

NAME

Font::TTF::Ttopen – OpenType superclass for standard OpenType lookup based tables (GSUB and GPOS)

DESCRIPTION

Handles all the script, lang, feature, lookup stuff for a Font::TTF::Gsub/Font::TTF::Gpos table leaving the class specifics to the subclass

INSTANCE VARIABLES

The instance variables of an opentype table form a complex sub-module hierarchy.

Version

This contains the version of the table as a floating point number

SCRIPTS

The scripts list is a hash of script tags. Each script tag (of the form `$t->{'SCRIPTS'}{$tag}`) has information below it.

OFFSET This variable is preceded by a space and gives the offset from the start of the table (not the table section) to the script table for this script

REFTAG This variable is preceded by a space and gives a corresponding script tag to this one such that the offsets in the file are the same. When writing, it is up to the caller to ensure that the REFTAGs are set correctly, since these will be used to assume that the scripts are identical. Note that REFTAG must refer to a script which has no REFTAG of its own.

DEFAULT

This corresponds to the default language for this script, if there is one, and contains the same information as an itemised language

LANG_TAGS

This contains an array of language tag strings (each 4 bytes) corresponding to the languages listed by this script

`$lang` Each language is a hash containing its information:

OFFSET This variable is preceded by a space and gives the offset from the start of the whole table to the language table for this language

REFTAG This variable is preceded by a space and has the same function as for the script REFTAG, only for the languages within a script.

RE-ORDER This indicates re-ordering information, and has not been set. The value should always be 0.

DEFAULT This holds the index of the default feature, if there is one, or -1 otherwise.

FEATURES This is an array of feature tags for all the features enabled for this language

FEATURES

The features section of instance variables corresponds to the feature table in the opentype table.

FEAT_TAGS

This array gives the ordered list of feature tags for this table. It is used during reading and writing for converting between feature index and feature tag.

The rest of the FEATURES variable is itself a hash based on the feature tag for each feature. Each feature has the following structure:

OFFSET This attribute is preceded by a space and gives the offset relative to the start of the whole table of this particular feature.

PARMS If FeatureParams are defined for this feature, this contains a reference to the corresponding FeatureParams object. Otherwise set to null.

LOOKUPS

This is an array containing indices to lookups in the LOOKUP instance variable of the table

INDEX This gives the feature index for this feature and is used during reading and writing for converting between feature tag and feature index.

LOOKUP

This variable is an array of lookups in order and is indexed via the features of a language of a script. Each lookup contains subtables and other information:

OFFSET This name is preceded by a space and contains the offset from the start of the table to this particular lookup

TYPE This is a subclass specific type for a lookup. It stipulates the type of lookup and hence subtables within the lookup

FLAG Holds the lookup flag bits

FILTER Holds the MarkFilteringSet (that is, the index into GDEF->MARKSETS) for the lookup.

SUB This holds an array of subtables which are subclass specific. Each subtable must have an **OFFSET**. The other variables described here are an abstraction used in both the **GSUB** and **GPOS** tables which are the target subclasses of this class.

OFFSET This is preceded by a space and gives the offset relative to the start of the table for this subtable

FORMAT Gives the sub-table sub format for this **GSUB** subtable. It is assumed that this value is correct when it comes time to write the subtable.

COVERAGE Most lookups consist of a coverage table corresponding to the first glyph to match. The offset of this coverage table is stored here and the coverage table looked up against the **GSUB** table proper. There are two lookups without this initial coverage table which is used to index into the **RULES** array. These lookups have one element in the **RULES** array which is used for the whole match.

RULES The rules are a complex array. In most cases, each element of the array corresponds to an element in the coverage table (governed by the coverage index). In a few caess, such as when there is no coverage table, then there is considered to be only one element in the rules array. Each element of the array is itself an array corresponding to the possibly multiple string matches which may follow the initial glyph. Each element of this array is a hash with fixed keys corresponding to information needed to match a glyph string or act upon it. Thus the **RULES** element is an array of arrays of hashes which contain the following keys:

MATCH This contains a sequence of elements held as an array. The elements may be glyph ids (**gid**), class ids (**cids**), or offsets to coverage tables. Each element corresponds to one glyph in the glyph string. See **MATCH_TYPE** for details of how the different element types are marked.

PRE This array holds the sequence of elements preceding the first match element and has the same form as the **MATCH** array.

POST This array holds the sequence of elements to be tested for following the match string and is of the same form as the **MATCH** array.

