Psycle Developer Guide

C-Version, Feb 2021 (unfinished)

Introduction

C-Psycle is a remake of Psycle in "C" language (instead of C++) with a slightly different concept. It is located under the "cpsycle" folder in the SVN.

This new version has the goal of being compatible with mfc-psycle and sharing as much features as possible, at the same time than adding new features, like scopes and pianoroll, and a "rack" view rather than a wire view.

Work is done to reach the point of releasing a 1.0 version, although it is not there yet.

If anyone wants to try it, it should be easier to compile with visual studio 2019 than Psycle, but at this point it still requires Psycle plugins and some manual changes to code for correct configuration.

Other than that, it starts to be usable, even if it still can't play all songs correctly.

The core of cpsycle is sequencer based (with events), not tracker based. Nearly all timing problems could be resolved but need more testing. Main work on compatibility needs still be done on XMSampler, MIDI support and the extension oft he file format.

Crossform portability should be reached in a step by step process, separating win32 from platform independend code. This was already done with the audio core implementing audio drivers and the ui separating first from the audio core and putting win 32 calls behind a bridge. After the arrival of the new programmers the ui implementation could be exchanged by a linux toolkit.

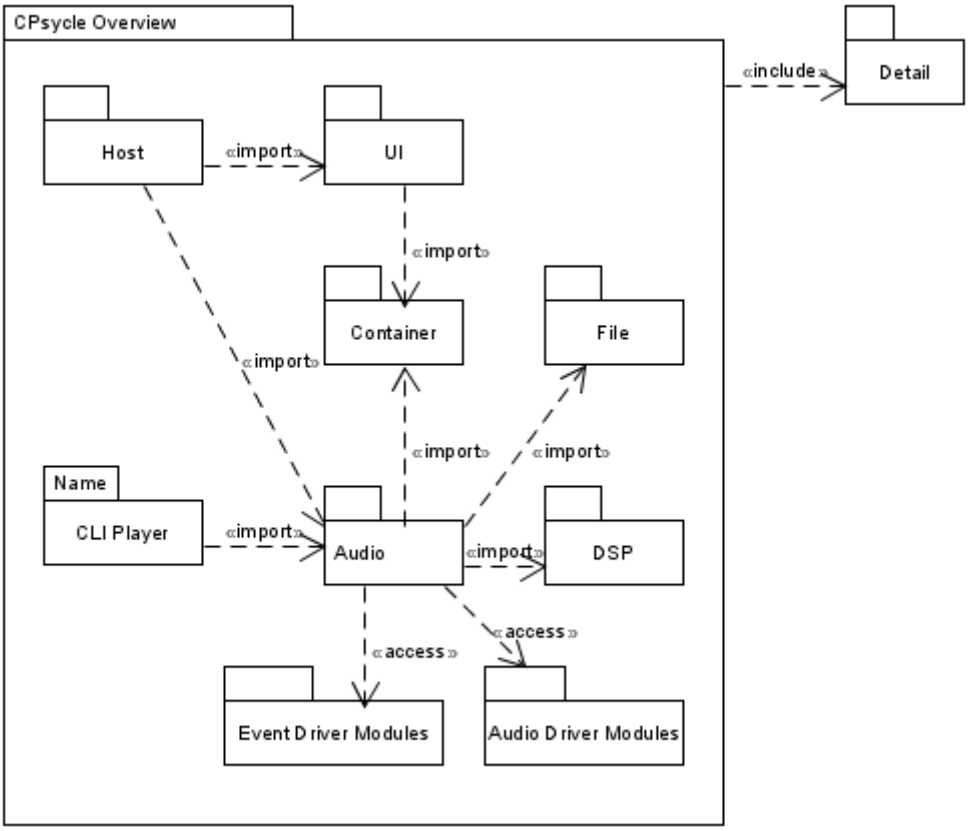
Structure of Psycle

Psycle is divided in two subsystems,

- the audio engine and
- the ui

The helpers dsp, container and file assist these subsystems.

The audio and event drivers will be loaded by the audio engine at runtime.


