

Text Mode Layer (tml) - ein Python-Modul für HP Prime zur Simulation eines Terminals mit Festbreitenschrift Version 1.00

von Piotr Kowalewski (komame), August-Oktober 2024

Was ist tml und wie funktioniert es?

Das *tml*-Modul (Text Mode Layer) dient als komfortabler Ersatz für das integrierte Terminal auf dem HP Prime für Python und bietet zusätzliche Funktionen zur Verwaltung des Terminals.

Das Modul verwendet den Grafikmodus, um ein Terminal mit Festbreitenschrift zu simulieren. Da das integrierte Terminal nach Beendigung des Python-Programms wiederhergestellt wird, ist es nicht möglich, *tml* direkt über die Befehlszeile zu verwenden. Daher muss es in ein Programm importiert werden und kann nur funktionieren, solange das Programm ausgeführt wird.

Aus diesem Grund enthält *tml* eine eigene Implementierung zur Ausgabe von Text, zum Scrollen des Bildschirms sowie für die Benutzer-Eingabe-Mechanismen. Die Benutzereingabe wurde so implementiert, dass sie sich ähnlich wie auf einem PC verhält, sodass Daten an jeder beliebigen Position auf dem Bildschirm eingegeben werden können, anstatt auf ein dediziertes Eingabefeld am unteren Rand des Bildschirms beschränkt zu sein (was eine Einschränkung des integrierten Terminals auf dem HP Prime darstellt).

Eine Statusleiste kann am unteren Rand des Terminals angezeigt werden (standardmäßig aktiviert), die jede Textnachricht anzeigen kann. Die Statusleiste zeigt außerdem Tastaturindikatoren (Shift/Alpha) an.

Das *tml*-Modul ermöglicht die Verwendung verschiedener monospaced Bitmap-Schriften (unterschiedliche Stile und Größen), aber pro *tml*-Instanz kann nur eine Schriftart verwendet werden. Nach dem Laden einer Schriftart initialisiert *tml* ein Terminal mit der Anzahl an Zeilen und Spalten, die die gewählte Schriftgröße erlaubt. Die maximale unterstützte Zeichenbreite beträgt 35 Pixel und die minimale 4 Pixel, wodurch ein Terminal mit 9 bis 80 Zeichen pro Zeile möglich ist.

Es ist möglich, benutzerdefinierte Schriftarten zu erstellen und eigene Tastenbelegungen für bestimmte Zeichen (Symbole) zu definieren, um eine direkte Eingabe von Symbolen über die Tastatur zu ermöglichen und diese korrekt darzustellen, beispielsweise für diakritische Zeichen. Weitere Details finden Sie im Abschnitt „Erweiterte Funktionen“

Wie startet man mit tml?

Wenn Sie die integrierte Python-App verwenden oder Ihre eigene Anwendung auf Basis der Python-App erstellen, kopieren Sie die Datei *tml.py* und eine der ausgewählten Schriftarten (aus dem Ordner *Fonts*) in Ihre Anwendung. Importieren Sie dann im Hauptfile Ihrer Anwendung (üblicherweise *main.py*) das *tml*-Modul, was Ihnen erlaubt, das Terminal zu initialisieren.

Es ist auch möglich, *tml* über den PPL-Wrapper zu verwenden, aber bei diesem Ansatz ist es entscheidend, dass alle PPL-Prozeduraufrufe von Python aus gemacht werden, nicht umgekehrt.

Initialisierung des Terminals

Die einfachste Art, das Terminal zu initialisieren, ist, einfach eine Instanz von *tml* zu erstellen, ohne irgendwelche Argumente anzugeben:

```
import tml
t = tml.tml()
```

Diese Herangehensweise initialisiert das *tml* mit den folgenden Standardwerten:

- Verwendung der ersten gefundenen Schriftart (Details unten)
- Statusleiste aktiviert, ohne Inhalt (")
- Dunkelmodus deaktiviert
- Tabulatorgröße: 4 Zeichen
- Erweiterte Zeichenzuordnung: leer (siehe Abschnitt „Erweiterte Funktionen“)
- Tastenbelegung für Symbole: leer (siehe Abschnitt „Erweiterte Funktionen“)
- Schriftartpuffer: 9 (G9)

Aus dem Obigen geht hervor, dass *tml* 7 Argumente hat, jedes mit seinem eigenen Standardwert; daher wird empfohlen, sie beim Namen zu nennen.