ACTION This array holds information regarding what should be done if a match is found. The array may either hold

glyph ids (which are used to replace or insert or whatever glyphs in the glyph string) or 2 element arrays consisting of:

OFFSET	Offset from the start of the matched string that the lookup should start at when processing the substring.
LOOKUP_INDEX	The index to a lookup to be acted upon on the match string.
CLASS	For those lookups which use class categories rather than glyph ids for matching this is the offset to the class definition used to categories glyphs in the match string.
PRE_CLASS	This is the offset to the class definition for the before match glyphs
POST_CLASS	This is the offset to the class definition for the after match glyphs.
ACTION_TYPE	This string holds the type of information held in the ACTION variable of a RULE. It is subclass specific.
MATCH_TYPE	This holds the type of information in the MATCH array of a RULE. This is subclass specific.
ADJUST	This corresponds to a single action for all items in a coverage table. The meaning is subclass specific.
CACHE	This key starts with a space A hash of other tables (such as coverage tables, classes, anchors, device tables) based on the offset given in the subtable to that other information. Note that the documentation is particularly unhelpful here in that such tables are given as offsets relative to the beginning of the subtable not the whole GSUB table. This includes those items which are stored relative to another base within the subtable.

METHODS

`$t->read`

Reads the table passing control to the subclass to handle the subtable specifics

`$t->read_sub($fh, $lookup, $index)`

This stub is to allow subclasses to read subtables of lookups in a table specific manner. A reference to the lookup is passed in along with the subtable index. The file is located at the start of the subtable to be read

`$t->extension()`

Returns the lookup number for the extension table that allows access to 32-bit offsets.

`$t->out($fh)`

Writes this OpenType table to the output calling `$t->out_sub` for each sub table at the appropriate point in the output. The assumption is that on entry the number of scripts, languages, features, lookups, etc. are all resolved and the relationships fixed. This includes a LANG_TAGS list for a script, and that all scripts and languages in their respective dictionaries either have a REFTAG or contain real data.

`$t->num_sub($lookup)`

Asks the subclass to count the number of subtables for a particular lookup and to return that value. Used in `out()`.

`$t->out_sub($fh, $lookup, $index)`

This stub is to allow subclasses to output subtables of lookups in a table specific manner. A reference to the lookup is passed in along with the subtable index. The file is located at the start of the subtable to be output

\$t->dirty

Setting GPOS or GSUB dirty means that OS/2 may need updating, so set it dirty.

\$t->maxContext

Returns the length of the longest opentype rule in this table.

\$t->update

Perform various housekeeping items:

For all lookups, set/clear 0x0010 bit of flag words based on 'FILTER' value.

Sort COVERAGE table and RULES for all lookups.

Unless `$t->{' PARENT'}` is true, update will make sure that GPOS and GSUB include the same scripts and languages. Any added scripts and languages will have empty feature sets.

Internal Functions & Methods

Most of these methods are used by subclasses for handling such things as coverage tables.

copy(\$ref)

Internal function to copy the top level of a dictionary to create a new dictionary. Only the top level is copied.

\$t->read_cover(\$cover_offset, \$lookup_loc, \$lookup, \$fh, \$is_cover)

Reads a coverage table and stores the results in `$lookup->{' CACHE'}`, that is, if it has not been read already.

ref_cache(\$obj, \$cache, \$offset [, \$template])

Internal function to keep track of the local positioning of subobjects such as coverage and class definition tables, and their offsets. What happens is that the cache is a hash of sub objects indexed by the reference (using a string mashing of the reference name which is valid for the duration of the reference) and holds a list of locations in the output string which should be filled in with the offset to the sub object when the final string is output in `out_final`.

Uses tricks for `Tie::Rehash`

out_final(\$fh, \$out, \$cache_list, \$state)

Internal function to actually output everything to the file handle given that now we know the offset to the first sub object to be output and which sub objects are to be output and what locations need to be updated, we can now generate everything. `$cache_list` is an array of two element arrays. The first element is a cache object, the second is an offset to be subtracted from each reference to that object made in the cache.

If `$state` is 1, then the output is not sent to the filehandle and the return value is the string to be output. If `$state` is absent or 0 then output is not limited by storing in a string first and the return value is "";

\$self->read_context(\$lookup, \$fh, \$type, \$fmt, \$cover, \$count, \$loc)

Internal method to read context (simple and chaining context) lookup subtables for the GSUB and GPOS table types. The assumed values for `$type` correspond to those for GSUB, so GPOS should adjust the values upon calling.

\$self->out_context(\$lookup, \$fh, \$type, \$fmt, \$ctables, \$out, \$num)

Provides shared behaviour between GSUB and GPOS tables during output for context (chained and simple) rules. In addition, support is provided here for type 4 GSUB tables, which are not used in GPOS. The value for `$type` corresponds to the type in a GSUB table so calling from GPOS should adjust the value accordingly.

BUGS

- No way to share cachable items (coverage tables, classes, anchors, device tables) across different lookups. The items are always output after the lookup and repeated if necessary. Within lookup sharing is possible.

AUTHOR

Martin Hosken <<http://scripts.sil.org/FontUtils>>.

LICENSING

Copyright (c) 1998–2016, SIL International (<http://www.sil.org>)

This module is released under the terms of the Artistic License 2.0. For details, see the full text of the license in the file LICENSE.