CPsycle Overview
«include»
Detail
Host
«import»
UI
«import»
Container
File
«import»
«import»
«import»
Name
CLI Player
«import»
Audio
«import»
DSP
«access»
«access»
Event Driver Modules
Audio Driver Modules

The Host

The host is the application the user starts. The program begins in psycle.c, first, initializing the ui, creating the mainframe and starting the ui event loop and finally leaves the program, if a close event occurs.

Depending on the platform the main entry point differs. Windows uses WinMain as entry point, other toolkits will use int main(int argc, char** argv)

DIVERSALIS, a collection of platform specific defines, is used to detect the platform and needed main entry. To leave the platform dependend startup, psycle_run is called as platform independed main entry point.

psycle_run:

1. The app is added to the enviremont path, that the scilexer module can be found.

Scilexer is an open source editor component, psycle uses since version 1.12.

2. psy_ui_app_init(&app, psy_ui_DARKTHEME, instance);

The ui is initialized with a darktheme. (psy_ui_LIGHTHEME starts a light theme).

3. mainframe = (MainFrame*)malloc(sizeof(MainFrame));

Mainframe is found in mainframe.c and is the composite of all views psycle will present to the user. Workspace holds the project song and configurations.

4. psy_ui_component_showstate(&mainframe->component, SW_MAXIMIZE);

Psycle is displayed.

5. Now the app main loop is started:

err = psy_ui_app_run(&app);

The main loop waits for an event (mouse, keyevent..) and triggers callbacks to the host. In case of a close event the main loop will be terminated.

6. The heap storage of the mainframe is cleaned up and the env path is restored

7. The app returns the status to the operation system and the process is terminated.

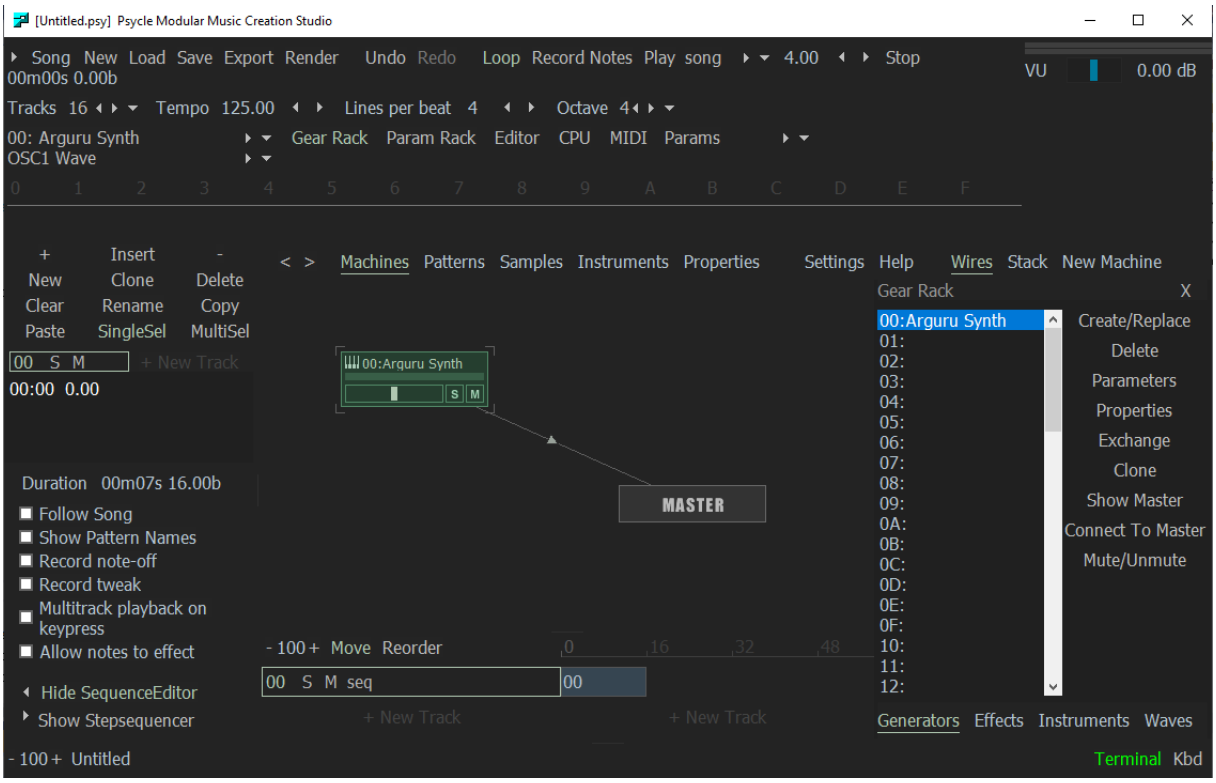
The Mainframe

The mainframe consists oft two basic systems:

The ui views and the workspace.

