XTrkCad Developer’s Guide

Table of Contents

[Introduction 2](#__RefHeading___Toc890_3359772709)

[Source Control 4](#__RefHeading___Toc779_2817623836)

[Branches 4](#__RefHeading___Toc781_2817623836)

[Releases 5](#__RefHeading___Toc783_2817623836)

[Text Files 7](#__RefHeading___Toc350_3696126145)

[Parameter File Maintenance 8](#__RefHeading___Toc171_3696126145)

[Independence of Parameters in Layout files 8](#__RefHeading___Toc342_3696126145)

[CONTENTS Label 8](#__RefHeading___Toc173_3696126145)

[Adding / Deleting Parameter files 8](#__RefHeading___Toc344_3696126145)

[Changing CONTENTS Label and/or FileName 8](#__RefHeading___Toc346_3696126145)

[Source Layout 10](#__RefHeading___Toc175_3696126145)

[Naming Convention 10](#__RefHeading___Toc473_82747535)

[WLib Interaction 11](#__RefHeading___Toc109_2394484777)

[Pretty Printing 12](#__RefHeading___Toc113_2394484777)

[Casts 13](#__RefHeading___Toc115_2394484777)

[Downcast Numeric values 13](#__RefHeading___Toc414_4191666527)

[Generic Values 13](#__RefHeading___Toc416_4191666527)

[Wlib wControl\_t conversion 13](#__RefHeading___Toc418_4191666527)

[GetTrkExtraData 13](#__RefHeading___Toc420_4191666527)

[Error Handling 14](#__RefHeading___Toc87_1320853591)

[User Input Errors 14](#__RefHeading___Toc372_3696126145)

[File Input Errors 14](#__RefHeading___Toc374_3696126145)

[Logic errors 14](#__RefHeading___Toc376_3696126145)

[System/Evironmental errors 15](#__RefHeading___Toc378_3696126145)

[Undo Processing 16](#__RefHeading___Toc366_3696126145)

[Draw Model 17](#__RefHeading___Toc368_3696126145)

[Headers 18](#__RefHeading___Toc358_3696126145)

[#ifdef WINDOWS 19](#__RefHeading___Toc360_3696126145)

[VERSION handling 20](#__RefHeading___Toc362_3696126145)

[Release Version 20](#__RefHeading___Toc475_82747535)

[File Format Version 20](#__RefHeading___Toc477_82747535)

[Segment Version 20](#__RefHeading___Toc479_82747535)

[Extensible Segment Format 21](#__RefHeading___Toc481_82747535)

# Introduction

XTrkCad is a large program and like most large program, it has developed an infrastructure of how things are done and a base level of functionality. Some are general to any program (Error Handling, Source Formatting) and some particular to XtrkCad (Undo Processing, Parameter Files).

This document describes a fairly random set of features. It is expected to evolve as needs be.

If you figure out some interesting, tricky or obscure things, please add them to this document.

You’ll see a number of place holders (pages with just a header). These will be addressed n future.

**$SRCDIR** refers to the root of your source tree. **$BINDIR** refers to the base of your Build tree.

# Source Control

XTrkCad source code is managed using Mercurial on SourceForge. See the Fetch/Build wikka page (<https://xtrkcad-fork.sourceforge.net/Wikka/BuildNotes>).

## Branches

Normal development is handled in the ‘default’ branch.

It is recommended that any non-trivial work occur in a sub-branch from ‘default’. Once the new work is complete and stable, the sub-branch is merged back into ‘default’. Merging from the ‘default’ branch should be done frequencely to keep the new code in sync, especially just before merging back to ‘default’r

If you are using **thg** (a graphical interface to hg) you can use the following to make a new branch.

At the **tip** of the **default** branch, click **Branch: default:**



Then pick the name of your branch. If you are fixing a bug, include the bug number.



1. Make your changes and **Commit** them. This will create the new branch. Furthjer changes are made in the new branch.

When done, **Close** your branch, switch to Default and **Merge** from the new branch. Closing first avoids dangling heads.

