals}
```

Here's a German version in the file `animals-german.ldf`:

```
\TrackLangProvidesResource{german}

\def\germananimals{%
  \def\catname{Katze}%
  \def\dogname{Hund}%
  \def\ladybirdname{Marienk\"afer}%
}

\TrackLangAddToCaptions{\germananimals}
```

This means that if babel or polyglossia are loaded, the redefinitions are automatically performed whenever the language is changed, but if there's no caption mechanism the user can switch the fixed names using the `\...animals` commands.

Here's an example LaTeX document that doesn't have any caption hooks:

```
\documentclass[english,german]{article}

\usepackage{animals}

\begin{document}
\englishanimals

\catname.
\dogname.
\ladybirdname.

\germananimals

\catname.
\dogname.
\ladybirdname.
\end{document}
```

Here's a babel example document:

```
\documentclass[american,german,british]{article}

\usepackage{babel}
\usepackage{animals}

\begin{document}
\selectlanguage{american}

\catname.
\dogname.
\ladybirdname.

\selectlanguage{german}

\catname.
\dogname.
\ladybirdname.

\selectlanguage{british}
```

```
\catname.
\dogname.
\ladybirdname.
\end{document}
```

There is some redundancy with the above resource files. Consider the babel example above. The `american` dialect is the first option, so in that case `animals-en-US.ldf` is loaded followed by `animals-english.ldf`. This means that the `\captionsamerican` hook now includes

```
\englishanimals
\enUSanimals
```

Since `\enUSanimals` includes `\englishanimals`, there is redundant code. However, when the `british` dialect is processed, this loads the file `animals-en-GB.ldf` but not the file `animals-english.ldf` (since it's already been loaded). This means that `\captionsbritish` contains `\enGBanimals` but not `\englishanimals`.

If this redundancy is an issue (for example, there are so many redefinitions needed that it significantly slows the document build process), then it can be addressed with the following modifications. The `animals-en-GB.ldf` file is now:

```
\TrackLangProvidesResource{en-GB}

\def\enGBanimals{%
  \englishanimals
  \def\ladybirdname{ladybird}%
}

\TrackLangRequireResourceOrDo{english}%
{
  \TrackLangAddToCaptions{%
    \def\ladybirdname{ladybird}%
  }%
}
{
  \TrackLangAddToCaptions{\enGBanimals}
}
```

The `animals-en-US.ldf` file is now:

```
\TrackLangProvidesResource{en-US}

\providecommand*{\enUSanimals}{%
  \englishanimals
  \renewcommand*{\ladybirdname}{ladybug}%
}

\TrackLangRequireResourceOrDo{english}
{
  \TrackLangAddToCaptions{%
    \renewcommand*{\ladybirdname}{ladybird}%
  }%
}
{
  \TrackLangAddToCaptions{\enUSanimals}
}
```

This means that the document that has the dialects listed in the order `american`, `british` now has

```
\englishanimals
\def\ladybirdname{ladybird}
```

in the `\captionsbritish` hook and just `\enUSanimals` in the `\captionsamerican` hook, which has removed most of the redundancy.

Note that polyglossia has a `\captionsenglish` hook but not `\captionsamerican` or `\captionsbritish`, so this code doesn't allow for switching between variants of the same language with polyglossia.

5.1.2. `regions.sty`

Earlier on page 49, I mentioned the search order for `\IfTrackedLanguageFileExists` where if, for example, the dialect is `british`, the file search (v1.4+) will be:

1. `mypackage-en-GB.ldf` (language tag)
2. `mypackage-british.ldf` (dialect label)
3. `mypackage-en-Latn-GB.ldf` (639-1 language code, script, region)
4. `mypackage-en-GB.ldf` (639-1 language code, region)
5. `mypackage-en-Latn.ldf` (639-1 language code, script)

6. `mypackage-en.ldf` (639-1 language code)
7. `mypackage-eng-Latn-GB.ldf` (639-2 language code, script, region)
8. `mypackage-eng-GB.ldf` (639-2 language code, region)
9. `mypackage-eng-Latn.ldf` (639-2 language code, script)
10. `mypackage-eng.ldf` (639-2 language code)
11. `mypackage-GB.ldf` (region)
12. `mypackage-english.ldf` (language label)

You may have wondered why `mypackage-GB.ldf` is included in the search given that some countries have multiple official languages, which means that the country code on its own may not indicate the language.

The reason for including just the country code as the ⟨*localeid*⟩ in the file search is to allow for region rather than language dependent settings. For example, suppose I want to write a package that needs to know whether to use imperial or metric measurements in the document, but I also want to provide multilingual support. The language alone won't tell me whether to use imperial or metric (for example, the US uses imperial and the UK uses metric for most product attributes). I could provide `ldf` files for every language and region combination, but this would result in a lot redundancy.

`\TrackLangRequireDialect` has an optional argument for adjusting the way the resource files are loaded. Suppose I have `regions-`⟨*localeid*⟩`.ldf` resource files, then

```
\TrackLangRequireDialect{regions}{\this@dialect}
```

loads the resource file for the dialect given by `\this@dialect` using:

```
\TrackLangRequireResource{\CurrentTrackedTag}
```

I can use the optional argument to also load the resource file for the root language as well:

```
% custom file loader for regions.sty
\newcommand*{\RequireRegionsDialect}[1]{%
 \TrackLangRequireDialect
   [\TrackLangRequireResource{\CurrentTrackedTag}%
    \TrackLangRequireResource{\CurrentTracked-
Language}%
   ]%
   {regions}{#1}%
}
```

Now the dialect `british` can load both `regions-GB.ldf` and `regions-english.ldf`. The example package (`regions.sty`) below illustrates this.

```
% Example package regions.sty
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{regions}

\RequirePackage{tracklang}[2016/10/07]% v1.3+

\DeclareOption*{\TrackLanguageTag{\CurrentOption}}
\ProcessOptions

\newcommand*{\weightunit}{kg}
\newcommand*{\lengthunit}{mm}
\newcommand*{\currencyunit}{EUR}

\newcommand*{\unitname}{units}

\newcommand*{\RequireRegionsDialect}[1]{%
 \TrackLangRequireDialect
  [\TrackLangRequireResource{\CurrentTrackedTag}%
   \TrackLangRequireResource{\CurrentTracked-
Language}%
  ]%
  {regions}{#1}%
}

\AnyTrackedLanguages
{%
 \ForEachTrackedDialect{\this@dialect}{%
   \RequireRegionsDialect\this@dialect
 }%
}
{% no tracked languages, default already set up
}

\endinput
```

There are separate `ldf` files for region and language. First are the regions.

- `regions-BE.ldf` (Belgium):

```
\TrackLangProvidesResource{BE}

\providecommand*{\BEunits}{%
  \renewcommand*{\weightunit}{kg}%
  \renewcommand*{\lengthunit}{mm}%
  \renewcommand*{\currencyunit}{EUR}%
}

\TrackLangAddToCaptions{\BEunits}
```

- `regions-CA.ldf` (Canada):

```
\TrackLangProvidesResource{CA}

\providecommand*{\CAunits}{%
  \renewcommand*{\weightunit}{kg}%
  \renewcommand*{\lengthunit}{mm}%
  \renewcommand*{\currencyunit}{CAD}%
}

\TrackLangAddToCaptions{\CAunits}
```

- `regions-GB.ldf` (Great Britain):

```
\TrackLangProvidesResource{GB}

\providecommand*{\GBunits}{%
  \renewcommand*{\weightunit}{kg}%
  \renewcommand*{\lengthunit}{mm}%
  \renewcommand*{\currencyunit}{GBP}%
}

\TrackLangAddToCaptions{\GBunits}
```

- `regions-US.ldf` (USA):

```
\TrackLangProvidesResource{US}

\providecommand*{\USunits}{%
  \renewcommand*{\weightunit}{lb}%
  \renewcommand*{\lengthunit}{in}%
  \renewcommand*{\currencyunit}{USD}%
}

\TrackLangAddToCaptions{\USunits}
```

Now the language files:

- `regions-dutch.ldf`:

```
\TrackLangProvidesResource{dutch}

\providecommand*{\dutchnames}{%
  \renewcommand*{\unitname}{meeteenheden}%
}

\TrackLangAddToCaptions{\dutchnames}
```

- `regions-english.ldf`:

```
\TrackLangProvidesResource{english}

\providecommand*{\englishnames}{%
  \renewcommand*{\unitname}{units}%
}

\TrackLangAddToCaptions{\englishnames}
```

- `regions-french.ldf`:

```
\TrackLangProvidesResource{french}
```

```
\providecommand*{\frenchnames}{%
  \renewcommand*{\unitname}{unit\'es}%
}

\TrackLangAddToCaptions{\frenchnames}
```

- `regions-german.ldf`:

```
\TrackLangProvidesResource{french}

\providecommand*{\germannames}{%
  \renewcommand*{\unitname}{Ma\ss einheiten}%
}

\TrackLangAddToCaptions{\germannames}
```

Here's an example document that uses this package:

```
\documentclass[canadien]{article}

\usepackage{regions}

\begin{document}

\unitname: \weightunit, \lengthunit, \currencyunit.

