XCSoar 7

the open-source glide computer

Developer Manual

August 13, 2021
For XCSoar version 7.12
http://www.xcsoar.org

Contents

Preface

This manual applies to XCSoar version 7.0. The authors reserve the right to update this manual as enhancements are made throughout the life of this product.

Warnings and precautions

IT IS THE USER'S RESPONSIBILITY TO USE THIS SOFTWARE PRUDENTLY. THIS SOFTWARE IS INTENDED TO BE USED ONLY AS A NAVIGATION AID AND MUST NOT BE USED FOR ANY PURPOSE REQUIRING PRECISE MEASUREMENT OF DIRECTION, DISTANCE, LOCATION, OR TOPOGRAPHY. THIS SOFTWARE SHOULD NOT BE USED AS AN AID TO DETERMINE GROUND PROXIMITY FOR AIRCRAFT NAVIGATION. THIS SOFTWARE SHOULD NOT BE USED AS A TRAFFIC COLLISION AVOIDANCE SYSTEM.

Legal notices

Software license agreement

This software is released according to the GNU General Public License Version 2. See Appendix B for the full text of the agreement and warranty notice.

Limited liability

In no event shall XCSoar, or its principals, shareholders, officers, employees, affiliates, contractors, subsidiaries, or parent organizations, be liable for any incidental, consequential, or punitive damages whatsoever relating to the use of the Product.

Disclaimer

This product, and all accompanying files, data and materials, are distributed "as is" and with no warranties of any kind, whether express or implied. This product is used entirely at the risk of the user. Although great care has been taken to eliminate defects during its development it is not claimed to be fault-free. No claims are made regarding its correctness, reliability or fitness

1 Policy

1.1 Git Work Flow

1.1.1 Version Numbering

Each release of XCSoar is denoted by a unique version number. A version number consists three sequential numerical fields separated a period and prefixed by a 'v'. The significance of the three fields are <major>.<minor>.<patch>. So, for instance, the version number 'v7.1.3' indicates XCSoar major version 7, minor version 1, and patch 3.

Release versions are determined and maintained by the release manager.

1.1.2 Git Repository Enduring Branches

At any time the XCSoar contains two enduring branches. The principle enduring branch is 'master'. The lifetime of this branch is unbounded. The second enduring branch is the current minor version branch. The lifetime of this branch is the lifetime of the current minor version. The name of the minor version branch is in the form 'v<major>.<minor>.x' So, for instance, the current minor branch may be called 'v7.1.x'.

With the commencement of a new minor version the release manager will create the required branch. The old minor branch will be retired. All XCSoar releases are made from the current minor branch.

The master branch serves as the development branch for the next minor release (which may also be the first minor of the next major release).

Developers should commit changes for the next minor release to the master branch. Bug fixes of the current minor release should be committed to the current minor branch, in preparation for the next patch release.

1.2 Writing Patches

There are two methods to summit patches to the upstream XCSoar repository.

1.2.1 GitHub

The XCSoar repository is currently hosted at XCSoar GitHub repository. GitHub provides a method for submitting patches using their Web interface. The easiest way to use this method is to make another XCSoar repository by 'forking' from 'XCSoar/XCSoar'. Make sure that your local repository is up to date with the upstream XCSoar/XCSoar repository. Then make a feature branch from the master or current minor branch as appropriate for the change you are making and commit the changes to that branch. To make the upstream merge easier it is best to re base this feature branch with the appropriate upstream branch from time to time.

When ready to submit the feature branch to the upstream push your local repository to your GitHub XCSoar repository and use the GitHub Web interface to make a 'pull request' of your feature branch to XCSoar/SCSoar.

1.2.2 Developers' Mail List

The second method is to submit patches or `git pull` requests to the developer mailing list (`xcsoar-devel@lists.sourceforge.net`).

Patch files can be generated by running

`git␣diff␣>␣patch`

1.2.3 Basic Patch Requirements

A patch should be self-explanatory, it needs a good description. The subject line specifies the subsystem/library name and a brief description of what is changed, followed by an empty line. Then write a longer description if needed, and explain why this change is needed.

Each patch must compile and must not introduce a regression (as far as we know at the time).

Each patch must be self-contained and should only change one thing. Split larger patches into smaller pieces. Don't refactor and add/modify/remove features in the same patch.

Don't rewrite code unless you need to. Migrate incrementally to a new concept. Keep patches small and easy to understand.

1.3 Code Style

79 columns, reasonable exceptions allowed. Indent 2 spaces, no tabs. No indent for `namespace` blocks (a compromise to avoid excessive indentation).

Comments: write enough code comments (in English). All workarounds must be documented. Everybody must be able to understand your code, even when you're gone. Don't abuse multiple single-line comments ("//") to write mult-line comments.

API documentation: non-trivial functions should be documented in a doxygen comment.

Names: class/function names in `CamelCase` (not `camelCase`); attributes/variables lower case, separated with underscore (e.g. `foo_bar`); constants (including `enum` values) all upper case (e.g. `FOO_BAR`).

Exception: when a foreign API is being mimicked (e.g. STL containers), we adopt its naming conventions.

Files: `*.cpp` and `*.hpp` for C++. Files should be named after the main class which is provided. Each class should have a separate source file and a separate header. UNIX text format.

Be `const`-correct. Use `constexpr` instead of `const` whenever possible.

Use `static` whenever possible. Functions and global variables that are only used in one source file should not be exported. Methods that do not use any instance method/variable should be static to avoid the overhead of passing the implicit `this` parameter.

Make methods `virtual` only after careful consideration. A destructor should only be virtual if necessary. All overrides must use the `override` keyword. Use `final` often.

Compile with `WERROR=y` and fix all warnings.

Don't write large functions. Split them up when they become too large.

Avoid dynamic allocation. Dynamic allocation means overhead, more locking and heap fragmentation. Use `StaticArray` and `StaticString` if possible.

Asterisks belong to the variable name, not to the type name. Consider "`Foo* a, b`". "`Foo *a, b`" or "`Foo *a, *b`" is easier to understand.

