.gitignore
biome.json
bun.lockb
tsconfig.json
# @cartesia/cartesia-js

## 1.1.0

### Minor Changes

- ed6be65: Add WebSocket continue method for input streaming with contexts

## 1.0.3

### Patch Changes

- 9b157d6: Support Bun by adding socket binaryType

## 1.0.2

### Patch Changes

- Make voice creation and cloning fully compatible with API.

## 1.0.1

### Patch Changes

- cb7adc2: Introduces support for continuations, timestamps, voice control and multiple output formats. Improves typing and error handling for the package.

## 1.0.0

### Major Changes

- 3ee5bfc: Initial release of Cartesia client with voices and WebSocket support

### Minor Changes

- e49f73a: Stabilize audio playback in the browser to support play/pause functionality.

### Patch Changes

- c98a0c7: Fix typo in README
- 38af01f: Fix how URLs are constructed, solving WebSocket connection failure
- 8ecf940: Add provisional Node.js support

- 585d2c9: Makes JS client compatible with the Cartesia Stable API (2024-06-10)

## 1.0.0-alpha.4

### Patch Changes

- c98a0c7: Fix typo in README

## 1.0.0-alpha.3

### Patch Changes

- 38af01f: Fix how URLs are constructed, solving WebSocket connection failure

## 1.0.0-alpha.2

### Patch Changes

- 585d2c9: Makes JS client compatible with the Cartesia Stable API (2024-06-10)

## 1.0.0-alpha.1

### Major Changes

- 3ee5bfc: Initial release of Cartesia client with voices and WebSocket support

## 0.0.4-alpha.0

### Patch Changes

- 8ecf940: Add provisional Node.js support

## 0.0.3

### Patch Changes

- 8b671ff: Revert queuing feature

## 0.0.2

### Patch Changes

- 9a31c66: Improve error handling logic, add connection info, make audio downloadable, add latency metrics, more robust reconnection handling, and misc. changes

## 0.0.1

### Patch Changes

- 0ea132f: Publish Cartesia JS client

# Cartesia JavaScript Client

![NPM Version](https://img.shields.io/npm/v/%40cartesia%2Fcartesia-js?logo=npm)
[![Discord](https://badgen.net/badge/black/Cartesia/icon?icon=discord&label)](https://discord.gg/cartesia)

This client provides convenient access to [Cartesia's TTS models](https://cartesia.ai/). Sonic is the fastest text-to-speech model around—it can generate a second of audio in just 650ms, and it can stream out the first audio chunk in just 135ms. Alongside Sonic, we also offer an extensive prebuilt voice library for a variety of use cases.

The JavaScript client is a thin wrapper around the Cartesia API. You can view docs for the API at docs.cartesia.ai.

## Installation

```bash
# NPM
npm install @cartesia/cartesia-js
# Yarn
yarn add @cartesia/cartesia-js
# PNPM
pnpm add @cartesia/cartesia-js
# Bun
bun add @cartesia/cartesia-js
```

## Usage

### CRUD on Voices

```js
import Cartesia from "@cartesia/cartesia-js";

const cartesia = new Cartesia({
  apiKey: "your-api-key",
});

// List all voices.
const voices = await cartesia.voices.list();
console.log(voices);

// Get a voice.
const voice = await cartesia.voices.get("<voice-id>");
console.log(voice);

// Clone a voice from a file.
const clonedVoiceEmbedding = await cartesia.voices.clone({
  mode: "clip",
  clip: myFile, // Pass a File object or a Blob.
});

// Mix voices together.
const mixedVoiceEmbedding = await cartesia.voices.mix({
  voices: [
    { id: "<voice-id-1>", weight: 0.6 },
    { id: "<voice-id-2>", weight: 0.4 },
  ],
});

// Localize a voice.
const localizedVoiceEmbedding = await cartesia.voices.localize({
  embedding: Array(192).fill(1.0),
  original_speaker_gender: "female",
  language: "es",
});

// Create a voice.
const newVoice = await cartesia.voices.create({
  name: "Tim",
  description: "A deep, resonant voice.",
  embedding: Array(192).fill(1.0),
});
console.log(newVoice);
```

### TTS over WebSocket

```js
import Cartesia from "@cartesia/cartesia-js";

const cartesia = new Cartesia({
  apiKey: "your-api-key",
});

// Initialize the WebSocket. Make sure the output format you specify is supported.
const websocket = cartesia.tts.websocket({
  container: "raw",
  encoding: "pcm_f32le",
  sampleRate: 44100,
});

try {
  await websocket.connect({
    // If using Node.js, you can pass a custom WebSocket constructor, such as from `ws`.
    // This is not needed for browser usage, so you can call connect() without any arguments.
    WebSocket: WS,
  });
} catch (error) {
  console.error(`Failed to connect to Cartesia: ${error}`);
}

// Create a stream.
const response = await websocket.send({
  model_id: "sonic-english",
  voice: {
    mode: "id",
    id: "a0e99841-438c-4a64-b679-ae501e7d6091",
  },
  transcript: "Hello, world!",
  // The WebSocket sets output_format on your behalf.
});

// Access the raw messages from the WebSocket.
response.on("message", (message) => {
  // Raw message.
  console.log("Received message:", message);
});

// You can also access messages using a for-await-of loop.
for await (const message of response.events("message")) {
  // Raw message.
  console.log("Received message:", message);
}
```

#### Input Streaming with Contexts

```js
const contextOptions = {
  context_id: "my-context",
  model_id: "sonic-english",
  voice: {
    mode: "id",
    id: "a0e99841-438c-4a64-b679-ae501e7d6091",
  },
};

// Initial request on the context uses websocket.send().
// This response object will aggregate the results of all the inputs sent on the context.
const response = await websocket.send({
  ...contextOptions,
  transcript: "Hello, world!",
});

// Subsequent requests on the same context use websocket.continue().
await websocket.continue({
  ...contextOptions,
  transcript: " How are you today?",
});
```

See the [input streaming docs](https://docs.cartesia.ai/reference/web-socket/stream-speech/working-with-web-sockets#input-streaming-with-contexts) for more information.

#### Timestamps

To receive timestamps in responses, set the `add_timestamps` field in the request object to `true`.

```js
const response = await websocket.send({
  model_id: "sonic-english",
  voice: {
    mode: "id",
    id: "a0e99841-438c-4a64-b679-ae501e7d6091",
  },
  transcript: "Hello, world!",
  add_timestamps: true,
});
```

You can then listen for timestamps on the returned response object.

```js
response.on("timestamps", (timestamps) => {
    console.log("Received timestamps for words:", timestamps.words);
    console.log("Words start at:", timestamps.start);
    console.log("Words end at:", timestamps.end);
});

// You can also access timestamps using a for-await-of loop.
for (await const timestamps of response.events('timestamps')) {
    console.log("Received timestamps for words:", timestamps.words);
    console.log("Words start at:", timestamps.start);
    console.log("Words end at:", timestamps.end);
}
```

#### Speed and emotion controls [Alpha]

The API has experimental support for speed and emotion controls that is not subject to semantic versioning and is subject to change without notice. You can control the speed and emotion of the synthesized speech by setting the `speed` and `emotion` fields under `voice.__experimental_controls` in the request object.

```js
const response = await websocket.send({
  model_id: "sonic-english",
  voice: {
    mode: "id",
    id: "a0e99841-438c-4a64-b679-ae501e7d6091",
    __experimental_controls: {
      speed: "fastest",
      emotion: ["sadness", "surprise:high"],
    },
  },
  transcript: "Hello, world!",
});
```

### Multilingual TTS [Alpha]

You can define the language of the text you want to synthesize by setting the `language` field in the request object. Make sure that you are using `model_id: "sonic-multilingual"` in the request object.

Supported languages are listed at [docs.cartesia.ai](https://docs.cartesia.ai/getting-started/available-models).

### Playing audio in the browser

(The `WebPlayer` class only supports playing audio in the browser and the raw PCM format with fp32le encoding.)

```js
// If you're using the client in the browser, you can control audio playback using our WebPlayer:
import { WebPlayer } from "@cartesia/cartesia-js";

console.log("Playing stream...");
```

```
// Create a Player object.
const player = new WebPlayer();

// Play the audio. (`response` includes a custom Source object that the Player can play.)
// The call resolves when the audio finishes playing.
await player.play(response.source);

console.log("Done playing.");
```

## React

We export a React hook that simplifies the process of using the TTS API. The hook manages the WebSocket connection and provides a simple interface for buffering, playing, pausing and restarting audio.

```jsx
import { useTTS } from "@cartesia/cartesia-js/react";

function TextToSpeech() {
  const tts = useTTS({
    apiKey: "your-api-key",
    sampleRate: 44100,
  });

  const [text, setText] = useState("");

  const handlePlay = async () => {
    // Begin buffering the audio.
    const response = await tts.buffer({
      model_id: "sonic-english",
      voice: {
        mode: "id",
        id: "a0e99841-438c-4a64-b679-ae501e7d6091",
      },
      transcript: text,
    });

    // Immediately play the audio. (You can also buffer in advance and play later.)
    await tts.play();
  };

  return (
    <div>
      <input
```

```
        type="text"
        value={text}
        onChange={(event) => setText(event.target.value)}
      />
      <button onClick={handlePlay}>Play</button>

      <div>
        {tts.playbackStatus} | {tts.bufferStatus} | {tts.isWaiting}
      </div>
    </div>
  );
}
```

```
{
  "$schema": "https://biomejs.dev/schemas/1.9.0/schema.json",
  "organizeImports": {
    "enabled": true
  },
  "linter": {
    "enabled": true,
    "rules": {
      "recommended": true
    }
  },
  "files": {
    "ignore": ["*.json"]
  }
}
```

bun.lockb

BASE64:

IyEvdXNyL2Jpbi9lbnYgYnVuCmJ1bi1sb2NrZmlsZS1mb3JtYXQtdjAKAgAAABjvjz8+y+CjO60sV8
kqdjHlKVREB+CqDHNKhEi7sBGR6fkAAAAAAACpAAAAAAAAAAgAAAAAAAAACAAAAAAAAACAAAAAAAAA
AIWnAAAAAAAAAAAAAAAAFQAAgHJlYWN0AAAAlgAAAAwAAIDyAAAACQAAgFgAAAALAACAcwEAABEAAI
BodW1hbi1pZGVtaXR0ZXJ5TQAAAAsAAICHAgAACgAAgM0CAAAKAACAEwMAABIAAIB0cjQ2AAAAAEQA
AAAJAACAOgAAAAoAAIB0c3VwAAAAANYEAAAJAACAzAQAAAoAAIB2BQAACQAAgGZkaXIAAAAAc3Vjcm
FzZQB4BgAAFAAAgHBpcmF0ZXMAbXoAAAAAAABXBwAACwAAgHRoZW5pZnkAPwcAAAsAAIBKBwAADQAA
gGcGAAARAACAZ2xvYgAAAAAaCQAAFgAAgAoJAAAQAACA1wkAAAsAAIDMCQAACwAAgHdoaWNoAAAAaX
NleGUAAABeCgAADwAAgDELAAANAACAcGF0aC1rZXn/CAAACwAAgBEMAAAJAACAbWluaXBhc3P2CAAA
CQAAgMcMAAAPAACAHA0AAA4AAIDtCAAACQAAgLUNAAAQAACAqA0AAA0AAIChDgAACQAAgHQOAAAKAA
CATA8AAAoAAIBBDgAADAAAgN0PAAAXAACA0g8AAAsAAIAFDwAACwAAgMYQAAANAACAFREAAAoAAICh
DgAACQAAgHQOAAAKAACATA8AAAoAAIBBDgAADAAAgE0SAAAOAACA0g8AAAsAAIAFDwAACwAAgF4GAA
AJAACARwYAABcAAIDNEwAAGwAAgLQTAAAZAACAiRQAABcAAICfEwAAFQAAgMIEAAAKAACAzQIAAAoA
AIATAwAAEgAAgHRyNDYAAAAAcHVueWNvZGWvFQAADQAAgHJvbGx1cAAAzBgAAB0AAICwGAAAHAAAgJ
MYAAAdAACAehgAABkAAIBcGAAAHgAAgDgYAAAkAACAGBgAACAAAID6FwAAHgAAgNcXAAAjAACAtRcA
ACIAAICWFwAAHwAAgHcXAAAfAACAWRcAAB4AAIA6FwAAHwAAgB4XAAAcAACAAxcAABsAAIBmc2V2ZW
50c/YWAAANAACAqgQAAAwAAICXBAAAEwAAgMYfAAAJAACAjQQAAAoAAIBqb3ljb24AAGV4ZWNhAAAA
4yAAABMAAIDWIAAADQAAgMogAAAMAACAviAAAAwAAIDXCQAACwAAgLQgAAAKAACAqyAAAAkAAIBvbm
V0aW1lAG1pbWljLWZuZXNidWlsZAAzJQAAFwAAgB0lAAAWAACAByUAABYAAIDxJAAAFgAAgNskAAAW
AACAxSQAABYAAICwJAAAFQAAgJwkAAAUAACAiCQAABQAAIB0JAAAFAAAgGAkAAAUAACATCQAABQAAI
A4JAAAFAAAgCQkAAAUAACAECQAABQAAID9IwAAEwAAgOojAAATAACA1yMAABMAAIDEIwAAEwAAgLIj
AAASAACAoCMAABIAAICOIwAAEgAAgHwjAAASAACAaiMAABIAAIBkZWJ1ZwAAAG1zAAAAAAAAY29uc2
9sYQBjaG9raWRhcu8sAAAOAACA4SwAAA4AAICFLQAAEQAAgNYsAAALAACAaXMtZ2xvYgBULgAACgAA
gHJlYWRkaXJwdgUAAAkAAIBhbnltYXRjaGJyYWNlcwAAeC8AAAoAAIC+LwAADgAAgBAwAAAJAACAY2
FjAAAAAAB/BAAADgAAgMYwAAANAACALgAAAAwAAIBQMQAAEQAAgGNzc3R5cGUAIwAAAAsAAIAUMgAA
DAAAgBUAAAAOAACATDMAAB0AAIAxMwAAGwAAgBkzAAAYAACAAzMAABYAAIDqMgAAGQAAgNMyAAAXAA
CAuzIAABgAAIClMgAAFgAAgA4VVM/+pBxEkOfY3hmQxJQC13GAl0EyK+ioig/W1gZwaYYmE1JWrVAQ
0ygfbSRXEjMaJq72rIDpqMGg1ldWOVqBOwfYujafTpD2paC6HQOCERnFlxjQONakKt6LdUkpSjpm4d
iZtYRDV1KcAP7v+sqfJYAsK4VlYpPKyoqsCSb4KpqukKR6bHszdLsU0RYn+VVg7P/6kGH2zUc447DR
40y2xXALm00VcupRY3tQiyh+f+7reTxKFp/SUtSF8Vh1QwX3ZOTq44kw8vOs6ahfwY2l9nhkUw7GDV
KJ1q4eWEoKetsfEMxjLXlmV43uGllORaoM1v7/Q9U0GRISbFQjD9S+UmnDAv7TkFB+bLzcsbOszG2O
96NP8kaoLFfaUxLq10AV35g6vwh9CulXHT+gjPrSc7/
+1VI8CYUQXGxshAGc8MR9cJzXzNNzBpB9Y5g6eB8C4l5R7u5wWb/uBZze7ShXUdRNXfLJ4cAaJQa7X
2DgqHn+bzOgG0GeqpLrFhXShVXT7CR6bxBZHSLhI8AJIrlaAZzbRvefcP1sAbBwIkTarG8xtGzuY57
sZFpV3TzARE4vwqIZysMUYnEkPUALDzjRWKrQSfSf5VXT7CR6bxBZHSLhI8AJIrlaAZzbRvefcP1sA
bBwIkTaWcY2UupK+ezsZFpV3TzARE4vwqIZysMU1g45bhIVS0mbFbfqGoW8a/
kRZWv+NM2ZefG12Ns/vH5gSWMLIolV2uH+za4oNA2Y5Oy6GsAg/
dERGcWXGNA41qQq3ot1SSlKOmbh2Jm1hEMpPmKJNw4FXc6vIjuZVLHBUGzbSA+lQZGIcpsn3YhJDQ/
i0TEs/Mv8HXpPAeV81YA/B1FmcojJZFTev6GH9IEakoEcWX529Sg3J604MvvQRuZBJQj1Q0TE+F6Zf
jScyv+Lj/2ctyMR6jjwe08OEIfRsyUEXTDohRMGdNJkbM7zAlGa7u5AoNH+tf/
s8goayiW7FBYSXY+ZgriQgheF7ojePl4CvNTvCN/s+GLdXxMINE/5w7W8BGKaeBxp0OGIXvJziUj+t

KUkfPMLdoQgFPmSfVlZHWD0kxxwVc9Y51dyf6WWnuwFIGBBdlcyHY18TsrLCOkmSk7KzL5SacMC/
tOQKPHe73MX8LfWTkan64/hNFc6q1+NEQGXr11EIwuXnOoQIz1Ddwfzt9GhEB3EKOlPMhF9VuG6UmJ
sYKL/1pG5QatGaV1yPyecDRdE6Z6udsdSPtArHB0Rp54bMLrT6WNayPaAH3ykVZOTO+wjao2aiGIW/
AkPHxT2KhIoMwGL46Hbg2ZGMZbawcIA7kcm6QJAGPr6MVLKNmXIB1CXRuhtwggRPiu4AZTMiASsaF8
ROniiqX5q7ggl0jBCml+3Bg8CNkzMtqGStHrB5ZGoExdH9+eWdPKhP/a7abk8qDQGbebe3r+dqzK9O
Y6DADThGsrGpLTE1bXXj0jcP32d+YXBNmfAil0bPy+6fAZdmz2MpW5ED7CndI3QimSMtlZp3J7unoa
7Q6j9DqZ3hGbmLk91NQqd4dqEB+MzIh+XGR0iUotVYOz/
+pBh9vh1+YRV65K1wBTYBrrD5x8988CxyhA1aARYMlTNBqrFKi9I/PhbFPLuvAb+deoPPh3Xg1uY38
iILgUDmlSmFiYaFSURgSNTzL2M9EYU73nGOfZ0DAt5k5GRL1qNJrU9OQlvPMbqUZq7opxsqHHDJq5V
j0CCrkriI7qaZRhHlGrDPy2pCGraHu/l+TNr5CMDAr2sSD8cVzifD/OXlh8+z6DRNrNCv87yXqzaFj
XB9k7RAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAIAAAAAAAAAAYwAAADMAAIASAAAAAwAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAACAAAAAAAAAKIAAABAAACAAQAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAD7AAAAOgAAgAQAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAANQEAAD4AAIABAAAAAAAAAAAIAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAIQBAABKAACABgAAAAAAAAAACAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADZAQAAOAAAgAQAAAAB
AAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAEQIAADgAAI
ABAAAAAAAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAEkC
AAA+AACABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAA
AAAACRAgAAPAAAgAIAAAAHAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAIAAAAAAAAA1wIAADwAAIAFAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAACAAAAAAAAACUDAABMAACAAwAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAgAAAAAAAABxAwAAMAAAgAAAAAAAAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAoQMAADoAAIABAAAABQAAAAEAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAOcDAAA8AACABQAAAAYAAAADAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABPBAAAMAAAgAgAAAADAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAAUAADoAAIABAAAAAgAAAA
IAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAADoFAAA8AACAAAAA
AAIAAAAJAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAB/BQAAOg
AAgAQAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAA
uQUAADAAAIAGAAAABAAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAA
AAAAAAABAGAAA3AACAAwAAACMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAgAAAAAAAACMBgAAUQAAgAAAAAABAAAADQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAIAAAAAAAAA3QYAADYAAIAEAAAAAAAAAAYAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAACAAAAAAAAABMHAAAsAACAAgAAAAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABiBwAAPgAAgAEAAAAGAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAArAcAADYAAIADAAAAAwAAAAEAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAOIHAAA+AACAAQAAAAMAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAgCAAAQgAAgAQAAAABAA
AAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAYggAAEoAAIAB
AAAAAgAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAALwIAA
AxAACACgAAAAQAAAAFAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAA
AAAwCQAAVAAAgAEAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