Der vollständige Konstruktor lautet wie folgt:

```
import tml
t = tml.tml(font, status = '', dark_mode = False, tab_size = 4, ext_char_map = {},
           symb_key_map = {}, grob = 9)
```

Argumente:

- Schriftart:

Während der Initialisierung sucht *tml* automatisch nach einer Schriftartdatei (eine Datei mit der Erweiterung „.font“) und lädt diese. Es ist jedoch auch möglich, die zu ladende Schriftart explizit mit dem Schriftartparameter anzugeben (wenn der Dateiname der Schriftart explizit angegeben wird, lassen Sie die Erweiterung „.font“ weg). Wenn die Schriftart nicht angegeben wird und die Anwendung mehr als eine Schriftartdatei enthält, wird die erste in alphabetischer Reihenfolge geladene.

Das *tml*-Modulpaket enthält mehrere Beispiel-Schriftarten in verschiedenen Stilen und Größen:

- atari8x8 (40 Spalten, 28 oder 30 Zeilen) - Atari 8-Bit-Stil
- std5x10 (64 Spalten, 26 oder 24 Zeilen) - Standard-Schriftart
- std5x12d (64 Spalten, 18 oder 20 Zeilen) - Standard-Schriftart mit diakritischen Zeichen
- med6x12 (53 Spalten, 18 oder 20 Zeilen) - mittelgroße Schriftart
- med10x12d (32 Spalten, 18 oder 20 Zeilen) - mittelgroße Schriftart mit diakritischen Zeichen
- mini4x7 (80 Spalten, 32 oder 34 Zeilen) - hp48/49/50 Mini-Schriftstil

Benutzerdefinierte Schriftarten können ebenfalls verwendet werden, wie im Abschnitt „Erweiterte Funktionen“ beschrieben.

- status:

Akzeptiert einen Textwert, der in der Statusleiste angezeigt wird. Standardmäßig ist es ein leerer Wert ("), was dazu führt, dass die Statusleiste erscheint, aber keinen anfänglichen Inhalt enthält. Die Eingabe eines Textes in diesen Parameter führt dazu, dass er in der Statusleiste angezeigt wird, die sich am unteren Rand des Bildschirms befindet. Wenn die Statusleiste sichtbar ist, belegt sie die letzten zwei Zeilen des Terminals und ist nicht scrollbar. Um die Statusleiste zu deaktivieren und das Terminal im Vollbildmodus zu verwenden, setzen Sie den Parameter *status* während der Initialisierung auf *None*. Beachten Sie jedoch, dass dies auch die Tastaturstatusanzeigen (Zustände der Shift- und Alpha-Taste) ausblendet.

- dark_mode:

Standardmäßig ist der Dunkelmodus deaktiviert, was bedeutet, dass der Terminalhintergrund weiß und die angezeigten Zeichen dunkel sind. Wenn der Dunkelmodus aktiviert ist, wird der Hintergrund schwarz und die Zeichen sind hell. Um den Dunkelmodus zu aktivieren, setzen Sie das

Argument auf `True`. Dieser Parameter kann nur während der Initialisierung des Terminals gesetzt werden und kann später nicht mehr geändert werden.

- `tab_size`:

Die Tabulatorgröße ermöglicht es, den Bildschirm in vertikale Segmente einer bestimmten Größe zu unterteilen, zu denen der Cursor springt, wenn ein Tabulatorzeichen (`\t`) gedruckt wird. Dies erleichtert die Textausrichtung oder die Spaltenteilung.

- `grob`:

Da die Schriftart aus einer „.png“-Datei geladen wird, muss sie in einem der grafischen Objekte (Grafikvariablen) des HP Prime gepuffert werden. Standardmäßig ist dies 9 (G9), was während der Initialisierung mit dem Argument `grob` angegeben werden kann. Wie bei `dark_mode` kann der Wert von `grob` nur während der Initialisierung des Terminals gesetzt werden und kann später nicht mehr geändert werden.

Beispiel für die Initialisierung eines Terminals mit 53 Spalten und 20 Zeilen im Dunkelmodus, mit der Statusleiste eingestellt auf 'Hello World!' und der Anzeige des Textes 'This\n\tis\n\t\ta test' mittels `print`, welches die Zeilenumbruch- und Tabulatorzeichen interpretiert:

```
import tml

t = tml.tml(status='Hello World!', font='med6x12', dark_mode=True)
t.print('This\n\tis\n\t\ta test')
t.print('\n\nPress any key...')
t.read_key()
```

Erzielter Effekt auf dem Bildschirm:



```
This
  is
    a test

Press any key...

Hello World!
```

Verfügbare Methoden

Das `tml`-Modul stellt die folgenden Methoden bereit:

- `print(*args)`

Ermöglicht die Ausgabe von Text und anderen Werten im Terminal. Es akzeptiert Argumente und funktioniert wie der eingebaute `print`-Befehl in MicroPython auf dem HP Prime, wobei zusätzlich das Tabulatorzeichen (`\t`) interpretiert wird.