Some sample code to demonstrate our code style:

```
/**
 * API documentation for this class.
 */
struct TheStruct {
  unsigned an_attribute;
  bool second_attribute;

  TheStruct();

  /**
   * API documentation for this method.
   *
   * @param foo documentation for this parameter
   * @return documentation for the return value
   */
  bool TheMethod(int foo);
};

TheStruct::TheStruct()
  :an_attribute(0),
   second_attribute(true)
{
}

static bool
FooBar(int a_parameter, unsigned another_parameter,
       const TheStruct *next_row)
{
  switch (a_parameter) {
  case 0:
    break;
  }

  if (a_parameter == 2 && another_parameter == 3 &&
      next_row != NULL)
    return true;

  return a_parameter == 42;
}
```

1.4 C++

XCSoar is written in C++17.

XCSoar's standard compilers are `gcc` (at least version 6) and `clang` (at least version 4.0).

Avoid preprocessor macros, because they are obscure, error prone, not type-safe, hard to read and hard to debug. Use `inline` functions and `constexpr` variables instead.

1.4.1 Other rules

In a class declaration, attributes come first, then constructor/destructor, and finally the methods. Having all attributes in one place gives a good overview of the nature of a class.

Avoid expensive and bloated STL containers if there are cheaper solutions (e.g. `StaticArray`, `StaticString` if the maximum size is predictable).

Avoid template hell. Keep templates readable. Keep in mind that excessive template use may bloat the binary.

1.5 Graphical User Interface

1.5.1 Letter Cases

Following the guidline should prevent the GUI from mixtures of "ON" and "On" text elements, and lead to a systematic GUI text presentation. The goal is to recognize GUI text fast and reliable.

Captions : Captions (button captions, windows titles) to use capitalization. E.g. ,"Pan On", "The Display Of ...".

Abbreviations : Generally known abbreviation use upper case like "MC", "ETA", "V"; or they can use CamelCase, especially when using synthetic words like "GoTo", "InfoBox". Abbreviated words by simply cutting the end of the word needs a dot, e.g. "Max. temp."

Plain text : Longer help texts are to write like prose: "This is the help page for ...".

Labels : Label text has the least systematic constraints:

- Captions for text (input) fields, e.g. "Wing loading"
- Info text on widgets. E.g. "No data" on an empty analysis page.
- Label text for radio or check boxes.
- Selections on Combo-boxes, selectors, Pull-down menus.

All those should go like prose, whereas exceptions might be meaningful.

Gauge caption : Also the appearance of the gauge caption should be covered with that. They are currently mapped to upper case all over. I think the most readable also here is a CamelCase approach. E.g. to distinct "WP Dist", "WP AltD", and "WP AltR". Another good example would be MACCREADY, which should be MacCready, or just MC.

Units : Units have their own specific appearance. A profound paper is http://physics.nist.gov/cuu/pdf/checklist.pdf we could just refer to.

2 Architecture

This chapter describes XCSoar's internal code architecture.

2.1 Source Organisation

XCSoar's source code is stored in the `src` directory. This section tries to give a rough overview where you can find what.

- `Util/`: generic C++ utilities that do not depend on external libraries, such as data structures, string operations
- `Math/`: math data types (fixed-point math, angles) and generic formulas
- `Geo/`: geographic data structures and formulas
- `Formatter/`: code that formats internal values to strings
- `Units/`: conversion from SI units ("System" units) to configured user units
- `NMEA/`: data structures for values parsed from NMEA
- `Profile/`: user profiles, loading from and saving to
- `IGC/`: support for the IGC file format
- `Logger/`: all loggers (NMEA, IGC, flights)
- `Thread/`: multi-threading support (OS specific)
- `Screen/`: base library for the graphical user interface
- `Renderer/`: various graphical renderers, for map and analysis
- `MapWindow/`: the map
- `Form/`: modal dialogs and their controls (based on the screen library)
- `Dialogs/`: modal dialogs implementations (based on the form library)
- `Net/`: networking code (OS specific)
- `Operation/`: generic code to support cancellable long-running operations

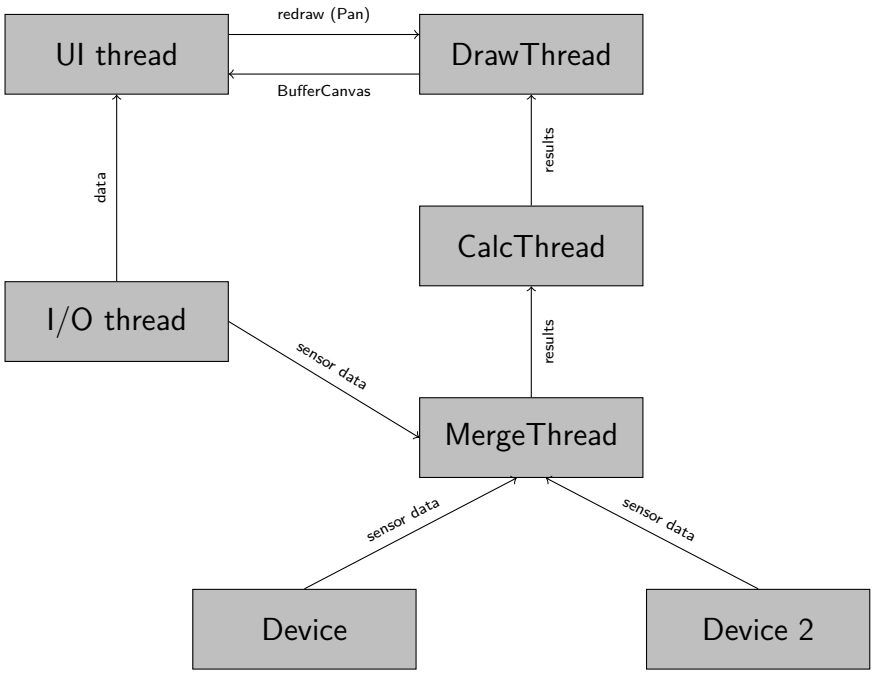
- `Android/`: code specific to Android (the native part only; Java code is in `android/src/`
- `Engine/PathSolvers/`: an implementation of Dijkstra's path finding algorithm, for task and contest optimisation
- `Engine/Airspace/`: airspace data structures and airspace warnings
- `Engine/Waypoint/`: waypoint data structures
- `Engine/GlideSolvers/`: a MacCready implementation
- `Engine/Task/`: task data structures and calculations
- `Engine/Contest/`: contest optimisation
- `Engine/Route/`: the route planner (airspace and terrain)

2.2 Threads and Locking

2.2.1 Threads

XCSoar runs on multiple threads, to make the UI responsive but still allow expensive background calculations.