IAAAAAAAAAhAkAAEgAAIADAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAACAAAAAAAAAOIJAAA+AACABAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAgAAAAAAAAAgCgAAPgAAgAcAAAAAAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAhwoAADIAAIACAAAAAAAAAAIAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAALkKAAAyAACAAgAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADrCgAARgAAgAIAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAPgsAAEIAAIADAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAIALAAA4AACAAwAAAA
EAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAC4CwAAPwAA
gAEAAAALAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAGg
wAADsAAIAKAAAABAAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAA
AAAAAFUMAAA4AACABwAAAAEAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAgAAAAAAAACNDAAAOgAAgAkAAAAAAAAAABQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAIAAAAAAAAAA1gwAAEYAAIACAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAACAAAAAAAAAACoNAABEAACAAQAAAAAAAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABuDQAAOgAAgAMAAAAEAAAAAwAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAxQ0AAEIAAIAAAAAACwAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAAcOAAA6AACACAAAAAAAAAAACAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADLDgAAOgAAgAcAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAEA8AADwAAIAGAA
AAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAFYPAAA8
AACABQAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAA
CSDwAAQAAAgAQAAAACAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIA
AAAAAAAA9A8AAFYAAIADAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAACAAAAAAAAAAEoQAAA+AACACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAgAAAAAAACIEAAAPgAAgAQAAAADAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAIAAAAAAAAAA0xAAAEIAAIACAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAB8RAAA8AACAAQAAAAEAAAAEAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABbEQAAOgAAgAgAAAABAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAlREAADwAAIAHAAAAQAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAANERAAA8AACABgAAAAEA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAANEgAAQAAAgA
UAAAABAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAWxIA
AEQAAIAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAA
AAAJ8SAAA+AACACQAAAAIAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AgAAAAAAAADdEgAAPgAAgAYAAAACAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAIAAAAAAAAAAGxMAADoAAIAEAAAAQAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAACAAAAAAAAAAFUTAABKAACAAAAAAAMAAAAFAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADoEwAAUgAAgAEAAAAFAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAOhQAAE8AAIAAAAAAAwAAABkAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAKAUAABKAACAAwAAAAEAAAACAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADqFAAARgAAgAEAAAACAAAAAQ
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAMBUAAEMAAIAAAAAA
CAAAAAAAAAAAAAAYmV0YS4wAAC9kOrFfikE+QAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAHMVAAA8AA

CABwAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAC8
FQAATAAAgAQAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAA
AAAAAACBYAADAAAIABAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAACAAAAAAAAADgWAAA4AACAAgAAAAMAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAgAAAAAAAABwFgAAQgAAgAQAAAAHAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAIAAAAAAAAAwRYAADUAAIAEAAAAGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAOkYAABbAACABAAAABgAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABEGQAAWQAAgAQAAAAYAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAnRkAAFsAAIAEAAAAGAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAPgZAABTAACABAAAABgAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABLGgAAXQAAgAQA
AAAYAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAqBoAAG
kAAIAEAAAAGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAA
ABEbAABhAACABAAAABgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAg
AAAAAAAAByGwAAXQAAgAQAAAAYAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAIAAAAAAAAAzxsAAGcAAIAEAAAAGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAACAAAAAAAAADYcAABlAACABAAAABgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAgAAAAAAAACbHAAAXwAAgAQAAAAYAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAA+hwAAF8AAIAEAAAAGAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAFkdAABdAACABAAAABgAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAC2HQAAXwAAgAQAAAAYAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAFR4AAFkAAIAEAAAAGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAG4eAABXAACA
BAAAABgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADFHg
AAOAAAgAIAAAADAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAA
AAAA/R4AADsAAIABAAAAAAAAAAYAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ACAAAAAAAAADgfAABAAACABQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAgAAAAAAAAB4HwAATgAAgAYAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAIAAAAAAAAAzx8AADoAAIADAAAAAQAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAkgAAA8AACAAQAAAAEAAAABAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABFIAAANAAAgAMAAAABAAAAAQAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAeSAAADIAAIAFAAAAAQAAAAEAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAPYgAABOAACAAgAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABEIQAAQgAAgAIAAA
ABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAhiEAAEAA
AIAEAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAM
YhAABAAACAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAA
AAAAAAAGIgAAPgAAgAMAAAAAAAAABwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAIAAAAAAAAAARCIAADwAAIAGAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAACAAAAAAAAAIAiAAA6AACAAgAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAgAAAAAAAAC6IgAANgAAgAUAAAABAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAA8CIAADgAAIACAAAAAQAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAADMjAAA3AACAAAAABcAAAABAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABKJQAATgAAgAAAAAAXAAAAAQAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAmCUAAEwAAIAAAAAAFwAA
AAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAOQlAABMAACAAA
AAABcAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAwJgAA
TAAAgAAAAAAXAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAA
AAfCYAAEwAAIAAAAAAFwAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAC
AAAAAAAAAMgmAABMAACAAAAAABcAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAgAAAAAAAAAUJwAASgAAgAAAAAAXAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAIAAAAAAAAAXicAAEgAAIAAAAAAFwAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAACAAAAAAAAAKYnAABIAACAAAAAABcAAAABAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADuJwAASAAAgAAAAAAXAAAAAQAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAANigAAEgAAIAAAAAAFwAAAAEAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAH4oAABIAACAAAAAABcAAAABAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADGKAAASAAAgAAAAAAX
AAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAADikAAEgAAI
AAAAAAFwAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAFYp
AABIAACAAAAAABcAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAA
AAAACeKQAARgAAgAAAAAAXAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAIAAAAAAAAA5CkAAEYAAIAAAAAAFwAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAACAAAAAAAAACoqAABGAACAAAAAABcAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAgAAAAAAAABwKgAARgAAgAAAAAAXAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAtioAAEQAAIAAAAAAFwAAAAEAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAPoqAABEAACAAAAAABcAAAABAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAA+KwAARAAAgAAAAAAXAAAAAQAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAgisAAEQAAIAAAAAAFwAAAA
EAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAMYrAABEAACAAAAA
ABcAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAKLAAAMg
AAgAQAAAADAAAABwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAA
PCwAACwAAIACAAAAAQAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAA
AAAAAAAGgsAAA2AACAAwAAAAIAAAADAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAgAAAAAAAACeLAAAOAAAgAMAAAAGAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAIAAAAAAAAA/SwAAEQAAIADAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAACAAAAAAAAAEEtAABEAACAAgAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAACWLQAASgAAgAIAAAADAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAA4C0AAD4AAIAFAAAAAQAAAAIAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAB4uAAA2AACABAAAAAAAAAADAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABeLgAAPAAAgAIAAAABAA
AAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAmi4AADgAAIAD
AAAABgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAANIuAA
A6AACAAgAAAAMAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAA
AAAMLwAAOAAAgAMAAAABAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
IAAAAAAAAARC8AADQAAIADAAAAAAAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAACAAAAAAAAAIIvAAA8AACABwAAAAEAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAgAAAAAAAADMLwAARAAAgAUAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAGTAAADoAAIAHAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAFMwAAAvAACABgAAAAcAAAAOAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAACCMAAARAAAgAUAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAA0zAAAEIAAIAAAAAAAgAAAUA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAABUxAAA7AACAEgAAAA
MAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAABhMQAARQAA
gA8AAAAHAAAADQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAApj
EAADYAAIADAAAAAQAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAA
AAAAANwxAAA4AACAFgAAAAcAAAAJAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAgAAAAAAAAgMgAAQQAAgAYAAAATAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAIAAAAAAAAAajIAADsAAIABAAAACQAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAACAAAAAAAAAGkzAABZAACAAQAAAAkAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAADCMwAAVQAAgAEAAAAJAAAABAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAFzQAAE8AAIABAAAACQAAAAQAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAGY0AABLAACAAQAAAAkAAAAEAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAACxNAAAUQAAgAEAAAAJAAAA
BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAjUAAE0AAIABAA
AACQAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAE81AABP
AACAAQAAAAkAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAA
CeNQAASwAAgAEAAAAJAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAALAAAACwAAAAEAAAAMAAAAAQAAAA0AAAAAAAADQAAAAEAAAAOAAAAAAAAAAA4AAAAAAAAAADgAAAA
AAAAAOAAAAAQAAAA8AAAACAAAAEQAAAAIAAAATAAAAAAAAAABMAAAAAAAAAAEwAAAAAAAAAATAAAAAAAAAA
ABMAAAAUAAAAJwAAAAAAAAAAnAAAAAgAAACkAAAAAAAAAAKQAAAAEAAAAqAAAABwAAADEAAAAAAAAAAMQ
AAAAAAAAAxAAAAAwAAADQAAAABAAAANQAAAAEAAAA2AAAAAAAAAADYAAAAAAAAAAANgAAAAAAAAAA2AAAA
BgAAADwAAAAAAAAAAPAAAAAIAAAA+AAAAAAAAAAD4AAAADAAAAQQAAAAEAAABCAAAAAAAAAAEIAAAABAA
AAQwAAAAAAAAAABDAAAAAAAAAAEMAAAACAAAARQAAAAAAAAAABFAAAAAAAAAAEUAAAABAAAARgAAAAEAAABH
AAAAAAAAAEcAAAACAAAASQAAAAAAAAABJAAAABgAAAE8AAAADAAAAUgAAAAEAAABTAAAAAAAAAAFMAAA
ADAAAAVgAAAAAAAAAABWAAAAAAAAAAFYAAAABAAAAVwAAAAEAAABYAAAAAAAAAAFgAAAADAAAAWwAAAAEA
AABcAAAAAAAAAAAFwAAAADAAAAXwAAAAAAAAABfAAAAAAAAAAAF8AAAAAAAAAAAXwAAAAAAAAABfAAAAAwAAAG
IAAAAAAAAAAAYgAAAAIAAABkAAAAAAAAAAGQAAAAAAAAAAAZAAAAAEAAABlAAAAAwAAAGgAAAAAAAAAAaAAA
AAEAAABpAAAAAAAAAAAGkAAAAAAAAAAaQAAABIAAAB7AAAAAAAAAAHsAAAAAAAAAAAewAAAAAAAAAAB7AAAAAA
AAAHsAAAAAAAAAAAewAAAAAAAAAAB7AAAAAAAAAAHsAAAAAAAAAAewAAAAAAAAAAB7AAAAAAAAAAAHsAAAAAAAAAAA
ewAAAAAAAAAAB7AAAAAAAAAAAHsAAAAAAAAAAewAAAAAAAAAAB7AAAAAAAAAAAHsAAAAAAAAAAewAAAAAAAAAAB7AA
AAAAAAAHsAAAAFAAAAgAAAAAAAAAAACAAAAAAAAAAAAIAAAAAAAAAAAgAAAAAkAAACJAAAAAAAAAAIkAAAAA
AAAAiQAAAAEAAACKAAAAAAAAAAAIoAAAAAAAAAAAigAAAAAAAAAACKAAAAAAAAAAIoAAAABAAAAiwAAAAAAAA
CLAAAAGAAAAKMAAAAAAAAAAAowAAAAAAAAAACjAAAAAAAAAAAKMAAAAAAAAAAAowAAAAAAAACjAAAAAAAAAAKMA
AAAAAAAAowAAAAAAAAAACjAAAAAAAAAAAKMAAAAAAAAAAowAAAAAAAAAACjAAAAAAAAAAAKMAAAAAAAAAAowAAAA
AAAACjAAAAAAAAAAAKMAAAAAAAAAAowAAAAAAAAAACjAAAAAAAAAAAKMAAAAAAAAAAowAAAAAAAAAACjAAAAAAAAAA
AKMAAAAAAAAAAAowAAAAAAAAAACjAAAAAAAAAAAKMAAAABAAAApAAAAAAAAAACkAAAAAAAAAAKQAAAAIAAAArA
AAAAAAAACsAAAAAQAAAK0AAAAAAAAAArQAAAAEAAACuAAAAAQAAAK8AAAAAAAAAArwAAAAEAAACwAAAA
AAAAALAAAAACAAAAsgAAAAEAAACzAAAAAQAAALQAAAABAAAAtQAAAAAAAAC1AAAAAAAAAALUAAAACAA
AAtwAAAAAAAAAAC3AAAAAgAAALkAAAAAAAAAAuQAAAAAAAAAAC5AAAAAQAAALoAAAAAAAAAAugAAAAgAAADC
AAAAAAAAAMIAAAAAAAAAAwgAAAAAAAAAADCAAAAAAAAAAMIAAAAAAAAAAAwgAAAAAAAAAADCAAAAAAAAAAMIAAA
AAAAAAAAAAAAsAAAALAAAAAQAAAAwAAAABAAAADQAAAAAAAAAANAAAAAQAAAA4AAAAAAAAAADgAAAAAA
AAAOAAAAAAAAAAAA4AAAABAAAADwAAAAIAAAARAAAAAgAAABMAAAAAAAAAAEwAAAAAAAAAAATAAAAAAAAAAB

