Better EMR Phone Documentation

Release v3.2.0

Elckarow

Jan 28, 2024

CONTENTS

This framework is intended for Ren'Py 7.4.9+ only. Any Ren'Py version below 7.4.9 will lead to errors and unexpected behavior.

As the name indicates, the first version of this phone was first seen in the DDLC mod Doki Doki: Exit Music REDUX.

It was pretty good (visually that is). The back-end code was… *a bit spaghetti not gonna lie*. Since I needed a phone for my own (well not my *own* per se, I simply am the lead coder) DDLC mod Doki Doki Undercurrents (go check it out), I decided to… remake the whole thing. Took a few tries but here we are I guess.

Credit goes to Wretched Team for the original version.

If you are not using this framework for a DDLC mod, the use of Monika's icon, Natsuki's icon, Yuri's icon, Sayori's icon and MC's icon is forbidden.

Be sure to credit Elckarow#8399 for making this framework (or I will haunt your dreams).

CHAPTER

ONE

CHARACTERS

The following functions and classes are defined in the **phone.character** *namespace.*

1.1 The phone `Character` class

`class Character(object)`

Not to confuse with Ren'Py's `Character` objects (we're in the `phone.character` namespace remember), these objects form the core of the framework.

When talking about a phone `Character` object, I will (most of the time) use this notation:

`*character*`

which means the phone `Character` object itself or its `key` (you'll see what it is just below).

`def __init__(self, name, icon, key, cps, color)`

- `name`: a string, the name of the character.
- `icon`: a displayable.
- `key`: any hashable object that is not `None`. this must be a unique object proper to this phone `Character` object.
- `cps`: an integer.
- `color`: any valid color value.

Once the object has been created, the `name`, `icon` and `cps` fields can be safely changed. The `key` and `color` fields are read-only.

`def get_textbox(self)`

Returns a solid with rounded corners of the character's color and the radius given by `phone.config.textbox_radius`.

`def is_pov(self)` ***(property)***

Returns whether the character's key is equal to the store variable `store.pov_key`.

`def get_typing_delay(self, message, substitute=True)`

Returns a number of second this character would be typing out `message` (a string). If `substitute` is true, text substitution occurs before computing the time.

These objects are *hashable* (their key will be hashed).

When creating a phone `Character` object, you must use `default` and not `define` .

1.2 Functions

def character(x)

Returns the `*character*` `x`. If `x` is a phone `Character`, returns it, otherwise it is taken to be the key to a phone `Character` object and will return that object, or raise `KeyError` if no phone `Character` object has a key like this (if `None` is passed, `store.pov_key` is used).

def has_character(key)

Returns true if there is a phone `Character` with the key `key`, or fals if there is not.

def get_textbox(color)

Returns a solid with rounded corners of color `color` and the radius given by `phone.config.textbox_radius`.

def get_all()

Returns a list of all phone `Character` objects defined.

1.3 Example

```
# default /!\
default p_eileen = phone.character.Character("Eileen", phone.asset("default_icon.png"),
↪"eileen", 20, "#fff")
```

CHAPTER

TWO

GROUP CHATS

The following functions and classes are defined in the **phone.group_chat** *namespace.*

2.1 The `GroupChat` class

`class GroupChat(object)`

The core of the discussion part of this framework.

`def __init__(self, name, icon, key, transient=False)`

- `name`: a string, the name of the character.
- `icon`: a displayable.
- `key`: any hashable object that is not `None`. this must be a unique object proper to this `phone.group_chat.GroupChat` object.
- `transient`: a boolean. If true, the group chat is cleared once the discussion is over.

Once created, the following fields can be read and safely modified:

- `name`
- `icon`

while following fields should be read only:

- `unread`
- `date`
- `transient`

`def add_character(self, char)`

Adds the `*character*` `char` to this group chat, saves the group chat in the `*character*`'s list of known group chats, and returns the group chat.

`def remove_character(self, char)`

Removes the `*character*` `char` from this group chat, and removes the group chat from the `*character*`'s known group chats.

`def number_of_messages_sent(self, char)`

Returns the number of messages sent by the `*character*` `char`. If `None` is passed, returns the total number of messages sent.

`def clear(self)`

Clears the group chat's history.

These objects are *hashable* (their key will be hashed).

2.2 Functions

def group_chat(x)

If `x` is a `phone.group_chat.GroupChat` object, will return that object, otherwise, the group chat with the same key will be returned, or raise a `KeyError` if it wasn't found.

def has_group_chat(key)

Returns if a group chat with the key `key` exists.

def get_all()

Returns a list of every group chats defined.

2.3 Example

```
# default /!\
default eileen_gc = phone.group_chat.GroupChat("Eileen", phone.asset("default_icon.png"),
↪ "eileen_gc").add_character("eileen")
```

or use the `define` clause of the `init phone register` statement.

```
init phone register:
    define "Hello":
        icon "icon.png" add "eileen" key "hello"
```

CHAPTER

THREE

EMOJIS

This framework comes with a built-in emoji system (used in text messages but I guess creators can find usage elsewhere).

The following functions are defined in the **phone.emojis** *namespace.*

This namespace is considered *constant* and should not be modified outside of init time.

`def add(name, emoji)`

This function adds an emoji to the list of known emojis. It take a string `name` (which amy only contain letters, numbers and underscores) and a displayable `emoji`.

`def get(name)`

Returns the emoji with the name `name`. Raises a `KeyError` in case the emoji wasn't found.

A custom text tag is also defined: the `emoji` text tag.

`"This framework is {emoji=poggers}!"` will use the `poggers` emoji. The displayable will be scaled up or down so that it fits in a line of text.

`def format_emoji_tag(s)`

This function formats the string `s` by replacing each `emoji` text tag with a more readable version (basically it replaces the tag with how you'd write it in discord).

```
phone.emoji.format_emoji_tag("This framework is {emoji=poggers}!")
>>> "This framework is :poggers:!"
```

Default emojis include:

- The most used twemojis
- Some emojis from the Doki Doki Undercurrents discord server.

CHAPTER

FOUR

DISCUSSION

probably the longest part of the doc have fun

4.1 Functions

The following functions and variables are defined in the **phone.discussion** *namespace.*

`def remove_text_tags(s)`
Formats the `emoji` text tag in `s` according to `phone.emojis.format_emoji_tag` and removes all other text tags.

`def discussion(gc)`
Starts a discussion with the `*group chat* gc`. If `None` is passed, the current group chat is used. The python equivalent of the `phone discussion` statement.

`def end_discussion()`
Ends the current discussion. The python equivalent of the `phone end discussion` statement.

`def message(sender, message, delay=None)`
Sends a message by the `*character* sender` to the current group chat. Text tags should not be used. Pauses for `delay` seconds after the message's been saved. The python equivalent of the default discussion statement.

`def image(sender, image, time=2.0, delay=None)`
Sends an image by the `*character* sender` to the current group chat. `time` is the time the image is being sent for. The python equivalent of the `image` discussion statement.