This is how it looks like on Windows and Linux/SDL (software rendering):



The UI thread is the main thread. It starts the other threads and is responsible for the UI event loop. No other thread is allowed

to manipulate windows. The UI thread has a timer which does regular house keeping twice per second (`ProcessTimer.cpp`).

The calculation thread (`CalculationThread.cpp`, `GlideComputer*.cpp`) does all the expensive calculations in background. It gets data from the devices (through `MergeThread`) and forwards it together with calculation results to the drawing thread and the main thread.

Each device has its own thread (`SerialPort.cpp`). This is needed because Windows CE does not support asynchronous COMM port I/O. The thread is stopped during task declaration (which happens in the UI thread).

When new data arrives on the serial port, the `MergeThread` gets notified, which will merge all sensor values into one data structure. It will then run cheap calculations, and forwards everything to the `CalculationThread`.

With OpenGL, the map is rendered live without a buffer. There is no DrawThread.

On Android, the UI thread is not the main thread - the main thread is implemented in Java, managed by Android itself. The UI thread listens for events which the Java part drops into the event queue (`NativeView.java` and others). The internal GPS does not need a thread, it is implemented with Java callbacks. For Bluetooth I/O, there are two threads implemented in Java (`InputThread.java` and `OutputThread.java`, managed by `BluetoothHelper`

2.2.2 Locking

Some data structures are rarely modified. There is no lock for them. For a modifications, all threads must be suspended. Example: waypoints, airspaces.

Other data structures are modified so often that correct locking would be too much overhead. Each thread and each instance has its own copy. The lock needs to be obtained only for making the private copy. The private copy can be used without locking. Example: `NMEA_INFO`, `DERIVED_INFO`.

There are objects which are too expensive to copy. Normal locking applies to them. We have a template class called `Guard` to enforce proper read/write locking. Example: the task.

2.3 Accessing Sensor Data

Much of XCSoar deals with obtaining sensor data and visualising it.

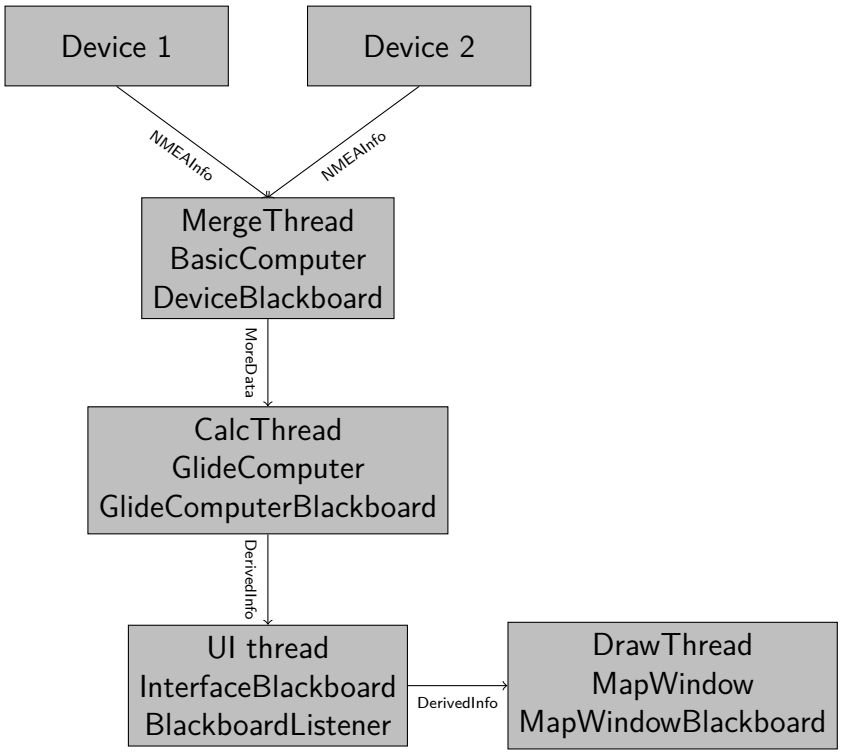
Suppose you want to write a dialog that needs the current GPS location, where do you get it? The short and simple answer is: from `CommonInterface::Basic()` (the `InterfaceBlackboard`). Example:

```
#include "Interface.hpp"

...
  const auto &basic = CommonInterface::Basic();
  if (basic.location_available)
    current_location = basic.location;
```

This is true for the main thread (aka the "user interface thread"). Other threads must not use the `Interface.hpp` library, because the `InterfaceBlackboard` is not protected in any way. It contains copies of various data structures just for the main thread.

This is how sensor data moves inside XCSoar:



The device driver parses input received from its device into its own `NMEAInfo` instance inside `DeviceBlackboard` (i.e. `per_device_data`). Then it wakes up the `MergeThread` to merge the new data into the central `NMEAInfo` instance. The `MergeThread` hosts the

BasicComputer which attempts to calculate missing data (for example, derives vario from GPS altitude).

The CalculationThread wakes up and receives the MoreData object from DeviceBlackboard. Here, expensive calculations are performed (GlideComputer: task engine, airspace warnings, ...), resulting in a DerivedInfo object. The CalculationThread runs no more than twice per second.

Finally, the UI thread wakes up and receives MoreData and DerivedInfo via DeviceBlackboard. This updates InfoBoxes and other UI elements. On Windows, the map is drawn in a separate thread, so there's another layer.

Let's get back to the question: where do I get sensor data? That depends on who you are:

- you are the user interface: (InfoBoxes, dialogs, any Window callback): InterfaceBlackboard (see above). To get notified on changes, register a BlackboardListener (and don't forget to unregister it).
- you are the MapWindow: depends! If you're being called from OnPaintBuffer (i.e. inside the DrawThread), you must use the MapWindowBlackboard, all others must use the InterfaceBlackboard.
- you are a "computer" library: you will get the values as a parameter. Don't try to use the GlideComputerBlackboard directly.
- you are a device driver: implement the method OnSensorUpdate or OnCalculatedUpdate if you need to know values from other devices or calculation results.
- everybody else may use the DeviceBlackboard, but be sure to lock it while using its data.