\end{document}
```

This works because the ⟨*localeid*⟩ search looks for the country code before the root language label. However, this will fail if the dialect label is the same as a root language label that has an associated territory, marked with † in Table 1.2 on page 5, as then it will be picked up before the country code.

In the above example, `regions-CA.ldf` is matched rather than `regions-french.ldf`, so `regions-CA.ldf` is loaded by

```
\TrackLangRequireResource{\CurrentTrackedTag}
```

After this, the language file `regions-french.ldf` is then loaded:

```
\TrackLangRequireResource{\CurrentTrackedLanguage}
```

This assumes that there's a country code `ldf` file available. This example needs a little modification to use default units in case the region is missing:

```
% Modified example package regions.sty
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{regions}

% Pass all options to tracklang.sty:
\DeclareOption*{\PassOptionsToPackage
{\CurrentOption}{tracklang}}
\ProcessOptions

\RequirePackage{tracklang}

\newcommand*{\weightunit}{kg}
\newcommand*{\lengthunit}{mm}
\newcommand*{\currencyunit}{EUR}

\newcommand*{\unitname}{units}

\newcommand*{\defaultunits}{%
  \renewcommand*{\weightunit}{kg}%
  \renewcommand*{\lengthunit}{mm}%
  \renewcommand*{\currencyunit}{EUR}%
}

\newcommand*{\RequireRegionsDialect}[1]{%
  \TrackLangRequireDialect
   [\TrackLangRequireResource{\CurrentTrackedTag}%
     \ifx\CurrentTrackedTag\CurrentTrackedLanguage
       \TrackLangAddToCaptions{\defaultunits}%
     \else
       \TrackLangRequireResource{\CurrentTracked-
Language}%
     \fi
   ]%
   {regions}{#1}%
}
```

```
\AnyTrackedLanguages
{%
  \ForEachTrackedDialect{\this@dialect}%
    \RequireRegionsDialect\this@dialect
  %
}
{% no tracked languages, default already set up
}

\endinput
```

Note that we still have a problem for dialect labels that are identical to root language labels with an associated territory (such as `manx`). This case can be checked with the following adjustment:

```
\newcommand*{\RequireRegionsDialect}[1]{%
  \TrackLangRequireDialect
  [\TrackLangRequireResource{\CurrentTrackedTag}%
   \ifx\CurrentTrackedTag\CurrentTrackedLanguage
     \ifx\CurrentTrackedRegion\empty
       \TrackLangAddToCaptions{\defaultunits}%
     \else
       \TrackLangRequireResource{\CurrentTracked-
Region}%
     \fi
   \else
     \TrackLangRequireResource{\CurrentTracked-
Language}%
   \fi
  ]%
  {regions}{#1}%
}
```

In the case where both the dialect and root language label are `manx` with the resource files `regions-manx.ldf` and `regions-IM.ldf`, then `\CurrentTrackedTag` will be `manx` (the dialect label) so `regions-manx.ldf` will be loaded with:

```
\TrackLangRequireResource{\CurrentTrackedTag}
```

In this case `\CurrentTrackedRegion` is `IM` (that is, it's not empty) so then `regions-IM.ldf` will be loaded with:

```
\TrackLangRequireResource{\CurrentTrackedRegion}
```

Here's another document that sets up dialects with tracklang labels that aren't recognised by babel. This means that there's no corresponding \captions⟨*dialect*⟩ hook for either the dialect label or the root language label, so mappings need to be defined from the tracklang dialect label to the matching babel dialect label.

```
\documentclass{article}

\usepackage{tracklang}

\TrackLanguageTag{de-US-1996}
\SetTrackedDialectLabelMap{\TrackLangLastTracked-
Dialect}{ngerman}

\TrackLanguageTag{en-MT}
\SetTrackedDialectLabelMap{\TrackLangLastTracked-
Dialect}{UKenglish}

\usepackage[main=ngerman,UKenglish]{babel}
\usepackage{regions}

\begin{document}
\selectlanguage{ngerman}

\unitname: \weightunit, \lengthunit, \currencyunit.

\selectlanguage{UKenglish}

\unitname: \weightunit, \lengthunit, \currencyunit.

\end{document}
```

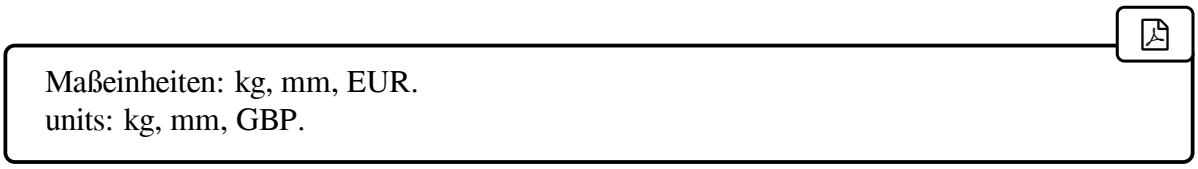
This produces:

Maßeinheiten: lb, in, USD.
units: kg, mm, EUR.

Compare this with:

```
\documentclass{article}

\usepackage[main=ngerman,UKenglish]{babel}
\usepackage{regions}

\begin{document}
\selectlanguage{ngerman}

\unitname: \weightunit, \lengthunit, \currencyunit.

\selectlanguage{UKenglish}

\unitname: \weightunit, \lengthunit, \currencyunit.

\end{document}
```

which produces:



Note that these mappings aren't needed if babel is loaded with the root language labels instead. For example:

```
\documentclass{article}

\usepackage{tracklang}

\TrackLanguageTag{de-US-1996}
\SetTrackedDialectLabelMap{\TrackLangLastTracked-
Dialect}{ngerman}

\TrackLanguageTag{en-MT}

\usepackage[main=ngerman,english]{babel}
\usepackage{regions}

\begin{document}
\selectlanguage{ngerman}
```

```
\unitname: \weightunit, \lengthunit, \currencyunit.

\selectlanguage{english}

\unitname: \weightunit, \lengthunit, \currencyunit.

\end{document}
```

No mapping is required for the `en-MT` locale as it can pick up `\captionsenglish` when `\TrackLangAddToHook` (used by `\TrackLangAddToCaptions`) queries the root language label after failing to find the language hook from the dialect label.

Some of the predefined tracklang dialects come with a mapping to the closest matching babel dialect label. For example, the option `ngermanDE` listed in Table 1.3 on page 7 automatically provides a mapping to `ngerman`. Since a tracklang dialect label can only map to one babel label, this can be problematic for synonymous labels such as `british`/`UKenglish` or `american`/`USenglish`. The default mappings used by tracklang are shown in Table 1.3 on page 7.

6. Adding Support for Language Tracking

If you are writing a package that *sets up* the document languages (rather than a package that provides multilingual support if the user has already loaded a language package) then you can load tracklang and use the commands below to help other packages track your provided languages. (See also: Integrating `tracklang.tex` into Language Packages.[1])

The tracklang package can be loaded using

```
\input tracklang
```

or (LaTeX only)

```
\RequirePackage{tracklang}
```

When using LaTeX, there's a difference between the two. The first case prevents tracklang from picking up the document class options but skips the check for known language packages. This check is redundant since your package is the language package, so you need to decide whether or not to allow the user to set up the localisation information through the document class options.

There's a hook that, if defined, is performed by `tracklang.sty` after the package options have been loaded but before known language packages are checked:

```
\@tracklang@prelangpkgcheck@hook
```

If you prefer `\RequirePackage` over `\input` but you want to make `tracklang.sty` skip the check for known language packages then (as from v1.3.8) define the pre-language package check hook as follows:

```
\providecommand\@tracklang@prelangpkgcheck@hook
{\endinput}
\RequirePackage{tracklang}[2019/10/06]% v1.3.8+
```

This will still pick up languages supplied through the document class options.

[1] dickimaw-books.com/latex/tracklang/langpkg.shtml

If you just use \input, there's a test at the start of tracklang.tex to determine if it's already been loaded, so you don't need to worry if the document has already input it.

To integrate tracklang into your language package, you need to consider the following steps:

1. Does tracklang define your supported ISO 15924 language scripts in the tracklang-scripts.tex file?

   If yes, then skip this step. Otherwise create a file with the relevant \TrackLangScriptMap command for each unknown script and identify this new file with \TrackLangAddExtraScriptFile (see §6.3). This usually won't be necessary unless you have a custom script or a child script (a script that's a sub-category of another script).

2. Does tracklang recognise the root language?

   If yes, then skip this step.

   If your package is setting up a language that tracklang doesn't recognise then you will need to define the root language using \TrackLangNewLanguage (see §6.5).

   This usually won't be the case as tracklang should support all languages that have an official ISO 639-1 alpha-2 code.

   If you simply have a different label from tracklang identifying the root language, then you can just set up your label as a dialect using \TrackLangProvidePredefinedDialect.

3. Does tracklang define the relevant ISO 3166-1 region codes in the tracklang-region-codes.tex file?

   If yes, then skip this step. Otherwise create a file with the relevant \TrackLangRegionMap command for each new region and identify this new file with \TrackLangAddExtraRegionFile (see §6.4). This usually won't be necessary as tracklang should recognise all countries that have an alpha-2 region code, but you may require it if you need a broader region, such as EU.

4. Do you want to define some convenient dialect labels that can be used with \TrackPredefinedDialect?

   If no, then skip this step. Otherwise you can use \TrackLangProvidePredefinedLanguage for root languages and \TrackLangProvidePredefinedDialect for dialects with additional information, such as a region, sub-language or script (see §6.6).

5. In your language initialisation code, add the tracklang code to track the particular dialect (for example, use \TrackPredefinedDialect for recognised dialect labels or use the \AddTracked⟨*Xxx*⟩ set of commands). See §6.1.

6. In your language selection code (such as \selectlanguage), add \SetCurrentTrackedDialect{⟨*label*⟩} to allow the document author to easily query the current localisation settings (such as the region). See §6.2.

6.1. Initialising a New Language or Dialect

When the user requests a particular dialect through your language package, you can notify tracklang of this choice using

```
\TrackPredefinedDialect{⟨dialect label⟩}
```

provided the dialect label is recognised by tracklang (all those listed in tables 1.1, 1.2 & 1.3).

If there's no matching dialect predefined by tracklang, you can just use `\TrackLocale` or `\TrackLanguageTag` (described in §3) with the appropriate ISO codes *if you're not providing caption hooks*.

If you are providing a captions hook mechanism in the form `\captions`⟨*dialect*⟩, then if ⟨*dialect*⟩ doesn't match the corresponding tracklang dialect label, you can provide a mapping using `\SetTrackedDialectLabelMap`, described below.

6.2. Switching Language or Dialect

When the document author switches to a different language or dialect, the current localisation information can be set with:

```
\SetCurrentTrackedDialect{⟨dialect⟩}
```

where ⟨*dialect*⟩ may the tracklang dialect label, or the mapped label previously set through `\SetTrackedDialectLabelMap`, described below, or the language label (in which case the last dialect to be tracked with that root language will be assumed).

This will make the following commands available which may be of use to other packages:

- `\CurrentTrackedDialect` The dialect label recognised by tracklang (which may not be the same as ⟨*dialect*⟩).
- `\CurrentTrackedLanguage` The root language label used by tracklang.
- `\CurrentTrackedDialectModifier` The dialect modifier.
- `\CurrentTrackedDialectVariant` The dialect variant.
- `\CurrentTrackedDialectScript` The dialect script. Note that if tracklang−scripts is also loaded, this allows the script direction to be accessed using

  ```
  \TrackLangScriptAlphaToDir{\CurrentTrackedDi-
  alectScript}
  ```

  See §4 for further details.

- `\CurrentTrackedDialectSubLang` The dialect sub-language code.
- `\GetTrackedDialectAdditional` The dialect's additional information.
- `\CurrentTrackedIsoCode` The dialect's root language ISO code. (The first found in the sequence 639-1, 639-2, 639-3.)
- `\CurrentTrackedRegion` The dialect's ISO 3166-1 region code.
- `\CurrentTrackedLanguageTag` The dialect's language tag.

(Without this automated use of `\SetCurrentTrackedDialect`, the same information can be picked up using commands like `\GetTrackedDialectScript`, but that's less convenient, especially if `\languagename` needs to be converted to ⟨*dialect*⟩. See the accompanying sample file `sample-setlang.tex` for an example.)

6.3. Defining New Scripts

The `tracklang-scripts.tex` file isn't automatically loaded, but if it is then, as from v1.4, it contains a hook at the end of the file that can be used to load additional files that define supplementary scripts. This entails creating a file called, say, `mypackage-scripts.tex` that contains:

```
\TrackLangScriptMap{⟨alpha code⟩}{⟨numeric code⟩}{⟨name⟩}
{⟨direction⟩}{⟨parent script⟩}
```

The first argument ⟨*alpha code*⟩ is the four-letter ISO 15924 code (such as `Latn`), the second argument is the numeric code (such as 215), the third argument ⟨*name*⟩ is the name of the script (such as `Latin`), the fourth argument is the direction (such as `LR` for left-to-right) and the final argument is the parent script (leave blank if there's no parent). Note that this command will override any previous mapping for those codes. No check is performed to determine if they have already been defined.

The supplementary file should be identified with:

```
\TrackLangAddExtraScriptFile{⟨filename⟩}
```

Additional information can be found in §4.

6.4. Defining New Regions

The `tracklang-region-codes.tex` file isn't automatically loaded, but if it is then, as from v1.4, it contains a hook at the end of the file that can be used to load additional files that define supplementary regions. This entails creating a file called, say, `mypackage-regions.tex` that contains:

```
\TrackLangRegionMap{⟨numeric code⟩}{⟨alpha-2 code⟩}{⟨alpha-3 code⟩}
```

where the first argument is the numeric region code (such as 826), the second argument is the alpha-2 region code (such as GB) and the third argument is the alpha-3 region code (such as GBR). Note that this command will override any previous mapping for those codes. No check is performed to determine if they have already been defined.

The supplementary file should be identified with:

```
\TrackLangAddExtraRegionFile{⟨filename⟩}
```

Additional information can be found in §4.

6.5. Defining a New Language

(New to version 1.3.) If the root language isn't recognised by tracklang (not listed in Table 1.2 on page 5), then it can be defined (but not tracked at this point) using:

```
\TrackLangNewLanguage{⟨language label⟩}{⟨639-1 code⟩}{⟨639-2 (T)⟩}{⟨639-2 (B)⟩}{⟨639-3⟩}{⟨3166-1⟩}{⟨default script⟩}
```

where ⟨*language label*⟩ is the root language label, ⟨*639-1 code*⟩ is the ISO 639-1 code for that language (may be empty if there isn't one), ⟨*639-2 (T)*⟩ is the ISO 639-2 (T) code for that language (may be empty if there isn't one), ⟨*639-2 (B)*⟩ is the ISO 639-2 (B) code for that language (may be empty if it's the same as ⟨*639-2 (T)*⟩), ⟨*639-3*⟩ is the ISO 639-3 code for that language (empty if the same as the 639-2 code), ⟨*3166-1*⟩ is the territory ISO 3166-1 code for languages that are only spoken in one territory (should be empty if the language is spoken in multiple territories), and ⟨*default script*⟩ is the default script (empty if disputed or varies according to region).

You can then track this language using:

```
\AddTrackedDialect{⟨dialect label⟩}{⟨root language label⟩}
```

for dialects (where ⟨*dialect label*⟩ is the dialect label and ⟨*root language label*⟩ is the root language label) or, if no regional variant is needed, you can instead use:

```
\AddTrackedLanguage{⟨root language label⟩}
```

This is equivalent to

```
\AddTrackedDialect{⟨root language label⟩}{⟨root language label⟩}
```

Note that \AddTrackedDialect defines:

```
\TrackLangLastTrackedDialect
```

to the dialect label, which makes it easier to reference the last dialect to be tracked.

6.6. Defining New tracklang Labels

A dialect label may be predefined with associated information that allows that particular combination to be easily tracked with \TrackPredefinedDialect. In the case of a dialect label that only requires the information provided in \TrackLangNewLanguage you can use:

```
\TrackLangProvidePredefinedLanguage{⟨language label⟩}
```

where ⟨*language label*⟩ corresponds to the language label used in \TrackLangNewLanguage. This allows

```
\TrackPredefinedDialect{⟨label⟩}
```

to not only track the root language but also the associated ISO codes.

If the dialect label doesn't match the root language label then use:

```
\TrackLangProvidePredefinedDialect{⟨dialect label⟩}{⟨root
language label⟩}{⟨3166-1 code⟩}{⟨modifier⟩}{⟨variant⟩}{⟨map⟩}{⟨script⟩}
```

where ⟨*dialect label*⟩ is the new tracklang dialect label, {root language label} is the tracklang root language label, ⟨*region*⟩ is the ISO 3166-1 region code (may be empty), ⟨*modifier*⟩ is the modifier (may be empty), ⟨*variant*⟩ is the variant information (may be empty), ⟨*map*⟩ is your package's language label that corresponds to the tracklang dialect label supplied in the first argument (may be empty if identical), and ⟨*script*⟩ is the ISO 15924 alpha-4 script code (may be empty if it's the same as the default script for the root language).

For compatibility with pre version 1.3, if the dialect isn't predefined by tracklang, then you can use:

```
\AddTrackedDialect{dialect}{root language label}
```

where ⟨*root language label*⟩ is the label for the dialect's root language (Table 1.2 on page 5) and ⟨*dialect*⟩ matches the captions hook. If the dialect is already in the tracked dialect list, it won't be added again. If the root language is already in the tracked language list, it won't be

added again. As from version 1.3 this additionally defines `\TrackLangLastTrackedDialect` to ⟨*dialect*⟩ for convenient reference if required. Note that `\AddTrackedDialect` is internally used by commands like `\TrackPredefinedDialect`, `\TrackLocale` and `\TrackLanguageTag`.

(New to version 1.3.) Many of the tracklang dialect labels don't have a corresponding match in various language packages. For example, tracklang provides `ngermanDE` but the closest match in babel is `ngerman`. This means that the caption hook `\captionsngerman` can't be accessed through:

```
\csname captions\CurrentTrackedDialect\endcsname
```

in the resource files. In this case, a mapping may be defined between the tracklang dialect label and the closest matching label used by the language hooks. This is done through

```
\SetTrackedDialectLabelMap{⟨tracklang-label⟩}{⟨hook-label⟩}
```

where ⟨*tracklang-label*⟩ is the tracklang label and ⟨*hook-label*⟩ is the language hook label. For example:

```
\TrackLanguageTag{de-AR-1996}
\SetTrackedDialectLabelMap{\TrackLangLastTracked-
Dialect}{ngerman}
```

Since `\TrackLanguageTag` internally uses `\AddTrackedDialect` the dialect label created by tracklang can be accessed using `\TrackLangLastTrackedDialect`. This means that `\TrackLangAddToCaptions` can now find the `\captionsngerman` hook even though the tracklang dialect label isn't `ngerman`.

(New to version 1.3.) If the root language label is recognised by tracklang, you can add the ISO codes using:

```
\AddTrackedLanguageIsoCodes{⟨root language label⟩}
```

As from v1.3, you can also provide a modifier for a given dialect using:

```
\SetTrackedDialectModifier{⟨dialect⟩}{⟨value⟩}
```

where ⟨*dialect*⟩ is the dialect label and ⟨*value*⟩ is the modifier value. For example:

```
\AddTrackedDialect{oldgerman}{german}
\AddTrackedLanguageIsoCodes{german}
\SetTrackedDialectModifier{oldgerman}{old}
```

Note that no sanitization is performed on ⟨*value*⟩ when the modifier is set explicitly through `\SetTrackedDialectModifier`, since it's assumed that any package that specifically sets the modifier in this way is using a sensible labelling system. If the modifier is obtained through commands like `\TrackLocale`, then the modifier is sanitized as the value may have been obtained from the operating system and there's no guarantee that it won't contain problematic characters.

The modifier is typically obtained by parsing locale information in POSIX format.

⟨*language*⟩`[_`⟨*territory*⟩`][.`⟨*codeset*⟩`][@`⟨*modifier*⟩`]`

whereas the variant is typically obtained by parsing the language tag.

The information provided in the commands below (such as the script) are typically obtained by parsing the language tag. For example, with Serbian in the Latin alphabet the modifier would be `latin` whereas the script would be `Latn`:

