amfbrowser implementation details

SCHORNACK LAB

Abstract

The standalone interface `amfbrowser` overlays root images and their corresponding annotations. It facilitates browsing and curation of `amf` predictions and computes statistics about fungal root colonisation and occurrence of intraradical hyphal structures.

Contents

Prepared with ocamldoc and LaTeX

1 Module AmfAnnot : Tile annotations.

```
class type annot = object..end
```

Contents:

```
method is_empty : ?level:AmfLevel.level → unit → bool
```

Returns `true` when the tile has no annotation.

```
method has_annot : ?level:AmfLevel.level → unit → bool
```

Indicates whether the tile has annotation.

```
method get : ?level:AmfLevel.level → unit → Morelib.CSet.t
```

Returns the annotations associated with the given level, or the current level if `?level` is `None`.

```
method add : ?level:AmfLevel.level → char → unit
```

`add chr` adds `chr` to the tile annotation. Only valid when annotating intraradical structures.

```
method remove : ?level:AmfLevel.level → char → unit
```

`remove chr` removes `chr` from the tile annotation.

```
method mem : char → bool
```

`mem chr` returns `true` when `chr` is part of the tile annotation.

```
method all : string
```

Returns all assigned annotations, irrespective of annotation level.

```
method hot : ?level:AmfLevel.level → unit → int list
```

Returns a one-hot vector containing tile annotations at a given level. Uses current level when `?level` is `None`.

```
method editable : bool
```

Indicates whether the given tile annotation is editable. A tile is always editable at the root segmentation level. Otherwise, they have to carry the annotation flag `'Y'`.

```
method erase : ?level:AmfLevel.level → unit → unit
```

Removes all annotations at the given level.

```
val create : unit → annot
```

Creates an empty annotation.

```
val of_string : string → annot
```

Creates an annotation and initializes it with the given string.

2 Module `AmfCallback` : Event manager.

```
type image_ref = AmfImage.image option Stdlib.ref
```

```
module Magnifier : sig..end
```

Contents:

```
val capture_screenshot : AmfCallback.image_ref → unit
```

Captures a snapshot of the magnified view.

Magnifier.

```
module Predictions : sig..end
```

Contents:

```
val update_list : AmfCallback.image_ref → unit
```

```
val update_cam : AmfCallback.image_ref → unit
```

```
val convert : AmfCallback.image_ref → unit
```

```
val select_list_item : AmfCallback.image_ref → unit
```

```
val move_to_ambiguous_tile : AmfCallback.image_ref → unit
```

Predictions.

```
module Window : sig..end
```

Contents:

```
val cursor : AmfCallback.image_ref → unit
```

Arrow keys move cursor.

```
val annotate : AmfCallback.image_ref → unit
```

Letter change annotations.

```
val save : AmfCallback.image_ref → unit
```

Saves current image.

Window events.

```
module DrawingArea : sig..end
```

Contents:

```
val cursor : AmfCallback.image_ref → unit
```

Updates the current tile after a mouse click.

```
val annotate : AmfCallback.image_ref → unit
```

Updates the interface to reflect current tile change.

```
val repaint : AmfCallback.image_ref → unit
```

Redraws the current mosaic when the active layer changes.

```
val repaint_and_count : AmfCallback.image_ref → unit
```

Same as `repaint`, but also computes statistics.

Events affecting the current tile displayed in the drawing area.

```
module ToggleBar : sig..end
```

Contents:

```
val annotate : AmfCallback.image_ref → unit
```

```
module Toolbox : sig..end
```

Contents:

```
val snap : AmfCallback.image_ref → unit
```

Take a snapshot of the current window.

```
val export : AmfCallback.image_ref → unit
```

Copy statistics to clipboard.

Misc tools, and application settings.

3 Module `AmfColor` : Color management.

```
type red = float
```

```
type blue = float
```

```
type green = float
```

```
type alpha = float
```

```
val opacity : alpha
```

Alpha channel value used for annotations and predictions.

```
val parse_rgb : string → red * green * blue
```

`parse_rgb "#rrggbbaa"` returns normalized floating-point values corresponding to the red, green, and blue components of the input color.

```
val parse_rgba : string → red * green * blue * alpha
```

`parse_rgba "#rrggbbaa"` returns normalized floating-point values corresponding to the red, green, blue, and alpha components of the input color.

```
val parse_desaturate : string → red * green * blue * alpha
```

`parse_desaturate "#rrggbbaa"` returns normalized floating-point values corresponding to the closest grayscale value to color `"#rrggbbaa"`. Conversion relies on the standard luminosity formula (i.e. `0.30 * r + 0.59 * g + 0.11 * b`).

4 Module `AmfConst` : AMFinder constants.

4.1 Grid settings

```
val _EDGE_ : int
```

Edge of a tile when displayed in the interface, in pixels.

```
val _WMAX_ : int
```

Number of tiles to be displayed on the X axis.

```
val _HMAX_ : int
```

Number of tiles to be displayed on the Y axis.

```
val _WMAT_ : int
```

Width of the displayed tile matrix, in pixels.

```
val _HMAT_ : int
```

Height of the displayed tile matrix, in pixels.

```
val _BGCOLOR_ : string
```

Background color used in the GtkDrawingArea.