3 Developing

3.1 Debugging XCSoar

The XCSoar source repository contains a module for the GNU debugger (gdb). It contains pretty-printers for various XCSoar types, including `Angle`, `GeoPoint` and others. These are helpful when you print values in the debugger. To use it, start the debugging session and load the module:

```
$ gdb -ex "source tools/gdb.py" output/UNIX/bin/xcsoar
(gdb) run
```

The module will automatically convert fixed-point to floating point, radian angles to degrees and more. You can now do fancy stuff like:

```
(gdb) p basic.location
$1 = GeoPoint(7.93911242887 51.1470221074)
(gdb) p basic.date_time_utc
$2 = DateTime(2012/12/23 21:41:57)
(gdb) p basic.track
$3 = 55.2254197961
(gdb) p basic.external_wind
$4 = GeoVector::ZERO
(gdb) p current_leg.vector_remaining
$5 = GeoVector(267.899420345 107957.109724)
```

4 User interface guidelines

4.1 General

- Minimise the number of colours, and re-use colour groups already defined.
- Too much use of colour where it is not required serves only to reduce the effectiveness of bright colours for important items.
- High colour saturation elements should be reserved for high importance items
- High contrast against background should be reserved for high importance items
- Attempt to adopt colours that are intuitive based the function of the item
- Minimise the clutter where possible — readibility is essential for use in flight
- Use colours defined in `Graphics` according to functional name, not their actual colour.
- Try to maintain consistent use of colours in all uses of that function, such as dialogue graphics as well as map overlays and infoboxes.
- Text should always be monochrome.

Use aviation conventions or adopt best aviation human factors standards where possible, in particular:

- ICAO Internation Standards and Recommended Practices, Annex 4 to the Convention on International Civil Aviation (Aeronautical Charts).
- NASA Colour Usage recommendations and design guidelines: `http://colorusage.arc.nasa.gov/`
- DOT/FAA/AR-03/67 *Human Factors Considerations in the Design and Evaluation of Electronic Flight Bags (EFBs)* `http://www.volpe.dot.gov/hf/aviation/efb/docs/efb_version2`
- FAA Human Factors Design Standards `http://hf.tc.faa.gov/hfds/`.

- DOT/FAA/AM-01/17 *Human Factors Design Guidelines for Multifunction Displays*

Check for performance with respect to colour blindness. This site has a useful tool that can be used to convert screenshots to how they would look to a person with common color blindness: `http://www.etre.com/tools/colourcheck/`.

For safety purposes, avoid use of elements that may encourage or require the user to stare at the screen continuously.

For safety purposes, avoid user controls that have significant risk of producing unsafe results if misconfigured by the pilot.

4.1.1 General colour conventions

Colour conventions generally in use throughout the program:

- Red for indicator of warning
- Orange for indicator of caution
- Green for positive indicator of safety
- Blue for neutral indicator of safety

4.1.2 Displayed data

- Where data is invalid, indicate this by not presenting the data or showing dashes.
- Present data in user-defined units.
- Display numerical data with significant digits appropriate to the accuracy of the calculations, or its functional use by the pilot, whichever is lower.

4.2 Dialogs and menu buttons

4.2.1 Colors

Colour conventions in use are:

- Grey for buttons
- Buttons and other widgets rendered with an evenly shaded border
- Yellow for clicked items
- Light blue for the key focused item

- Medium blue for dialogue title bar
- Text is black if the item is enabled
- Text is greyed out (but still visible) if the item is disabled

4.2.2 dialogue types and navigation buttons

There are four types of dialogs in XCSoar, and the navigation buttons for each are different. Navigation buttons are the Close, OK, Cancel and Select buttons.

- Dialogs that modify and save data when the dialogue closes.

  These shall usually have a Close button (no Cancel) and may have context specific function buttons

- Dialogs that modify data where Cancel would be important for the user.

  These shall have OK and Cancel buttons. This may include dialogs with children dialogs where hitting Cancel from the parent dialogue cancels all the changes made in the children dialogs

- Dialogs that have a list of values, one of which can be selected to return to the parent dialogue.

  These shall have Select and Cancel buttons

- Dialogs that display information that cannot be modified.

  These shall have a Close button

4.2.3 dialogue button placement and size

- The Close and Cancel buttons will never appear in the same dialogue and are always located in the same place. This location will be:

  For portrait: lower right

  For landscape: lower left

- The Select button will be accompanied with a Cancel button. The locations will be:

  For portrait: Select in lower left, Cancel in lower right

  For landscape: Cancel in lower left, Select immediately above it

- Buttons will be 35 (scaled) pixels high

- Buttons will be flush with the bottom of the screen and with the sides of the screen and against each other (no margins)
- In portrait, buttons will be 33
- In landscape, buttons will be 65 to 80 (scaled) pixels wide, as wide as the frame permits. They will generally be a vertical row of buttons flush left of the screen
- If text won't fit on a button, the buttons can be made larger consistently for a screen, but this should be the exception because if it must contain that much text consider using a different type of control.
- Exceptions to all the dialogue concepts above are encouraged, but should be mocked up and reviewed with the development community prior to implementing and possibly documenting in the developers guide.

4.2.4 Usability

- Minimum size of buttons should be X by Y mm
- Ensure all dialogs are navigable using cursor keys only
- Ensure the focussed item is clearly identified. The rectangle of the widget on the canvas may be drawn using the `fill_focus` method of `Canvas`.

4.3 Main graphics

4.3.1 Colors

Colour conventions in use, in order of priority, are:

- Aircraft black and white, for neutrality but clear identification
- Traffic (FLARM) use alarm green, orange, and red.
- Lift is vibrant green, sink is copper orange.
- Aircraft navigation (route, best cruise track) is (ICAO) dark purple-blue
- Task navigation lines and areas are (ICAO) magenta.
- Updraft sources and other updraft derived data is sky blue.

(Todo) airspace alert colours

Map culture (topography) and terrain rendering should conform to ICAO Annex 4 where appropriate. Note that some modifications are reasonable for electronic use given that Annex 4 deals with paper charts. Nevertheless, the colour conventions are useful to adopt as they are likely to be intuitive and are designed for aviation use.