## Releases

Releasing a new version of XTrackCad, once the code is stable, feature complete and free of serious bugs:

* Create in a new branch (vX.Y.Z) which is branched from **default**
* Update *$SRCDIR****/****ProgramVersion.cmake*, in the **default** branch, with the next 3rd level version number, by incrementing XTRKCAD\_RELEASE\_VERSION. XTRKCAD\_VERSION\_MODIFIER will be “Dev”.
* Update *$SRCDIR/ProgramVersion.cmake*, in the **new** branch with the new version number. XTRKCAD\_VERSION\_MODIFIER will be “Beta1”
* Handle new or modified param files.
* All platforms build and upload their beta install packages as described in <https://xtrkcad-fork.sourceforge.net/Wikka/BuildNotes>
* Source tar balls are created and uploaded.
* Update *CHANGES.md, Readme.md* in *$SRCDIR/app/lib* and upload.
* Update the XTrackCad Manual (see *$SRCDIR/app/doc/mk-xtrkcad-manual-htm*l) and upload.
* Update the 'XTrackCAD Version' field with the new version at the [SourceForge Bug Field Managment](https://sourceforge.net/p/xtrkcad-fork/admin/bugs/fields)  page
* Announce the new release on the XTrackCad mailing litst.

This process (except for the first two actions) is repeated for subsequent Betas, until all issues are resolved or defered. XTRKCAD\_VERSION\_MODIFIER is incremented: Beta2, Beta3, ...

For General Availability, XTRKCAD\_VERSION\_MODIFIER will be “GA” and GA install packages are built and uploaded.

Critical bugs discovered after GA are fixed in Hot Site releases with XTRKCAD\_VERSION\_MODIFIER set to “HS1”, “HS2”,...

The same branch is used for all releases. A tag is created for each build: vX.Y.Z-B1..., vX.Y.Z-GA, vX.Y.Z-HS1...

**Version string sequence**

Version numbers are for illustration.

The choice of release level is made when the first Beta is released

* 1st level: major UI, technological or organizational changes
* 2nd level: significant new functionality, major redesign
* 3rd level: bug fixes, minor functionality

This shows the sequence for a minor (3rd level) release.

|  |  |  |  |
| --- | --- | --- | --- |
| Event | Rlse Version | Beta Version | Dev Version |
| Normal dev, between releases | 5.6.2GA | N/A | 5.6.3Dev |
| Start Release - Beta1 | 5.6.2GA | **5.6.3Beta1** | **5.6.4Dev** |
| Beta2 | 5.6.2GA | **5.6.3Beta2** | 5.6.4Dev |
| Release – 5.6.3 | **5.6.3GA** | N/A | 5.6.4Dev |
| Hot Site | **5.6.3HS1** |  | 5.6.4Dev |

This shows the sequence for a 2nd level release. The sequence for a 1st level release is an exercise for the reader

|  |  |  |  |
| --- | --- | --- | --- |
| Event | Rlse Version | Beta Version | Dev Version |
| Normal dev, between releases | 5.6.2GA | N/A | 5.6.3Dev |
| Start Release - Beta1 | 5.6.2GA | **5.7.0Beta1** | **5.7.1Dev** |
| Beta2 | 5.6.2GA | **5.7.0Beta2** | 5.7.1Dev |
| Release – 5.6.3 | **5.7.0GA** | N/A | 5.7.1Dev |
| Hot Site | **5.7.0HS1** | N/A | 5.7.1Dev |

There should be minimal changes in the Beta or Hot Site releases.

There should be no changes between the last Beta and the GA release.

These releases are for bug fixes, not new features.

Fixes in Beta and Hot Site releases are duplicated in the **default** branch. It is recommended that these fixes be made in the Release branch and merged into the **default** branch.

The merge may complain about a conflict in ProgramVersion.cmake. The VERSION values should be as described above.

We only support the last released version.

# Text Files

All text files (Params, Demos, Layout,...) use the UTF-8 character set and Unix-style line terminators (Single New Line, no Carriage Return).

# Parameter File Maintenance

## Independence of Parameters in Layout files

When a Turnout or Structure is place on the layout, all of its information is copied to layout file. There is no dependency on the Parameter file.

If a change is made to Parameter file, such as adding art-work to a Turnout d0efinition, the existing copies of that definition can be updated by the *Manage/Update Turnouts and Structures* command.

## CONTENTS Label

Each Parameter file must have a least one CONTENTS line. The first is used as the CONTENTS Label. Any others as used as described below.

Parameter files are distinquished by their CONTENTS Label. A map from CONTENTS Label to file name is maintained. This map is implemented by Preferences entries:

Parameter File Map.Kato Unitrack HO Scale: ../params/HO-Kato.xtp

If the current XTrkCad library directory has changed, then these entries are updated to the new location.

These entries accumulate, they are never purged. But unreferenced CONTENTS Labels are ignoreed.