`def label(label, delay=0.5)`
Adds a label to the current group chat. The python equivalent of the `label` discussion statement.

`def date(month, day, year, hour, minute, second, delay=0.5, auto=False)`
Adds a date as label to the current group chat. The date is saved to the group chat using `datetime.datetime`. If any of these values are `None`, they are taken from the currently saved date. If any of these values are `True`, they are taken from the date returned by `phone.system.get_date()`. If `auto` is true, sets every values to `True`. The python equivalent of the `time` discussion statement.

`def typing(sender, value, delay=None)`
Simulates the `*character* sender` typing for `value` seconds. If `value` is a string, `sender.get_typing_delay` is called. The python equivalent of the `type` discussion statement.

`def choice(captions, delay=0.3)`
Causes a choice to occur inside the phone, and returns the caption that has been chosen. `captions` is a list of strings. The python equivalent of the `menu` discussion statement.

def audio(sender, audio, time=2.0, delay=None)
Sends an audio by the `*character*` `sender` to the current group chat. `time` is the time the audio is being sent for.

def register_message(group, sender, text)
Saves a message sent by the `*character*` `sender` in the `*group chat*` `group`. This is called automatically by the `phone.discussion.message` function. The python equivalent of the default register statement.

def register_image(group, sender, image)
Saves an image sent by the `*character*` `sender` in the `*group chat*` `group`. This is called automatically by the `phone.discussion.image` function. The python equivalent of the `image` register statement.

def register_label(group, label)
Saves a label in the `*group chat*` `group`. This is called automatically by the `phone.discussion.label` function. The python equivalent of the `label` register statement.

def register_date(group, month, day, year, hour, minute, second, auto=False)
Saves a date in the `*group chat*` `group`. This is called automatically by the `phone.discussion.date` function. The python equivalent of the `time` register statement.

def register_audio(group, sender, audio)
Saves an audio sent by the `*character*` `sender` in the `*group chat*` `group`. This is called automatically by the `phone.discussion.audio` function. The python equivalent of the `audio` register statement.

def sort_messages(key)
Sorts the group chats of the `*character*` `key` in descending order according to their last registered date.

4.2 The statements

The framework comes with its load of custom statements aiming at making it easier to use.

4.2.1 The discussion statements

phone discussion
Used to start a phone discussion. If a simple expression is given, it must be a `*group chat*`. If no simple expression is given (or that `None` is given), it's assumed that a discussion is already going on, and will therefore use the current group chat.

If a block is given, the following statements can be used:

- `` ` ` ``

  The default statement, equivalent of the `phone.discussion.message` function. It expects a `*character*` and a string (flagged as translatable). It accepts the `delay` property (defaults to `None`) which is the time to wait before the next statement executes. A `None` delay will wait for user input. A negative value delay won't wait.

- **image**

  The equivalent of the `phone.discussion.image` function. It expects a `*character*` and a simple expression (the image). It accepts the `time` property (defaults to `2.0`), which is the time the image is being sent for.

- **label**

  The equivalent of the `phone.discussion.label` function. It expects a string (the string is flagged as being translatable). It also accepts the `delay` property (defaults to `0.5`).

- **time**

  The equivalent of the `phone.discussion.date` function. It expects at least one of the following property; `year`, `month`, `day`, `hour`, `minute`, `second`; which can be a number, `None` or `True`, as well as the `auto` property. If one of these is missing, it is retrieved from the current date registered. It also accepts the `delay` property (defaults to 0.5).

- **`type`**
  The equivalent of the `phone.discussion.typing` function. It expects a `*character*` and a `value` property, which can be a number or a string. The string is NOT flagged as translatable. It also accepts the `delay` property (defaults to `None`).
- **`if/elif/else`**
  Does exactly what you'd expect from this statement.
- **`menu`**
  The equivalent of the `phone.discussion.choice` function. It expects a block which can contain the following: The `delay` property, which has to be given before the menu items (defaults to `0.3`). A series of menuitems. A menuitem is a string (flagged as translatable) which may be followed by an `if` clause and a simple expression. If the expression is false, the choice won't appear. The line ends with a colon `:` and must be followed by a block that contains any of the phone discussion statements.
- **`$`**
  The one-line python statement. Executes code in the global store.
- **`python`**
  Works the same way as the normal `python` statement except for one thing: If the `in` clause is given, the substore is created at init 0, unlike the regular `python` statement which does it at early time.
- **`pass`**
  Does nothing.
- **`pause`**
  Same as the regular `pause` statement.

If no block is given, it behaves as if a single `pass` statement was given.

`phone end discussion`
Used to end a phone discussion. It doesn't expect anything.

4.2.2 The register statements

`phone register`
Used to register messages in a group chat. It expects a `*group chat*` and a block (see the part above). It doesn't accept the `type`, `menu`, `$` nor `python` statements, nor the properties related to time (`delay`, `time`, `cps` …).

`init phone register`
Used to register messages in a group chat at init time and / or create a new group chat. The statement is run at init priority 700.

If a `*group chat*` is given, it behaves the same way as the `phone register` statement. If no `*group chat*` is given, the block expects a `define` clause.

The `define` clause expects a string, the name of the group chat, and a block which can contain the following statements:

- **`add`**
  Expects a `*character*`. Will add this `*character*` to the group chat when created.
- **`key`**
  Expects a simple expression. The key of the group chat.

- **icon**
  Expects a displayable. The icon of the group chat.
- **as**
  Expects a dotted name. The group chat will be saved in the global store under this name (as if the group chat was manually created using the `default` statement).
- **transient**
  Optional. If present, the group chat becomes transient. Transient group chats are cleared once the discussion is over.

4.3 Example

```
# create two phone Character objects
default phone_sayori = phone.character.Character("Sayori", phone.asset("sayori_icon.png
↪"), "s", 21,   "#22Abf8")
default phone_mc = phone.character.Character("MC", phone.asset("mc_icon.png"), "mc", 35,
↪"#484848")

# create a group chat manually
default mc_sayo_gc = phone.group_chat.GroupChat("Sayori", phone.asset("sayori_icon.png"),
↪ "mc_sayo"). add_character("mc").add_character("s")

# create another group chat using `init phone register`
# and add a few messages
init phone register:
    define "goofy ahh chat":
        icon phone.asset("sayori_icon.png") key "goofy"
        add "mc" add "s" as goofy
        transient

    time month 1 day 26 year 2013 hour 14 minute 31
    "mc" "Ah!"
    "s" "Boo!"
    "mc" "Ah!"

label phone_discussion_test:
    scene expression "#fdfdfd"
    phone register mc_sayo_gc:  # using the group chat object directly
        time month 5 day 12 year 2015 hour 20 minute 40
        image "s" Solid("#000", xysize=(50, 50))
        "s" "oops"

    "..."
    "Hmm?"
    "A message from Sayori?"

    phone discussion "mc_sayo": # using the gc's key
        pause

    "..."
    "... Really now?"
```

(continues on next page)