4.3.2 Pen styles

- Map culture should be rendered with a thin pen
- Thicker pens used for important (e.g. task, navigational, airspace) lines
- Dashed lines are used to increase perceptual priority

4.3.3 Map overlays

Elements on the map that are not part of the map layer, such as additional informational widgets (final glide bar, wind, north arrow) should be rendered so as to help those elements be visually separated from the map:

- Generally adopt higher contrast (higher colour saturation or darker shade) than the background map layer elements.
- For elements covering an area (non line), draw the entire element or a border with a luminosity contrasting pen, of width `IBLSCALE(1)`.
- Consider whether the widget is required in all flying states and display modes. if it does not serve a direct functional purpose in some states/modes, do not render it.
- Avoid locating widgets at the aircraft symbol (ownship symbol). It is important to keep this area clear so the aircraft symbol can be easily found.

Elements that may be rendered over each other should be organised in order of priority, particularly with alert warning items above caution items above non-alert items.

4.4 Terminology

4.4.1 Glide Ratio

'Glide ratio' is a non-specific term which can refer to the ratio of horizontal to vertical motion with reference to either the surrounding airmass or the ground.

To reduce confusion, ground-referenced glide ratios (eg distance travelled over ground vs altitude lost) should be referred to by the term 'glide ratio over ground' when space allows, or 'glide ratio' / 'GR'.

Air-referenced glide ratios (eg airspeed vs sink rate) should be specified as 'lift/drag ratio' / 'L/D ratio' / 'LD'. The lift/drag ratio is numerically equal to the air-referenced glide ratio when flying at constant speed.

If usage spans both air-referenced and ground-referenced glide ratios, the non-specific term 'glide ratio' / 'GR' should be used. 'Lift/drag ratio' should never be used to refer to ground-referenced glide ratios.

5 File formats

5.1 Input Events

The Input System is deprecated! It is being replaced by a Lua scripting engine.

5.1.1 Introduction

The Input System is actually a large number of things all bunched into one.

Primarily it is about giving the user control of what button does what and when. There is a new concept called Input Mode - this is a the mode the GUI is in for input. For example, you can click on the info boxes and you are now in "infobox" mode. Clicking on the map is called "default". But it doesn't stop there, you can create a new mode called anything you like. This may not mean much - but wait till you combine it with the rest of the features...

Input is not restricted to hardware buttons any more. You can map all your hardware buttons (currently support for APP1 to APP6, Left, Right, Up, Down and Enter, although I believe we can do some more) but also any key code at all. This feature allows those with a built in keyboard to use any key to map to any function in XCS. Where it comes into real advantage is in external keyboards. There are a number of bluetooth devices out there (eg: http://shop.brando.com.hk/btgamepad.php) which can map each of their buttons to any key code - that key code can then be mapped to any feature in XCS. You can then add to the hardware buttons the buttons available to you on external devices. Other inputs (eg: Serial) are also being looked at - and support is in the code for that extension.

We are striving towards a platform which is not only easier to use and more intuitive, but also faster and easier to use in flight as well. As such, another new feature as part of input is the concept of Button Labels. Combined with the modes mentioned above, you can create any arbitrary set of functions to map to any number of buttons. Think about it like creating a tree, or a multiple level menu.

This produces two benefits that I know will be appreciated by people with limited inputs. The first is that you can create menus,

where by you press one button to get to the next level (eg: pressing on APP1 brings up AutoZoom, Pan Mode, Full screen on the other buttons. Press APP1 again and it goes to Terrain, Marker and Auto MacCready. Press APP1 again and the menu is gone) - but more importantly for those with touch screens and limited buttons, each of these labels can optionally be assigned a key and you can touch the button area as if it was a button. This means that we can actually control on a touch screen model the entire system without buttons - press an area of the screen and the buttons pop up, click through - change options and more.

The combined features of labels, configurable buttons (including from external hardware), hierarchical menus (for lack of a better name), touch screen buttons has allowed us to configure XCS - without recompile - for an enormous range of hardware, and personal preference. And all configurable as plane text, simple files. There is no need for a file, the defaults internally will probably be a combination of a 4 button bottom system with one button always shown on screen for no/few button display.

The screen layout - location of the labels - is also totally configurable - allowing us to vary the layout of buttons depending on the type of organiser or desired look and feel.

There is a great unexpected benefit in the development of the input system.

We can execute any number of events attached to an input with only 2 extra lines of code. This worked perfectly. So now we have a basic macro system, allowing many more events to be attached to a single input event.

But it doesn't stop there, this has lead to some more excellent developments. The idea of Glide Computer Events things like "Maximum Altitude Reached". Currently we play a sound effect for that. But you may choose to play a sound, bring up a message box and write to the log file.

One nice feature of XCS is the ability to change things such as Zoom and North when Circling. Now you can do so much more. You could choose to point North, Zoom to 1.0 (rather than a relative change), Turn on Vario Sounds, Start a timer. When switching back to Cruise mode, you can bring up the stats box for 30 seconds. The options are limited by your imagination.

This is also contributing to a major reduction in complex code. We can move out these complex tests into one centrally, easier to manage system, reducing bugs and improving maintainability.

Another side benefits of these Macros is User Defined Flight Modes. One idea was a button which switched to Zoom 1.0, Pan ON, Pan Move to Next Waypoint. Basically the ability to jump and see the next waypoint. And in the previous we can change the Input Mode to "ViewWaypoint" - at which point you can redefine the same button to switch back to your original settings.

The flexibility of this system comes with only one small price. We can't provide an interface within XCS to fully customise all of these near infinitely variable possibilities. However I believe that is unnecessary anyway, you are not likely to change these sort of features very often, and definitely not on the field. That does not mean you can't, you can of course edit the plane(sic) text file to change functions.

What this really means is that we can have people in the project helping and contributing to the customising of XCS, without having to change the code. This, especially on an open source project is fantastic as it nicely separates the user interface changes from the highly reliable part of the code. It also involves people who can develop new interfaces and functions that are expert gliders but not necessarily programmers.

For information on file formats see Common/Data/Input/template.xci and the web site documentation.