The active Parameter file list is implemented by Preferences entries:

Parameter File Names.File1: Kato Unitrack HO Scale

Parameter File Names.File2:

The list is terminated by an empty value

## Adding / Deleting Parameter files

A new Parameter file is added to the *parms/* directory and is available when the user accesses the Parameter library.

A deleted Paramter is removed from the params/ directory. If the user had loaded that Parameter file, then an error will be raised saying the Parameter file can not be found and will be moved from the Parameter file list.

## Changing CONTENTS Label and/or FileName

Occansionally you may need to rename a parameter file or change its CONTENTS Label. This can lead to missing file errors.

For example, we have a paramter file **XYZ.xtp** with CONTENTS Label

**CONTENTS XYZ Parameters**

We want to rename this file to **HO-XYZ.xtp** and change its CONTENTS Label

CONTENTS HO XYZ Parameters

You need to do 2 things when you submit an updated file;

1 Add the old CONTENTS line to the new param file (HO-XYZ.xtp). Note: the first line is the new Contents Label

CONTENTS HO XYZ Parameters

**CONTENTS XYZ Parameters**

2. Update xtrkcad.upd with the current date and append the name of the new param file

**...**  
**20220108**  
**...**

**HO-XYZ.xtp**

If you are just updating the Contents Label, you only need to do step 1

If you are just changing the file name, you only need to do step 2.

# Source Layout

## Naming Convention

Source files use the following naming convention, somewhat loosely followed as not all of these are appropriate for all objects:

- c\<object\>.c: This is similar to a class file. It contains the code and data structures for creating, manipulating and deleting the object.

- d\<object\>.c: This code handles the dialog(s) for creating, changing and deleting the object.

- t\<object\h.c: This code handle the drawing of the object.

- c\<object\>.h: Definitions for the methods of the object that need to be visible outside the object class.

For example, there are files ccurve.c, ccurve.h and tcurve.c. There is no dialog for curved track.

# WLib Interaction

**Wlib** packages the interface between the core XTrkCad code and the underlying windowing tool kit (MS-Windows or GTK2). There is currently work to migrate to GTK3, which will be the same toolkiit for all platforms.

* 1. WLib calls mostly create and interact with a variety window objects (wButton, wString, wDraw, collectively known as wControl). Many of these objects notify the core code when certain events occur (button push, text entry,...) via Callback function pointers specified when the object is created.
  2. The Callback also is passed a Context value which was specified at creation. This Context value is a void **\*** value, but sometimes the creator of the object needs a integral valiue. A pair of defines **I2VP()** and **VP2I()** handle the conversion of integral value to the void \* Context. Typicall I2VP() is used when creating window object and VP2L in callbacks.
  3. Sometimes we use a **void \* context** in other places as a generic value(eg paramData\_t) and use L2VP()/VP2L() pairs to access integral values.

# Pretty Printing

Code may be formatted (pretty-printed) using the [astyle](http://astyle.sourceforge.net/) tool with the config file *astylerc* or *AStyle.cfg* found in *$SRCDIR/app/lib*.

If a file is pretty-printed, that must be to only change made for that commit. Do not mix non-formatting changes with pretty-printing.

# Casts

Casts are generally to be avoided. There is no need to cast to or from **void\***. There are a few situations where casts are required:

## Downcast Numeric values

Down casting a wide numeric to a narrower numeric (including floating point to integral)

## Generic Values

Casting a generic call-back value from or to a (void\*) using **I2VP()** and **VP2L**

## Wlib wControl\_t conversion

Converting a wlib callback.

## GetTrkExtraData

# Error Handling

* 1. There are four types of errors:

## User Input Errors

This includes invalid entries on dialogs, invalid object selection / manipulation.

The user is directed to correct the error or reset to a valid state. Invalid dialog values are highlighted and balloon help identifies the problem (typically value out of range),

Otherwise ErrorMessage() can explain the problem. If it is complex, NoticeWindow() can provide details. The user should be returned to a state where they can try again. The message should explain, in user terms, what the problem is and it’s fix. “Ptr == NULL” is not a useful User Input error.

Most messages are in *help/messages.in*. These entries provide a user centric description of the problem and its resolution. The errors are added to the ‘Help/Recent Messages’ list which can direct the user to additional help. If your error does not fit this model (user-centric description and resolution) then you should consider alternatives: InfoMessage(), Logic Error or LOG().

## File Input Errors

These could be corrupt files or new files with unsupported features

Generally we try to recover by skipping past the problem after a error message. InputError() is called which offers the option of terminating processing the file.