MAAAAAAAAAEwAAABQAAAAnAAAAAAAAACcAAAACAAAAKQAAAAAAAAApAAAAAQAAACoAAAAHAAAAMQAA
AAAAAAAxAAAAAAAAADEAAAADAAAANAAAAAEAAAA1AAAAAQAAADYAAAAAAAAANgAAAAAAAAA2AAAAAA
AAADYAAAAGAAAAPAAAAAAAAAA8AAAAAgAAAD4AAAAAAAAAPgAAAAMAAABBAAAAAQAAAEIAAAAAAAAA
QgAAAAEAAABDAAAAAAAAAEMAAAAAAAAAQwAAAAIAAABFAAAAAAAAAEUAAAAAAAAARQAAAAEAAABGAA
AAAQAAAEcAAAAAAAAARwAAAAIAAABJAAAAAAAAAEkAAAAGAAAATwAAAAMAAABSAAAAAQAAAFMAAAAA
AAAAUwAAAAMAAABWAAAAAAAAAFYAAAAAAAAAVgAAAAEAAABXAAAAAQAAAFgAAAAAAAAAWAAAAAMAAA
BbAAAAAQAAAFwAAAAAAAAAXAAAAAMAAABfAAAAAAAAAF8AAAAAAAAAXwAAAAAAAABfAAAAAAAAAF8A
AAADAAAAYgAAAAAAAABiAAAAAgAAAGQAAAAAAAAAZAAAAAAAAABkAAAAAQAAAGUAAAADAAAAaAAAAA
AAAABoAAAAAQAAAGkAAAAAAAAAaQAAAAAAAABpAAAAEgAAAHsAAAAAAAAAewAAAAAAAAB7AAAAAAAA
AHsAAAAAAAAAewAAAAAAAAB7AAAAAAAAAHsAAAAAAAAAewAAAAAAAAB7AAAAAAAAAHsAAAAAAAAAew
AAAAAAAAB7AAAAAAAAAHsAAAAAAAAAewAAAAAAAAB7AAAAAAAAAHsAAAAAAAAAewAAAAAAAAB7AAAA
AAAAAHsAAAAAAAAAewAAAAUAAACAAAAAAAAAAIAAAAAAAAAAgAAAAAAAAACAAAAACQAAAIkAAAAAAA
AAiQAAAAAAAAACJAAAAAQAAAIoAAAAAAAAAigAAAAAAAACKAAAAAAAAAIoAAAAAAAAAigAAAAEAAACL
AAAAAAAAAIsAAAAYAAAAowAAAAAAAACjAAAAAAAAAKMAAAAAAAAAowAAAAAAAACjAAAAAAAAAKMAAA
AAAAAAowAAAAAAAACjAAAAAAAAAKMAAAAAAAAAowAAAAAAAACjAAAAAAAAAKMAAAAAAAAAowAAAAAA
AACjAAAAAAAAAKMAAAAAAAAAowAAAAAAAACjAAAAAAAAAKMAAAAAAAAAowAAAAAAAACjAAAAAAAAAK
MAAAAAAAAAowAAAAAAAACjAAAAAAAAAKMAAAAAAAAAowAAAAEAAACkAAAAAAAAAKQAAAAAAAAApAAA
AAgAAACsAAAAAAAAAKwAAAABAAAArQAAAAAAAACtAAAAAQAAAK4AAAABAAAArwAAAAAAAACvAAAAAQ
AAALAAAAAAAAAAsAAAAAIAAACyAAAAAQAAALMAAAABAAAAtAAAAAEAAAC1AAAAAAAAALUAAAAAAAAA
tQAAAAIAAAC3AAAAAAAAALcAAAACAAAAuQAAAAAAAAC5AAAAAAAAALkAAAABAAAAugAAAAAAAAC6AA
AACAAAAMIAAAAAAAAAwgAAAAAAAADCAAAAAAAAAMIAAAAAAAAAwgAAAAAAAADCAAAAAAAAAMIAAAAA
AAAAwgAAAAAAAAAAP4P/gEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAQD+D/4BAAABAAAAAAAAAAA
AAAEwS+hAgJShR0KhEvPJArfuPVN1+Hz1t0Y6n5jLrGQbkb4urgPE/0Rve+1kMB1v/
oWgHgm4WIcV+i7F2pTVj+2iQEAAAEA/g/+AQAAAgAAAAAAAAAAAAAABJcrsTxq/1n4a5Xptgi/1HJ1
HNc3KigCJgQ87pGO2ORf8kIjXZKOvn0S3r5cNR4DMksO3+tdVCGONPBLcUUqCt0BAAABAP4P/gEAAA
MAAAAAAAAAAAAAAARF0lR+VwTdxTMqIypCCwK7ToU+71R0gk7Rt5hs+Ec3iTpqmAm2J9ygK1P1tzE6
lgG2kPaQIzpJvODgJq6xb88pAQAAAQD+D/4BAAAEAAAAAAAAAAAAAAAErAFOUKImaq+VBk2B+2RTBs
WEvlnarEP53nchoUHzpVs8V6fG2/estihOTslTQUWHVuHEKDL5k8htG8K3TngyFAEAAAEA/g/
+AQAABQAAAAAAAAAAAAAABPKty7GY0Xs2xWdjzfrj/kO6Zx1DJgDs4lNrwNjsKBWiiF8ZcZd/
CMpYUkoFKK1rv63+Hq+u8NIpAaowRZP6aJgBAAABAP4P/gEAAAYAAAAAAAAAAAAAAATeAqb+AJJSKl
4uxFhltt0DUEPzeUGhZSRnsb1UWuFY8BTvorBijkpHxXbxumg+b/KDLTHn8YJulFB1jtUtgKgiAQAA
AQD+D/4BAAAHAAAAAAAAAAAAAAAEtJdCJitoy2lrC2ldJcqN4vkqJ00lT+tOWNT1hBJjO/3FDMJa5T
TIiYGCKGkn/WfCyOzUMObeohbVTj00fhiLiAEAAAEA/g/+AQAACAAAAAAAAAAAAAAABHuGuTfJVb7i
4AFoJwqy69jz9GMgzkx2vR/0Y+eHYq6etal5zauOpvNm0YTaiAOVXHCbVgbz0xUVd07bx1CjwdYBAA
ABAP4P/gEAAAkAAAAAAAAAAAAAAARzgVF9Sb91txZntT7VarQDALIUv7ee3ZsTDTnB/FLP4NalayK2
CZKBibLZ1B1bIoLXr3gQs+oyz9jNRI2jMu3wAQAAAQD+D/4BAAAKAAAAAAAAAAAAAAAEsaE57nupxk
6v3HY35+jzBwYa0rKSy0XR8JSxZPwgLr7ys0IBzhGviA1/
TUGJLmSVqs8pb9AnvICXEuOHLprYTwEAAAEA/g/+AQAACwAAAAAAAAAAAAAABNiQJ98/
AEeq4yvEpvKK0QtIf23Jfw6i+7UT3RmeCNQo3RfhGjC5mMQR8l7ii/OPXrnDxYbwaMTLH5Xzm/
JMWnkBAAABAP4P/gEAAAwAAAAAAAAAAAAAAAQ3dYyy6pXrqVPfQKtc1sSPHgYTCWjDe/quviYJy/
qmud/CFLTWuSDIV2M80Fh31uvsuldXX4SaHTV8eerYZ2CvAQAAAQD+D/4BAAANAAAAAAAAAAAAAAAE
AKpaYlHn8t4SVbOHCy+b5+KKgvR4vrsD8vbvrbiQJps7fKDTkjkDry6ji0rUJjC0kzbNePLwzxq8iy
po41qeWAEAAAEA/g/+AQAADgAAAAAAAAAAAAAABIY3EtZoX7souFlvCFrYz+26w6xtnLg2bpMq2K0m