1. Machines, Patterns, Samples, Instruments, Help, SequencerView, and more …
2. The workspace contains the currenct song, owns the audio players and the configuration files.

Psycle is mainview orientated decorated with sidebars. The mainview structure is realized with a psy_ui_Notebook, that allows to switch components. Tabbar controls the notebook and offers the user tabs to switch it.



Workspace

The workspace connects the player with the psycle host ui and configures both

psy_audio_MachineCallback

^

Workspace

<>---- PsycleConfig; host

<>---- ViewHistory

<>---- psy_audio_Player; audio imports

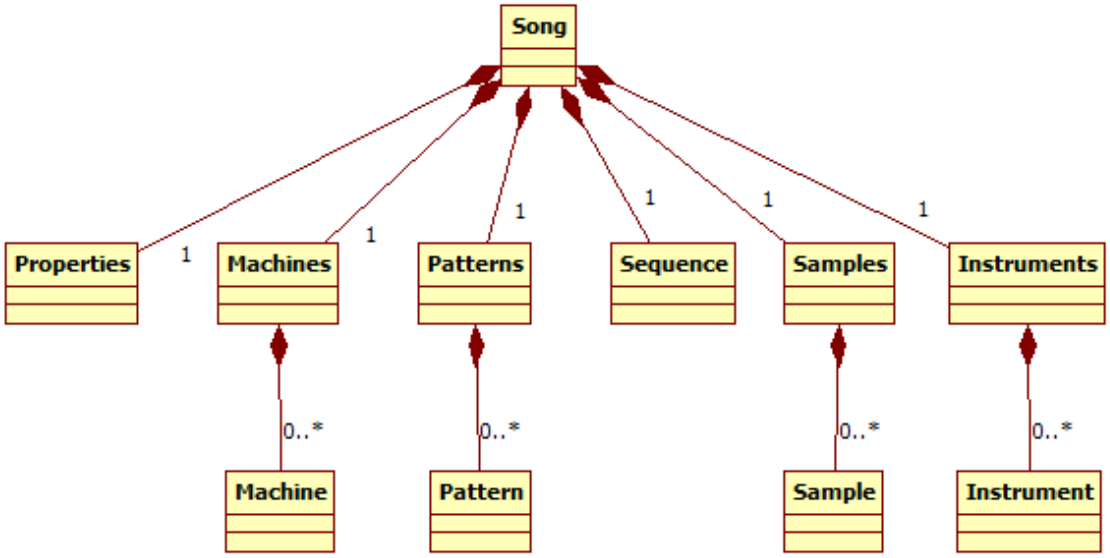
<>---- psy_audio_MachineFactory

<>---- psy_audio_PluginCatcher

<>---- psy_audio_Song

The Song

psy_audio_Song hold everything comprising a "tracker module", this include patterns, pattern sequence, machines and their initial parameters and coordinates, wavetables, ...



Song hold everything comprising a "tracker module", this include patterns, sequence, machines, samples(wavetables), and instruments

The Player

The player controls the audio drivers. It loads a driver and has a work callback. The audio driver will call them if the soundcard buffer needs to be filled. Usually the player work callback will be in the audio driver thread (different than the ui thread).

Work callback:

psy_dsp_amp_t* psy_audio_player_work(psy_audio_Player* self, int* numsamples, int* hostisplaying)

The Player first fills its own buffer defined in player:

static psy_dsp_amp_t bufferdriver[MAX_SAMPLES_WORKFN];

with the numsamples the driver asked for. The buffer will be interleaved (mix of the channels in one stream.) Work returns a pointer to this interleaved buffer and the audio driver can use (copy) the needed frames. To fill this buffer the player has a reference to a song. With help oft the song, the player has a helper, psy_audio_Sequencer, that processes the patterns oft the song. The player will then work the machines with the current events, the sequencer has collected. psy_audio_Machines compute the machine path determined by psy_audio_Connections, the wires of the Machines, from their leafs to the Master.