5.1.2 Defaults and Files

The file in the source Common/Data/input/template.xci is used to generate automatically the C code necessary for the default configuration. However it is in the exact same format as can be read in by XCS and therefore can be used literally as a template for a more complicated file.

When you create your own file, you will need to select it as the Input File within XCSoar Menu/Config/System/Look/Language,Input/Events. Choose the custom file you would have previously created, and then restart XCS.

5.1.3 File format

The file is plain text, with key=value pairs and a blank line to indicate the end of a record.

```
mode=default
type=key
data=APP1
```

```
event=StatusMessage My favorite settings are done
event=ScreenModes full
event=Sounds on
event=Zoom 1.0
event=Pan off
label=My Prefs
location=1
```

The record above demonstrates remapping the first hardware key on your organiser to change Pan to off, Zoom to 1.0 Sounds on, ScreenModes full, and then a status message to tell you it is done.

Lines are terminated by the stanard DOS newline which is CRLF (Carrage Return then Line Feed). Records are terminated by an extra new line.

5.1.4 Event order

Until further work is done on processing, events are actually done in reverse order - also known as RPN. This is because the events work on the stack principle. Each one is pushed onto the stack for execution, and then executed by popping back off the stack. This has reduced complexity of the code base.

When writing input events, have a look where you put the StatusMessage and make sure that it is at the top, not the bottom (if you have one).

5.1.5 Event list

Event	*Description*
`MainMenu`	
`MarkLocation`	Mark a location.
`Mode` *M*	Set the screen mode.
`Pan` *P*	Control pan mode. Possible arguments: on (enable pan), off (disable pan), up, down, left, right
`PlaySound` *S*	Play the specified sound.
`SnailTrail` *S*	Change snail trail setting. Possible arguments: off, short, long, show.
`ScreenModes` *M*	Set the screen mode. Possible arguments: normal, auxilary, toggleauxiliary, full, togglefull, toggle.
`Sounds` *S*	Change vario sounds. Possible arguments: toggle, on, off, show.
`StatusMessage` *M*	Display the specified status message.
`Zoom` *Z*	Everything about zoom of map. Possible arguments: auto toogle, auto on, auto off, auto show, in, out, +, ++, -, –.

5.1.6 Modes

XCSoar now has the concept of Modes. These are an arbitrary string that associates with where and what XCS is doing.

Note: a mode entry in a record can have multiple entries by using a space between eg: "infobox menu1 menu2"

List of known modes

`default` : Really map mode, where you mostly are

`infobox` : An info box has been selected on the scrreen

`*` : Any other arbitrary string

Mode precedence has been tricky, so instead of solving the problem it is being worked around. XCS will choose to set a global variable to specify what mode it thinks it is in. This can then be used by the input code to decide what to do. This mode could get out of sink with the real world, and careful checking will be required, but at this stage it seems like the only sensible option.

The code will review first if an entry exists in the current mode, and then in the default mode. This allows you to do one of the following example: Define a default action for button "A" to be "Zoom In" but make that button increase Bugs value in infobox mode only. You can do this by making an "default" and a "infobox" entry. You can also put an entry in for Button "A" for every mode and have complete control.

Special Modes - eg: the level of a menu (Think File vs Edit, vs Tools vs Help)

have special modes, such as the level of the menu you are at. You press one button, then another set become available (like pressing menu and seeing Settings etc). This will be very useful in non-touch screen models. The menu configuration can then be read from this same file and configured, allowing any number of levels and any number of combinations.

The only hard part is what mode to go back to. We need a "Calculate Live Mode" function - which can be called to calculate the real live mode (eg: finalglide vs curse) rather than the temporary mode such as Menu, Special Menu Level, Warning etc.

The label and location values are examples of what can be done here to allow input button labels to be displayed. What needs to be considered is a simple way of mapping the locations and the size. In some models it may be that buttons are 4 across the top

of the screen, where as others it is 3 or 2 or even 6. So both size and location needs to be considered.

The label itself will go through gettext to allow language translations.

5.1.7 Keys

The key type can have the following possible values:

`APP1-APP6` : Hardware key on pocket pc

`F1-F12` : Standard function keys

`LEFT, RIGHT, UP, DOWN, RETURN` : Mapped to arrow keys - joystick on organisers

`A-Z, 0-9` : and other possible keyboard buttons (case is ignored)

XXX Review... Input Types

Types:

hardware These are the standard hardware buttons on normal organisers. Usually these are APP1..6.

keyboard Normal characters on the keyboard (a-z etc)

nmea A sentence received via NMEA stream (either)

virtual Virtual buttons are a new idea, allowing multiple buttons to be created on screen. These buttons can then be optionally mapped to physical buttons or to a spot on the screen (probably transparent buttons over the map).

Modifiers

It is a long term goal of this project to allow modifiers for keys. This could include one of the following possibilities:

- Combination presses (although not supported on many devices)
- Double Click
- Long Click

Modifiers such as the above will not be supported in the first release.

`Functions/Events - what it does`

AutoZoom on, off, toggle FullScreen on, off, toggle SnailTrail on, off, long, toggle VarioSound on, off Marker optional text to add MenuButton on, off, toggle Menu open, close, toggle MenuEntry task, b+b, abortresume, abore, resume, pressure logger, settings,

status, analysis, exit, cancel NOTE: Some of the above may be separate functions Settings (each setting, bring up to that point) Bugs add, subtract, 0-100Ballast add, subtract, 0-100Zoom add, subtract, 0-nn (set value) Wind up, down, 0-nn (set value, left, right, "n","ne","e","se","s","sw","w","nw"... MacCready add, subtract, 0-nn (set value) WaypointNext "String" to specific waypoint eg: WayPointNext "home" WayPoint??? "reverse" - reverse, from last passed back to start (ie: from here to home) "drop next" - drop the next "restore" - restore all - from start of flight but XXX This needs more thought flight "startstop", "start", "stop", "release" Start/Stop of flight - Can be automatic, but pressing will override automatic part. release markse the point of release from tow

5.1.8 Glide Computer Events

These are automatically triggered events. They work in exactly the same way, but instead of the user pressing a key, the glide computer triggers the events.