(continued from previous page)

```
    phone discussion: # no gc. uses the one used before
        menu:
            "a square?":
                "mc" "a square?"
            "a black square?":
                "mc" "a black square?"

    "..."

    phone discussion:
        time minute 50 # year, month, day, hour are all taken from the date before
        "s" "missinput"
    phone end discussion

    "What an airhead..."

    return
```

CHAPTER

FIVE

PHONE CALLS

The following functions are defined in the **phone.calls** *namespace (not* **phone.call**, *Ren'Py doesn't like me using that).*

Phone calls require a certain type of sayer. The need **those 3** properties:

- `screen` set to `"phone_say"`
- `who_style` set to `"phone_say_label"`
- `what_style` set to `"phone_say_dialogue"`

The rest is as usual.

```
define phone_eileen = Character("Eileen", screen="phone_say", who_style="phone_say_label
↪", what_style="phone_say_dialogue")
```

5.1 Functions

`def call(caller, video=False, nosave=False)`

Starts a phone call with the `*character*` `caller`. If `video` is true, a video call is started (this is ignored if the game is on a version prior to 7.5.0). If `nosave` is true, the call won't be saved to the call history. Replaces the `narrator` with a special narrator. The python equivalent of the `phone call` statement.

`def end_call()`

Ends the current phone call, and registers it for both `*character*`s (the caller and the current pov). Sets the `narrator` back and clears the video call layer. The python equivalent of the `phone end call` statement.

`def register_call(char1, char2, duration=None)`

Saves a call between the `*character*`s `char1` and `char2`. If `duration` is not `None`, it's a float, a number of seconds the call lasted. This is called automatically by the `phone.calls.end_call` function.

5.2 Statements

`phone call`

Used to start a phone call. It expects a `*character*`. If the `video` clause is given, a video call is started. If the `nosave` clause is given, the call won't be saved to the call history.

`phone end call`

Used to end a phone call. It doesn't expect anything.

5.3 Example

```
# define two phone sayers
define phone_s  = Character("Sayori", screen="phone_say", who_style="phone_say_label",␣
↪what_style="phone_say_dialogue")
define phone_mc = Character("MC", screen="phone_say", who_style="phone_say_label", what_
↪style="phone_say_dialogue")

# create the two phone characters
default pc_sayori = phone.character.Character("Sayori", phone.asset("sayori_icon.png"),
↪"s", 21, "#22Abf8")
default pc_mc     = phone.character.Character("MC", phone.asset("mc_icon.png"), "mc", 35,
↪ "#484848")

label phone_call_test:
    phone call "s"
    phone_s "Ohayouuu!!!!!!!!!!!!!!!"
    phone_mc "Hey!"
    "Why is she always this energetic?"
    phone end call
    "..."

    return

label phone_video_call_test:
    show sayori onlayer phone_video_call
    show bg sayori_room onlayer phone_video_call
    phone call "s" video
    phone_s "Hey! That's my room"
    phone end call
    "..."

    return
```

CHAPTER

SIX

WANNA MAKE YOUR OWN APP?

In order to do that, you need **two** things:

- A phone screen
- A `phone.application.Application` object (not necessary but if you want your app to appear on the `phone` screen then yes you need this)

6.1 The _phone screen

To create a phone screen, simply `use` the `_phone` screen, intended as base for all phone screen.

```
screen my_phone_screen():
    use _phone():
        # your screen code goes there
```

The `_phone` screen has 5 parameters:

- `xpos` The xpos of the phone.
- `ypos` The ypos of the phone.
- `xanchor` The xanchor of the phone.
- `yanchor` The yanchor of the phone.
- `horizontal` If true, the phone is displayed horizontally. If false, it is displayed vertically.

Once you've coded your screen, simply call it with `phone.call_screen` or the `PhoneMenu` action and there you go!

6.2 Applications

If you want your app to appear on the `phone` screen, you need to create a `phone.application.Application` object and add it to your `*character*`s.

The following functions and classes are defined in the **phone.application** *namespace.*

`class Application(object)`

`def __init__(self, name, icon, action)`

- `name` A string. The name of the app.
- `icon` A `phone.application.Icon` or `phone.application.GradientBackground` object.

- `action` An action to run when clicking on the app (most of the time it's a `PhoneMenu` action, but any valid action works).

`def Icon(d, size=None, background=None)`
Retuns a displayable used as icon for applications. `d` is a displayable to add on to the icon. `size` is the size that displayable takes. `background`, if not `None`, is a `phone.application.IconBackground` or `phone.application.GradientBackground` object displayed behind `d`.

`def GradientBackground(start_color, end_color, theta=0)`
Retuns a `phone.application.IconBackground` object with a `Gradient` as displayable.

`def IconBackground(d, **kwargs)`
Returns `d` with rounded corners of size `gui.phone_application_icon_size`.

6.3 Example

```
init python:
    my_app = phone.application.Application(
        "my application",
        phone.application.GradientBackground("#5bf676", "#04be25"),
        PhoneMenu("my_phone_screen")
    )
```

Now that you've done this, it's time to add the app to the `*character*`s you've defined, using those two functions.

`def add_application(app, page=0, key=None)`
Adds the application `app` to the known applications for `*character*` `key`. Returns `True` if it succesfully added the app, `False` if it failed, or `None` if Ren'Py is still in init phase.

`def add_app_to_all_characters(app, page=0)`
Same as above but for every `*character*` known at execution time.

To add the app you've created, simply

```
phone.application.add_app_to_all_characters(my_app)
```

and ta-da, your app should appear on the `phone` screen.

6.4 Functions

`def move_application(start, end, key=None)`
`start`/`end` are 3-tuples containing a page, a column and a row. They represent the start/end point. This function swaps the application of coordinates `start` with the one of coordinates `end`.

CHAPTER SEVEN

CONTACTS

Loops over the group chats known to the `*character*` `store.pov_key` at `phone.data[store.pov_key]["group_chats"]` and, for each group chat, displays a button that, when clicked, allows the player to read the chat.

Uses the `phone_contacts` screen.

That's it.

CHAPTER

EIGHT

CALL HISTORY

Not much to say about this one. Loops over the list of saved call at `phone.data[store.pov_key]["call_history"]` and display them alongside the duration if one was given.

Uses the `phone_call_hisory` screen.

That's it.

CHAPTER

NINE

CALENDARS

This one requires a bit more explaination.

Loops over the list of calendars saved at `phone.data[store.pov_key]["calendars"]` and displays them. A day passed is rendered with a gray background. A day that has a description has a `?` added in the top right.

Uses the `phone_calendars` screen.

9.1 The `Calendar` class

The following functions, variables and classes are defined in the **phone.calendar** *namespace.*

`class Calendar(calendar.Calendar)`

`def __init__(self, month, year=2017, first_day=_default_first_day)`

- `month` A valid month from 1-12.
- `year` An integer.
- `first_day` An integer, one of the day constants described below.

`def is_day_passed(self, day)`

Returns whether the day `day` of this calendar is passed, compared to `phone.system.get_date`. `day` has to be a valid day (from 1 to whatever the last day is).