rqFxjYMdEu83Hj9O2nWJCcnBK+egTlEzT82yJ6KIjd359asBAAABAP4P/gEAAA8AAAAAAAAAAAAAAA
QAuxwR7JLTciiZy410V1zRhYZUiTAfVazud9NsA+6ggu8RXaGkZABjvYsg6OCUajhKFall+kGo/
+OlBULOphBqAQAAAQD+D/4BAAAQAAAAAAAAAAAAAAEL0Orpi8qGpRG//Nd+H90vFB+3iHnue1zSSG
mNOOCh1GLJ7rUKVwV2HvijphGQS2UmhUZewS9VgvxYIdgr+fG1AEAAAEA/g/+AQAAEQAAAAAAAAAAAAA
AAABPKK9fgRhHZNWZJMNooteqkku7la0CT63uQPaP15FjwE5JNtxWYf9eQShcT55U8ur5JY2cS6mAs
8wTI99Ga0r48BAAABAP4P/gEAABIAAAAAAAAAAAAAAAQzsEBXpGVzLm76bqg+EA8WAlPMCKhf/
oHQPHK8OWj2Xz5PecsputzODZYtTLN3jkvxGp9QzIY/N6Rsy9fYt2TCAQAAAQD+D/4BAAATAAAAAAA
AAAAAAAAEKnhMXsKSPZlAhp7+IjUkRZKPb4fUyccpDrdFXbi4QL1qkmFh9kVY09Yox+n4MaOb3lHZ1
Tv829C3oaaXoMYPDQEAAAEA/g/+AQAAFAAAAAAAAAAAAAAABPBG1Q4rvYjf5xEsMXksQyntHbp7WtR
jpR7n5kkl8TA9s9v7TGaQzKb10BrHPmoxqPMtrmFJosWkkVHP0D6ENBgBAAABAP4P/gEAABUAAAAAAA
AAAAAAAAARj9qu9uf6uvPckIqX0LiRU19N9Kbb+YSnkVLPkSxlIA0Y9KVCulEjkzg8oX6YmcTnaM47
3Q+c9JzdSvdtNDDSAAQAAAQD+D/4BAAAWAAAAAAAAAAAAAAAEsaLsH7WeYYPiD25LDuLRRY/
i+6HaPYr6G1OUlN39otzkSTxKnubR9RTxS3/Kk50s1g2JTgFwWQDQyplC5/SHZgEAAAEA/g/
+AQAAFwAAAAAAAAAAAAAABM/NRjTu552DBIaxofS3spqBOPmK9Fp+THByGTCuXH0ApfjQ19PLAmYFH
Pf+jB54vSFrhS5tWdx0wl7ts/XzetkBAAABAP4P/gEAABgAAAAAAAAAAAAAAARE3FAf+ojz+3e2Fck
PByy1Q7jN6qjrj5TL/6w1VEHHhefY5f45n2g/6Imc0WqmUWtrZlRV4oJJrahVaLdPi5WYAQAAAQD+D
/4BAAAZAAAAAAAAAAAAAAAERVZSIV5IG10Hk3enotrhvz0T9em6cyHBLkH/
YAZuKqd8hRkKhSfCGIcP2KUY0EPxndzANBmNllzWPwak+bheSwEAAAEA/g/+AQAAGgAAAAAAAAAAAAAA
AABO1L5impVkbdcIIy9UaxsaEiVv9EGRSHoKXhr2RvZI6fL60bueV0x28J6qthqV5vbi23LocZtyKl
/TgeDGUdW9gBAAABAP4P/
gEAABsAAAAAAAAAAAAAAASsmBNCeRScfWwXDzJPpVJTfMPexaa7qxmEix5jxVf4ZG7c/oXsW74k0Oh
d+SUSVsslKdzcVRAdV7hxTmGP4FxSAQAAAQD+D/4BAAAcAAAAAAAAAAAAAAAE7ylylesZQ/
PV29jhEDl3Ufjo6ZX7gCqJr5F7PKrqc93v7fzSymt1BpwEU8nAUXs8qzzvqhbjhK5QZg6Mt/
HkBgEAAAEA/g/+AQAAHQAAAAAAAAAAAAAABOwb/
ERdJOsY6O3eAPz8WC21An2+nPlaXdv5gdskQ5XsOyW+YRF4gg/
Ym3zu8KZPIuLHryugxZ8vYexGGzN/7B4BAAABAP4P/
gEAAB4AAAAAAAAAAAAAAARQRkhLf9vLg4Ly8vc/Z1NdEROl5ssjY2Ijm8iuNoP/lS2uQVf+01vCNNJ
EAYL/7sICjakhwFpGBaZwEEdyxoIjAQAAAQD+D/4BAAAfAAAAAAAAAAAAAAAELd2g8rrAyMYFXBhEq
Mz8ZAHBi4J4uS1i/CxGMDnjyFWddMXLVcDp051DZfu+t7+ab7Wv6SMqpWmyFIj5UbfFvgEAAAEA/g/
+AQAAIAAAAAAAAAAAAAAABG88mdXvPMPTtYjSWypzpb2E61jw5eOjtWxtA91yJ7/
vbZD68azfI1FE4hZQ5JJiloJ9TOgnyANd0rhqjmvSqK8BAAABAP4P/
gEAACEAAAAAAAAAAAAAAASJEM8kpQ9UQ0Pt0c87yuRs6c+nIPKBwMW1aOl5Y0KDLxY/atdzFcvxOyR
F5CXo6sHYbv5QmtqCzWrXkW51zsbrAQAAAQD+D/4BAAAiAAAAAAAAAAAAAAAEBLI3Tl1TW3Pvl70l3
yq3Y64i+awpwXqsGBYWkkqMtnbXgrMD+yj7rhW0kuEDxzJaYXGjEW5ogapKNMEKNMjibAEAAAEA/g/
+AQAAIwAAAAAAAAAAAAAABER8TC6fZZyhxh0Z4PUBYUQjG2AHFaZ+vbJkhnKt39+sY4FVVk4Y+Kqi2
0y5au0rI/AfnyENRLghBiNpSrMkHiMBAAABAP4P/gEAACQAAAAAAAAAAAAAAASQfGvbNmli12as3Wo
OOutf9nWtHWQbwPH6CSkrUbh5ea9ezCZwTWFNYFZhTOWtpjDX/
Jmnpi4Njvti29s3R2YMAQAAAQD+D/4BAAAlAAAAAAAAAAAAAAAE7+
+dFhtcx3353uBaq8DDR4NuxBetBzC7ZQOhmTQInHEd6bSrXdiEyzCvG07Z44UYdLShWUyXt5M/
yhz8ekcb1AEAAAEA/g/+AQAAJgAAAAAAAAAAAAAABKI5njdKnfstI7MxLaGOPK9D3quXcDBJCJQjru
kOX+NZX5LMF7irWK4YKE6S58iHB5tuFIasfuU6ptiJ0sC4ROkBAAABAP4P/gEAACcAAAAAAAAAAAAA
AARdrg3DXsVL0ClAUn26YuIlLiisaObtnPBSvBqZwZC4dLMPK2H1ugoNrJxh0NxkO6pgBNfDgcVeBq
pZNy1b+/UcAQAAAQD+D/4BAAAoAAAAAAAAAAAAAAAEJNAzZcXrCt42VGLuYz0zfAzDfAvJWW6AfYlD
BQyDV5DClI2m5sAmK+OIO7s59XfsRsWHp02jAJrRadPRGTt6SQEAAAEA/g/