4.2 Magnified view

```
val _MAGN_ : int
```

Edge (in pixels) of the tiles displayed in the magnified view.

```
val _BLANK_ : GdkPixbuf.pixbuf
```

Blank tile to be displayed in the magnified view where appropriate.

4.3 Icon size

```
val _LARGE_ : int
```

Edge of a large icon, in pixels.

```
val _SMALL_ : int
```

Edge of a small icon, in pixels.

5 Module AmfIcon : Interface icons.

Icons are stored as PNG files and loaded as GdkPixbuf objects at startup. Some of them have two states (type `style`), while others also occur in small and large sizes (type `size`).

```
type size =
  [ `LARGE
  | `SMALL ]
```

Icon size.

```
type style =
  [ `GRAY
  | `RGB ]
```

Color mode.

```
type id =
  [ `APP
  | `ATTACH
  | `CAM of style
  | `CLASS of char * size * style
  | `CONVERT
  | `DETACH
  | `EXPORT
  | `LOW_QUALITY
  | `PALETTE
  | `SETTINGS
  | `SNAP ]
```

Custom amfbrowser icons.

```
val get : id → GdkPixbuf.pixbuf
```

`get ico` returns the GdkPixbuf associated with icon `ico`.

6 Module AmfImage : Image management.

```
class type image = object..end
```

Contents:

```
method ui : ImgUI.cls
```

User interface.

```
method file : ImgFile.cls
```

File settings.

```
method source : ImgSource.cls
```

Source image settings.

```
method brush : ImgBrush.cls
```

Drawing toolbox.

```
method draw : ImgDraw.cls
```

Drawing functions.

```
method cursor : ImgCursor.cls
```

Cursor position manager.

```
method pointer : ImgPointer.cls
```

Mouse pointer tracker.

```
method small_tiles : ImgTileMatrix.cls
```

Small-sized tiles to be used in the right pane.

```
method large_tiles : ImgTileMatrix.cls
```

Large-sized tiles to be used in the left pane.

```
method annotations : ImgAnnotations.cls
```

User-defined annotations.

```
method predictions : ImgPredictions.cls
```

Computer-generated predictions (probabilities).

```
method predictions_to_annotations : ?erase:bool → unit → unit
```

Converts predictions to annotations.

```
method show_predictions : unit → unit
```

Displays the active prediction layer.

```
method show : unit → unit
```

Displays background, tiles as well as the activate annotation layer.

```
method uncertain_tile : unit → unit
```

Draw the next uncertain tile image.

```
method mosaic : ?sync:bool → unit → unit
```

Displays the tile matrix.

```
method magnified_view : unit → unit
```

Updates the magnified view.

```
method update_statistics : unit → unit
```

Updates statistics (for layers).

```
method at_exit : (unit → unit) → unit
```

Registers a function to be run before the image gets destroyed.

```
method save : unit → unit
```

Saves the current image.

```
method screenshot : unit → unit
```

Saves a screenshot of the current magnified view.

```
val create : string → image
```

`load ~edge:n path` loads image `path`, using `n`x`n` squares as tiles.
Raises `Invalid_argument` if the file does not exist.

7 Module AmfLang : English translation.

```
val en_stage_A : string
```

```
val en_stage_A_label : string
```

```
val en_stage_B : string
```

```
val en_stage_B_label : string
```

```
val en_layer : string
```

```
val en_overlay : string
```

```
val en_predictions : string
```

```
val en_attach : string
```

```
val en_detach : string
```

8 Module `AmfLevel` : Annotation levels.

The first level corresponds to root segmentation into colonized versus non-colonized areas, and background. The second levels aims to predict AM fungal structures (arbuscules, vesicles, intraradical hyphae, and hyphopodia) within colonized root areas.

```
type level = bool
```

The type alias for annotation levels. Value `true` corresponds to root segmentation, while `false` is used for mycorrhizal structures.

```
val col : level
```

Root segmentation into colonized versus non-colonized areas, and background.

```
val myc : level
```

Mycorrhizal structures (arbuscules, vesicles, hyphopodia, intraradicular hyphae).

```
val all : level list
```

List of available annotation levels, sorted from lowest to highest.

```
val is_col : level → bool
```

Indicates whether the given level is root segmentation.

```
val is_myc : level → bool
```

Indicates whether the given level is intraradical structures.

```
val to_string : level → string
```

`to_string t` returns the textual representation of the level `t`.

```
val of_string : string → level
```

`of_string s` returns the level corresponding to the string `s`.

```
val to_header : level → char list
```

`to_header t` returns the header associated with annotation level `t`.

```
val of_header : char list → level
```

`of_header t` returns the annotation level associated with header `t`.

```
val chars : level → Morelib.CSet.t
```

Returns the string set containing all available chars at a given level.

```
val char_index : level → char → int
```

Index of the given char at the given level.

```
val all_chars_list : char list
```

All available annotations.

```
val lowest : level
```

Least detailed level of mycorrhiza annotation.

```
val highest : level
```

Most detailed level of mycorrhiza annotation.

```
val others : level → level list
```

`other t` returns the list of all annotations levels but `t`.

```
val colors : level → string list
```

`colors t` returns the list of RGB colors to use to display annotations at level `t`.

```
val tip : char → string
```

Keyboard shortcuts for the given annotation class.

```
val symbols : level → string list
```

Short symbols for annotation legend.

```
val icon_text : level → (char * string) list
```

Short symbols for icons.

```
module type ANNOTATION_RULES = sig..end
```

Contents:

```
val add_add : char → Morelib.CSet.t
```

Returns the annotations to add when a given annotation is added.

```
val add_rem : char → Morelib.CSet.t
```

Returns the annotations to remove when a given annotation is added.

```
val rem_add : char → Morelib.CSet.t
```

Returns the annotations to add when a given annotation is removed.

```
val rem_rem : char → Morelib.CSet.t
```

Returns the annotations to remove when a given annotation is removed.