`def lenght(self, offsets=False)`

If `offsets` is false, returns the number of days. If true, also returns the number of "out of range" days (take the 2023 June calendar for instance, the first 4 days and the last day are considered "out of range").

`def get_week_days(self)`

Returns an generator iterating over the days of the week (that are strings, so `"Monday"`, `"Tuesday"`, `etc...`).

These objects are *iterable*. Each iteration will either return `None` if it's an "out of range" day or will return an object that has the following fields:

- `day` The number of the day. Read-only.
- `description` If not `None`, a string.

The dunder method `__getitem__` is also defined. It takes an integer, a valid day, and will return an object as described above, or raise an `IndexError` if it's not a valid day (`my_calendar[1]` will return the first day object, `my_calendar[0]` will raise an `IndexError`).

9.2 Functions and Variables

`days = (...)`

A tuple containing strings corresponding to the week days name. The strings are flagged as translatable.

`months = (...)`

A tuple containing strings corresponding to the months name (similar to `calendar.month_name`, it follows normal convention of January being month number 1, so it has a length of 13 and `months[0]` is the empty string). The strings are flagged as translatable.

The constants `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, `SATURDAY`, `SUNDAY` all represent a week day.

`def get_week_days(first_day=_default_first_day)`

Returns an generator iterating over the days of the week (that are strings, so `"Monday"`, `"Tuesday"`, `etc...`).

`def day_name(year, month, day)`

Returns the day name of the corresponding date.

`def add_calendar(year, month, key=None, first_day=SUNDDAY)`

Creates and adds a calendar to the list of calendars for the `*character*` `key`.

`def add_calendar_to_all_characters(year, month, first_day=SUNDAY)`

Same as above but for every `*character*` known at execution time.

`def get_calendar(year, month, key=None)`

Returns the calendar for the `*character*` `key` that has the corresponding year and month number. `None` is returned if no such calendar was found.

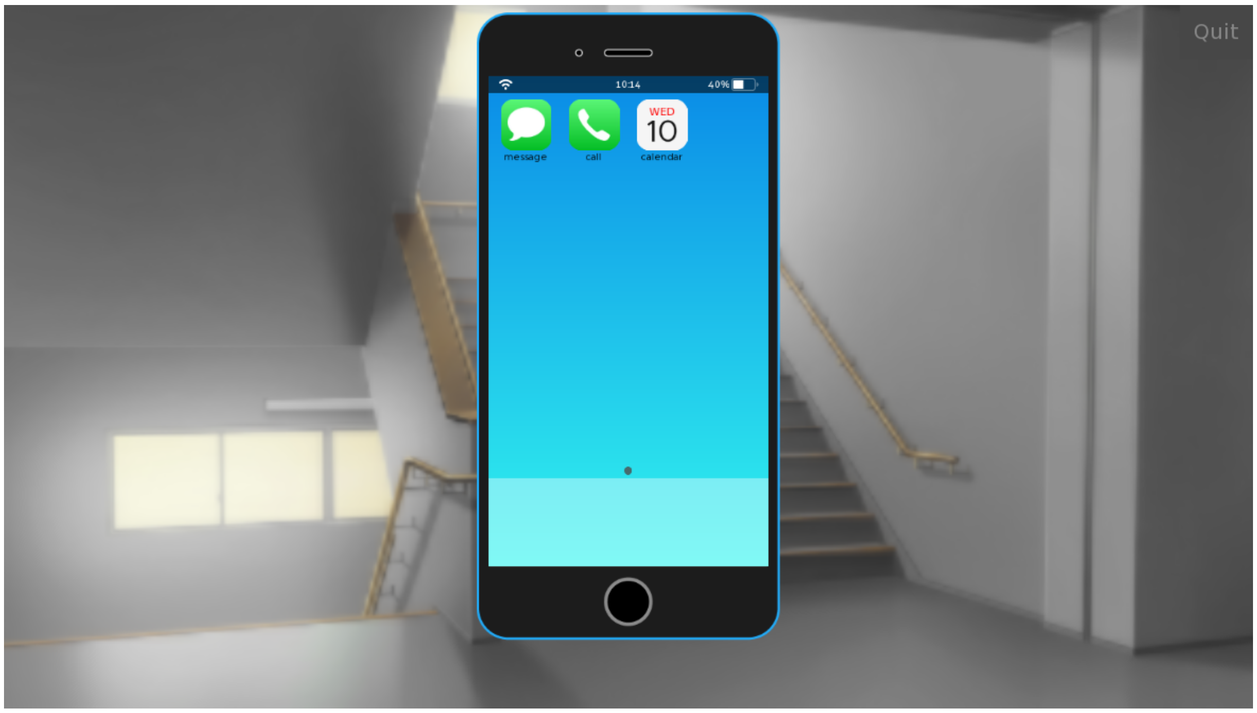
CHAPTER

TEN

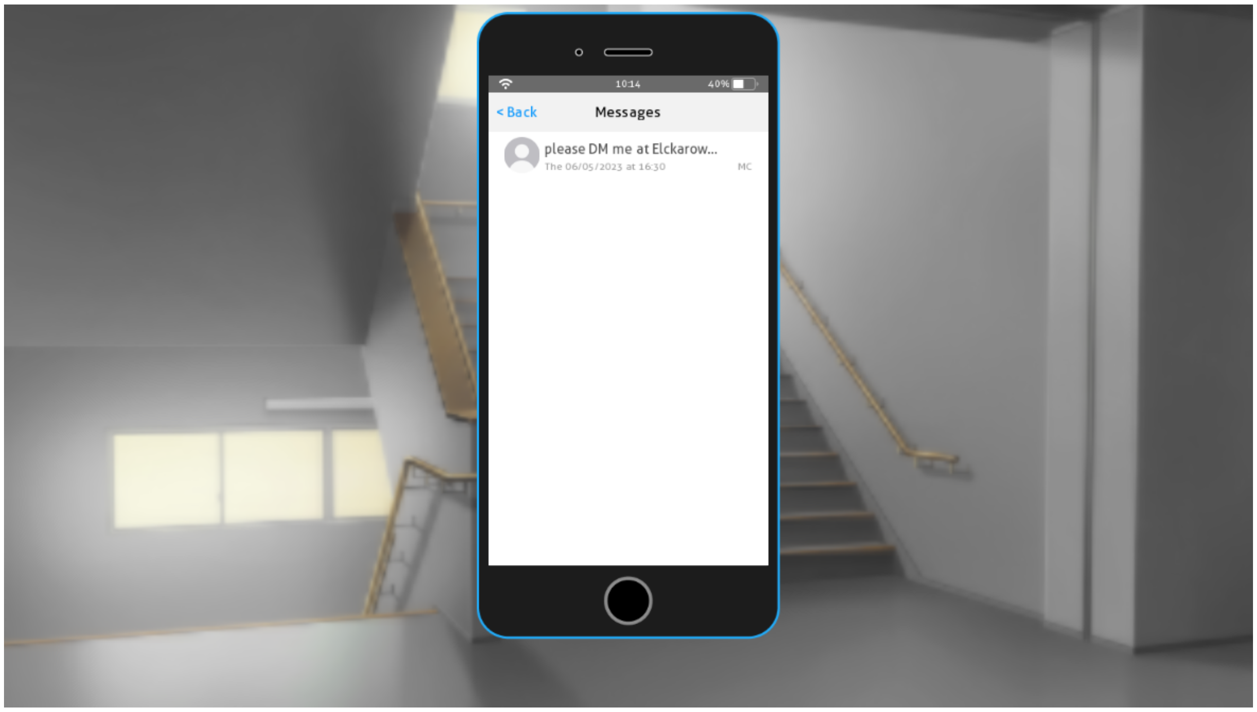
BUILT-IN SCREENS

This framework comes with its set of built-in screens.