+AQAAKQAAAAAAAAAAAAAAABKjjs0tXAU1mBeAR/
H1XjwwTjvYqbTJxlBGcDXP3DFp01dp1S2e1aDVhDx5GHM2QNKXaAO3ZenuxS+ufZ1/
UtmsBAAABAP4P/gEAACoAAAAAAAAAAAAAAAQbpPRlfjzGCjPHvnzuSh5f1izY1jLoaa//8/
z2wS171X3CEhqkw0XiJ0rGdbZC0JwuJNaVv/B8JpsC0AVaGEGjAQAAAQD+D/4BAAArAAAAAAAAAAAA
AAAEXnAIvQ8eM+kC6aULx6wuQiwVsnzsi9d3WxzV3FpWTGA19F621kwdbsAcFKXgKUHZWsy+mY6iL1
sHTxWEFCytDAEAAAEA/g/+AQAALAAAAAAAAAAAAAAABN6EnlDtEzFeu4TdQJm17CuMmqlO7Y4h5W8U
Q2TqR9Clvfgnl+G0QGl9AJ8bdLcdjK6UaVsEGj8CJSEhCYWFOT8BAAABAP4P/gEAAC0AAAAAAAAA
AAAAQ4aVlCnPbJ9qED9F3VjwJ31IgSyq9eQtWhLD9yDCGeEUwNuxAV5ligkntshkFL0FxqZRb3pqyr
+ek9a6Az5FAHAQAAAQD+D/4BAAAuAAAAAAAAAAAAAAAAE+1VkjdD0QBLPodGrJUeqarH8VAIvQODIbw
h9XpP5Syisf7YoQgsJKPNFoqqLQlu+VQ/tVSshMR6loPMn8U+dPgEAAAEA/g/
+AQAALwAAAAAAAAAAAAAABDvI3I2m12pXjhvQ0NPgEV1mQU35z+FjQKs7oiSu5ZeOAJsRir/
ydjOEz48Y2N85wQn7wVxc7nJtbcHchcmxahABAAABAP4P/
gEAADAAAAAAAAAAAAAAAARhUYiPaRqYtJPHDo2xmOgHF9LCyfTJx16yZzin5DbVznM+5nWmX41/FV3
E+10e+Y1U5DpdJgbgBS3K38WLsPXpAQAAAQD+D/4BAAAxAAAAAAAAAAAAAAAEY38VPSHcqkFrCpFnQ
9vuSXmquuv5oXOKpGeT6aGrr3o3Gc9AlVa6JBfUSOCnbxGGZF+/0ooI7KrPuUSztUdU5AEAAAEA/g/
+AQAAMgAAAAAAAAAAAAAABKriUF5U0lBi9ix/
UlF6PFcLGOLKGp4YKOizUpvOBNSwXBPLNztMKXYkc8kfc/2WSTJTFr9+6jjm/
aXSZTFBChUBAAABAP4P/gEAADMAAAAAAAAAAAAAAATArJBFCmMnSwinrYStJl0ayMwlaxqnmhE2KEe
G7obslU7/2MgHpTJ68v61e46queDyP9zEpNbJZTC9JOuKJnP+AQAAAQD+D/4BAAA0AAAAAAAAAAAA
AAEzymm5+u+sCsSWyD9qNaejV3DFvhCKclKdizYaJUuHA83RLjb7nSuGnddCHGv0hk+KY7BMAlsWeK
4Ueg6EV6XQgEAAAEA/g/+AQAANQAAAAAAAAAAAAAABDEo2M3FjTgNHsAB6c9DMaWBb8IOso8tTRt8b
XqKs+uOFQqP0T4J69fxhrfonN4iU80PBLt03TNeEmsJ1VJhhOgBAAABAP4P/gEAADYAAAAAAAAAAA
AAATNsH2sIkBPWtuOJUNvaGooUc1gvGC2Tw1RHFnchnAPcXo23FtdlAKedKLUuTH4gOiF0+UWnbbbB
UAsiF5klBISAQAAAQD+D/4BAAA3AAAAAAAAAAAAAAAERRECPsj7iu/
xb5oKYcsFHSppFNnsj/52OVTRKb4zP5onXwVF3zVmmToNcOfGC+CRDpfK/
U584fMg38ZHCaElKQEAAAEA/g/+AQAAOAAAAAAAAAAAAAAABHTsvtwLlt2tsDW2RyLjGaU3IIxri1P
7gS/7m3GRfTl2w6PH3+DvMlaeQX9Hn0vLhKGKOauBce3WPToEBl4ALEABAAABAP4P/gEAADkAAAAAA
AAAAAAAAASyLtBYjrNQyrnpsRIW9qC2bMx0Y62jF9H5J7PXUyht9zu2b5WRRyST1tbZR599MZVRs6S
zGZLDQADaCzyDvU0JAQAAAQD+D/4BAAA6AAAAAAAAAAAAAAAEiq6eVVI64nQQTRYq2KtEg2d2uU7LE
lhTJwsH4YzIHZshxlgZms/wIc4VoDQTlG/IvVIrBKG06CrZnp0qv7hkcQEAAAEA/g/
+AQAAOwAAAAAAAAAAAAAABOx0l+EEG+ArKXIi6VRcMkXu/
Tt8bCGQwyxEdtZBEUO9aGj6HRfIy+9uQICTBQGG6KCKqJSaES7jPNUqXlJKZLwBAAABAP4P/gEAADw
AAAAAAAAAAAAAAAQecs4JHe+Nxjxt6g0u1yNnn+fGfZp+YwTqWGsOt5uiSoxqn5dt5byf1Nek8M6p0
YrmpwjehPQYpNbrALsQyJWoAQAAAQD+D/4BAAA9AAAAAAAAAAAAAAAEI88TYZWc9XiYHRQ4/3c5rjj
fgkjhLyW2luGIheGERbNQ6OY7yTybanSpDXZa8y7VUP9YmDcYa+eyq4ca7iLqWAEAAAEA/g/
+AQAAPgAAAAAAAAAAAAAABC9fA2ibF0lJNvuNqb/Ji7OYyU9oahZBROI9tcDpoG1KrGdoS+9jbFFO/
OYPUV4KN7NGTYFZeNk4h6d2bTr/1coBAAABAP4P/gEAAD8AAAAAAAAAAAAAAARs3v3yAV9Bf6+LDdH
vKsZZGqes3ahGQSRSOOXgk2fgTwbHFuO0bcVusQghjeXz+GvBTAh4Jm+LhC45M/gwStW6AQAAAQD+D
/4BAABAAAAAAAAAAAAAAAAENOKm8xhkzAjzFx8B2v5OAHT+u5pRQc2UCa2Vq9jYL/31o2wi9mxBA7L
IFs3sV5VSC49z6pEhfbMULvShKj26WAEAAAEA/g/+AQAAQQAAAAAAAAAAAAAABCMy/
GaBAyAUVhM5QnEYTmgrqWMjeYHSCvkOn2xXTw4Oh6l+o6ZCLZ+wxSKVvS0M1xug3/bAO/jip6tKpc/
xmkIBAAABAP4P/gEAAEIAAAAAAAAAAAAAAASC/dlFohJTd+M8CA2yuIFGoZZAvqq4XHThgw9b/
MPxcwuxTfaaEIJt9s7opkUuO9ikJnzPIMSCqyB/4/
A9ozsZAQAAAQD+D/4BAABDAAAAAAAAAAAAAAAEvNk6aEwybGtawWmy/