- **clear()**

Löscht den Terminalbildschirm und setzt den Cursor auf die Position (0,0), die sich in der oberen linken Ecke des Bildschirms befindet. Es nimmt keine Argumente entgegen.

- **set_cursor(x, y)**

Setzt den Cursor an die Position (x, y). Die nächste Verwendung von *print* oder *input* wird von dieser Position aus starten.

- **input(prompt, length, alpha, shift, new_line)**

Die *input*-funktion ermöglicht die Dateneingabe durch den Benutzer, ähnlich der eingebauten *input*, jedoch mit einigen Erweiterungen, die zusätzliche Aktionen während der Dateneingabe erlauben. Wenn das Eingabefeld erscheint, kann der Benutzer den Cursor innerhalb des eingegebenen Textes nach links und rechts bewegen, Zeichen einfügen und sie mit der Backspace- oder Del-Taste löschen. Während der Dateneingabe können die Modifier-Tasten Shift und Alpha (und ihre Lock-Zustände) gemäß dem Standardverhalten in der Startansicht verwendet werden. Wenn eine Modifier-Taste aktiviert ist, wird der Tastaturstatus in der Statusleiste angezeigt (falls aktiviert), wobei "SL" für Shift-Lock steht, "AL" oder "al" je nach Fall für Alpha-Lock steht und "A" für aktives Shift (zur einmaligen Verwendung).

Es ist auch möglich, benutzerdefinierte Zeichen einzugeben, indem die Symb-Taste gehalten und andere Tasten gedrückt werden. Dies erfordert jedoch das Definieren zusätzlicher Tastenzuordnungen, wie im Abschnitt 'Erweiterte Funktionen' erwähnt.

Wenn während der Bearbeitung Esc oder Clear (Shift+Esc) gedrückt wird, wird das Bearbeitungsfeld geleert und der Cursor kehrt zur Startposition zurück. Das Drücken der Enter-Taste beendet die Eingabe und gibt die eingegebenen Daten als Zeichenkette zurück.

Hinweis: Die Tastatureingabe funktioniert möglicherweise nicht korrekt auf dem Virtual Calculator, wenn Buchstaben- oder Symboltasten gedrückt werden. Wenn Sie die Eingabemethode testen möchten, verwenden Sie ausschließlich zugeordnete Tasten (siehe den Abschnitt zur Tastaturzuordnung am Computer in der VC-Hilfe für Details) oder verwenden Sie einen physischen HP Prime.

Input akzeptiert die folgenden Argumente:

- **prompt:** Bestimmt den Hinweistext, der im Dateneingabefeld auf dem Bildschirm angezeigt wird.
- **length:** Standardmäßig erstellt *input* ein Bearbeitungsfeld, das die gesamte Zeile umspannt (von der x-Position des Cursors bis zum rechten Rand des Terminals). Es kann jedoch eine maximale Bearbeitungslänge festgelegt werden, wodurch verhindert wird, dass mehr Zeichen als durch das *length*-Argument angegeben, eingegeben werden.
- **alpha:** Aktiviert bei der Anzeige des Bearbeitungsfeldes zunächst Alpha-Lock, sodass Text sofort eingegeben werden kann, ohne die Alpha-Taste drücken zu müssen.
- **shift:** Aktiviert zunächst Shift oder Shift-Lock (wenn zuvor Alpha-Lock aktiviert wurde).
- **new_line:** Dieses Argument legt fest, ob das Drücken der Enter-Taste zum Wechsel in eine neue Zeile führen soll. Wenn *new_line* auf **False** gesetzt wird, ist dies nützlich, wenn Daten in der letzten Zeile des Terminals gesammelt werden und der Bildschirm nach dem Drücken dieser Tasten nicht scrollen soll.

- **read_key(code=False)**

Hält das Programm an und wartet, bis eine beliebige Taste gedrückt wird (außer Alpha und Shift). Standardmäßig gibt es das Symbol der gedrückten Taste zurück, wobei der aktuelle Alpha- und Shift-Zustand berücksichtigt wird. Wenn *code* auf **True** gesetzt ist, wird der Tastencode der gedrückten Taste (0 - 51) zurückgegeben.