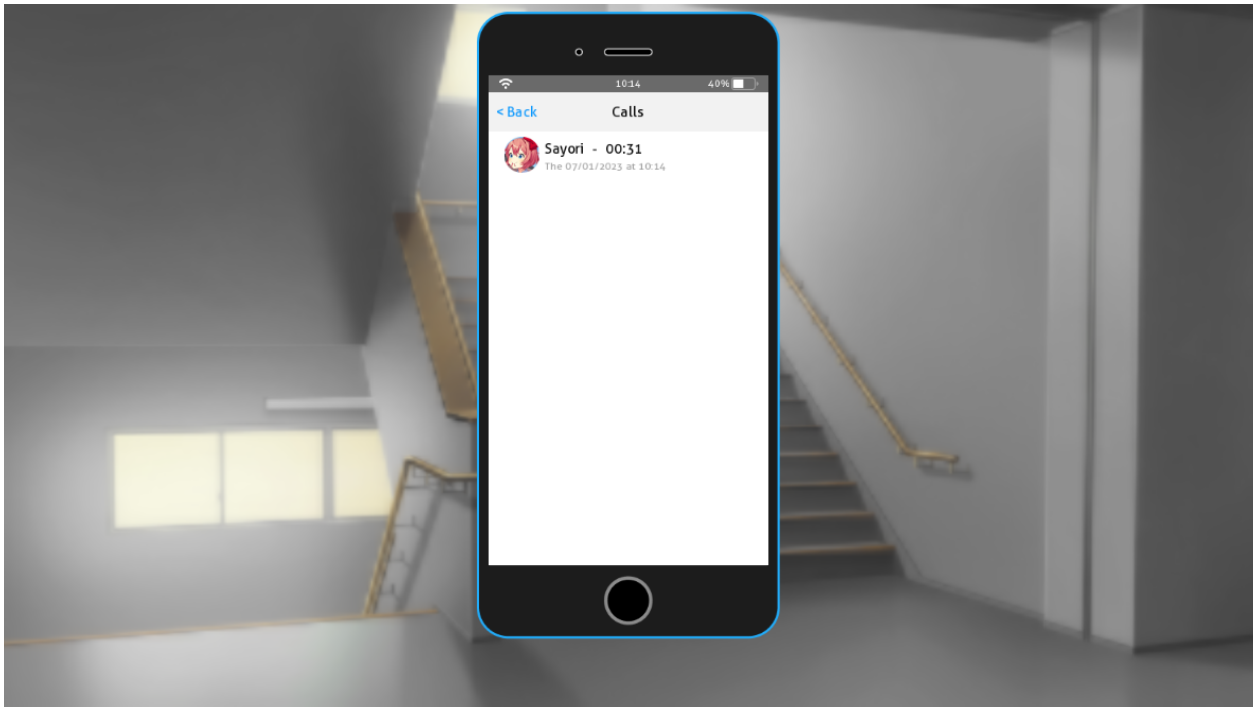
`phone()`



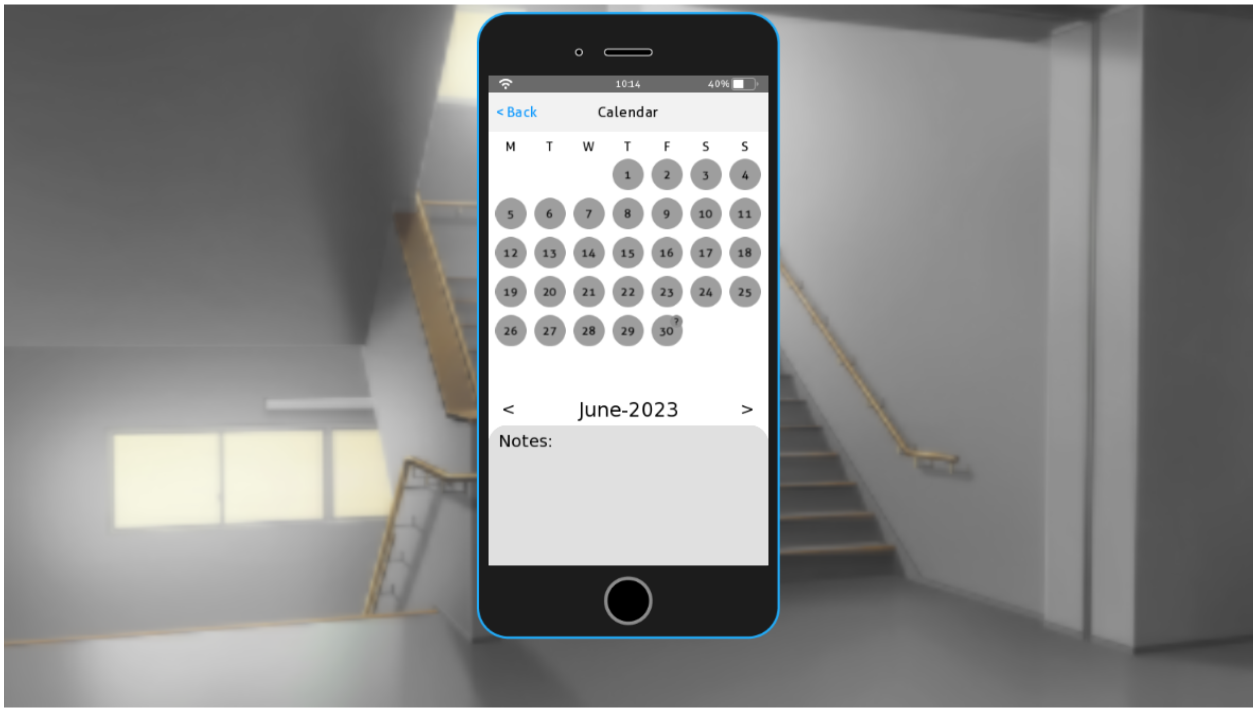
`phone_contacts()`



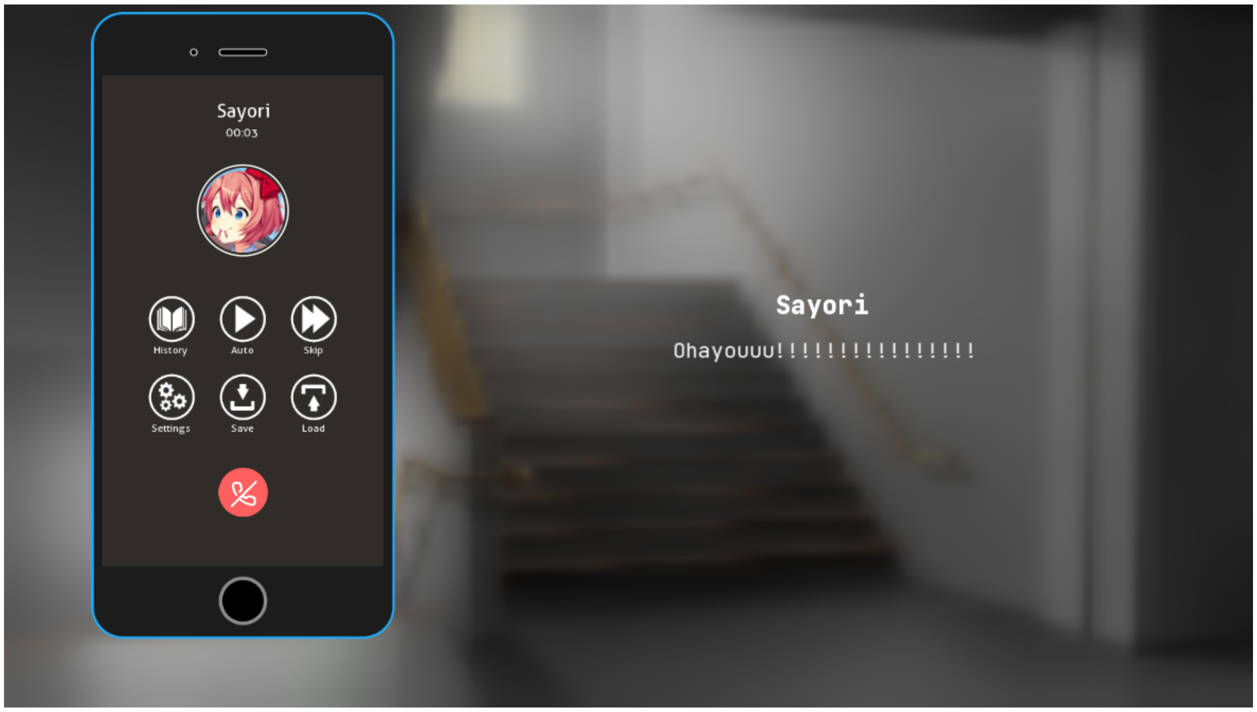
phone_call_history()



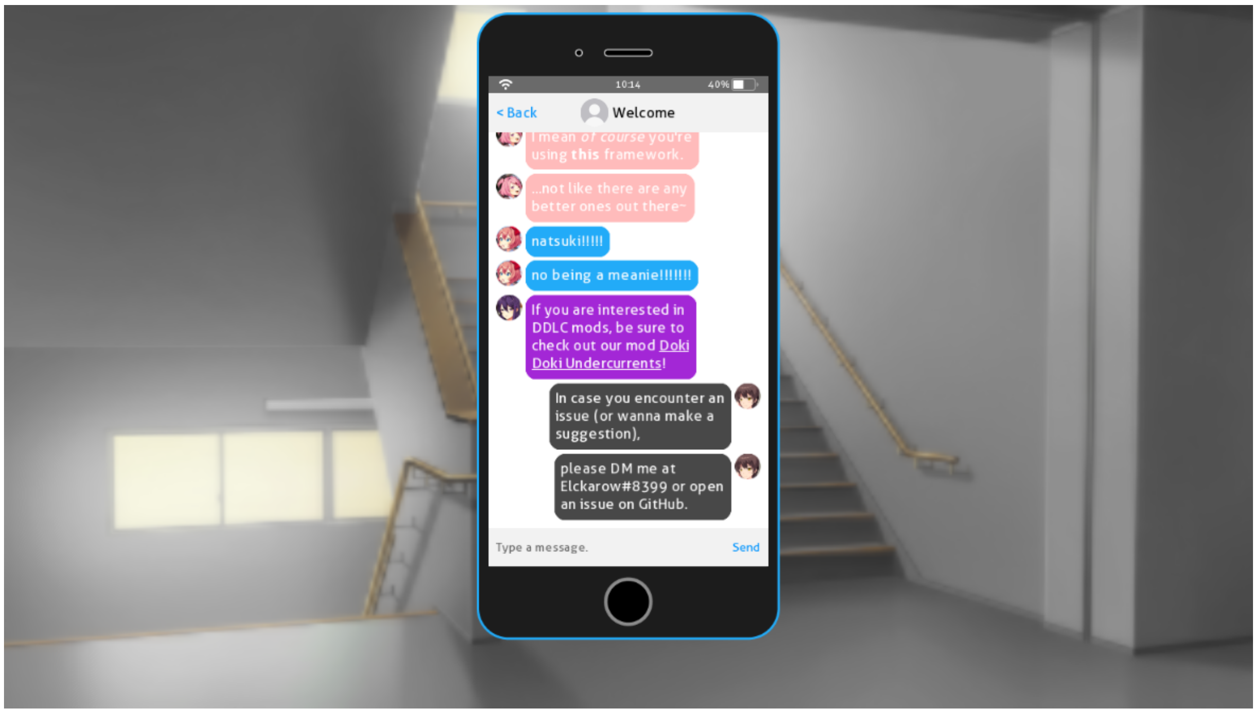
phone_calendars()



`phone_call(video=False)` *(used during phone calls)*



`phone_discussion()` *(used during phone discussions)*



I mean *of course* you're using **this** framework.

...not like there are any better ones out there~

natsuki!!!!!

no being a meanie!!!!!!!

If you are interested in DDLC mods, be sure to check out our mod Doki Doki Undercurrents!

In case you encounter an issue (or wanna make a suggestion),

please DM me at Elckarow#8399 or open an issue on GitHub.

CHAPTER

ELEVEN

FUNCTIONS AND ACTIONS

11.1 Utility Functions

The following functions, variables and Actions are defined in the **phone** *namespace.*

`def format_date(month, day, year)`

Returns a formatted string using `phone.config.date_format`.

`def format_time(hour, minute)`

Returns a formatted string using `phone.config.time_format`.

`def show_layer_at(at_list, layer="master", camera=True, reset=False)`

A wrapper around `renpy.show_layer_at`. If `at_list` is a string (or `None`), it is looked up in `phone.config.layer_at_transform`. The transform or list of transforms is then passed to `renpy.show_layer_at` along with the other parameters.

`def short_name(s, length)`

Shortens the string `s` after translating it. The string is sliced to `length - 3` and `"..."` is appended to it.