Player will not work all samples at once, but splits the fill amount into psy_audio_MAX_STREAM_SIZE chunks (256). This is done for psycle plugins to maintain compatibility. (e.g. the sampler ps1 slides used this amount as tick). To maintain compatibility the amount is further splitted if a new trackerline event occurs to notify the machines of this change. This is done because psycle was a pure tracker and worked this way:

1. Work machines till all frames of a line are worked
2. Notify machine line has ended
3. Execute global events (e.g. bpm change)
4. Notify all machines a newline has started
5. Execute line events of the pattern (send a seqtick to the machines).The plugins will setup their voices to work the events
6. Repeat step 1 until all numsamples are worked and return buffer to the audiodriver

The new version is sequencer orientated. This has two mayor differences.

1. The timestamp of an event isn't the line number, but a position in beats.
2. The player playposition is in beats, not in line numbers, frames or ticks

Events can occur at any time, not only at line start.

The sequencer works this way:

1. Calculate amount of frames of the interval to beats using the current bpm tempo.
2. Update linetickcount (in beats) and split current amount to notify linecount and send events that are on linestart. (for compatibility)
3. Search for all pattern events in this time interval
4. Timestamp all events of this interval with an offset in frames relative to the splitted interval start.
5. Work machine with this event list
6. Increase playposistion with interval beat time.
7. Repeat step 1 until all numsamples are worked and return buffer to the audiodriver

Vst plugins and psycle effects work without problems, because they dont rely on ticks. Problememetic are the samplers psycle uses. Sampler PS1 was rewritten, XMSampler isn't tested enough. For this machines the linetickcount (in beat unit) splits the work amount again to ensure the samplers get their events at linestart. For events occuring inside, the samplers will have problems with the effect processing. On the other side, this newline split can make problems at some vsts that usually are getting equal process intervals. This problem have all programs that maintain tracker functionality. A vst plugin should handle every amount of samples correct.

Retrigger/Delay/etc..

Since psycle used lines/ticks instead of beat timing and events occured only at line start, positioning in one line was done using a tick offset and a global retrigger delay array in the player. Each plugin api had an own part to handle this retrigger delay code. The sequencer insted delays events changing the beat position and new retrigger events are cloned and put at the later beat positions. Smooth tweaks (tws) are generated the same way generating 64 new tws events matching the line.

Ticks occur in the samplers, too. The samplers alter the effects each tick (standard is 24 ticks per beat). The samplers used the global player retrigger array to time this ticks. This was replaced by a new class TickTimer, that is resetted at each newline event. Each sampler has an own instance of a TickTimer. This removes the need to work with the player but adds redundancy counting the ticks. The Extraticks per beat were introduced to handle bpm timings better. To reimplement this, the sequencer lpb speed will be altered to match this extra ticks. Normally this isn't needed anymore because the play time can be changed continious by the lpb speed in the sequencer.

Accuracy

Since the tracker engine counted frames and the sequencer adds up real values, rounding errors occur. This means, the tick timer in the sampler may differ by one or more samples over the timer or the seqtick at line start may be some samples offseted. This is corrected by the ticktimer class resetting the counter at each newline notify.

The Sequencer

The sequencer helps the player:

1. Increase the playposition
2. Find the newline position
3. Prepare event lists, the machine works in the interval

To use the sequencer the player will call

- void psy_audio_sequencer_frametick(psy_audio_Sequencer*, uintptr_t numsamples)

and before machine_work is called

- events = psy_audio_sequencer_timedevents(&self->sequencer, slot, amount);

to get the timed events, the machine should process. The sequencer collects the pattern events in the sequence oft the current song.

Sequence

Sequence is part of Song. A sequence has a ordered lists of patterns. A sequence item is called SequenceEntry. A SequenceEntry has a pattern index and a reposition offset to allow breaks between two entries. The pattern index refers to a pattern in patterns. A Pattern contains a list of PatternEntries with a time offset in beats and a trackerchannel index. A PatternEntry has a list of PatternEvents a machine can process.