A simple example is moving from Cruise to Climb mode. We want to zoom in, change our track up to north up and switch to full screen. You may also choose to drop a marker with the words "entered thermal". The choicese are up to your imaginations - the GCE (Glide Computer Events) allow you to control what happens.

These are represented as "type=gce" and data=* - as listed below.

Event	*Description*
COMMPORT_RESTART	The comm port is restarted.
FLIGHTMODE_CLIMB	The flight mode has switched to "climb".
FLIGHTMODE_CRUIS	The flight mode has switched to "cruise".
FLIGHTMODE_FINALGLIDE	The flight mode has switched to "final glide".
GPS_CONNECTION_WAIT	Waiting for the GPS connection.
GPS_FIX_WAIT	Waiting for a valid GPS fix.
HEIGHT_MAX	Maximum height reached for this trip.
LANDING	You are at landing.
STARTUP_REAL	First message - this happens at startup of the real XCS.
STARTUP_SIMULATOR	Startup first message. This happens during simulator mode.
TAKEOFF	You have taken off.

5.2 Map Data file formats

The map data is typically downloaded from the map generator server and consists of a single `.xcm` file. It is a zip file which contains several separate files for terrain, topography and waypoint data:

`info.txt`	General map information
`terrain.jp2`	Terrain (elevation) data, georeferenced in `terrain.j2w`
`waypoints.cup`	Waypoint data
`topology.tpl`	Topography data file index
`*.shp` / `*.dbf` / `*.shx`	A set of ESRI format shape file sets with actual topography voctor data information, as listed and defined in `topology.tpl` (Coasts, rivers, roads, cities etc.)

5.2.1 Map information

`info.txt` Contains information about the map as a whole, such as creator, creation time, and lat/lon range.

5.2.2 Terrain data files

The map cointains a digital elevation model of the map area. It is stored as an JPEG2000 compressed image in the file `terrain.jp2`. The projection information (lat/lon boundaries) of the DEM file are contained in the text file `terrain.j2w`, in decimal degree latitude/longitude format. Water is defined as elevation lower than `TERRAIN_WATER_THRESHOLD=-30000`, therefore care has to be taken that JPEG compression parameters and algorithms are used which do not generate artefacts at the coastlines due to the potentially big jump in elevation value.

5.2.3 Waypoints

A map database file can contain waypoints. They reside in the `waypoints.cup` file, which has regular `.CUP` format.

5.2.4 Topography data

Shape files

Non-elevation topography data is stored in standard ESRI shape files. For each type of topographic shape (road, river, city outline, etc.) there is one shape filein `.shp`, which containes all shapes of this type. For each `.shp` file, there has to be an associated `.dbf` file containing shape metadata (such as the name of the city) in

dBASE format, and an index file of `.shx` file type which contains the index that relates the metadata to the shapes.

All of this is defined in the ESRI shape file standard. The official definition of the standard can be found at http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf, but there are more compact descriptions available on the web, see for example wikipedia info and links at http://en.wikipedia.org/wiki/Shapefile.

There can be more files associated with each shape file, such as `.prj`, `.qix`, `.atx`, which are not used by XCSoar.

The set of shapefiles actually used by XCSoar and the attributes of each file are defined in the topography layer description file `topology.tpl`. All shape files used by the map must be listed there.

Topography layer description file (topology.tpl) format

Each line of the topography layer description file (`topology.tpl`) contains a comma separated list (CSV) of information needed for rendering of an individual topography layer. Lines starting with '*' are ignored.

XCSoar v6.6 and earlier will display at most 20 topography layers. XCSoar v6.7 and later will display at most 30 topography layers.

Column name	Data type	Valid range
filename	string	
range	double (nm)	-
icon	string	
label index	int	0-1
color (red component)	int	0-255
color (green component)	int	0-255
color (blue component)	int	0-255
pen width	int	0-31
label range (nm)	double	-
important label range (nm)	double	-
alpha	int	0-255

Table 5.1: Topography file format

filename : The filename of the Topography layer within the container file.

icon : XCSoar v6.5 and earlier, Only the value 219 is recognised, for town icons. From XCSoar v6.6, the name of the icon to display. Optional. See below for a list of available names.

range : Zoom level threshold. Layer elements will not be drawn unless zoomed in closer than this threshold.

pen width : Lines contained within this layer are drawn with pen width.

label range : Label display zoom level threshold. Labels contained in the layer file will not be rendered unless zoomed in closer than this threshold.

important label range : A zoom level threshold. Labels contained in the layer file will be rendered in standard style when the display zoom level is greater than this threshold.

alpha : The alpha component controls transparency of polygons... 0 means polygons are completely transparent, 255 means they are completely opaque. Only used by XCSoar v6.7 and later.

`Versions of XCSoar running on Windows and WinCE ignore any item where transparency is specified.`

Point Features

Prior to XCSoar v6.6, this could contain the value 219 to display an icon for a town From XCSoar v6.6, a user can put an optional string into the icon column in topology.tpl in the .XCM file (e.g.)

- SpotHeight,5,`mountain_top`,1,64,64,64,1,5,
- Mast,10,`obstacle`,,,,1,10,

This can be used for Shapefiles containing point features or polygons or linestrings, but is probably only useful for point features.

The icon of the corresponding image and optional label will be displayed. In the first example, the "mountain_top" icon and a label will be displayed for each point in the SpotHeight shapefile. My SpotHeight Shapefile has been generated with the point elevation in feet as the label value). For the second example, only "obstacle" icons (no labels) will be displayed for points in the Mast Shapefile..

Icon names are detected in TopographyStore.cpp. Names must be given in lowercase. If the icon name given is unknown, or no icon name is given, then icons are not displayed for that Shapefile.

Names correspond to images which have been linked into XCSoar, although it is envisaged that in future these will be names of icon files. Available icon names are:

- mountain_top
- bridge
- tunnel
- tower
- power_plant
- obstacle
- mountain_pass
- weather_station
- thermal_hotspot
- town
- mark
- turnpoint
- small
- cruise
- terrainwarning
- logger
- loggeroff
- target
- teammate_pos
- airspacei
- traffic_safe
- traffic_warning
- traffic_alarm
- taskturnpoint
- marginal
- landable
- reachable
- airport_reachable

- airport_unreachable
- airport_marginal
- airport_unreachable2
- airport_marginal2
- outfield_unreachable2
- outfield_marginal2
- outfield_reachable
- outfield_unreachable
- outfield_marginal

Adding new Icons

At the moment, adding new icons requires a rebuild of the XCSoar application.It is envisaged that, in future, this process won't be required...users will include icon files in their .XCM map container files, and refer to them by name. However, that has not yet been implemented.