Annotation rules.

```
val rules : level → (module AmfLevel.ANNOTATION_RULES)
```

Returns the rules associated with a given annotation level.

9 Module AmfLog : Logging functions.

```
val info : (α, unit, string, string, string, unit) Stdlib.format6
    → α
```

Prints a piece of information to stdout.

```
val info_debug :
  (α, unit, string, string, string, unit) Stdlib.format6 → α
```

Same, but only prints when _DEBUG_ is set to **true**.

```
val warning : (α, unit, string, string, string, unit) Stdlib.
   format6 → α
```

Prints a warning message to stderr.

```
val error :
  ?code:int → (α, unit, string, string, string, β) Stdlib.format6
      → α
```

Prints an error message to stderr and terminates with the given code.

```
val usage : unit → α
```

Specialized error function with application message.

10 Module AmfMemoize

Memoization.

```
val create : (α → β) → α → β
```

`create f` returns a memoized version of function `f`.

10.1 Memoized Cairo functions

```
val cursor : AmfSurface.pixels → Cairo.Surface.t
```

Square with a thick red stroke.

```
val dashed_square : AmfSurface.pixels → Cairo.Surface.t
```

Rounded square with a dashed stroke.

```
val locked_square : AmfSurface.pixels → Cairo.Surface.t
```

Rounded square with a lock.

```
val empty_square : AmfSurface.pixels → Cairo.Surface.t
```

Empty square.

```
val palette : int → AmfSurface.pixels → Cairo.Surface.t
```

Circle filled with color from a given palette.

```
val layer : AmfLevel.level → char → AmfSurface.pixels → Cairo.
  Surface.t
```

Rounded square with a symbol and filled with a specific color.

11 Module `AmfPar` : Command-line settings.

```
val edge : int Stdlib.ref
```

Edge size, in pixels.

```
val path : string option Stdlib.ref
```

```
val threshold : float Stdlib.ref
```

Annotation probability threshold.

```
val verbose : unit → bool
```

`verbose ()` indicates whether the application runs in verbose mode. It is recommended to switch on this mode for debugging.

```
val initialize : unit → unit
```

`initialize ()` reads command-line arguments.

12 Module `AmfSurface`

Cairo surfaces for drawing.

```
type pixels = int
```

```
type color = string
```

```
val rectangle :
  ?rgba:color →
  w:pixels → h:pixels → Cairo.context * Cairo.Surface.t
```

`rectangle ?rgba:c ~w ~h` draws a rectangle with a width and height of `w` and `h` pixels, respectively, filled with color `c`.

```
val square : ?rgba:color →
  pixels → Cairo.context * Cairo.Surface.t
```

`square ?rgba:c ~e` draws a square with an pixels of `e` pixels, filled with color `c`.

```
module Arrowhead : sig..end
```

Contents:

```
val top :
  ?bgcolor:AmfSurface.color →
  ?fgcolor:AmfSurface.color → AmfSurface.pixels → Cairo.Surface.t
```

Draw an arrowhead toward the top.

```
val bottom :
  ?bgcolor:AmfSurface.color →
  ?fgcolor:AmfSurface.color → AmfSurface.pixels → Cairo.Surface.t
```

Draw an arrowhead toward the bottom.

```
val left :
  ?bgcolor:AmfSurface.color →
  ?fgcolor:AmfSurface.color → AmfSurface.pixels → Cairo.Surface.t
```

Draw an arrowhead to the left.

```
val right :
  ?bgcolor:AmfSurface.color →
  ?fgcolor:AmfSurface.color → AmfSurface.pixels → Cairo.Surface.t
```

Draw an arrowhead toward the right.

Arrowheads.

module Annotation : sig..end

Contents:

```
val cursor : AmfSurface.pixels → Cairo.Surface.t
```

Square with a thick stroke.

```
val filled :
  ?symbol:string →
  ?force_symbol:bool →
  ?base_font_size:float →
  ?grayscale:bool → AmfSurface.color → AmfSurface.pixels → Cairo.Surface.t
```

Rounded, filled square with a centered symbol.

```
val colors :
  ?symbol:string →
  ?symbol_color:string →
  ?base_font_size:float →
  AmfSurface.color list → AmfSurface.pixels → Cairo.Surface.t
```

Rounded square filled with multiple colors.

```
val dashed : AmfSurface.pixels → Cairo.Surface.t
```

Square with a dashed stroke and an eye symbol.

```
val locked : AmfSurface.pixels → Cairo.Surface.t
```

Square with a dashed stroke and a red lock.

```
val empty : AmfSurface.color → AmfSurface.pixels → Cairo.Surface.t
```

Rounded square with a solid stroke and a white background.