To address a SequenceEntry, two methods can be used

1. a OrderIndex (trackindex, order(row)index)
2. a SequenceIterator

OrderIndexes are used by the ui to have a simple access to the sequence

Iterators are used by the Sequencer for a fast traverse over the sequence, a concept developed in Freepsycle.

Mulisequence structure of a sequence

The sequence is written as a mulisequence structure and stores a list of psy_audio_SequenceTrack. The number of tracks is called sequencewidth. Each track contains a list of SequenceEntries.

A SequenceTrackIterator refers to one SequenceTrack and defines a pattern entry position inside the track. With a SequenceTrackIterator the sequencer can advance through a sequence track.

psy_audio_SequenceTrackIterator:

```
typedef struct {
    psy_audio_Patterns* patterns;
    psy_audio_SequenceEntryNode* sequencentrynode;
    psy_audio_PatternNode* patternnode;
    psy_audio_Pattern* pattern;
} psy_audio_SequenceTrackIterator;
```

- Sequencentrynode points to the current sequenceentry of the track
- patternode points to the current pattern entry in the pattern
- pattern is a reference to the current pattern
- patterns is a reference to the pattern pool.

The Sequencer walks through the sequence with

- psy_audio_sequencetrackiterator_inc(psy_audio_SequenceTrackIterator*);
  and
  Increases the position to the next pattern entry
- psy_audio_sequencetrackiterator_inc_entry(psy_audio_SequenceTrackIterator*)
  Jumps to the start of the next sequence entry

Sequencer work

The Sequencer operates with the SequenceTrackIterators and stores them in a list of SequencerTracks.

```
typedef struct {
    psy_audio_SequenceTrack* track;
    psy_audio_SequencerTrackState state;
    psy_audio_SequenceTrackIterator* iterator;
    uintptr_t channeloffset;
```

} psy_audio_SequencerTrack;

The psy_audio_SequencerTrackState is used to handle retrigger commands and the channeloffset is used for the multi sequence.

To achieve polyphony a psycle plugin has separate pattern channels from 0 to 64. Each channel plays independently from the other and a plugin allocates a new voice for it. If the Sequencer would play a sequence track on the same pattern channel than a other sequence track already adressed, they would collide and share the voices. To avoid that a channeloffset is added to each channel on a new sequencer track.

Since the psycle native plugins can only handle 64 channels, a special class, Logicalchannel, will keep a list of free channels and map the channeloffset to the 64 physical channels. LogicalChannel is part of the plugin hosts, that can only handle 64 tracks.

Sequencer Work Flow:

To start the sequencer first a position needs to be set:

```
void psy_audio_sequencer_setposition(psy_audio_Sequencer* self,
                psy_dsp_big_beat_t offset)
```

The Sequencer collects events with different lists:

event list : List for the current interval

delayed list: List for retrigger or delayed events

The difference between delayed and the event list is the timestamp. A event list has a delta offset from the interval start. The delayed list stores the absolute song position. In both cases delta is used to store the timestamp but meaning something different.

Now the event lists will be cleared, the trackiterators cleared and new ones created.

- psy_audio_sequencer_clearevents(self);
- psy_audio_sequencer_cleardelayed(self);
- psy_audio_sequencer_clearcurrtracks(self);
- psy_audio_sequencer_makecurrtracks(self, offset);

```
void psy_audio_sequencer_start(psy_audio_Sequencer*);
```

Resets the loops and updates beatspersample, the sequencerspeed:

psy_audio_sequencer_compute_beatspersample(self);

The player calls in work:

```
void psy_audio_sequencer_frametick(psy_audio_Sequencer* self, uintptr_t
        numframes)
```

frametick converts the frames to beats and calls:

```
psy_audio_sequencer_tick(self, psy_audio_sequencer_frametooffset(self,
                numframes));
```

Tick will mainly do two things:

psy_audio_sequencer_insertevents(self);

This will add the events inside the interval

psy_audio_sequencer_insertdelayedevents(self);