Layout files are versioned so new feature that are not backwards compatible will be caught. If the format of the file has been changed and is not backwards comatible, the file VERSION number has to be updated (See **VERSION handling**)

The user is not expected to understand these errors. The corrupt files must be fixed.

## Logic errors

Null pointer checks, array index range, parameter assertions, invalid case/default, ‘Can not happen’ conditions, unable to recover.

*ASSERT(cond)* is called which, if cond is false, displays an Error Message (wNoticeEx), with the condition and location. *ASSERTEX(cond, (mesg) )* includes (mesg) in the report, if extra information is required After the user has the chance of saving their layout, XTrkCad aborts.

The user is not expected to understand this error, but report it to the developers. In most cases the condition and location is sufficient.

Most likely, these errors indicate a bug in the code.

## System/Evironmental errors

Typically, Out-Of-Memory. We exit ASAP.

# Undo Processing

# Draw Model

# Headers

# #ifdef WINDOWS

# VERSION handling

## Release Version

See **Source Control / Releases**

## File Format Version

The format for .xtc, .xtr, .xtp and xtrkcad.cus files is described on the **File Format** page of the XtrkCad wikka page [XTrkCad File Formats](https://xtrkcad-fork.sourceforge.net/Wikka/FileFormats).

**$SRCDIR/**xtrkcad-config.h.in contains the following defines

#define XTRKCAD\_PARAMVERSION (12)

#define XTRKCAD\_PARAMVERSIONVERSION “5.2.0”

#define XTRKCAD\_MIN\_PARAMVERSION (1)

XTRKCAD\_PARAMVERSION is the current version of the File Formats. If the VERSION line in the .xtc or .xtr is greater then this value then file can only be opened by an updated version of XTrkCad.

XTRKCAD\_PARAMVERSIONVERSION is the first version of XTrkCad that supports the specified File Format. This allows us to tell the user the minimum version they need to upgrade to.

XTRKCAD\_MIN\_PARAMVERSION is the first File Format version that that we support. Its value (1) inidicates that we support all versions.

Incrementing XTRKCAD\_PARAMVERSION is a major change as it requires users to upgrade when they load a new .xtc file. It would be required if we add an new object or make major changes to existing objects. We have mechanisms (below) to avoid this.

Parameter files (.xtp and xtrkcad.cus) do not have VERSIONs. Version handling is handled by the remaining options.

## Segment Version

Most objects are defined by Segments such as

M1 0 0.500000 20.375000 15.500000 46.625000 20.250000

M3 0 0.500000 20.375000 15.500000 0 46.625000 20.250000 0 1

This defines 2 segments (type M – Dimension Line) with two Segment Versions (1 and 3). The difference is the first segment is missing elevation values (0) following the Co-ordinates and the flag in the last position.

ReadSegs() will read 6 or 9 values based on the Segment Version. For version 1, default values will be used for the elevations and flag.

The code to process this looks like (hasElev is a flag which is true if the Segment Version is >= 3)

if ( !GetArgs( cp, hasElev?"lwpfpfl":"lwpYpYZ",

&rgb, &s->lineWidth,

&s->u.l.pos[0], &elev0,

&s->u.l.pos[1], &elev1, &option ) ) {

The ‘**Y**’ and ‘**Z**’ (and ‘**X**’) format codes specify that no value is read from the input string, but the corresponding arguments are set to 0. (See GetArgs() definition in fileio.c)

If an old version of XTrkCad reads a new file with segment version it doesn’t understand an InputError will be raised and the segment is ignored. If the segment is part of a TURNOUT definition and is followed by any track segments then Path processing will fail.

Note: ReadSegs() checks that the Segment Version is not greater than the known versions.

## Extensible Segment Format

Most changes to segment definitions are the addtion of values. We can use the **c** format code to handle this.

Consider ReadSegs() for SEG\_FILCRCL;

if (!GetArgs(cp, hasElev ? "lwfpf**c**" : "lwfpY**c**",

...

&cp)) { rc = FALSE; break;}

if (cp) {

if (!GetArgs(cp, "ffd",

... ) { rc = FALSE;break; }

}

The **c** format code places the address of any remaining text (if any) in the corresponding argument **cp** or NULL if there is none. If **cp** is not NULL then there is more to read, otherwise we’re reading an old file.

This may require the definition argurments to be specified in an unnatural order. In this example radius1 is in position 3 and radius2 is in position 6. This is localized to ReadSegs/WriteSegs.

The old code will ignore the extra parameters in the new file allowing for backwards and forwards compatibility.