Square tile drawings corresponding to annotations.

```
module Prediction : sig..end
```

Contents:

```
val filled :
  ?margin:float → AmfSurface.color → AmfSurface.pixels → Cairo.Surface.t
```

Circle.

```
val pie_chart :
  ?margin:float →
  float list → AmfSurface.color list → AmfSurface.pixels → Cairo.Surface.t
```

Pie chart for root segmentation.

```
val radar :
  ?margin:float →
  float list → AmfSurface.color list → AmfSurface.pixels → Cairo.Surface.t
```

Same, but for prediction of intraradical structures.

Circular tile drawings corresponding to predictions.

```
module Legend : sig..end
```

Contents:

```
val palette : ?step:int → AmfSurface.pixels → Cairo.Surface.t
```

Prediction values (continuous color palette). Colors are retrieved from `AmfUI.Predictions.get_colors ()`.

```
val classes : unit → Cairo.Surface.t
```

Annotation classes (discrete colors).

Legends.

13 Module AmfUI : Graphical user interface.

```
val window : GWindow.window
```

Application main window.

```
val status_icon : GMisc.status_icon
```

Status icon to be displayed on the system tray.

```
module Levels : sig..end
```

Contents:

UILevels.S

Annotation types.

```
module Toggles : sig..end
```

Contents:

UIToggleBar.S

The horizontal toolbox (left pane) contains the toggle buttons used to set mycorrhizal annotations. Users can switch between three sets of toggle buttons which correspond to basic (`COLONIZATION), intermediate (`ARB_VESICLES) and full (`ALL_FEATURES) annotation modes. Only one set is active at a given time.

```
module Magnifier : sig..end
```

Contents:

UIMagnifier.S

Magnified view of the cursor area.

```
module Drawing : sig..end
```

Contents:

UIDrawing.S

Whole image (right pane).

```
module Layers : sig..end
```

Contents:

UILayers.S

Annotation layers.

```
module Predictions : sig..end
```

Contents:

UIPredictions.S

Prediction manager.

```
module FileChooser : sig..end
```

Contents:

UIFileChooser.S

File chooser dialog to allow for selection of the JPEG/TIFF image to open within the CastAnet editor. This dialog shows at startup (unless an image is provided on the command line) as well as when the main window is closed. Users can therefore load successive images without closing the application, but only one image is active at a time.

```
module Tools : sig..end
```

Contents:

UITools.S

Misc tools such as snapshot or export, and application settings.

14 Module Err

```
val out_of_bounds : int
```

```
val invalid_argument : int
```

```
module Level : sig..end
```

Contents:

> ```
> val unknown_level : int
> ```
>
> ```
> val invalid_level_header : int
> ```

```
module Annot : sig..end
```

Contents:

> ```
> val invalid_character : int
> ```

```
module Image : sig..end
```

Contents:

> ```
> val unknown_annotation_file : int
> ```

15 Module ImgActivations : Class activation maps (CAMs).

```
class type cls = object..end
```

Contents:

```
method active : bool
```

Indicates whether class activation maps are to be displayed.

```
method get : string → char → r:int → c:int → GdkPixbuf.pixbuf option
```

Return the CAM associated with a given tile.

```
method dump : Zip.out_file → unit
```

Saves activations.

```
val create : ?zip:Zip.in_file → ImgSource.cls → cls
```

Builder.

16 Module ImgAnnotations : Annotation manager.

```
class type cls = object..end
```

Contents:

```
method get : r:int → c:int → unit → AmfAnnot.annot
```

Returns the item at the given coordinates and annotation level.

```
method iter : (r:int → c:int → AmfAnnot.annot → unit) → unit
```

Iterates over items at the given coordinates and annotation level.

```
method iter_layer :
  char → (r:int → c:int → AmfAnnot.annot → unit) → unit
```

Iterates over items at the given coordinates and annotation level.

```
method statistics : ?level:AmfLevel.level → unit → (char * int) list
```

Returns the current statistics.

```
method has_annot : ?level:AmfLevel.level → r:int → c:int → unit → bool
```

Indicates whether the given tile has annotation. By default, checks the current annotation layer.

```
method dump : Zip.out_file → unit
```

Saves annotations.

```
val create : ?zip:Zip.in_file → ImgSource.cls → cls
```

Builder.

17 Module `ImgBrush` : Image drawing functions.

```
class type cls = object..end
```

Contents:

```
method edge : int
```

Returns tile size.

```
method x_origin : int
```

Returns drawing origin on X axis.

```
method y_origin : int
```

Returns drawing origin on Y axis.

```
method set_update : (unit → unit) → unit
```

Registers a function to update the overall view.

```
method has_unchanged_boundaries : r:int → c:int → unit → bool
```

Ensure the given tile is within the visible window.
Returns True when drawing has to be updated.

```
method r_range : int * int
```

Returns the range of visible rows.

```
method c_range : int * int
```

Returns the range of visible columns.

```
method backcolor : string
```

Returns image background color.

```
method set_backcolor : string → unit
```

Defines image background color.

```
method background : ?sync:bool → unit → unit
```

Draws a white background on the right image area.

```
method pixbuf : ?sync:bool → r:int → c:int → GdkPixbuf.pixbuf → unit
```

`pixbuf ?sync ~r ~c p` draws pixbuf `p` at row `r` and column `c`.

```
method empty : ?sync:bool → r:int → c:int → unit → unit
```

Draws an empty tile.

```
method surface : ?sync:bool → r:int → c:int → Cairo.Surface.t → unit
```