This will add delayed events, if they are in the time interval

Finally psy_audio_sequencer_sortevents(self); sorts the events by timestamp and track.

psy_audio_sequencer_insertevents:

1. traverses the sequencer tracks

   for (p = self->currtracks; p != NULL; psy_list_next(&p))

2. gets the currents sequence entry

   psy_audio_sequencetrackiterator_entry(track->iterator);

3. increments the iterator:

psy_audio_sequencetrackiterator_inc_entry

4. Checks if the iterator position is in the current time interval:

   psy_audio_sequencetrackiterator_offset(track->iterator);

   psy_audio_sequencer_isoffsetinwindow(self, offset)

5. If : psy_audio_sequencer_executeline

psy_audio_sequencer_executeline:

First execute global commands:

psy_audio_sequencer_executeglobalcommands

Now execute all events:

psy_audio_sequencer_addsequenceevent(self,

psy_audio_patternnode_entry(p), track, rowoffset);

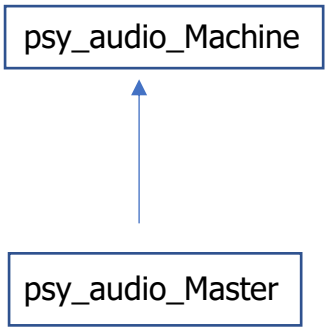
psy_audio_Machine

MachineInterface

MachineHostCallback

Internal Machines

Master



The Master is the root node of the Machines. Any machine that wants to produce sound must be connected to the master. The player copies and interleaves the output buffer of the master to the current audio driver buffer. The master handles global volume changes generated by the sequencer. Connected input wires can be tweaked.

Virtual Generators

In essence, a Virtual generator is an alias for a sampler number and instrument number. Virtual Generators have an index number from 81 to FE, and this index is associated to a pair of "Sampled instrument" and "Machine index".

But a Virtual generator is not only an easier way to use a sampled instrument. It also offers another great feature: a volume column. Since the auxiliary(inst) column remains unused with a virtual generator (because the instrument index is already

known), it is possible now to add a command here that will get translated as the 0Cxx command if the machine is a sampler, or as the 1Exx command, if the machine is Sampulse. (see Psycle Help v. 1.3.0)

A VirtualGenerator works like the NoteDuplicator, delegates and translates the inst to volume column in VirtualGenerator::Seqtick to the real machine with help of the stored inst/machine pair Additional the buffer memory of the real machine will be returned by VirtualGenerator::buffermemory. (Todo: return instrument voice buffer, not recorded separatly yet, of the sampler). The buffer memory is used in the scope monitors. (WireView, PatternTrackScopes)

AudioDriver

The Interface

EventDriver

The Interface

Dsp

Envelope

Filter

Resampler

Container

List

Hashtable

Property

Signals

File

Dir

PropertiesIO

UI

Crossplatform

The UI is organized as bridge allowing different implementations and a win32 implementation is so far developed.

Screen Resolutions

The UI operates with two units, px and em. Em uses the size of the current font instead of pixel values.

Skinning

There exist different independend themes:

1. MachineView
2. PatternView
3. ParamView
4. App Theme: light or dark

The Bridge

Host side

psy_ui_App:

Controls the event main loop, initializes the ImpFactories and stores defaults for the app theme.

ImpFactory:

Factory for the implementations:

psy_ui_Component: Base component of the host side components of the bridge. Delegates the functionality to an implementation.

Implementation of the Bridge

psy_ui_ComponentImp:

Base implementation component of the implementation side of the bridge. Receives the delegated calls of the host side of the bridge.

psy_ui_CheckBoxImp: Displays a checkbox

psy_ui_ListBoxImp: Displays a listbox

Object-oriented techniques in C

Although written in C, Psycle is object orientated written. The techniques used are derived from Dmitry Frank Object-oriented techniques in C and explained at https://dmitryfrank.com/articles/oop_in_c

Psycle avoids the use of multiple inheritance and uses only a subset of that techniques. It follows a short summary of that subset and some common namings psycle uses:

All classes are realized with a struct typedef and organized this way.

typedef : avoids the need to use the struct keyword
public section:
– if inherits: the base class
- signals (with public access)
internal
-private members
-references: pointers to shared data

Construction

init/dispose:

All structs have a

- init: initializes the members
- dispose: cleans up the members

method. The first parameter of all object methods is a context pointer „self" setting the object (instance) identity. The context pointer „self" must point to an already allocated space either on the heap or on the stack. Init and dispose don't do any allocation or deallocation of the own struct memory but data members can be allocated on the heap and dispose needs to clean them up.