PzwnGDOjCkLWSD2wqvjGGAgOAwCGWySYXfYoxt00IJkTF+8Lb57DwOb3Aa0o9CApepiYQEAAAEA/g/
+AQAARAAAAAAAAAAAAAAAABG0SEoAiIz9tP7W1kj1jBIueEFT0WRMZLg/ZSS/
lCMVCrcFSQPMFtU629YzLNURV6NQgUzWf+YaQvUL5ilnaKSsBAAABAP4P/
gEAAEUAAAAAAAAAAAAAAAARHyAtFNl7KnTfKbM//ouKX/by0Z4YTOHHWraTvTcoZZEAjVV288hd0bvR
UlzakAzDc0DokovmGEW7WwlfQyef8AQAAAQD+D/4BAABGAAAAAAAAAAAAAAAE2ymg6oRBpebeZi9UU
NsgQ89bhx01TcTkmNTGnNO88imTmbSgy4nfujrgVEFKWpMTEGA11EDkTt7mqObTPdigIAEAAAEA/g/
+AQAARwAAAAAAAAAAAAAAABFlLu0YNQ66DO6ELxlnzogCttZ+
+BoPk+HpVxEGJDobci3loiRaDhi1zyiP/YMyInvQrcFS5vMLcOmCXSssUo3oBAAABAP4P/
gEAAEgAAAAAAAAAAAAAAAARhD4GbG5OB3pRdlbf4gIZ/
KpHIddWUPka1Cvn6qOI1btsXRyqvNfnTQdVc8E6+BdvlifMN36HTOrK/
rUpQPv5KAQAAAQD+D/4BAABJAAAAAAAAAAAAAAAAEdTpowEjclQ7Kgx5SdBkqRzVhERQXov8/
l9Ft9dVM9fmg0W0KQSVaXX9T4i6twCPNtYiZM53lpSSUAwJbFPOHxAEAAAEA/g/
+AQAASgAAAAAAAAAAAAAAABL2Le1A9VPVoOtd/LIS7Szr3QLvvA7Av4pRbRFR3B/
sMnXEqTRNtAH0jnbn+jJERWoS+RWO19aFO5ylWRbX6vBYBAAABAP4P/
gEAAEsAAAAAAAAAAAAAAAAQcNZcbyKwye0DJEXm9nvEq4S86FPgCGVG3/dzNNKO8ZTGNgIFVFBjjxWI
MfJ9jP1UigGRbhMwZA1qnH6XdFTB0AQAAAQD+D/4BAABMAAAAAAAAAAAAAAAEDOmrlGSXNk1DM0lji
QA+i+o0rSLhtii1je5wgk60j49d1jHT5YYttBv1iWOnYSTG+fZZESUOSNiAl89SIet+CgEAAAEAAAg
QAAAATQAAAAAAAAAAAAAAABMNYvi+5AF2TXWJfrxb80mcvGNWq0u7zICMvMLrFyc0az8qH+SwG9Nihq
9Tw9hJEXJe95B7pSCA9DOdNZTfr6G0BAAABAAAIEAAAAE4AAAAAAAAAAAAAAAARlcWTszvZHRhgU3mr
XeeJXQHsbz/l1xD0kOnyR4lJ+vcahl5cSgwc/8eW+pE4Dc4xsx1+vX0iMJ40Cc9Qb3dvoAQAAAQAAC
IAAAABPAAAAAAAAAAAAAAAAEfbMkAF7fufku0N2dE5TBXcNlg0pt0cJue4xBRE2Qc5Vqikxr4VCgKj/
ht6SMdFcOacVA9rqF70APJ8RN/4vMJwEAAAEAAAgEAAAAUAAAAAAAAAAAAAAAABF+v5zsKBTe0U9rLx
EFlLFv+Qv32DAXuH5yyvQ0JOEl+JoFQYkwPYeuJI1GD7xkNBf/ccCeVOVH025egRI2kHQUBAAABAAA
CEAAAAFEAAAAAAAAAAAAAAAASaMFqijihzEmmlOitlUpYWnLVt8WP0V4IwmrOUxAaOk09j0nBCLs65L
VrHM0iTScxfqNKDLplRENnbDRBboPnqAQAAAQCAABAAAABSAAAAAAAAAAAAAAAE2XFFPJ2XMEiF5Zi
2EBf4h73oR1V/lycirxZxHZNc93SqDN/IWhYYSYj8I9381ikUFXZrz2v7r2tOVk2NBwxrWwEAAAEAA
AAQAAAAUwAAAAAAAAAAAAAAABDNw4OIZcLp7VAnc1Oyo2Km23fgS6tyTADoQinQRN03DBjGWypA3SeY
rzXXoTrLDAiKJDRxTrL+wzuK5RDn8iqYBAAABAAgAgAAAAFQAAAAAAAAAAAAAAATGs1wYNTQ7FVw9M
f/yf9KEfhR69nEog7ockUrR7Xlci1k/ZRxqIsV3mSJfuEmeoJtLO1QXxCi34lYEL9BdE9hZAQAAAQA
CABAAAABVAAAAAAAAAAAAAAAAEit2BW6kKFVh8xk/
BnHfakEeoLPv8STIISekpoF+nBgWM4d55CZKc7T4Dx1pEbTnYm/
xEKMgy1MNtYuoA8RFIWwEAAAEAAgAQAAAAVgAAAAAAAAAAAAAAABNCl7yCUDDiJiwlQs/br73T4H8lq
Pd2hx6fLotxa7Tj+e7L6qxf66EkkeaY6i9oXDKdJY3hpVOKvaP60GgENwGABAAABAAIAAAEAAFcAAA
AAAAAAAAAAAARDocl3tjrF0HjzHxk1UM6qxuHa4HIGGArw+JCA+HCQjD8cL9hfkaGraZBjmxExlYdv
dWwWRaihRU5kcqvsOYMgAQAAAQAEABAAAABYAAAAAAAAAAAAAAAE9E6MKUJhDuDh604Qco5yP/3qn3
y7SLXYuiC0Rpr89aMScS2UAmK1wHP2b7KAa1nSjWJc/f/Lc0Wl1L47qjiyQwEAAAEABAAQAAAAWQAA
AAAAAAAAAAAABItMUy146mqdnkX7pRVZUpzOXRG01arnv9z4uINsu3Ma7O/mvQ3uNs+F8Si2Ce/joR
NyIMb30DO70HON28E4D8ABAAABAAQAgAAAAFoAAAAAAAAAAAAAAARVcGuc9aAGlUMJVjpcXcsRbSk5
TnUptoeFxwfDoBLq8FYgLrR6qwfX0MB545uX6DycoDPThAV2X8J5+1rTFC1RAQAAAQAEAAABAABbAA
AAAAAAAAAAAAEijLnS1qFId8xhKjT81uBHuuJp2lU4x2yxa4ctFPtG+MqEE6+C5f/+X/
bStmxapgmwLwiL3ih122xv8kVARNAZAEAAAEABAAEAAAAXAAAAAAAAAAAAAAAABGyL/l/
cXkrNVwpOg1b5Dvkv8/P9gDLf9pTKnUGzK5laYBb15ZnXvZi3G0Kc7V5tbWowt62hQ6noLB9fB0DzG
LQBAAABAP4PBAAAAF0AAAAAAAAAAAAAAAATnGgN9f58vt9oBOdqCZY+lsW3CH9HvtaYwyqocZLrkLe+
8HRgeuAX4HViZnfjjW0yPmfreTTbXZc2gnDOWF99DAgAAAQD+D/4BAABeAAAAAAAAAAAAAAAAEAYnb1

nQyY49te+VRAVgmzfcgjYS91mY5P0TKUDCLEM+gNnA+3T6rWITXRLYCpahpqSQbN5cE+gHpnPyXjHW
xcwEAAAEA/g/+AQAAXwAAAAAAAAAAAAAAABKmIPSj9uHQ+apGvSeO3dGlZMtDfm+H01PPSzfYg54wec
GpLIguPa7zAdD61CUBqE5h+dFz4qjrwIw32ooxsWGcBAAABAP4P/
gEAAGAAAAAAAAAAAAAAAASg+1Mzih6sv5RebH73fK1lU3zZGuVj/A9RXwh4PEWvMiNc4sXWk34SYo8
tu5u8odPYcLbTFewIAfZn4IpXo7P+AQAAAQD+D/4BAABhAAAAAAAAAAAAAAAEeop+wDAvpItUys0FW
kHIKeC9ybYrTGbU41U5K7+bttZZeohvnY7M9dZ5kB21GNWiFT2q1OoPTvncPCgSOVO5owEAAAEA/g/
+AQAAYgAAAAAAAAAAAAAABMXHh9rJ4bW+TPZYqg7JhMOepXt+
+pk2ZBF/4xG/0cTRcnoDbpe3jbJQlz/RQ4/y3LtF/ChMjHHj9p7aWh6wxFQBAAABAP4P/gEAAGMAAA
AAAAAAAAAAAATfjAH9jsxbtvOMpGNQpNrjojZnt5XrZGSG9r4qSilbtC+/85JYGq+RJju+6w4+s25l
xQbe0Cm91WA0Hw86PdI/AQAAAQD+D/4BAABkAAAAAAAAAAAAAAAE8uSpZZocAZRBAPIEINJj3Lo9Hy
Gitllczc27Eh5YYojjMFMn8yHMDMaUHE2Jqfq05D/wucwI4JGURyXt1vchygEAAAEA/g/+AQAAZQAA
AAAAAAAAAAAABAa6b3zQBN3XL6u5Zd8Vbps4yo2UObSNbBFCCq91KJLNF1JeOUrdxZWrVann/
aa5OI0Q84VulmYPt25Pd8uqS4wBAAABAP4P/gEAAGYAAAAAAAAAAAAAAAQHgUVnqr9PaOGGSyCRsRb
ccG9Yh8NbzmyeRCBrC3TtLsnlBdOToGQ1X7TIB5mszlCkwB1iWhwaiWOfSwn9ZCQXAQAAAQD+D/4BA
ABnAAAAAAAAAAAAAAAES48WzZW777zhNIrn7gxOlISNAqi9ZC/
uQFnRdbeIHhZhCA6UqpkOT8T1G7BvfdgP4Er8gF4sUbaS0i7QvIfCWwEAAAEA/g/
+AQAAaAAAAAAAAAAAAAAABGm7/6jnLj35N1ET3w85mVNSyprsPJE/
tJyB7yqyoBa8In6Jf3aFnHQOGarFkPBDaxSpHeuzH6aPy6L2yFLG7d8BAAABAP4P/
gEAAGkAAAAAAAAAAAAAAATCcPZkT6X5I8L+6hLS9d4T0vX7TC5oyoqV/P0AxSjfwmzItIFZIVwdHVG
uLrYtlzXa8uvWBveOXuLBCGDCkBsZAQAAAQD+D/4BAABqAAAAAAAAAAAAAAAEts6Wi+2j3jQjqi70w
5AlN8DFnkSwC+MqmxEzdEALB2qXZYV3X/b1CTfgPLGJNMeAWxdPfU8FO1ms3NUfaHCPYgEAAAEA/g/
+AQAAawAAAAAAAAAAAAAABIRaIiYk5et55/pLLRxgbXsFkip0C6cm9eeSh4XgNZd/
br7TvZ1iKKdad7naj3FHf8WxdVSzDuJ+ziOqe0W54A4BAAABAP4P/gEAAGwAAAAAAAAAAAAAAASRul
pJIYlNZ0Bjko9V4w4pdKs+2vwLwLvCh0ltyx3nWNGeYP4Zm7xjRWhToOblni9avQiD/U0q5ZEp/uPl
pphKAQAAAQD+D/4BAABtAAAAAAAAAAAAAAAEOqbOk5oEQeAZ8WXWydlu9HJjz9WVdEIvamMCcXmuqU
YjTknH/sqsWvhQ3vgwKFRR1HpjvNBKQ37nbJgYzGqGcgEAAAEA/g/
+AQAAbgAAAAAAAAAAAAAABFVTc//
UmtG7OUhC7Z93+6oSQgzwheaDBJeXDvf0kop6RPYWz1b8W3ee9hx+QCoj+1g/CDTudxoHww4UY5MpR
xICAAABAAAAEAAAAG8AAAAAAAAAAAAAAASesXPMxDtbhZEs1Q5Tl2VYCprrJV4jQ/
vxYoZSjuWDYdKGr/LH5RiR41ITRJJ9K6t7RKqCezE++YBS7+
+7QT7dAQAAAQAEACAAAABwAAAAAAAAAAAAAAAE3x37szhLexNA4bXhLrCC/LImN/YtWis6WXr1VESl
fVtVeoFJBRINPJ3f0a/6LV8zpikqoUg4hyXw0sFBt5Cr+QEAAAEAAAAQAAAAcQAAAAAAAAAAAAAAABO
QFeD86fNH4Ry/NyTOi6AOqU0FVoAdR6LzAbC0PTmViw6lWPDFaGwVzUpgCYv/wrIerXGO3uy58th4N
bRJTIyABAAABAAAAEAAAAHIAAAAAAAAAAAAAAARXHT0vMSgFrnMOeJ4fwtIxFOiPuKv8l6rQa7qon/
UZwGr0aR+gQChe8uJbVT0nWoT9j87JqrffDmnZlas00c0TAQAAAQAEAAgAAABzAAAAAAAAAAAAAAAE
h1k6yS8/pN/NHlMl5+v4XPfikhJulk4G+tKGFIOwURBSFzE8bixw1ebjluLOjfwtLqY0kewfjLSrO6
tN2MgIhAEAAAEABAAAAQAAdAAAAAAAAAAAAAAABMcOdIqcpF3C7LS3lh+1kXULsrCano7gBj3d9L3d
Yuc0siit1hPoo8x5iysXYgjLR9aR8+/KzoNASkmDkgf9iocBAAABAAQABAAAAHUAAAAAAAAAAAAAAA
RixLZ7da2CfDtarnddcEtxXomFGwSryGR+D3URJVp0ivOv3A7qMkMxmcJBsebUa5zm38J0OIVJJ9Y/
nwU269PdAQAAAQAEAIAAAAB2AAAAAAAAAAAAAAAE4O+gPR5rEBe2FpKOVyiJ7wNDPA8nGzDuJ6gN4o
kSA1gEOYZ67N8JPk58tkWtdtPeLz7lBnY6I5L3jdsr3S+A6AEAAAEAAAggAAAAdwAAAAAAAAAAAAAA
BGmNoDJipj4cX1PtBHZzflzbRm430IMGY+N5PBt38qwzP1HWcMOsa1sZg5Ls9ngmXIalXgAY1eQaYp
XTq1GwU6gBAAABAAACEAAAAHgAAAAAAAAAAAAAAAT3KCzve24UJ7p/
qb9Nv70u1jFZHhz3KDvOO3ZxiSMNsOlX/FWf7rxxDExcKNSljASTN7ZFxT+syd8hc1EWZKgPAQAAAQ