- `get_keys()`

Gibt eine Liste der Codes für aktuell gedrückte Tasten zurück (einschließlich Alpha und Shift sowie bei gleichzeitigem Drücken mehrerer Tasten). Gibt eine leere Liste zurück, wenn keine Taste gedrückt wird.

- `set_status(text)`

Ermöglicht das Festlegen des in der Statusleiste angezeigten Textes. Um die Statusleiste zu leeren, verwenden Sie einen leeren String: `set_status('')`

- `print_xy(x, y, text)`

Zeigt den angegebenen Text an der Position (x, y) des Terminals an, ohne den Cursor zu bewegen. Für diese Funktion werden Tabulator- und Zeilenumbruchzeichen nicht interpretiert.

Erweiterte Funktionen

Dieser Abschnitt ist für fortgeschrittene Benutzer gedacht und beschreibt drei Funktionen von *tml*:

1. Erstellen von benutzerdefinierten Schriftartdateien.
2. Zuordnung grafischer Symbole aus einer Schriftartdatei zu einzelnen darstellbaren Zeichen.
3. Zuordnung von Tastaturtasten für die direkte Eingabe benutzerdefinierter Symbole.

Hier finden Sie auch Informationen darüber, wie *tml* auf einer niedrigeren Ebene funktioniert.

Jede *tml*-Schriftartdatei ist eine „.png“-Datei, jedoch leicht modifiziert, da sie zusätzliche Informationen über die Schriftbreite und die Konfigurationsnummer der spezifischen Schrift enthalten muss. Intern wird eine solche „.png“-Datei, die das Aussehen einzelner Schriftzeichen enthält, als Array von 0 bis 94 behandelt, wobei 0 ein Leerzeichen und 94 eine Tilde ist. Alle Zeichen sind nebeneinander in einer einzigen Reihe mit einer festen Pixelbreite angeordnet, sodass beim Anzeigen von Zeichen mit der *print*-Methode jedes Zeichen dem entsprechenden Index zugeordnet werden kann und der entsprechende Abschnitt der Schriftartdatei angezeigt wird, um das Aussehen des spezifischen Zeichens darzustellen.

Standardmäßig können nur Zeichen innerhalb des grundlegenden ASCII-Tabellenbereichs in *tml* angezeigt und eingegeben werden, d.h. vom Leerzeichen (Code 32) bis zur Tilde (Code 126). Alle Zeichen in diesem Bereich werden durch die eingebaute Zuordnung gehandhabt, sodass es nicht notwendig ist, zusätzliche Zuordnungen für das Anzeigen dieser Zeichen oder Tastenzuordnungen für deren Eingabe zu definieren. Es ist jedoch möglich, Schriftartdateien mit einer größeren Zeichensatzmenge zu verwenden. Dies erfordert die Verwendung von zwei zusätzlichen Argumenten bei der Initialisierung von *tml*: **`ext_char_map`** und **`symb_key_map`**.

Erstellung benutzerdefinierter Schriftartdateien

Um eine benutzerdefinierte Schriftart zu erstellen, erstellen Sie ein Bitmap mit den Abmessungen $x = 95 * \text{Zeichenbreite}$ und $y = \text{Zeichenhöhe}$. Das Bitmap sollte mindestens die ASCII-Zeichen von Code 32 (Leerzeichen) bis 126 (Tilde) in einer einzigen Reihe enthalten. Die Zeichen sollten schwarz auf weißem Hintergrund sein. Graustufen sind erlaubt, andere Farben sollten jedoch vermieden werden, da sie bei der Textbearbeitung auf dem Bildschirm oder bei aktiviertem Dunkelmodus Artefakte verursachen können.

Wenn Ihre Schriftart mehr als 95 Zeichen enthalten soll (z. B., wenn Sie diakritische Zeichen oder spezielle Symbole hinzufügen möchten, die in Ihrem Programm benötigt werden), können Sie diese nach dem Tilde-Zeichen in beliebiger Reihenfolge hinzufügen, was bedeutet, dass der Index des ersten zusätzlichen Zeichens 95 ist (zur Erinnerung: die Tilde ist das 95. Zeichen, aber da wir von Null an

zählen, ist der Tilde-Index 94). Sobald alle Zeichen gestaltet sind, speichern Sie die Bitmap-Datei im „png“-Format (Sie können die Farbtiefe auf 1-Bit setzen, wenn nur Schwarz-Weiß-Farben verwendet wurden).