`surface ?sync ~r ~c s` draws surface `s` at row `r` and column `c`.

```
method locked_tile : ?sync:bool → r:int → c:int → unit → unit
```

Specialized method for locked tiles.

```
method cursor : ?sync:bool → r:int → c:int → unit → unit
```

Draws the cursor.

```
method annotation :
  ?sync:bool → r:int → c:int → AmfLevel.level → Morelib.CSet.t → unit
```

Draws a tile annotation.

```
method annotation_other_layer : ?sync:bool → r:int → c:int → unit → unit
```

Draw the frame of an annotation, for tiles that gets annotated but not in the active layer (to distinguish with non-annotated tiles).

```
method prediction :
  ?sync:bool → r:int → c:int → float list → char → unit
```

Draws a tile prediction.

```
method palette : ?sync:bool → unit → unit
```

Displays a full palette.

```
method classes : ?sync:bool → unit → unit
```

Displays the annotations.

```
method show_probability : ?sync:bool → float → unit
```

Displays the probability cursor.

```
method hide_probability : ?sync:bool → unit → unit
```

Hides probability.

```
method sync : string → unit → unit
```

Synchronize drawings between the back pixmap and the drawing area.

```
val create : ImgSource.cls → cls
```

Builder.

18 Module `ImgCursor` : Cursor management.

```
class type cls = object..end
```

Contents:

```
method get : int * int
```

Returns the current cursor position.

```
method at : r:int → c:int → bool
```

Indicates whether the cursor is at the given coordinates.

```
method key_press : GdkEvent.Key.t → bool
```

Monitors key press.

```
method mouse_click : GdkEvent.Button.t → bool
```

Monitors mouse click.

```
method set_erase : (?sync:bool → r:int → c:int → unit → unit) → unit
```

Sets the function used to repaint tiles below cursor.

```
method set_paint : (?sync:bool → r:int → c:int → unit → unit) → unit
```

Sets the function used to paint the cursor.

```
method update_cursor_pos : r:int → c:int → bool
```

Updates cursor position.

```
val create : ImgSource.cls → ImgBrush.cls → cls
```

Builder.

19 Module ImgDraw : High-level drawing functions.

```
class type cls = object..end
```

Contents:

```
method tile : ?sync:bool → r:int → c:int → unit → bool
```

Draw tile image at the given coordinates.

```
method cursor : ?sync:bool → r:int → c:int → unit → unit
```

Draw cursor at the given coordinates.

```
method overlay : ?sync:bool → r:int → c:int → unit → unit
```

Draw annotation/prediction at the given coordinates.

```
val create :
  ImgTileMatrix.cls →
  ImgBrush.cls →
  ImgCursor.cls → ImgAnnotations.cls → ImgPredictions.cls → cls
```

Builder.

20 Module `ImgFile` : File settings.

```
class type cls = object..end
```

Contents:

```
method path : string
```

Path to the input image.

```
method base : string
```

File base name.

```
method archive : string
```

Name of the output archive.

```
val create : string → cls
```

Builder.

21 Module `ImgPointer` : Mouse pointer events.

```
class type cls = object..end
```

Contents:

```
method get : (int * int) option
```

Returns the current pointer position, if any.

```
method at : r:int → c:int → bool
```

Tells whether the pointer is at a given coordinate.

```
method track : GdkEvent.Motion.t → bool
```

Tracks pointer position.

```
method leave : GdkEvent.Crossing.t → bool
```

Detects pointer leaving.

```
method set_erase : (?sync:bool → r:int → c:int → unit → unit) → unit
```

Sets the function used to repaint tiles below pointer.

```
method set_paint : (?sync:bool → r:int → c:int → unit → unit) → unit
```

Sets the function used to paint the pointer.

```
val create : ImgSource.cls → ImgBrush.cls → cls
```

Builder.

22 Module ImgPredictions : Prediction tables.

```
class type cls = object..end
```

Contents:

```
method ids : AmfLevel.level → string list
```

Returns the list of predictions at the given annotation level.

```
method current : string option
```

Return the identifier of the active prediction table.

```
method current_level : AmfLevel.level option
```

Returns the level of the active predictions.

```
method set_current : string option → unit
```

Define the active prediction table.

```
method active : bool
```

Tells whether predictions are being displayed.

```
method count : int
```

Returns the number of predictions in the current dataset.

```
method next_uncertain : (int * int) option
```

Returns the coordinates of the next most uncertain prediction.

```
method get : r:int → c:int → float list option
```

Return the probabilities at the given coordinates.

```
method max_layer : r:int → c:int → (char * float) option
```

Return the layer with maximum probability at the given coordinates.

```
method iter :

  [ `ALL of r:int → c:int → float list → unit
  | `MAX of r:int → c:int → char * float → unit ] → unit
```

Iterate over all predictions.

```
method iter_layer : char → (r:int → c:int → float → unit) → unit
```

Iterate over tiles which have their top prediction on a given layer.

```
method statistics : (char * int) list
```

Statistics, currently based on the top prediction.

```
method to_string : unit → string
```

Return the string representation of the active prediction table.

```
method exists : r:int → c:int → bool
```

Indicates whether the given tile has predictions.

```
method dump : Zip.out_file → unit
```

Saves predictions.

```
val create : ?zip:Zip.in_file → ImgSource.cls → cls
```

Builder.