To add your own images to the list of icons:

1. Create a .svg file for the icon (e.g. mast.svg) and copy into `xcsoar/Data/icons`. For Android, the name must be lowercase.
2. Insert two (for normal and high-res) lines into `xcsoar/Data/XCSoar.rc`, (e.g.)

   ```
   BITMAP\_ICON(IDB\_MAST, "mast")
   BITMAP\_ICON(IDB\_MAST\_HD, "mast\_160")
   ```

3. Insert two lines into `xcsoar/src/Resources.hpp` (e.g.)

   ```
   MAKE_RESOURCE(IDB_MAST, 500);
   MAKE_RESOURCE(IDB_MAST_HD, 5500);
   ```

4. Add a corresponding line into the `icon_list` table in `xcsoar/src/Topogr`

   ```
   {"mast", IDB\_MAST},
   ```

5. Make XCSoar

After this, a line can be added in `topology.tpl` to connect the icon to the Shapefile using the icon name. (e.g.)

```
Mast,10,mast,,,,,1,10,
```

Note that unless these changes are merged into the main XCSoar repository, then only your specific build of XCSoar will be able to display your icon image.

Appendix A Setting up a development environment based on linux

This describes the setup of a development environment suitable to compile XCSoarfor most supported platforms. The manual focuses on recent releases of Debian-based flavors of GNU/Linux (including Ubuntu).

In the following instructions, `sudo` is used to execute commands with root privileges. This is not enabled by default in Debian (but on some Debian based distributions, like Ubuntu).

To install a virtual machine with the required, you can use `Vagrant`, see section **??**.

A.1 Download source code

To download the XCSoarsource code, make sure you have git installed:

```
sudo apt-get update
sudo apt-get install git
```

Download the source code of XCSoarby executing `git` in the following way in your project directory:

```
git clone --recurse-submodules git://github.com/XCSoar/XCSoar
```

A.2 Use provisioning scripts

If you are not using Vagrant, but an existing standard installation of a Debian-based Linux distribution, you can run the scripts from `ide/provisioning` subfolder of the XCSoarsource to install the build dependencies for various XCSoartarget platforms.

```
cd ide/provisioning
sudo ./add-debian-unstable.sh
sudo ./install-debian-packages.sh
./install-android-tools.sh
```

A.3 Optional: Eclipse IDE

One of the most widespread IDEs is eclipse. It is not limited to Android, and can be used for all targets. It is not required for XCSoar, but its installation is described here as an example. Eclipse is quite heavyweight, and many developers prefer other IDEs for XCSoardevelopment.

To install, download the eclipse installer (Sometimes called "*Ooomph!*" for some reason) from here:
`https://www.eclipse.org/downloads/`

Important: Install the CDT version of eclipse for C development, not the Android/Java package, even if you plan developing for Android. In addition, it is very convenient to install the git support (egit).

The current stable version is *eclipse mars* (4.5) and works with OpenJDK 7 or 8, the new *eclipse neon* 4.6, currently RC2, is also quite stable, and requires OpenJDK 8. Both can be installed with the installer.

You can also install the ADT (Android development tools) package for better integration with Android.

Next, create a new project, by generating a make project from existing sources files. Choose your xcsoar source directory which contains the makefile.

Important: After you have added the sources, eclipse will start indexing all files. If you have already started `make` before this time, then a lot of files have been downloaded for the various libraries which are exctracted/built within the XCSoardirectory (most notably the boost libraries). Indexing all these takes a very long time, and a lot of heap space, so you should probably stop the indexer right away. In addition you should probably exclude these directories from the indexer for the future.

For this, in the C/C++ scope, right-click on the "output" directory in the file tree on the left side, select "Properties", then "Resource/Resource Filters" and add a filter. In the "add filter" dialog, choose "exclude all", "files and folders", "all children (recursive)" and set the Filter details to "Name matches *". This will exclude the output tree from the indexer, leading to a minimal index.

A.4 Optional: modern LaTeX editor for editing the Manual

Most people today edit LaTeX files in specific editors, as this is much more comfortable and efficient. This is highly recommended especially if you are not very familiar with LaTeX: learning it is very easy with a modern editor. Here, we install TeXstudio as an example, as it is very widespread and supports the rather rare LuaLaTeX well.

To install, get the relevant package:

```
sudo apt-get install texstudio
```

As the directory tree of XCSoaris very unusual for a LaTeX project, we need to make some special configurations in order to allow for quick compiling from within the editor, and for full synctex functionality:

In "Options / Configure TeXStudio", enable "show advanced options".

In "Options / Configure TeXStudio / Commands / Commands / LuaLaTeX", replace

```
lualatex -synctex=1 -interaction=nonstopmode %.tex
```

with

```
lualatex -synctex=1 -interaction=nonstopmode
 -output-directory=?a)../../../output/manual %.tex
```

In "Options / Configure TexStudio / Build / Build Options / Addition Search Paths":
Enter in *both* fields ("Log file" and in the field "PDF File"):

```
../../../output/manual/
```

Add the following line to *both* the `.profile` and the `.bashrc` file of your user directory:

```
export TEXINPUTS="..:../../../output/manual:../../../output/manual/en:../../..:"
```

Finally, you need to run "make manual" in the XCSoarbase directory at least once from the command line before you can compile from within the TexStudio interface. This creates the path structure and generates the figure files which are included into the manual. Of course, if you change figures, you might have to run "make manual" again.

Inside TeXStudio, open the file "XCSoar-manual.tex" (or one of the other root files) and right-click on this file to "set as explicit

root document", in the structure view on the left. Now you are good to go. Make changes and press F5 to see the result immediately.

Appendix B GNU General Public License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we

want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

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

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms

of Section 1 above, provided that you also meet all of these conditions:

(a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

(a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the

terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copy-

ing, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

No warranty

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE 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.