```
\AddTrackedDialect{serbianlatin}{serbian}
\AddTrackedLanguageIsoCodes{serbian}
\SetTrackedDialectModifier{serbianlatin}{latin}
\SetTrackedDialectScript{serbianlatin}{Latn}
```

As from v1.3, you can provide a script (for example, `Latn` or `Cyrl`) using:

```
\SetTrackedDialectScript{⟨dialect⟩}{⟨value⟩}
```

where ⟨*dialect*⟩ is the dialect label and ⟨*value*⟩ is the ISO 15924 alpha-4 script identifier. For example:

```
\AddTrackedDialect{serbiancyrl}{serbian}
\AddTrackedLanguageIsoCodes{serbian}
\SetTrackedDialectScript{serbiancyrl}{Cyrl}
```

As from v1.3, you can provide a variant for a given dialect using:

```
\SetTrackedDialectVariant{⟨dialect⟩}{⟨value⟩}
```

For example:

```
\AddTrackedDialect{german1901}{german}
\SetTrackedDialectVariant{german1901}{1901}
```

As from v1.3, you can also provide a sub-language using:

```
\SetTrackedDialectSubLang{⟨dialect⟩}{⟨value⟩}
```

where ⟨*dialect*⟩ is the dialect label and ⟨*value*⟩ is the code. For example:

```
\AddTrackedDialect{mandarin}{chinese}
\AddTrackedLanguageIsoCodes{chinese}
\SetTrackedDialectSubLang{mandarin}{cmn}
\AddTrackedIsoLanguage{639-3}{cmn}{mandarin}
```

As from v1.3, you can also provide additional information using:

```
\SetTrackedDialectAdditional{⟨dialect⟩}{⟨value⟩}
```

where ⟨*dialect*⟩ is the dialect label and ⟨*value*⟩ is the additional information.

6.7. Example (`alien.sty`)

Suppose I want to create a language package `alien.sty` that defines the `martian` language with regional dialects `lowermartian` and `uppermartian`. First, let's suppose that tracklang recognises the root language `martian`:

```
\ProvidesPackage{alien}

\inputtracklang% v1.3

\DeclareOption{martian}{%
 \TrackPredefinedDialect{martian}
}
\DeclareOption{lowermartian}{%
 \AddTrackedDialect{lowermartian}{martian}
 \AddTrackedLanguageIsoCodes{martian}
 \AddTrackedIsoLanguage{3166-1}{YY}{lowermartian}
 % other attributes such as
 % \SetTrackedDialectVariant{lowermartian}{...}
}
\DeclareOption{uppermartian}{%
 \AddTrackedDialect{uppermartian}{martian}
```

```
 \AddTrackedLanguageIsoCodes{martian}
 \AddTrackedIsoLanguage{3166-1}{XX}{uppermartian}
 % other attributes such as
 % \SetTrackedDialectVariant{uppermartian}{...}
}

\ProcessOptions

\newcommand*{\selectlanguage}[1]{%
 \def\languagename{#1}%
 % other stuff
 \SetCurrentTrackedDialect{#1}%
}

\AnyTrackedLanguages
{
 \ForEachTrackedDialect{\thisdialect}
 {%
  \TrackLangRequireDialect{alien}{\thisdialect}
 }
}
```

The caption commands and language set up are in the files `alien-`⟨*localeid*⟩`.ldf` as in the examples from §5.1. This allows for the user having already loaded tracklang before alien and used `\TrackLangFromEnv` to pick up the locale from the operating system's environment variables. (For example, they may have `LANG` set to `xx_YY`.)

The resource files may need to set the mapping between the tracklang dialect label and the alien dialect label. For example, in `alien-xx-YY.ldf`:

```
\TrackLangProvidesResource{xx-YY}

\TrackLangRequireResource{martian}
% load common elements

\newcommand{\captionslowermartian}{%
 \captionsmartian
 \def\contentsname{X'flurp}% regional variation
}

\SetTrackedDialectLabelMap{\CurrentTrackedDialect}
{lowermartian}
```

Now let's consider the case where tracklang doesn't know about the `martian` language. In this case the user can't track the dialect until the root language has been defined, so the user can't use `\TrackLangFromEnv` before using the alien package.

With tracklang v1.3. The new root language can be defined with a minor adjustment to the above code:

```
\ProvidesPackage{alien}

\input{tracklang}% needs v1.3

\TrackLangIfKnownLang{martian}
{}
% tracklang already knows about the martian language
{
 % tracklang doesn't known about the martian language, so define
 % with ISO 639-1 (xx) and ISO 639-2 (xxx) codes:
 \TrackLangNewLanguage{martian}{xx}{xxx}{}{}{}{Latn}
}
```

The rest is as before.

Now other package writers who want to provide support for the Martian dialects can easily detect which language options the user requested through my package, *without needing to know anything about my alien package*.

Part II.

Summaries

A. Region and Script Mappings

Region mappings are listed in Table A.1, and script mappings are listed in Table A.2 on page 88.

Table A.1.: Region Mappings

Alpha-2	Alpha-3	Numeric	Alpha-2	Alpha-3	Numeric
AD	AND	020	AE	ARE	784
AF	AFG	004	AG	ATG	028
AI	AIA	660	AL	ALB	008
AM	ARM	051	AO	AGO	024
AQ	ATA	010	AR	ARG	032
AS	ASM	016	AT	AUT	040
AU	AUS	036	AW	ABW	533
AX	ALA	248	AZ	AZE	031
BA	BIH	070	BB	BRB	052
BD	BGD	050	BE	BEL	056
BF	BFA	854	BG	BGR	100
BH	BHR	048	BI	BDI	108
BJ	BEN	204	BL	BLM	652
BM	BMU	060	BN	BRN	096
BO	BOL	068	BQ	BES	535
BR	BRA	076	BS	BHS	044
BT	BTN	064	BV	BVT	074
BW	BWA	072	BY	BLR	112
BZ	BLZ	084	CA	CAN	124
CC	CCK	166	CD	COD	180
CF	CAF	140	CG	COG	178
CH	CHE	756	CI	CIV	384
CK	COK	184	CL	CHL	152
CM	CMR	120	CN	CHN	156
CO	COL	170	CR	CRI	188
CU	CUB	192	CV	CPV	132
CW	CUW	531	CX	CXR	162
CY	CYP	196	CZ	CZE	203
DE	DEU	276	DJ	DJI	262
DK	DNK	208	DM	DMA	212
DO	DOM	214	DZ	DZA	012

Table A.1.: Region Mappings (Continued)

Alpha-2	Alpha-3	Numeric	Alpha-2	Alpha-3	Numeric
EC	ECU	218	EE	EST	233
EG	EGY	818	EH	ESH	732
ER	ERI	232	ES	ESP	724
ET	ETH	231	FI	FIN	246
FJ	FJI	242	FK	FLK	238
FM	FSM	583	FO	FRO	234
FR	FRA	250	GA	GAB	266
GB	GBR	826	GD	GRD	308
GE	GEO	268	GF	GUF	254
GG	GGY	831	GH	GHA	288
GI	GIB	292	GL	GRL	304
GM	GMB	270	GN	GIN	324
GP	GLP	312	GQ	GNQ	226
GR	GRC	300	GS	SGS	239
GT	GTM	320	GU	GUM	316
GW	GNB	624	GY	GUY	328
HK	HKG	344	HM	HMD	334
HN	HND	340	HR	HRV	191
HT	HTI	332	HU	HUN	348
ID	IDN	360	IE	IRL	372
IL	ISR	376	IM	IMN	833
IN	IND	356	IO	IOT	086
IQ	IRQ	368	IR	IRN	364
IS	ISL	352	IT	ITA	380
JE	JEY	832	JM	JAM	388
JO	JOR	400	JP	JPN	392
KE	KEN	404	KG	KGZ	417
KH	KHM	116	KI	KIR	296
KM	COM	174	KN	KNA	659
KP	PRK	408	KR	KOR	410
KW	KWT	414	KY	CYM	136
KZ	KAZ	398	LA	LAO	418
LB	LBN	422	LC	LCA	662
LI	LIE	438	LK	LKA	144
LR	LBR	430	LS	LSO	426
LT	LTU	440	LU	LUX	442
LV	LVA	428	LY	LBY	434
MA	MAR	504	MC	MCO	492
MD	MDA	498	ME	MNE	499
MF	MAF	663	MG	MDG	450

Table A.1.: Region Mappings (Continued)

Alpha-2	Alpha-3	Numeric	Alpha-2	Alpha-3	Numeric
MH	MHL	584	MK	MKD	807
ML	MLI	466	MM	MMR	104
MN	MNG	496	MO	MAC	446
MP	MNP	580	MQ	MTQ	474
MR	MRT	478	MS	MSR	500
MT	MLT	470	MU	MUS	480
MV	MDV	462	MW	MWI	454
MX	MEX	484	MY	MYS	458
MZ	MOZ	508	NA	NAM	516
NC	NCL	540	NE	NER	562
NF	NFK	574	NG	NGA	566
NI	NIC	558	NL	NLD	528
NO	NOR	578	NP	NPL	524
NR	NRU	520	NU	NIU	570
NZ	NZL	554	OM	OMN	512
PA	PAN	591	PE	PER	604
PF	PYF	258	PG	PNG	598
PH	PHL	608	PK	PAK	586
PL	POL	616	PM	SPM	666
PN	PCN	612	PR	PRI	630
PS	PSE	275	PT	PRT	620
PW	PLW	585	PY	PRY	600
QA	QAT	634	RE	REU	638
RO	ROU	642	RS	SRB	688
RU	RUS	643	RW	RWA	646
SA	SAU	682	SB	SLB	090
SC	SYC	690	SD	SDN	729
SE	SWE	752	SG	SGP	702
SH	SHN	654	SI	SVN	705
SJ	SJM	744	SK	SVK	703
SL	SLE	694	SM	SMR	674
SN	SEN	686	SO	SOM	706
SR	SUR	740	SS	SSD	728
ST	STP	678	SV	SLV	222
SX	SXM	534	SY	SYR	760
SZ	SWZ	748	TC	TCA	796
TD	TCD	148	TF	ATF	260
TG	TGO	768	TH	THA	764
TJ	TJK	762	TK	TKL	772
TL	TLS	626	TM	TKM	795

Table A.1.: Region Mappings (Continued)

Alpha-2	Alpha-3	Numeric	Alpha-2	Alpha-3	Numeric
TN	TUN	788	TO	TON	776
TR	TUR	792	TT	TTO	780
TV	TUV	798	TW	TWN	158
TZ	TZA	834	UA	UKR	804
UG	UGA	800	UM	UMI	581
US	USA	840	UY	URY	858
UZ	UZB	860	VA	VAT	336
VC	VCT	670	VE	VEN	862
VG	VGB	092	VI	VIR	850
VN	VNM	704	VU	VUT	548
WF	WLF	876	WS	WSM	882
YE	YEM	887	YT	MYT	175
ZA	ZAF	710	ZM	ZMB	894
ZW	ZWE	716			

Table A.2.: Script Mappings

Alpha-2	Numeric	Direction	Description
Adlm	166	RL	Adlam.
Afak	439	varies	Afaka.
Aghb	239	LR	Caucasian Albanian.
Ahom	338	LR	Ahom, Tai Ahom.
Arab	160	RL	Arabic.
Aran	161	RL	Arabic (Nastaliq variant).
Armi	124	RL	Imperial Aramaic.
Armn	230	LR	Armenian.
Avst	134	RL	Avestan.
Bali	360	LR	Balinese.
Bamu	435	LR	Bamum.
Bass	259	LR	Bassa Vah.
Batk	365	LR	Batak.
Beng	334	LR	Bhaiksuki.
Blis	550	varies	Blissymbols.
Bopo	285	LR	Bopomofo.
Brah	300	LR	Brahmi.
Brai	570	LR	Braille.
Bugi	367	LR	Buginese.
Buhd	372	LR	Buhid.
Cakm	349	LR	Chakma.
Cans	440	LR	Unified Canadian Aboriginal Syllabics.
Cari	201	LR	Carian.
Cham	358	LR	Cham.
Cher	445	LR	Cherokee.
Cirt	291	varies	Cirth.
Copt	204	LR	Coptic.
Cprt	403	RL	Cypriot.
Cyrl	220	LR	Cyrillic.
Cyrs	221	varies	Cyrillic (Old Church Slavonic variant).
Deva	315	LR	Devanagari (Nagari).
Dsrt	250	LR	Deseret (Mormon).
Dupl	755	LR	Duployan shorthand, Duployan stenography.
Egyd	070	RL	Egyptian demotic.
Egyh	060	RL	Egyptian hieratic.
Egyp	050	LR	Egyptian hieroglyphs.
Elba	226	LR	Elbasan.
Ethi	430	LR	Ethiopic (Ge'ez).
Geok	241	LR	Khutsuri (Asomtavruli and Nuskhuri).
Geor	240	LR	Georgian (Mkhedruli).

Table A.2.: Script Mappings (Continued)

Alpha-2	Numeric	Direction	Description
Glag	225	LR	Glagolitic.
Goth	206	LR	Gothic.
Gran	343	LR	Grantha.
Grek	200	LR	Greek.
Gujr	320	LR	Gujarati.
Guru	310	LR	Gurmukhi.
Hanb	503	LR	Han with Bopomofo (alias for Han + Bopomofo).
Hang	286	LR	Hangul.
Hani	500	LR	Han (Hanzi, Kanji, Hanja).
Hano	371	LR	Hanunoo.
Hans	501	varies	Han (Simplified variant).
Hant	502	varies	Han (Traditional variant).
Hatr	127	RL	Hatran.
Hebr	125	RL	Hebrew.
Hira	410	LR	Hiragana.
Hluw	080	LR	Anatolian Hieroglyphs (Luwian Hieroglyphs, Hittite Hieroglyphs).
Hmng	450	LR	Pahawh Hmong.
Hrkt	412	varies	Japanese syllabaries (alias for Hiragana + Katakana).
Hung	176	RL	Old Hungarian (Hungarian Runic).
Inds	610	RL	Indus (Harappan).
Ital	210	LR	Old Italic (Etruscan, Oscan, etc.)
Jamo	284	LR	Jamo (alias for Jamo subset of Hangul).
Java	361	LR	Javanese.
Jpan	413	varies	Japanese (alias for Han + Hiragana + Katakana).
Jurc	510	LR	Jurchen.
Kali	357	LR	Kayah Li.
Kana	411	LR	Katakana.
Khar	305	RL	Kharoshthi.
Khmr	355	LR	Khmer.
Khoj	322	LR	Khojki.
Kitl	505	LR	Khitan large script.
Kits	288	TB	Khitan small script.
Knda	345	LR	Kannada.
Kore	287	LR	Korean (alias for Hangul + Han).
Kpel	436	LR	Kpelle.
Kthi	317	LR	Kaithi.
Lana	351	LR	Tai Tham (Lanna).
Laoo	356	LR	Lao.
Latf	217	varies	Latin (Fraktur variant).

Table A.2.: Script Mappings (Continued)

Alpha-2	Numeric	Direction	Description
Latg	216	LR	Latin (Gaelic variant).
Latn	215	LR	Latin.
Leke	364	LR	Leke.
Lepc	335	LR	Lepcha.
Limb	336	LR	Limbu.
Lina	400	LR	Linear A.
Linb	401	LR	Linear B.
Lisu	399	LR	Lisu (Fraser).
Loma	437	LR	Loma.
Lyci	202	LR	Lycian.
Lydi	116	RL	Lydian.
Mahj	314	LR	Mahajani.
Mand	140	RL	Mandaic, Mandaean.
Mani	139	RL	Manichaean.
Marc	332	LR	Marchen.
Maya	090	varies	Mayan hieroglyphs.
Mend	438	RL	Mende Kikakui.
Merc	101	RL	Meroitic Cursive.
Mero	100	RL	Meroitic Hieroglyphs.
Mlym	347	LR	Malayalam.
Modi	324	LR	Modi.
Mong	145	TB	Mongolian.
Moon	218	varies	Moon (Moon code, Moon script, Moon type).
Mroo	199	LR	Mro, Mru.
Mtei	337	LR	Meitei Mayek (Meithei, Meetei).
Mult	323	LR	Multani.
Mymr	350	LR	Myanmar (Burmese).
Narb	106	RL	Old North Arabian (Ancient North Arabian).
Nbat	159	RL	Nabataean.
Newa	333	LR	Newa, Newar, Newari.
Nkgb	420	LR	Nakhi Geba.
Nkoo	165	RL	N'Ko.
Nshu	499	LR	Nushu.
Ogam	212	varies	Ogham.
Olck	261	LR	Ol Chiki.
Orkh	175	RL	Old Turkic, Orkhon Runic.
Orya	327	LR	Oriya.
Osge	219	LR	Osage.
Osma	260	LR	Osmanya.
Palm	126	RL	Palmyrene.

Table A.2.: Script Mappings (Continued)

Alpha-2	Numeric	Direction	Description
Pauc	263	LR	Pau Cin Hau.
Perm	227	LR	Old Permic.
Phag	331	TB	Phags-pa.
Phli	131	RL	Inscriptional Pahlavi.
Phlp	132	RL	Psalter Pahlavi.
Phlv	133	RL	Book Pahlavi.
Phnx	115	RL	Phoenician.
Piqd	293	LR	Klingon (KLI plqaD).
Plrd	282	LR	Miao (Pollard).
Prti	130	RL	Inscriptional Parthian.
Qaaa	900	varies	Reserved for private use (start).
Qaai	908	varies	Private use.
Qabx	949	varies	Reserved for private use (end).
Rjng	363	LR	Rejang (Redjang, Kaganga).
Roro	620	varies	Rongorongo.
Runr	211	LR	Runic.
Samr	123	RL	Samaritan.
Sara	292	varies	Sarati.
Sarb	105	RL	Old South Arabian.
Saur	344	LR	Saurashtra.
Sgnw	095	TB	SignWriting.
Shaw	281	LR	Shavian (Shaw).
Shrd	319	LR	Sharada.
Sidd	302	LR	Siddham.
Sind	318	LR	Khudawadi, Sindhi.
Sinh	348	LR	Sinhala.
Sora	398	LR	Sora Sompeng.
Sund	362	LR	Sundanese.
Sylo	316	LR	Syloti Nagri.
Syrc	135	RL	Syriac.
Syre	138	RL	Syriac (Estrangelo variant).