23 Module ImgShared

Image-related shared functions.

```
val crop_pixbuf :
  src_x:int → src_y:int → edge:int → GdkPixbuf.pixbuf → GdkPixbuf
    .pixbuf
```

`copy_pixbuf ~src_x:x ~src_y:y ~edge:e pix` extracts from pixbuf `pix` the `e` × `e` tile with top left corner at `(x, y)`.

```
val resize_pixbuf :
  ?interp:GdkPixbuf.interpolation →
  int → GdkPixbuf.pixbuf → GdkPixbuf.pixbuf
```

`resize_pixbuf ?interp:i e pix` resizes pixbuf `pix` to a square of edge `e`, using interpolation method `i`. This function is intended for square input as it will otherwise modify aspect ratio.

24 Module ImgSource : Source image settings.

24.1 Image source

```
class type cls = object..end
```

Contents:

```
method width : int
```

Image width, in pixels.

```
method height : int
```

Image height, in pixels.

```
method edge : int
```

Tile size (in pixels) used to segment the source image.

```
method rows : int
```

Row count.

```
method columns : int
```

Column count.

```
method save_settings : Zip.out_file → unit
```

Saves settings.

```
val create : ?zip:Zip.in_file → GdkPixbuf.pixbuf → cls
```

Builder.

25 Module ImgTileMatrix : Tile matrices.

```
class type cls = object..end
```

Contents:

```
method get : r:int → c:int → GdkPixbuf.pixbuf option
```

Retrieves a specific tile.

```
method iter : (r:int → c:int → GdkPixbuf.pixbuf → unit) → unit
```

Iterates over tiles.

```
val create : GdkPixbuf.pixbuf → ImgSource.cls → int → cls
```

Builder.

26 Module ImgUI : Interaction with the user interface.

```
class type cls = object..end
```

Contents:

```
method set_paint : (unit → unit) → unit
```

Painting functions to update the current tile.

```
method set_refresh : (unit → unit) → unit
```

Sets the painting function to refresh the whole drawing area.

```
method update_toggles : unit → unit
```

Update annotations at the current cursor position.

```
method toggle :
  GButton.toggle_button → GMisc.image → char → GdkEvent.Button.t → bool
```

Update annotations when a toggle button is toggled.

```
method key_press : GdkEvent.Key.t → bool
```

Update annotations based on key press.

```
method mouse_click : GdkEvent.Button.t → bool
```

Update annotations based on mouse click.

```
val create : ImgCursor.cls → ImgAnnotations.cls → ImgPredictions.
   cls → cls
```

UI object generator.

27 Module `Morelib` : Lightweight extension of the OCaml standard library.

```
module CSet : sig..end
```

Contents:

```
Set.S with type elt = char
```

```
module StringSet : sig..end
```

Contents:

```
val union : string → string → string
```

`union s1 s2` returns a string containing one instance of all characters occurring in either `s1` or `s2`, sorted in alphabetical order.

```
val inter : string → string → string
```

`union s1 s2` returns a string containing one instance of all characters occurring in both `s1` or `s2`, sorted in alphabetical order.

```
val diff : string → string → string
```

`union s1 s2` returns a string containing one instance of all characters occurring in `s1` but not in `s2`, sorted in alphabetical order.

String sets.

```
module Matrix : sig..end
```

Contents:

```
type α t = α array array
```

The type of matrices.

```
val dim : α t → int * int
```

Returns the the number of rows and columns of the given matrix.

```
val get : α t → r:int → c:int → α
```

`get t ~r ~c` returns `t.(r).(c)`.
Raises `Invalid_argument` if the given indices are out of range.

```
val get_opt : α t → r:int → c:int → α option
```

Same as `get`, but returns `None` if indices are invalid.

```
val set : α t → r:int → c:int → α → unit
```

`set t ~r ~c x` stores `x` at row `r` and column `c` of matrix `t`.

```
val make : r:int → c:int → α → α t
```

`make ~r ~c x` returns a matrix containing `r` rows and `c` columns, with all elements initialized with `x`.

```
val init : r:int → c:int → (r:int → c:int → α) → α t
```

`init ~r ~c f` builds a matrix of `r` rows and `c` columns and initializes values using the function `f`.

```
val map : (α → β) → α t → β t
```

`map f m` builds a new matrix by applying `f` to all members of `m`.

```
val mapi : (r:int → c:int → α → β) → α t → β t
```

Same as `map`, but `f` receives row and column indexes as parameters.

```
val map2 : (α → β → 'c) →
  α t → β t → 'c t
```

Same as `map`, but iterates over two matrices in one go.

```
val iter : (α → unit) → α t → unit
```

`iter f m` applies `f` to all members of the matrix `m`.

```
val iteri : (r:int → c:int → α → unit) → α t → unit
```

Same as `iter`, but oasses row and column indexes as parameters.

```
val iter2 : (α → β → unit) → α t → β t → unit
```

Same as `iter`, but iterates over two matrices in one go.

```
val fold : (r:int → c:int → α → β → α) → α → β t → α
```

Fold function.

```
val to_string : cast:(α → string) → α t → string
```

Exports a matrix as string.

```
val to_string_rc : cast:(r:int → c:int → α → string) → α t → string
```

Same as `to_string`, but passes row and column indexes to `cast`.

```
val of_string : cast:(string → α) → string → α t
```

Imports a matrix from a string.

```
val of_string_rc : cast:(r:int → c:int → string → α) → string → α t
```

Same as `of_string`, but passes row and column indexes to `cast`.

```
val copy : ?dat:(α → α) → α t → α t
```

Returns a copy of a given matrix.