Alloc/allocinit/deallocate

If structs are mainly allocated by the heap, they have further:

alloc, allocinit and deallocate.

alloc: aquires enough size on the heap but doesn't initialize the class

allocinit: allocates the memory of the struct and calls the init method on it.

deallocate: calls first dispose and then frees the memory

3. Inheritance/Interfaces

If Psycle derives a base class, the first member of the subclass is the base:

typedef DerivedA { struct Base base; int a; } DerivedA;

typedef DerivedB { struct Base base; int b } DerivedB;

The C standard ensures that the base and derived struct will point to the same aligned memory block and makes a cast possible:

Base* base = (Base*)aDerivedA;

Base* base = (Base*)aDerivedB;

Each struct becomes a cast method _base

Base* deriveda_base(DerivedA* self) { return &self->base }

Base* derivedb_base(DerivedB* self) { return &self->base }

Virtual Methods

Virtual methods are realized with vtables. Each class that implements vtables has a helper struct with functionpointer declarations. The derived class itself only has a pointer of this helper struct. If the first time a derived object is initialized a static vtable will be initialized aswell. The function pointers oft the derived class will be set to its implementation. If the second time a derived object is initialized just the pointer to that vtable will be set to the new instance. Vtable calls are often wrapped inside an inline method:

- INLINE base_onload(Base* self) { self->vtable->onload(self); }
- a call: base_onload(deriveda_base(&adireveda))

Context Type:

The derived methods use as context the derived struct type and not the base one. The compiler doesn't know anything about that and warns or erros. A typecast is done in the vtable init:

vtable.onload = (fp_onload)derived_onload;

This has the drawbacks that wrong type casting and wrong signatures aren't detected and special care has to be taken.

(Another approach would be to use the base type in the derived implementation and do a cast at the begin of each method to the derived type. This method has the advantage that the signature is still checked but adds quite an overhead of code to each method and currently not used by psycle except in some audio drivers.)

Signals

Psycle uses at many points signals to avoid timer polling for state changes and reacts instead to the notification of the signal. At some places psycle uses them for specialization if inheritance is not appropiate, e.g. the ScrollZoom of Psycle can update custom drawing code with a signal. Signals allowing communication between objects not related to an inheritance hierachy. They have a container with slots containing a function callback and a context (destination object) pointer. In difference to c++ signals there is no typesafety for the context and signature and no special memory managment of autodisconnecting. Before the context becomes invalid (disposed) the context must disconnect itself from the signal. Depending of the lifetime of sender and destination object this may be or not nesessary. The slot must be disconnected, if the signal owner lives longer than the destination.

Usage

psy_signal_connect:

adds a slot (context, function callback pointer) to the slot list.

psy_signal_emit:

Calls each slot with the context and sender (passed to emit) pointer

and their additional argument(s) (passed to emit).

Example: psy_ui_Button <>---- psy_Signal signal_clicked

psy_Button:

- init/dispose signal
- App event loop generates a click event and calls
- psy_signal_emit(&self->signal_clicked, self, 0);

  1. the Signal, 2. sender (self/Button instance), 3 zero arguments.

  The slot list delegates the event to its destinations (MainFrame).

MainFrame <>----- psy_Button

init: psy_signal_connect(&self->button.signal_clicked, self, onclicked)

args: 1. the button signal, 2. destination of the click (self/mainframe),

      3. the function callback pointer called by the button signal

void onclicked(MainFrame* self, psy_ui_Button* sender) { }

Arguments:

psy_signal_emit: variable number of generic void* pointers

psy_signal_emit_int: int argument

psy_signal_emit_float: float argument

Project Organisation

Build

Header files for platform tweaks and configuration are located in:

/cpsycle/detail/

prefix.h

Definitions for psycle code that must come before any other header file. This is the first include in every c file, but not in header files.

source.c file:

1: #include "../../detail/prefix.h"

2:

3: #include "source.h "

At the moment it avoid warnings of MSVC about ISO C functions and enables/disables leak detection. To enable leak detection in MSVC remove the comments of:

// #define _CRTDBG_MAP_ALLOC

// #include <crtdbg.h>

Windows

Visual Studio 2019:

Open cpsycle.msvc-2019.sln and select a build configuration:

Debug Win32/Release Win32/Debug x64/Release x64 and build the solution.