Syrj	137	RL	Syriac (Western variant).
Syrn	136	RL	Syriac (Eastern variant).
Tagb	373	LR	Tagbanwa.
Takr	321	LR	Takri.
Tale	353	LR	Tai Le.
Talu	354	LR	New Tai Lue.
Taml	346	LR	Tamil.
Tang	520	LR	Tangut.
Tavt	359	LR	Tai Viet.

Table A.2.: Script Mappings (Continued)

Alpha-2	Numeric	Direction	Description
Telu	340	LR	Telugu.
Teng	290	LR	Tengwar.
Tfng	120	LR	Tifinagh (Berber).
Tglg	370	LR	Tagalog (Baybayin, Alibata).
Thaa	170	RL	Thaana.
Thai	352	LR	Thai.
Tibt	330	LR	Tibetan.
Tirh	326	LR	Tirhuta.
Ugar	040	LR	Ugaritic.
Vaii	470	LR	Vai.
Visp	280	LR	Visible Speech.
Wara	262	LR	Warang Citi (Varang Kshiti).
Wole	480	RL	Woleai.
Xpeo	030	LR	Old Persian.
Xsux	020	LR	Cuneiform, Sumero-Akkadian.
Yiii	460	LR	Yi.
Zinh	994	inherited	Inherited script.
Zmth	995	LR	Mathematical notation.
Zsye	993	varies	Symbols (emoji variant).
Zsym	996	varies	Symbols.
Zxxx	997	varies	Unwritten documents.
Zyyy	998	varies	Undetermined script.
Zzzz	999	varies	Uncoded script.

Symbols

Symbol	Description
	The syntax and usage of a command, environment or option etc.
	An important message.
	Prominent information.
	LaTeX code to insert into your document.
	How the example code should appear in the PDF.
>_	A command-line application invocation that needs to be entered into a terminal or command prompt.

Glossary

Command-line interface (CLI)

An application that doesn't have a graphical user interface. That is, an application that doesn't have any windows, buttons or menus and can be run in a command prompt or terminal.[1]

Shell escape

TEX Has the ability to run CLI applications while it's typesetting a document. Whilst this is a convenient way of using tools to help build the document, it's a security risk. To help protect users from arbitrary — and potentially dangerous — code from begin executed, TEX has a restricted mode, where only trusted applications are allowed to run. This is usually the default mode, but your TEX installation may be set up so that the shell escape is disabled by default. The unrestricted mode allows you to run any application from the shell escape. Take care about enabling this option. If you receive a document or package from an untrusted source, first run TEX with the shell escape disabled or in restricted mode and search the `log` file for "runsystem" before using the unrestricted mode. Note that LuaLATEX additionally requires the shellesc package.

[1] `dickimaw-books.com/latex/novices/html/terminal.html`

Command Summary

@

\@tracklang@declareoption{ ⟨*dialect*⟩ } tracklang v1.1+ §3; 15

Provided by `tracklang.sty` to declare ⟨*dialect*⟩ as a package option that tracks ⟨*dialect*⟩. Provided by `tracklang.tex`, if not already defined, to ignore its argument.

\@tracklang@for⟨*cs*⟩**:=**⟨*list*⟩`\do`{ ⟨*body*⟩ } `tracklang.tex` v1.0+ §5; 31

As LaTeX's `\@for`.

\@tracklang@prelangpkgcheck@hook §6; 63

If defined before `tracklang.sty` v1.3.8+ is loaded, this command will be done after package options have been processed but before the check for language packages, such as babel and polyglossia.

A

\AddTrackedCountryIsoCode{ ⟨*language*⟩ } `tracklang.tex` v1.3+

Adds the ISO 3166-1 code.

\AddTrackedDialect{ ⟨*dialect label*⟩ } { ⟨*root language label*⟩ } §6.5; 67
`tracklang.tex` v1.0+

Tracks a dialect. This command defines `\TrackLangLastTrackedDialect` to provide a convenient way to reference the last dialect to be tracked.

\AddTrackedIsoLanguage { ⟨*code type*⟩ } { ⟨*code*⟩ } { ⟨*language*⟩ }
tracklang.tex v1.0+

Adds a mapping between the given ISO code and language name.

\AddTrackedLanguage { ⟨*root language label*⟩ } tracklang.tex v1.0+ §6.5; 67

Shortcut for \AddTrackedDialect { ⟨*root language label*⟩ } { ⟨*root language label*⟩ }.

\AddTrackedLanguageIsoCodes { ⟨*root language label*⟩ } §6.6; 69
tracklang.tex v1.3+

Adds the ISO 639-1, 639-2 and 639-3 codes, which must have previously been declared using \TrackLangNewLanguage.

\AnyTrackedLanguages { ⟨*true*⟩ } { ⟨*false*⟩ } tracklang.tex v1.0+ §5; 29

Expands to ⟨*true*⟩ if there are any tracked languages, otherwise expands to ⟨*false*⟩.

C

\CurrentTrackedDialect tracklang.tex v1.3+ §5; 38

Defined by \SetCurrentTrackedDialect to the dialect label, which may be the supplied ⟨*dialect*⟩ label or the mapped label or, if ⟨*dialect*⟩ is a root language label, the last tracked dialect for the given root language.

\CurrentTrackedDialectAdditional tracklang.tex v1.3+ §5; 39

Defined by \SetCurrentTrackedDialect to the additional part associated with the dialect (may be empty).

\CurrentTrackedDialectModifier tracklang.tex v1.3+ §5; 38

Defined by \SetCurrentTrackedDialect to the associated modifier (may be empty).

\CurrentTrackedDialectScript tracklang.tex v1.3+ §5; 39

Defined by \SetCurrentTrackedDialect to the script associated with the dialect, or to the default script for the language.

\CurrentTrackedDialectSubLang tracklang.tex v1.3+ §5; 39

Defined by \SetCurrentTrackedDialect to the sub language associated with the dialect (may be empty).

\CurrentTrackedDialectVariant tracklang.tex v1.3+ §5; 38

Defined by \SetCurrentTrackedDialect to the associated variant (may be empty).

\CurrentTrackedIsoCode tracklang.tex v1.3+ §5; 38

Defined by \SetCurrentTrackedDialect to the ISO 639-1 or 639-2 or 639-3 language code (may be empty).

\CurrentTrackedLanguage tracklang.tex v1.3+ §5; 38

Defined by \SetCurrentTrackedDialect to the associated root language label.

\CurrentTrackedLanguageTag tracklang.tex v1.3+ §5; 39

Defined by \SetCurrentTrackedDialect to the language tag that identifies the dialect or und if no match.

\CurrentTrackedRegion tracklang.tex v1.3+ §5; 38

Defined by \SetCurrentTrackedDialect to the ISO 3166-1 region code associated with the dialect (may be empty).

\CurrentTrackedTag tracklang.tex v1.0+ §5; 37

Expands to the current tracked tag.

F

\ForEachTrackedDialect { ⟨*cs*⟩ } { ⟨*body*⟩ } tracklang.tex v1.0+ §5; 31

Iterates through the list of tracked dialects. On each iteration ⟨*cs*⟩ is set to the dialect tag and ⟨*body*⟩ is performed.

\ForEachTrackedLanguage { ⟨*cs*⟩ } { ⟨*body*⟩ } tracklang.tex v1.0+ §5; 31

Iterates through the list of tracked languages. On each iteration ⟨*cs*⟩ is set to the language tag and ⟨*body*⟩ is performed.

G

\GetTrackedDialectAdditional { ⟨*dialect*⟩ } tracklang.tex v1.3+ §5; 37

Expands to the extra information for ⟨*dialect*⟩.

\GetTrackedDialectFromLanguageTag { ⟨*tag*⟩ } { ⟨*cs*⟩ }
tracklang.tex v1.3+ §5; 29

Finds the tracked dialect that matches the given language tag and stores the dialect label in ⟨*cs*⟩. If no match found, ⟨*cs*⟩ will be empty.

\GetTrackedDialectModifier { ⟨*dialect*⟩ } tracklang.tex v1.3+ §5; 34

Expands to the modifier for the given dialect.

\GetTrackedDialectScript { ⟨*dialect*⟩ } `tracklang.tex` v1.3+ §5; 35

Expands to the script for ⟨*dialect*⟩.

\GetTrackedDialectSubLang { ⟨*dialect*⟩ } `tracklang.tex` v1.3+ §5; 36

Expands to the sub-language for ⟨*dialect*⟩.

\GetTrackedDialectVariant { ⟨*dialect*⟩ } `tracklang.tex` v1.3+ §5; 35

Expands to the modifier for ⟨*dialect*⟩.

\GetTrackedLanguageTag { ⟨*dialect*⟩ } `tracklang.tex` v1.3+ §5; 33

Gets the language tag for ⟨*dialect*⟩.

I

\IfHasTrackedDialectAdditional { ⟨*dialect*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+ §5; 37

Expands to ⟨*true*⟩ if there's extra information for ⟨*dialect*⟩, otherwise expands to ⟨*false*⟩.

\IfHasTrackedDialectModifier { ⟨*dialect*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+ §5; 34

Expands to ⟨*true*⟩ if there's a modifier for the given dialect, otherwise expands to ⟨*false*⟩.

\IfHasTrackedDialectScript { ⟨*dialect*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+ §5; 35

Expands to ⟨*true*⟩ if there's a script for ⟨*dialect*⟩, otherwise expands to ⟨*false*⟩.

\IfHasTrackedDialectSubLang { ⟨*dialect*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
tracklang.tex v1.3+

§5; 37

Expands to ⟨*true*⟩ if there's a sub-language for ⟨*dialect*⟩, otherwise expands to ⟨*false*⟩.

\IfHasTrackedDialectVariant { ⟨*dialect*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
tracklang.tex v1.3+

§5; 35

Expands to ⟨*true*⟩ if there's a modifier for ⟨*dialect*⟩, otherwise expands to ⟨*false*⟩.

\IfTrackedDialect { ⟨*dialect-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
tracklang.tex v1.0+

§5; 31