Matrix iterators.

```
module Text : sig..end
```

Contents:

```
val explode : string → char list
```

Converts a string into a list of characters.

```
val implode : char list → string
```

Converts a list of characters into a string.

Text operations.

```
module File : sig..end
```

Contents:

```
val read : ?binary:bool → ?trim:bool → string → string
```

Reads a file. Setting up option `binary` results in file being opened in binary mode (default: **`false`**). Option `trim` triggers trimming of leading and trailing spaces (default: **`true`**).

Operations on files.

```
module Memoize : sig..end
```

Contents:

```
val create : ?label:string → ?one:bool → (unit → α) → unit → α
```

Memoization function. The optional parameter `lbl` is used to report when data are being recomputed due to a previous call to `forget` (see below). The optional parameter `one` defines memoized values that are never recomputed, i.e. insensitive to `forget` invokation.

```
val forget : unit → unit
```

Triggers all memoized data to be recomputed when accessed.

Memoized values with possible reinitialization. Such values are computed only once within a given session. However, they can recomputed when a new session starts, for instance to take into account the modification of general settings that affect the computation of memoized values.

28 Module UIDrawing

```
module type PARAMS = sig..end
```

Contents:

```
val packing : GObj.widget → unit
```

```
module type S = sig..end
```

Contents:

```
val area : GMisc.drawing_area
```

Drawing area were the whole image is displayed.

```
val cairo : unit → Cairo.context
```

Cairo context used to overlay annotation information.

```
val pixmap : unit → GDraw.pixmap
```

Backing pixmap used to draw offscreen.

```
val width : unit → int
```

Width of the drawing area, in pixels.

```
val height : unit → int
```

Height of the drawing area, in pixels.

```
val synchronize : unit → unit
```

Synchronizes the backing `pixmap` with the foreground `area`.

```
module Make : sig..end
```

Contents:

```
functor (_ : PARAMS) →S
```

Generator.

29 Module UIFileChooser

```
module type PARAMS = sig..end
```

Contents:

```
val parent : GWindow.window
```

```
val title : string
```

```
val border_width : int
```

```
module type S = sig..end
```

Contents:

```
val jpeg : GFile.filter
```

File filter for JPEG images.

```
val tiff : GFile.filter
```

File filter for TIFF images.

```
val run : unit → string
```

Displays a dialog window to select the image to open (if not provided on the command line). The program terminates if no image is selected.

```
module Make : sig..end
```

Contents:

functor (_ : PARAMS) →S

Generator.

30 Module `UIHelper` : Helper functions for custom toolbars.

```
val separator : (GButton.tool_item_o → unit) → unit
```

Adds a separator (and some extra space).

```
val morespace : (GButton.tool_item_o → unit) → unit
```

Adds more space.

```
val label :
  ?vspace:bool → (GButton.tool_item_o → unit) → string → GMisc.
     label
```

Adds a Gtklabel.

```
val pango_small : string → string
```

Pango small text

```
val custom_tool_button :
  ?packing:(GButton.tool_item_o → unit) →
  AmfIcon.id → string → GButton.tool_button
```

Custom tool button with a GdkPixbuf icon.

31 Module UILayers

```
module type PARAMS = sig..end
```

Contents:

```
val packing : GObj.widget → unit
```

```
val remove : GObj.widget → unit
```

```
val current : unit → AmfLevel.level
```

```
val radios : (AmfLevel.level * GButton.radio_button) list
```

```
module type S = sig..end
```

Contents:

```
val current : unit → char
```

Indicates which layer is currently active.

```
val set_label : char → int → unit
```

Updates the counter of the given annotation.

```
val set_callback :
  (char → GButton.radio_tool_button → GMisc.label → GMisc.image → unit) →
  unit
```

Sets a callback function to call when a button is toggled. The callback function will be applied to all tool buttons.

```
module Make : sig..end
```

Contents:

functor (_ : PARAMS) →S

Generator.

32 Module `UILevels` : UI auxiliary module dealing with annotation levels.

```
module type PARAMS = sig..end
```

Contents:

```
val init_level : AmfLevel.level
```

```
val packing : GObj.widget → unit
```

```
module type S = sig..end
```

Contents:

```
val current : unit → AmfLevel.level
```

Returns the active annotation level.

```
val set_current : AmfLevel.level → unit
```

Changes the active annotation level.

```
val radios : (AmfLevel.level * GButton.radio_button) list
```

Radio buttons corresponding to the different annotation types.

```
val set_callback : (AmfLevel.level → GButton.radio_button → unit) → unit
```

Applies a callback function to all radio buttons.

```
module Make : sig..end
```

Contents:

`functor (_ : PARAMS) →S`

Generator.

33 Module `UIMagnifier` : GUI auxiliary module.

Implements a magnified view of the 3x3 tile square surrounding the cursor position.

```
module type PARAMS = sig..end
```

Contents:

```
val rows : int
```

Number of tiles to display on the Y axis.

```
val columns : int
```

Number of tiles to display on the X axis.

```
val tile_edge : int
```

Size, in pixels, of a magnified tile.

```
val window : GWindow.window
```

Main application window.

```
val packing : GObj.widget → unit
```

Packing function to bind the magnified view to the interface.