If the audio drivers aren‘t build at the first run, press Ctrl + Shift + B to start the build again. The drivers will be generated depending of the configuration in cpsycle\Release\x86 cpsycle\Debug\x86 cpsycle\Release\x64 or cpsycle\Debug\x64.

Plugins aren’t at the moment build and have tob e taken from an mfc-psycle release.

Some dlls like universalis.dll and boost dlls have to be copied to the cpsycle host.exe output directory. (See readme file in /cpsycle.)

Visual Studio 2008:

The project file is cpsycle.msvc-2008.sln but not often updated. If files are missing, add them in the project explorer.

VC2006: The project file is cpsycle.dsw and mostly out of date. If files are missing, add them in the project explorer.

GCC:

Currently out of date.

Psycle Glossary

Here are some names explained that occur quite often in Psycle.

Alk: A Psycle Coder, Tester and chip musician.

Auxslot: Psycle has an instrument pool all samplers and machines share. For playback and pattern edit an instrument index is stored. The Auxslot extends this global concept. Each machine can have its own slot for universal use, an own instrumentlist or something different like the selection of midichannels. Depending on the machine selected in the machineview, the host displays either the instrumentindex or the auxslot stored in the machine that uses auxcolumns.

Arguru: Creator of Psycle

Bohan: A Psycle developer and build master

Buffer: Either some amount of memory, like a buffer for text, or an audio buffer. An audio buffer is a container of wave pointers (at the moment float) combining one or more channels. A buffer can be shared or can be the owner oft he data.

BusMachine: todo (old psy2 concept abandoned in psy3 but naming still occurs)

Channel: A channel can refer to different things. A channel can mean a pattern channel/track or a channel can mean a channel of an sample(wavetable), e.g in case of stereo left or right channel.

CPsycle: C Version of Psycle trying to bring mfc psycle q-psycle and freepsycle

FreePsycle: Test Host by Bohan. Many ideas realized in mfc psycle.

FT2: Fast Tracker 2, a standard tracker. Psycle uses some keyboard mappings besides impulse tracker

GPL: An open source licence by Richard Stallman

IT2: Impulse Tracker 2, a standard tracker. Psycle uses some keyboard mappings besides FastTracker

JAZ: [JAZ]/JosepMa Developer since release 1.5

LPB: lines per beat. Defines the raster the pattern view shows tracker lines

Machine: Base Interface of generators and effects that can be processed and wired by the player and mixer.

MFC: C++ Windows Toolkit Psycle was primarily written in.

Module: Other word for song but more used to fasttracker or impulse tracker module import/export.

Psycedelics: All musicians and developers that use or write Psycle.

QPsycle: qt version of psycle.

Sequencer: Timing of events and playposition is in beats opposite to a tracker that operates on ticks.

Ticks: The name tick has multiple meanings:

1. line tick: refers to the advance from on tracker line tot he other
2. ticks per beat. Depending on the speed of one beat, the beat is divided by the tick number, at default 24, and the sequencer or the sampler use it as timing for effect changes, delays or retrigger timing of notes.
3. Sampler ps1 tick: mostly occurs after 256 samples, but not always.
4. Other trackers can have different tick definitions depending on their hardware.

Track: A track can refer to multiple things. Mostly it refers to a pattern channel, a column in the patternview that allow plugins to use polyphonic (parallel) voices. A Track can also be a SequenceTrack, a column in the SequenceView, that allows to play parallel different patterns.

Tracker: Line orientated player (opposite to Sequencer)

Work: Anything that must be updated very often, Machine work, player audio driver work callback, etc .. Vst Plugins use instead the name process meaning the same.