Does ⟨*true*⟩ if the dialect identified by ⟨*dialect-label*⟩ has been tracked, otherwise does ⟨*false*⟩.

\IfTrackedDialectIsScriptCs { ⟨*dialect*⟩ } { ⟨*cs*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
tracklang.tex v1.3+

§5; 36

If the given tracked dialect has an associated script and that script code matches the replacement text for the control sequence ⟨*cs*⟩ then do ⟨*true*⟩ otherwise to ⟨*false*⟩. If the tracked dialect doesn't have an associated script then the default script for the root language is tested.

\IfTrackedIsoCode { ⟨*code type*⟩ } { ⟨*code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
tracklang.tex v1.0+

§5; 32

Does ⟨*true*⟩ if the given ISO code has been defined otherwise does ⟨*false*⟩.

\IfTrackedLanguage { ⟨*language-label*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
tracklang.tex v1.0+

§5; 31

Does ⟨*true*⟩ if the language identified by ⟨*language-label*⟩ has been tracked, otherwise does ⟨*false*⟩.

\IfTrackedLanguageFileExists{⟨*dialect*⟩}{⟨*prefix*⟩}{⟨*suffix*⟩}{⟨*true code*⟩}{⟨*false code*⟩} tracklang.tex v1.0+ §5; 37

Does \SetCurrentTrackedDialect{⟨*dialect*⟩} and if the dialect is recognised, then determines if the file ⟨*prefix*⟩⟨*tag*⟩⟨*suffix*⟩ exists. If it does, \CurrentTrackedTag is set to ⟨*tag*⟩ and ⟨*true*⟩ is done, otherwise ⟨*false*⟩ is done.

\IfTrackedLanguageHasIsoCode{⟨*code type*⟩}{⟨*label*⟩}{⟨*true*⟩}{⟨*false*⟩} tracklang.tex v1.0+ §5; 32

Does ⟨*true*⟩ if the given language or dialect has a corresponding ISO code of the given type, otherwise does ⟨*false*⟩.

\ifTrackLangShowInfo ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iftrue tracklang.tex v1.3+

Conditional that indicates whether or not to show information messages.

\ifTrackLangShowVerbose ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iffalse tracklang.tex v1.4+

Conditional that indicates whether or not to show verbose messages.

\ifTrackLangShowWarnings ⟨*true*⟩\else ⟨*false*⟩\fi *initial:* \iftrue tracklang.tex v1.3+

Conditional that indicates whether or not to show warnings.

S

\SetCurrentTrackedDialect{⟨*dialect*⟩} tracklang.tex v1.3+ §6.2; 65

Sets the current tracked dialect.

\SetTrackedDialectAdditional{⟨*dialect*⟩}{⟨*value*⟩}
tracklang.tex v1.3+ §6.6; 70

Sets the extra information for ⟨*dialect*⟩ to ⟨*value*⟩.

\SetTrackedDialectLabelMap{⟨*tracklang-label*⟩}{⟨*hook-label*⟩}
tracklang.tex v1.3+ §6.6; 68

Defines a mapping between a tracklang dialect label and the corresponding dialect label used by a language hook, such as \captions⟨*dialect*⟩.

\SetTrackedDialectModifier{⟨*dialect*⟩}{⟨*value*⟩}
tracklang.tex v1.3+ §6.6; 69

Sets the modifier for the given ⟨*dialect*⟩ to ⟨*value*⟩.

\SetTrackedDialectScript{⟨*dialect*⟩}{⟨*value*⟩} tracklang.tex v1.3+ §6.6; 69

Sets the script for ⟨*dialect*⟩ to ⟨*value*⟩.

\SetTrackedDialectSubLang{⟨*dialect*⟩}{⟨*value*⟩} tracklang.tex v1.3+ §6.6; 70

Sets the sub-language for ⟨*dialect*⟩ to ⟨*value*⟩.

\SetTrackedDialectVariant{⟨*dialect*⟩}{⟨*value*⟩} tracklang.tex v1.3+ §6.6; 70

Sets the modifier for ⟨*dialect*⟩ to ⟨*value*⟩.

T

\ThreeLetterExtIsoLanguageCode tracklang.tex v1.3+ §5; 33

Expands to 639-3 (should not be redefined).

\ThreeLetterIsoLanguageCode tracklang.tex v1.0+ §5; 33

Expands to 639-2 (should not be redefined).

\TrackedDialectClosestSubMatch tracklang.tex v1.3.6+ §5; 29

Defined by \GetTrackedDialectFromLanguageTag to the closest match.

\TrackedDialectsFromLanguage{ ⟨*root language label*⟩ } §5; 32
tracklang.tex v1.0+

Expands to a comma-separated list of the tracked dialects with the given language.

\TrackedIsoCodeFromLanguage{ ⟨*code type*⟩ } { ⟨*label*⟩ } §5; 33
tracklang.tex v1.0+

Expands to the code associated with the given language or dialect identified by ⟨*label*⟩.

\TrackedLanguageFromDialect{ ⟨*dialect*⟩ } tracklang.tex v1.0+ §5; 32

Expands to the language from the given dialect.

\TrackedLanguageFromIsoCode{ ⟨*code type*⟩ } { ⟨*code*⟩ } §5; 33
tracklang.tex v1.0+

Expands to a comma-separated list of language or dialect labels associated with the given code.

\TrackIfKnownLanguage{ ⟨*tag*⟩ } { ⟨*success code*⟩ } { ⟨*fail code*⟩ } §3; 17
tracklang.tex v1.3.9+

As \TrackLanguageTag but does ⟨*fail code*⟩ if the tag doesn't contain a valid language code. If successful, does ⟨*success code*⟩ after tracking the language.

\TrackLangAddExtraRegionFile{⟨*file*⟩} `tracklang.tex` v1.4+ §4; 26

Adds ⟨*file*⟩ to the list of extra region code files that should be input by `tracklang-region-codes.tex`.

\TrackLangAddExtraScriptFile{⟨*file*⟩} `tracklang.tex` v1.4+ §4; 27

Adds ⟨*file*⟩ to the list of files that should be input by `tracklang-scripts.tex`.

\TrackLangAddToCaptions{⟨*code*⟩} `tracklang.tex` v1.3+ §5; 47

A shortcut that just does `\TrackLangAddToHook`{⟨*code*⟩}{`captions`}.

\TrackLangAddToHook{⟨*code*⟩}{⟨*type*⟩} `tracklang.tex` v1.3+ §5; 46

For use within resource files, this can be used to add ⟨*code*⟩ to the appropriate hook.

\TrackLangAlphaIIIToNumericRegion{⟨*alpha-3 code*⟩}
`tracklang-region-codes.tex` v1.3+ §4; 25

Expands to the numeric code corresponding to the given alpha-3 code.

\TrackLangAlphaIIToNumericRegion{⟨*alpha-2 code*⟩}
`tracklang-region-codes.tex` v1.3+ §4; 25

Expands to the numeric code corresponding to the given alpha-2 code.

\TrackLangDeclareDialectOption{⟨*dialect*⟩}{⟨*root language*⟩}{⟨*3166-1 code*⟩}{⟨*modifier*⟩}{⟨*variant*⟩}{⟨*map*⟩}{⟨*script*⟩}
`tracklang.tex` v1.3+

Defines a predefined dialect label that can be used by `\TrackPredefinedDialect`.

\TrackLangDeclareLanguageOption { ⟨*language name*⟩ } { ⟨*639-1 code*⟩ } { ⟨*639-2 (T)*⟩ } { ⟨*639-2 (B)*⟩ } { ⟨*639-3*⟩ } { ⟨*3166-1*⟩ } { ⟨*default script*⟩ }
`tracklang.tex` v1.3+

Defines a new root language that's declared as an option.

\TrackLangEncodingName `tracklang.tex` v1.6.1+ §5; 45

Expands to `\inputencodingname` if it has been defined or `utf8` otherwise.

\TrackLangEnv (user defined) §3; 20

May be defined using the same format as `LC_ALL` before using `\TrackLangParseFromEnv` to skip the environment variable query.

\TrackLangEnvCodeSet `tracklang.tex` v1.3+ §3; 21

Set by `\TrackLangParseFromEnv` to the code-set.

\TrackLangEnvLang `tracklang.tex` v1.3+ §3; 21

Set by `\TrackLangParseFromEnv` to the language code.

\TrackLangEnvModifier `tracklang.tex` v1.3+ §3; 21

Set by `\TrackLangParseFromEnv` to the modifier.

\TrackLangEnvTerritory `tracklang.tex` v1.3+ §3; 21

Set by `\TrackLangParseFromEnv` to the territory.

\TrackLangFromEnv `tracklang.tex` v1.3+ §3; 18

Queries environment variable if `\TrackLangEnv` not already set, parses `\TrackLangEnv` if it has been set, and adds the dialect if it's recognised.

\TrackLangGetDefaultScript { ⟨*language*⟩ } `tracklang.tex` v1.3+ §5; 35

Expands to the default script for the given language.

\TrackLangGetKnownCountry { ⟨*language*⟩ } `tracklang.tex` v1.3+

Expands to the ISO 3166-1 country code for the given language.

\TrackLangGetKnownIsoThreeLetterLang { ⟨*language*⟩ }
`tracklang.tex` v1.3+

Expands to the ISO 639-2 language code associated with ⟨*language*⟩.

\TrackLangGetKnownIsoThreeLetterLangB { ⟨*language*⟩ }
`tracklang.tex` v1.3+

Expands to the ISO 639-2 (B) language code associated with ⟨*language*⟩.

\TrackLangGetKnownIsoTwoLetterLang { ⟨*language*⟩ }
`tracklang.tex` v1.3+

Expands to the ISO 639-1 language code associated with ⟨*language*⟩.

\TrackLangGetKnownLangFromIso { ⟨*ISO code*⟩ } `tracklang.tex` v1.3+

Expands to the root language label from the given ISO code (639-1 or 639-2 or 639-3).

\TrackLangIfAlphaNumericChar { ⟨*tag*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+

Does ⟨*true*⟩ if the argument is a single alphanumeric character otherwise does ⟨*false*⟩.

\TrackLangIfHasDefaultScript { ⟨*language*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+

Does ⟨*true*⟩ if the given language has a default script (but is not necessarily tracked), otherwise does ⟨*false*⟩.

\TrackLangIfHasKnownCountry { ⟨*language*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+

Does ⟨*true*⟩ if the given language has an ISO 3166-1 country code (but is not necessarily tracked), otherwise does ⟨*false*⟩.

\TrackLangIfKnownAlphaIIIRegion { ⟨*alpha-3 code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang-region-codes.tex` v1.3+

§4; 26

Expands to ⟨*true*⟩ if there's a known mapping for the given ⟨*alpha-3 code*⟩, otherwise expands to ⟨*false*⟩.