Module parameters.

```
module type S = sig..end
```

Contents:

```
val event_boxes : GBin.event_box Morelib.Matrix.t
```

Matrix of event boxes.

```
val tiles : GMisc.image Morelib.Matrix.t
```

Matrix of GtkImage widgets used to display a magnified view of the tile square surrounding the cursor position. Square size is determined by `rows` and `columns` (see `PARAMS`).

```
val set_pixbuf : r:int → c:int → GdkPixbuf.pixbuf → unit
```

set_pixbuf ~r ~c p displays pixbuf p at row r and column c. Both values must be greater or equal to zero and strictly lower than rows and columns, respectively (see PARAMS).

```
val screenshot : unit → GdkPixbuf.pixbuf
```

Takes a screenshot of the whole magnifier area.

Output module.

```
module Make : sig..end
```

Contents:

functor (_ : PARAMS) →S

Generator.

34 Module `UIPredictions` : GUI auxiliary module.

Loads color palettes.

```
module type PARAMS = sig..end
```

Contents:

```
val parent : GWindow.window
```

```
val packing : #GButton.tool_item_o → unit
```

```
val border_width : int
```

```
val tooltips : GData.tooltips
```

```
module type S = sig..end
```

Contents:

```
val palette : GButton.tool_button
```

Displays a small utility window to select a palette.

```
val get_colors : unit → string array
```

Returns the colors associated with the current palette.

```
val set_choices : string list → unit
```

Sets prediction list.

```
val get_active : unit → string option
```

Tells which prediction is active.

```
val overlay : GButton.toggle_tool_button
```

Button that allows to select the predictions to display.

```
val palette : GButton.tool_button
```

Color palette selector.

```
val set_palette_update : (unit → unit) → unit
```

Defines the update function to use when the palette changes.

```
val cams : GButton.toggle_tool_button
```

Indicates whether CAMs are to be displayed or not.

```
val convert : GButton.tool_button
```

Converts predictions to annotations.

```
val ambiguities : GButton.tool_button
```

Emphasizes ambiguities.

```
module Make : sig..end
```

Contents:

functor (_ : PARAMS) →S

Generator.

35 Module UITileSet

```
module type PARAMS = sig..end
```

Contents:

```
val parent : GWindow.window
```

```
val spacing : int
```

```
val border_width : int
```

```
val window_width : int
```

```
val window_height : int
```

```
val window_title : string
```

```
module type S = sig..end
```

Contents:

```
val add : r:int → c:int → ico:GdkPixbuf.pixbuf → GdkPixbuf.pixbuf → unit
```

`add r c pix` adds the tile `pix` using coordinates `(r, c)` as legend.

```
val run : unit →
  [ `DELETE_EVENT
  | `OK
  | `SAVE ]
```

Displays the dialog and returns the output flag.

```
val set_title : (α, unit, string, unit) Stdlib.format4 → α
```

Sets tile set title, using `printf`-like style.

```
val hide : unit → unit
```

Hides the tile set and clears all tiles.

```
module Make : sig..end
```

Contents:

functor (_ : PARAMS) →S

Generator.

36 Module UIToggleBar : UI auxiliary module dealing with annotation toggle buttons.

```
module type PARAMS = sig..end
```

Contents:

```
val packing : GObj.widget → unit
```

```
val remove : GObj.widget → unit
```

```
val current : unit → AmfLevel.level
```

```
val set_current : AmfLevel.level → unit
```

```
val radios : (AmfLevel.level * GButton.radio_button) list
```

```
val tooltips : GData.tooltips
```

```
module type S = sig..end
```

Contents:

```
val is_active : char → bool option
```

Indicate whether the given annotation is active at the current level.

```
val iter_all :
  (AmfLevel.level → char → GButton.toggle_button → GMisc.image → unit) →
  unit
```

Iterate over all toggle buttons.

```
val iter_current :
  (char → GButton.toggle_button → GMisc.image → unit) → unit
```

Iterate over toggle buttons at the current level.

```
module Make : sig..end
```

Contents:

functor (_ : PARAMS) →S

Generator.

37 Module UITools : Toolbox

```
module type PARAMS = sig..end
```

Contents:

```
val packing : GObj.widget → unit
```

```
val border_width : int
```

```
val tooltips : GData.tooltips
```

Input module.

```
module type S = sig..end
```

Contents:

```
val toolbar : GButton.toolbar
```

GtkToolbar widget containing shared tools.

```
val snap : GButton.tool_button
```

Snapshot button.

```
val export : GButton.tool_button
```

Export button.

Output module.

```
module Make : sig..end
```

Contents:

```
functor (_ : PARAMS) →S
```

Generator.

GNU General Public License (GPL) version 3
29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. **Source Code.**

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

   The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

   The Corresponding Source for a work in source code form is that same work.

2. **Basic Permissions.**

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. **Protecting Users' Legal Rights From Anti-Circumvention Law**.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. **Conveying Verbatim Copies.**

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. **Conveying Modified Source Versions.**

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as

to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

(a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

(b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. **Additional Terms.**

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who

receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the

covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

39 Index