`def path_join(*args)`

Computes *os.path.join(*args).replace("\\", "/")*

`def asset(path)`

Computes *path_join(phone.config.basedir, path)*

`def execute_default(f, id)`

Mimics the behavior of the `default` statement by calling `f` if a function with the unique value `id` has never been called before. The function is added to `config.start_callbacks` and `config.after_load_callbacks`, and in the latter case, if the function is called, rollback will be blocked. A good *unique value*, for instance, is a tuple where the first component is a string describing what the function does, and where the remaining components are the actual unique value related to whatever the function does. For example, when calling `phone.calendar.add_calendar` during init phase, this function is called with the unique value `("_phone_add_calendar", month, year, key)`, where `month`, `year` and `key` are the values passed to `phone.calendar.add_calendar`.

`data = {...}`

The dictionnary that's storing all of the phone's in-game data. Each `*character*` has an entry (their key) in this dict and will return another dictionnary as described in `phone.config.data`.

11.2 Screen Functions and Actions

`class PhoneMenu(Action)`

The framework's equivalent of the `ShowMenu` action. Arguments given are passed to the `phone.call_screen` function.

`def call_screen(_screen_name, *args, **kwargs)`

The framework's equivalent of the `renpy.call_screen` function. Invokes `renpy.call_screen` in a new context, passing all arguments and keyword arguments to it. This ensures a sort of "depths" effect (going to screen A, then screen B, then screen C, returning brings you back to screen B, and so on).

`def PhoneReturn(value=None)`

The framework's equvalent of the `Return` action. It should be used to return from a phone screen.

`menu = False`

Indicates whether we're in a phone menu or not. This is useful when a button is used in a phone screen that's used both in-game and through the `PhoneMenu` action (for instance, the `Back` textbutton in the `phone_discussion` is disabled during a phone discussion). As it is set by `phone.call_screen`, this variable should be read-only.

The **PhoneReturn** *and* **PhoneMenu** *actions are available in the global store. If their values are overridden during init phase, an error is raised.*

CHAPTER TWELVE

GUI VARIABLES

Please note that this framework was developed on a 1280x720 project.

These variables can be found in the `phone_stuff.rpy` file.

`gui.phone_xsize = 389`
    The width available for the phone screens.

`gui.phone_ysize = 803`
    The height available for the phone screens.

`gui.phone_margin = (15, 81, 15, 94)`
    Margins added to the left / top / right / bottom of the phone's available area, respectively.

If you take **gui.phone_margin** *to be* **(l, t, r, b)** *,* **l + r + gui.phone_xsize** *and* **t + b + gui.phone_ysize** *should match the phone's background image's dimensions (the one at* **phone.config.basedir + "background.png"** *).*

`gui.phone_zoom = 0.8`
    The level of zoom applied to the phone. If your project uses a different aspect ratio than 1280x720, you might want to consider modifying this variable.

Other GUI variables exist but should not be changed by creators.

CONFIGURATION VARIABLES

The following variables are defined in the **phone.config** *namespace.*

This namespace is considered *constant* and should not be modified outside of init time.

`basedir = "phone/assets/"`
A path from the game folder to the directory where the assets are located.

`overlay_screens = [...]`
A list of screen names shown above the phone screens (but still inside the phone). As the framework uses it internally, creators should `append` their screen name to the list rather than replacing it entirely.

`quick_menu = True`
If true, the framework's quick menu is used during phone calls. If false, the game's quick menu is used.

`enter_transition = Dissolve(0.6, time_warp=_warper.ease)`
A transition used when showing the phone.

`exit_transition = Dissolve(0.6, time_warp=_warper.ease)`
A transition used when hiding the phone.

`intra_transition = Dissolve(0.1)`
A transition used when going from a phone screen to another.

`time_format = _("%H:%M")`
A string used to format a time. Passed to `time.strftime`.

`date_format = _("%m/%d/%Y")`
A string used to format a date. Passed to `datetime.datetime.strftime`.

`textbox_radius = 15`
The radius of the rounded corners of the text messages' textboxes.

`call_history_lenght = 20`
How many phone calls we save.

`message_text_tags = set(...)`
A set of text tags that are allowed in text messages. They should not affect the text's size.

`status_bar = True`
If true, the phone uses the status bar. If false, it doesn't.

`hide_status_bar_screens = [...]`
A list of screen names. When the status bar is used, it is hidden when any of these screens are showing.

`layer_at_transforms = {...}`
A dictionary mapping a screen name to a transform or a list of transforms. When a phone screen is shown, the screen name is looked up in the map (None is used if not found), and the layer "master" is shown at those transforms using `phone.show_layer_at`.

`applications_pages = 4`

How many "pages" of application we use in the `phone` screen.

`lowest_brightness = 0.3`

From 0.0 (complete darkness) to 1.0 (normal), how low can the brightness go?

`data = {...}`

A dictionnary mapping a name to a callable. Each *character* has an entry (their key) containing these values in the `phone.data` dictionnary. When the entry is created, the callables are called without arguments, and the values are set. tl;dr `collections.defaultdict` The following keys are documented:

- `"call_history"`
- `"group_chats"`
- `"background_image"`
- `"calendars"`
- `"applications"`

`discussion_callbacks = [...]`

A list of functions that are called whenever a phone discussion function executes. They are called with three arguments:

- the `*group chat*` the interaction is taking place in.
- an event:
  - `"start"` is delivered at the start of the interaction.
  - `"end"` is delivered just before the data has been saved.
  - `"save"` is delivered after the data has been saved (called after the `register_` function associated to what's happening).
- an object representing the data, which has thefollowing fields:
  - `source`, the `*character*` that's sending the data, or `None`.
  - `type`, one of the following constants (in the `phone.discussion` namespace): `TYPING`, `TEXT`, `IMAGE`, `LABEL`, `DATE`, `MENU`, `AUDIO`, `VIDEO` (if it ever gets implemented).
  - `data`:
    - For a typing, the time to wait for.
    - For a text message, the text that's been formatted by `phone.discussion.remove_text_tags`.
    - For an image, the displayable.
    - For a label, the text.
    - For a date, a tuple of (`month`, `day`, `year`, `hours`, `minutes`, `seconds`).
    - For a menu, a list of all the captions.
    - For an audio, the string of the audio.
    - For a video, `None`.

`video_call_layer = "phone_video_call"`

The name of the layer usied in video calls. It is appended to `config.detached_layers`

`video_call_layer_transform_properties = {...}`

A dict of transform properties applied to the `Layer` displayable (not the layer itself) used during a video call. The default dict centers the displayable and makes it fit the phone vertically.

`messages_displayed = 100`
: How many messages we display at the same time.

`messages_fill_if_lower = 30`
: If the next "load" of messages contains this many or less messages, add those messages to the current load.

`message_delay = 0.6`
: A number of seconds added to the pause before each message.

`unread_group_chat_pov = False`
: If true, a group chat's "unreadness" is determined on the pov the group chat was read in. I.e, if the group chat was read in the "mc" pov, then it won't be marked as read in the "s" pov. If false, it is determined by whether the player has opened the group chat or not.