Damit die Datei korrekt erkannt und vom *tml*-Modul importiert wird, muss am Ende der Datei ein Konfigurationsbyte hinzugefügt werden. Dieses Byte sollte die Konfigurationsnummer und die Breite eines einzelnen Zeichens in Pixeln enthalten. Die Konfigurationsnummer sollte in den drei am wenigsten signifikanten Bits dieses Bytes gespeichert werden und sollte derzeit immer auf 0 gesetzt sein (andere Konfigurationsnummern sind für zukünftige Verwendungen reserviert), während die Breite eines einzelnen Zeichens in den fünf am meisten signifikanten Bits als um 4 verringerte Wert gespeichert werden sollte. Dies ermöglicht Zeichenbreiten von 4 bis 35 Pixeln.

Zum Beispiel: Wenn die Schriftartdatei Zeichen mit einer Breite von 9 Pixeln enthält, sollte das Konfigurationsbyte in Binärform so aussehen: 00101000, was in Dezimalzahlen 40 ist. Sie können dieses Byte auf einem PC oder direkt auf dem HP Prime mit dem Befehl `AFilesB` hinzufügen.

Sobald dieses Byte hinzugefügt und die Dateierweiterung in `.font` geändert wurde, ist die Schriftartdatei bereit, in *tml* verwendet zu werden.

Zuordnung grafischer Symbole aus einer Schriftartdatei zu einzelnen darstellbaren Zeichen

Wenn Ihre Schriftart mehr als die standardmäßigen 95 Zeichen enthält, müssen Sie eine Zuordnung von Unicode-Zeichen zu den entsprechenden Indizes in Ihrer Schriftart definieren, damit *tml* weiß, welche Zeichen für die zusätzlichen Symbole außerhalb des Standard-ASCII-Bereichs angezeigt werden sollen.

Nehmen wir zum Beispiel an, das 95. Zeichen ist das Paragraphenzeichen (§), das nicht Teil des standardmäßigen ASCII-Zeichensatzes ist. Wenn die *print*-Methode auf dieses Zeichen im zu druckenden Text trifft, muss sie dessen Position innerhalb Ihrer Schriftart kennen, d.h., sie muss dessen Index kennen. Um das Zeichen einem Index zuzuordnen, müssen Sie ein Wörterbuch in der folgenden Form definieren: { '§' : 95 } und es als Argument *ext_char_map* übergeben.

Zum Beispiel nehmen wir an, Ihre Schriftart enthält Symbole für die Kartensymbole:

♠ (Pik), ♥ (Herz), ♦ (Karo), ♣ (Kreuz), jeweils von Index 95 bis 98.

In diesem Fall sollte das im *ext_char_map* definierte Wörterbuch folgendermaßen aussehen:

```
{ '♠' : 95, '♥' : 96, '♦' : 97, '♣' : 98 }
```

Wenn Sie benutzerdefinierte Symbole entworfen haben, die keinen entsprechenden Unicode-Zeichen zugeordnet sind, können Sie jedes Unicode-Zeichen verwenden, das Sie in Ihrem Programm nicht benötigen, und es durch Ihr Symbol ersetzen. Es ist nicht notwendig, alle entworfenen Zeichen im Wörterbuch zu definieren. Wenn jedoch ein Zeichen, das im Wörterbuch ausgelassen wurde, im Text erscheint, der der *print*-Methode übergeben wird, wird es einfach nicht auf dem Bildschirm angezeigt.

Zuordnung von Tastaturtasten für die direkte Eingabe benutzerdefinierter Symbole

Wenn eine Zuordnung für nicht-standardisierte Zeichen definiert wurde und sie auf dem Terminal mit der *print*-Methode angezeigt werden können, besteht manchmal die Notwendigkeit, sie von der Tastatur einzugeben. In der Startansicht, um ein sprachspezifisches Zeichen wie „ä“ einzugeben, können Sie das Werkzeug „Chars“ (Zeichen) öffnen, das Symbol finden und in das Bearbeitungsfeld einfügen. Dieses Werkzeug ist jedoch nicht zugänglich, wenn ein Programm ausgeführt wird, das das *tml*-Modul verwendet. In solchen Fällen bietet die Eingabemethode eine Möglichkeit, bestimmte in Ihrer Schriftart definierte Zeichen über Tastenkombinationen einzugeben: `[Symb]+[beliebige Taste]`. Ähnlich wie bei

der Zuordnung von Schrift zu Zeichen ist es notwendig, die Tastenzuordnungen zu definieren und sie als Argument `symb_key_map` zu übergeben.