\TrackLangIfKnownAlphaIIRegion { ⟨*alpha-2 code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang-region-codes.tex` v1.3+

§4; 25

Expands to ⟨*true*⟩ if there's a known mapping for the given alpha-2 region code, otherwise expands to ⟨*false*⟩.

\TrackLangIfKnownIsoThreeLetterLang { ⟨*language*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+

Does ⟨*true*⟩ if ⟨*language*⟩ has an ISO 639-2 code (but is not necessarily tracked), otherwise does ⟨*false*⟩.

\TrackLangIfKnownIsoThreeLetterLangB { ⟨*language*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang.tex` v1.3+

Does ⟨*true*⟩ if ⟨*language*⟩ has an ISO 639-2 (B) code (but is not necessarily tracked), otherwise does ⟨*false*⟩.

\TrackLangIfKnownIsoTwoLetterLang{ ⟨*language*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang.tex` v1.3+

Does ⟨*true*⟩ if ⟨*language*⟩ has an ISO 639-1 code (but is not necessarily tracked), otherwise does ⟨*false*⟩.

\TrackLangIfKnownLang{ ⟨*language*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang.tex` v1.3+

Does ⟨*true*⟩ if ⟨*language*⟩ is known (but not necessarily tracked), otherwise does ⟨*false*⟩.

\TrackLangIfKnownLangFromIso{ ⟨*ISO code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang.tex` v1.3+

Does ⟨*true*⟩ if the given language code (639-1 or 639-2 or 639-3) is recognised (but not necessarily tracked), otherwise does ⟨*false*⟩.

\TrackLangIfKnownNumericRegion{ ⟨*numeric code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang-region-codes.tex` v1.3+

§4; 25

Expands to ⟨*true*⟩ if there's a known mapping for the given numeric region code, otherwise expands to ⟨*false*⟩.

\TrackLangIfLanguageTag{ ⟨*tag*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang.tex` v1.3+

Does ⟨*true*⟩ if the argument is a language tag otherwise does ⟨*false*⟩.

\TrackLangIfRegionTag{ ⟨*tag*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang.tex` v1.3+

Does ⟨*true*⟩ if the argument is a region tag otherwise does ⟨*false*⟩.

\TrackLangIfScriptTag{ ⟨*tag*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang.tex` v1.3+

Does ⟨*true*⟩ if the argument is a script tag otherwise does ⟨*false*⟩.

\TrackLangIfVariantTag{ ⟨*tag*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ } `tracklang.tex` v1.3+

Does ⟨*true*⟩ if the argument is a variant tag otherwise does ⟨*false*⟩.

\TrackLangLastTrackedDialect `tracklang.tex` v1.3+ §6.5; 67

Expands to the label of the last tracked dialect.

\TrackLangNewLanguage{ ⟨*language label*⟩ } { ⟨*639-1 code*⟩ } { ⟨*639-2 (T)*⟩ } { ⟨*639-2 (B)*⟩ } { ⟨*639-3*⟩ } { ⟨*3166-1*⟩ } { ⟨*default script*⟩ } `tracklang.tex` v1.3+ §6.5; 67

Identifies a new language that may be tracked. Apart from ⟨*language label*⟩, the other arguments may be empty if the information is unavailable.

\TrackLangNumericToAlphaIIIRegion{ ⟨*numeric code*⟩ }
`tracklang-region-codes.tex` v1.3+ §4; 25

Expands to the alpha-3 code corresponding to the given numeric code.

\TrackLangNumericToAlphaIIRegion{ ⟨*numeric code*⟩ }
`tracklang-region-codes.tex` v1.3+ §4; 25

Expands to the alpha-2 code corresponding to the given numeric code.

\TrackLangParseFromEnv `tracklang.tex` v1.3+ §3; 22

Attempts to obtain locale information from the expansion of `\TrackLangEnv`.

\TrackLangProvidePredefinedDialect{ ⟨*dialect label*⟩ } { ⟨*root language label*⟩ } { ⟨*3166-1 code*⟩ } { ⟨*modifier*⟩ } { ⟨*variant*⟩ } { ⟨*map*⟩ } { ⟨*script*⟩ }
`tracklang.tex` v1.4+ §6.6; 68

Defines a predefined dialect label that can be used by `\TrackPredefinedDialect`.

\TrackLangProvidePredefinedLanguage{ ⟨*language label*⟩ }
tracklang.tex v1.4+

§6.6; 67

Sets up a language label for use with \TrackPredefinedDialect.

\TrackLangProvidesResource{ ⟨*tag*⟩ } [⟨*version info*⟩]
tracklang.tex v1.3+

§5; 44

Analogous to \ProvidesFile.

\TrackLangQueryEnv tracklang.tex v1.3+

§3; 21

Attempts to obtain locale information from the LC_ALL environment variable via the shell escape or, with LuaTEX, \directlua.

\TrackLangQueryOtherEnv{ ⟨*env-name*⟩ } tracklang.tex v1.3+

§3; 22

Attempts to obtain locale information from the LC_ALL environment variable and then by the ⟨*env-name*⟩ environment variable via the shell escape or, with LuaTEX, \directlua.

\TrackLangRedefHook{ ⟨*code*⟩ } { ⟨*type*⟩ } tracklang.tex v1.4+

§5; 47

Similar to \TrackLangAddToHook but redefines the hook rather than appending to it.

\TrackLangRegionMap{ ⟨*numeric*⟩ } { ⟨*alpha-2*⟩ } { ⟨*alpha-3*⟩ }
tracklang-region-codes.tex v1.3+

§4; 26

Establishes a mapping between a numeric region code and alpha-2 and alpha-3 codes.

\TrackLangRequestResource{ ⟨*tag*⟩ } { ⟨*not found code*⟩ }
tracklang.tex v1.3+

§5; 45

As \TrackLangRequireResource but does ⟨*not found code*⟩ if the file doesn't exist.

\TrackLangRequireDialect [⟨*load code*⟩] { ⟨*pkgname*⟩ } { ⟨*dialect*⟩ }
tracklang.tex v1.3+ §5; 37

Loads the dialect for the given package.

\TrackLangRequireDialectPrefix tracklang.tex v1.3+ §5; 38

Defined by \TrackLangRequireDialect.

\TrackLangRequireResource { ⟨*tag*⟩ } tracklang.tex v1.3+ §5; 45

Loads the appropriate ldf file if it hasn't already been loaded.

\TrackLangRequireResourceOrDo { ⟨*tag*⟩ } { ⟨*code1*⟩ } { ⟨*code2*⟩ }
tracklang.tex v1.3+ §5; 45

As \TrackLangRequireResource but does ⟨*code1*⟩ if the file is now loaded or ⟨*code2*⟩ if the file has already been loaded.

\TrackLangScriptAlphaToDir { ⟨*alpha code*⟩ }
tracklang-scripts.tex v1.3+ §4; 27

Expands to the direction associated with the given alpha script code.

\TrackLangScriptAlphaToName { ⟨*alpha code*⟩ }
tracklang-scripts.tex v1.3+ §4; 27

Expands to the name associated with the given alpha script code.

\TrackLangScriptAlphaToNumeric { ⟨*alpha code*⟩ }
tracklang-scripts.tex v1.3+ §4; 27

Expands to the numeric script code corresponding to the given alpha code.

\TrackLangScript⟨*Code*⟩ `tracklang-scripts.tex` v1.3+ §4; 26

Set by `\TrackLangScriptMap` to the associated alpha code ⟨*Code*⟩.

\TrackLangScriptGetParent { ⟨*alpha code*⟩ }
`tracklang-scripts.tex` v1.3+ §4; 27

Expands to the parent of the given alpha script code.

\TrackLangScriptIfHasParent { ⟨*alpha code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang-scripts.tex` v1.3+ §4; 27

Expands to ⟨*true*⟩ if the given alpha script code has a parent otherwise expands to ⟨*false*⟩.

\TrackLangScriptIfKnownAlpha { ⟨*alpha code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang-scripts.tex` v1.3+ §4; 27

Expands to ⟨*true*⟩ if there's a known mapping for the given alpha script code otherwise expands to ⟨*false*⟩.

\TrackLangScriptIfKnownNumeric { ⟨*numeric code*⟩ } { ⟨*true*⟩ } { ⟨*false*⟩ }
`tracklang-scripts.tex` v1.3+ §4; 27

Expands to ⟨*true*⟩ if there's a known mapping for the given numeric script code otherwise expands to ⟨*false*⟩.

\TrackLangScriptMap { ⟨*letter code*⟩ } { ⟨*numeric code*⟩ } { ⟨*script name*⟩ } { ⟨*direction*⟩ } { ⟨*parent script*⟩ } `tracklang-scripts.tex` v1.3+ §4; 26

Defines a mapping between an alpha code and a numeric code.

\TrackLangScriptNumericToAlpha { ⟨*numeric code*⟩ }
`tracklang-scripts.tex` v1.3+ §4; 27

Expands to the alpha script code corresponding to the given numeric code.

\TrackLangScriptSetParent { ⟨*alpha code*⟩ } { ⟨*parent alpha code*⟩ }
tracklang-scripts.tex v1.3+ §4; 27

Sets the parent for the given alpha script code.

\TrackLangShowWarningsfalse tracklang.tex v1.3+ §3; 19

Sets \ifTrackLangShowWarnings to false.

\TrackLangShowWarningstrue tracklang.tex v1.3+ §3; 19

Sets \ifTrackLangShowWarnings to true.

\TrackLanguageTag { ⟨*tag*⟩ } tracklang.tex v1.3+ §3; 16

Parse ⟨*tag*⟩, which should be a regular, well-formed RFC 5646 language tag (not an irregular grandfather tag) and track the dialect. Note that the tag must start with a language identifier and can't simply be a region code.

\TrackLocale { ⟨*locale*⟩ } tracklang.tex v1.3+ §3; 16

Tracks the dialect identified by the given ⟨*locale*⟩, which may either be a predefined language/dialect or in the same format as \TrackLangEnv.

\TrackPredefinedDialect { ⟨*dialect label*⟩ } tracklang.tex v1.0+ §3; 15

Tracks a predefined language or dialect.

\TwoLetterIsoCountryCode tracklang.tex v1.0+ §5; 33

Expands to 3166-1 (should not be redefined).

\TwoLetterIsoLanguageCode tracklang.tex v1.0+ §5; 33

Expands to 639-1 (should not be redefined).

Index

D

E

F

G

H

I

Q

R

S

T

U

V

W

X

Y

Z