`auto_emojis = True`
: If true, will define all images found in "assets/emojis" as emojis.

`default_label_delay = 0.5`
: The default value of the *delay* property for the *time* and *label* discussion statements.

CHAPTER FOURTEEN

SYSTEM

The following functions and variables are defined in the **phone.system** *namespace.*

`def get_battery_level()`
Returns the battery level. If the variable `battery_level` is `None`, the value is taken from the player's device.

`def get_internet_connection_state()`
Returns the state of the internet connection, represented by the following constants:

- `CONNECTED` wifi is on and connected to the internet
- `NO_INTERNET` wifi is on but no internet connection
- `NOT_CONNECTED` wifi is off
- `AIRPLANE_MODE` airplane mode is on
- `CELLULAR_DATA` wifi is off and cellular data is on

`def get_date()`
Returns the date. If the variable `date` is `None`, the date used is `datetime.datetime.now()`.

`date = None`
If not `None`, a `datetime.datetime` object.

`battery_level = None`
If not `None`, an integer.

`wifi = None`
If not `None`, a boolean.

`locked = False`
If true, some `Action`s (notably in the status bar screen) won't do anything, preventing the player from changing variables when they're not supposed to.

`at_list = []`
A transform or list of transforms applied to the phone screen (overlay screens excluded). Set back to `[]` when exiting the phone.

`cellular_data = False`
Is cellular data on?

`airplane_mode = False`
Is airplane mode on?

`bluetooth = False`
Is bluetooth on?

`internet_connection = True`

This can be turned on/off for more a realistic gameplay (say the main character is in a very rural place, they won't have access to the internet).

Other variables (`flashlight`, `rotation_locked`, `dark_mode`) currently don't have any use and should not be used by creators.

CHAPTER

FIFTEEN

CLICK EFFECTS

The framework comes with a built-in click effects that shows a displayable whenever a click/drag/release occurs.

It uses three variables

- phone_on_click_effect
- phone_on_drag_effect
- phone_on_release_effect

which are, if not None, a 2-tuple containing a displayable and a float.

Whenever the corresponding mouse event occurs

- a left/right click for phone_on_click_effect
- moving the mouse while holding click for phone_on_drag_effect
- releasing a click for phone_on_release_effect

the corresponding displayable will be added to the screen and then hidden after the amout of time passed as second element of the tuple.

say you have this

```
define phone_on_click_effect = (Solid("#f00", xysize=(50, 50)), 2)
```

left/right clicking will add a 50x50 red square to the screen that will be hidden after 2 seconds.

CHAPTER SIXTEEN

AUDIO

This framework includes a custom audio mixer `phone`, on which the `phone_music` channel has been registered. Not much you can do with it right now though…

CHAPTER SEVENTEEN

CREDITS

I'd like to thank to following people for their contributions:

- silversnow__ (MC's icon)
- lameman (Quick menu icons)
- negative.two (Shader code for the `CircleDisplayable` displayable)
- calebthepianist (Renaming all the twemojis)

as well as the project's contributors on GitHub:

- Galo223344 (transient group chats)

Various ressources used:

- twemojis
- Wintermute's Gradient displayable

CHAPTER

EIGHTEEN

CHANGELOG

See `Incompatible Changes` *for additional information.*

A list of all the changes throughout the versions, starting from 3.0.0.

18.1 3.2.2

- The `phone call` statement can now take a `nosave` clause.
- Added new default emojis.
- 4 new functions `phone.asset`, `phone.path_join`, `phone.short_name` and `phone.execute_default`.
- `GroupChat.short_name` and `Character.short_name` are now deprecated. See the new `phone.short_name` function.
- Reworked a bit the `phone_contacts` screen.
- Reset the yadjustment when starting a phone discussion.
- 7 new phone config variables.
- Fixed an issue where changes applied to a calendar would persist when going back to the main menu.

18.2 3.2.1

- Checks for the correct version before appending to `config.detached_layers`.

18.3 3.2.0

- Group chats can now be transient. A transient group chat will be cleared once a discussion is over.
- Added video phone calls.
- A battery level of 0% will now display an empty battery (as it should).
- Added the `pause` phone discussion statement.
- Can now start a phone discussion when another discussion is going on.

18.4 3.1.1

- Clearer error messages when a group chat / phone character isn't defined.
- Document some GUI variables.
- Phone definitions will now work on an already existing save. Before, if you had a save where no group chat was defined (this is just an example), that you created a group chat, and then loaded that save, the group chat wouldn't be registered. This change replicates the behavior of the `default` statement, allowing creators to, for instance, add this framework to an already released game. After loading the save, rollback will be blocked in such cases.
- Fixed an issue where running `PhoneMenu` during an interaction would block the player from advancing after returning.

18.5 3.1.0

- Improved lint.
- `phone.discussion.date` and `phone.discussion.register_date` now accept `None` and `True` values (before, only the `date` phone statement could use `None` values).
- `phone.discussion.date` and `phone.discussion.register_date` now accept two new arguments: `second` and `auto`.
- The `image` phone statement and `phone.discussion.image` can now take any displayable.
- Fixed an issue with `gui.phone_message_label_null_height`.
- Added `phone.config.discussion_callbacks`.

18.6 3.0.3

- Fixed the `calendar` app layout.

18.7 3.0.2

- Fixed the `pass` phone discussion statement.

18.8 3.0.1

- Phone messages now respect the `delay` property.
- Phone labels can now accept `None` values.
- The audio icon in the status bar now uses `preferences.get_mixer` on 7.6/8.1+ and `preferences.get_volume` on other Ren'Py versions.
- A new function has been introduced to the `phone.character` namespace: `get_all()`.
- A new function has been introduced to the `phone.group_chat` namespace: `get_all()`.

18.9 3.0.0

- None

CHAPTER

NINETEEN

INCOMPATIBLE CHANGES

A list of all changes that may require you to modify some of your code.

19.1 3.2.2

- The `phone.config` and `phone.emojis` namespaces now behave the same way as renpy's `config`. The `default` statement cannot be used to declare a variable in those namespaces.
- The `phone.calendar.add_calendar` and `phone.calendar.add_calendar_to_all_characters` functions have had their signature changed.

19.2 3.2.1

- None

19.3 3.2.0

- The `Quit` button in the `phone` screen has been removed in place of a dismiss-like button.
- The `pass` phone discussion statement now doesn't do anything, like a regular `pass` statement. To wait for a user input, see the `pause` phone discussion statement.

19.4 3.1.1

- The `phone register` statement can't be ran during init phase anymore. Use the `init phone register` statement instead.
- `auto` can't be used anymore for the `time` statement in the `init phone register` statement.

19.5 3.1.0

- None

19.6 3.0.3

- None

19.7 3.0.2

- None

19.8 3.0.1

- None

19.9 3.0.0

- Functions and classes related to phone characters have been moved in the `phone.character` namespace.
- Functions and classes related to group chats have been moved in the `phone.group_chat` namespace.