Diese Methode ermöglicht es Ihnen auch, mehrere Symbole unter einer Taste zu definieren, was nützlich sein kann, wenn Sie mit einer Gruppe ähnlicher Symbole umgehen und verhindern möchten, dass verschiedene Tasten für jedes zugeordnet werden müssen. Beispielsweise hat der Buchstabe „e“ im Französischen vier diakritische Varianten: **é, è, ê, ë**. In solch einem Fall ist es am besten, alle vier unter der Kombination `[Symb]+[e]` zu platzieren, was beim Eingeben von Text, der eines dieser Zeichen enthält, intuitiv und bequem sein wird.

Um eines dieser Zeichen in das Bearbeitungsfeld einzugeben, drücken und halten Sie die `[Symb]`-Taste und drücken zusätzlich die `[e]`-Taste. Das erste Symbol erscheint an der Cursorposition, aber der Cursor bewegt sich nicht zur nächsten Position, bis Sie die `[Symb]`-Taste loslassen. Zu diesem Zeitpunkt wird das wiederholte Drücken der `[e]`-Taste die Zeichen durchlaufen, die unter dieser Taste definiert sind, und das Loslassen der `[Symb]`-Taste bestätigt das derzeit angezeigte Zeichen.

Die Zuordnung von Tasten zu Symbolen beinhaltet die Verwendung der Tastennummer (von 0 bis 51) als Schlüssel im Wörterbuch. Der entsprechende Wert sollte eine Liste mit vier Elementen sein, wobei jedes Element entweder eine Liste von Symbolen oder `None` ist. Jedes dieser vier Elemente repräsentiert einen anderen Tastaturzustand, da die angezeigten Symbole je nach Aktivierung der Shift-Taste, der Alpha-Taste oder einer Kombination aus beiden variieren können. Daher muss der Wert für jede Taste im Wörterbuch eine Liste mit vier Fällen in der folgenden Reihenfolge sein:

1. wenn kein Modifikator aktiv ist,
2. wenn nur Alpha aktiv ist,
3. wenn nur Shift aktiv ist,
4. wenn beide aktiv sind.

Zum Beispiel könnten Sie für die Taste mit dem Buchstaben 'e' und den französischen diakritischen Zeichen das folgende Wörterbuch erstellen:

```
{ 18: [None, ['è', 'é', 'ê', 'ë'], None, ['Ê', 'É', 'Ë', 'È']] }
```

Dies sollte wie folgt verstanden werden: Wenn weder Alpha noch Shift aktiv sind, wird durch Drücken von `[Symb]+[e]` kein Symbol angezeigt (da Alpha nicht aktiv ist, gibt es keinen Grund dafür, dass Buchstaben erscheinen). Ist jedoch der Alpha-Modifikator aktiviert, zeigt das Drücken von `[Symb]+[e]` den Buchstaben „è“ (Kleinbuchstabe) an, und jedes weitere Drücken von `[e]` wird die Symbole in der Liste durchlaufen, bis die `[Symb]`-Taste losgelassen wird. Der dritte Wert ist ebenfalls `None`, da wir keine Buchstabensymbole anzeigen möchten, wenn nur Shift aktiv ist (dies könnte ein guter Ort sein, um einige einzigartige nicht-alphanumerische Symbole anzuzeigen). Der vierte Wert gibt an, was erscheinen soll, wenn sowohl Alpha als auch Shift aktiv sind; in diesem Fall werden die Symbole **Ê, É, Ë, È** (Großbuchstaben) angezeigt.

Bekannte Probleme und Einschränkungen

- Zum aktuellen Zeitpunkt ist es nicht möglich, den Inhalt des Terminalbildschirms wiederherzustellen, nachdem er durch externen Code überschrieben wurde. Dies bedeutet auch, dass es nicht möglich ist, zwischen zwei parallelen Instanzen von `tml` zu wechseln.
- Es wird keine Unterstützung für die Anzeige von Text und Hintergrundfarben in verschiedenen Farben angeboten (dies liegt an den Einschränkungen der `hprime`-Bibliothek).
- Es ist ebenfalls nicht möglich, den Zeichencode vom `tml`-Bildschirm an spezifischen Koordinaten zu lesen.

Wenn Sie Fehler finden, interessante Vorschläge haben oder einfach Kontakt aufnehmen möchten, senden Sie eine private Nachricht an den Benutzer **komame** auf www.hpmuseum.org/forum/.