CAABAAAAB5AAAAAAAAAAAAAAAEdKN8fgVqd0vUIjxuJI6P/9SSSe/mB9rvA98CSH2sJnlZ/
OCZWO1DJvxj8jvKTfYUdGfcq2dDxoKaC6bHuTlgcwEAAAEABAAQAAAAegAAAAAAAAAAAAAAABP/
d239ssXp3GA5yLTGFf79des33+lMk46OBLREI1AXholaTJn3wzVMg/VDds1fldy0Vd/6J+MXWdujSs
5+u+eIBAAABAAAICAAAAHsAAAAAAAAAAAAAAASUrV4l7KTVlfxSSWoV3/cDrRS5n/
pRV8ZPYu4ajdanj/Pt5FNFhj7TE67aSkKU/68xOL3Ao59eLa+NSk7VGjvqAQAAAQAACAABAAB8AAAA
AAAAAAAAAAAEnlN9B69St9BwUoB+jkyU090bru8L0NA3yFvAd7k8dNsVH8bi9a8cUAUSEcEEgTp2z3
dbEDGJGfP6VUnkQnlRegEAAAEAAgAAAQAAfQAAAAAAAAAAAAAAABLs+v7RMtiBZvUWDsrypfPM3c2X8
BHymZp1/Qo9QR43khjbhi7jA9ZEfMuW7XmJsKiptbY/JuHG/2ihKK6+nx1kBAAABAAgAgAAAAH4AAA
AAAAAAAAAAAAQFxovRWfpDALCt7djvXudUdlsBrhQESKDMWbpV2xV1P8xtnfzGiPVf0c+RmFuZoEZQ
PVby/EB999sN7xsDoAqdAQAAAQAACAAAAAB/AAAAAAAAAAAAAAAEaevEkCNu7KlPRpYLjwmdcuNz6b
DFiE7Z8XC4CPqExjTvrHugh28QzUXVOZtiYghciKUacNktqxdpymplil1beAEAAAEACAAQAAAAgAAA
AAAAAAAAAAAABFUzeBLjh8FxJF81+Z071WOLOxPyq8+wjgvMZH7W2ykWAJferz4lLLBpHjBC0DDpyz
ifJNwWNfmWOdrbbWZx3F0BAAABAAAIBAAAAIEAAAAAAAAAAAAAAARoKWp2BMNU9IQaCgJLxAMbfsa7
zJz2VMc0BCWzO1DoQ160oQ7/9vzLfUwy7Zlo9ufiRWwgAHNUtdguEIZ6QYOrAQAAAQAACIAAAACCAA
AAAAAAAAAAAAAEBHpFFeslkWrXWyUPnbKm+xYYVYruCinGcftSBaa8zoF9hZO4BcSCFUvHVTtzpIY6
YzUnYtuEhZ+C9iEXjxnasgEAAAEAAAhAAAAAgwAAAAAAAAAAAAAAABEQUU9oKhJdCCkAAU+F04+/LaZ
Pb+3oafpms59h21HmZdWDHWhgEKoWqYxkwnu7sO8tH6w4uIEtLh84b6RbiNbABAAABAAAIEAAAAIQA
AAAAAAAAAAAAAAQRXr6i9OwRcI8CnnyDZ0PrwvGg//ArmlG+DLQn8fE8YHTT2iYU+/dh63QeJqq+si
uA9MG4LrAIfUtM5j3UQ7mFAQAAAQACABAAAACFAAAAAAAAAAAAAAAECXXkzgn+dXAPs3WBwE+Kvnrf
4WECwBdfjfeYHpMeVxWE0EceB6vhWGShs6wi0IYEqMSIzdOF1XjQ/
Mkm5d7ZdQEAAAEAgAACAAAAhgAAAAAAAAAAAAAAABOlYWJNXYkaq4QQKqSRHYwEkf/
llUbX46airjcNdXC4mlqQE/1Lvp6JAflCZ3r88vz/Qv7cugDntHcLnjKxof4kBAAABAP4P/gEAAIcA
AAAAAAAAAAAAAAQSvadz8ftGsxdkEUISKbpMKYyTTZny8sLZFuLUoQGCCmjR9LqXRLWedqnCYiLfJb
/4Y4lqnUquDjDQeDzSgKqBAQAAAQD+D/4BAACIAAAAAAAAAAAAAAAE6FlzubTLZG3J2a/
NVCAleEhjzq5oxgHyaCU9yYXvcLsvoVaHJq/s5xXI6/XXP6tz7R9xAOtHnSO/tXtF3WRTlAEAAAEA/
g/+AQAAiQAAAAAAAAAAAAAAABCOasacy7/
rCYU1RKC2DXE1vm04Qwj7IrdLyzSkoWceqy7aYpTaZpnkz2nqiW/WRPXVdwxmnm8ulboa8N/
Fo3skBAAABAP4P/gEAAIoAAAAAAAAAAAAAAAATtVPXd+aOi0qkaKkJb0kRAC6wQ8T4SLyeXr+DgUECY
ibQY44sy5r00MOj8NanRkDEKvdw+rlmkGqY8BCAN1rY/AQAAAQD+D/4BAACLAAAAAAAAAAAAAAAAE6e
Zs5Ls3WtCisHWp9S2GUy8dqkpGi4BVSz3GaqiE6ezub0512ESztXUwUB6C6IKbQkY2Pnb/
mD4WYojCRwcwLAEAAAEA/g/+AQAAjAAAAAAAAAAAAAAAABGTBEWHrOqQ8ncrhonbHuzrB8bWyO1lXlB
KM4Ef4O63dMVIpmDZb2URPytjIGU41su9uSH3pS3lXBDPe5prURl8BAAABAP4P/gEAAI0AAAAAAAAAAA
AAAAAAQJ6H7ujHmp7ssm4sehjR96Heke5QMcBxFR7IvZViCFnB+mQ0jL/7w5yDRrdS5KhjNq+bKXC4
tZA5/eGXSOMwwjAQAAAQD+D/4BAACOAAAAAAAAAAAAAAAAEAOIgSQCepiJYwP3ARnGx+5VnTu2HBYdz
bGP45eLw1vr3zB3vZLeyed1sC9hnbcOc9/SrMyM5RPQrkGz4aS9ZowEAAAEA/g/+AQAAjwAAAAAAAAAA
AAAAAABMXpUmshx9+mYBO2VoZYu6Vt+ITWzZfDo7+SlZpCQ+IQXQ97YfE35Pb2GrCzPpl1jmYRZIGX
8YS0p68Ea+HpUkoBAAABAP4P/gEAAJAAAAAAAAAAAAAAAAARJspsA2Q3rTdWLiMRm/j0t5UkyfjIbCx
vNnCisSjISK62w3eclh1s7frN+EYnpAQOk5kgWQO2erklHGa+XeOyhAQAAAQD+D/4BAACRAAAAAAAAAAA
AAAAAAAEhOS089on8RduqdbhvQ5Z37A0ESjsqz6qnRcffsMU3495FuTdqSm+7bhJ29JvIOsBDEEnan
5DPu9t3To9VRlMzAEAAAEA/g/
+AQAAkgAAAAAAAAAAAAAAABCVN7Xh0zY5hNlQhhc7mPBF8wg1cBKgdmvH7CL8GkrR4QFiRHlXdaNUA/
NAlOvmXRF10i20rLi8CY5AgVqkUFFQBAAABAP4P/gEAAJMAAAAAAAAAAAAAAAAQoxF4VSvQHi34P44G
SNHcpiq+hynZdpLM7nlRwHqaBAx3cptwT6ZZPK9VXsP/
Ox0Rs2dXppxlS62SIdBe9OvVHAQAAAQD+D/4BAACUAAAAAAAAAAAAAAAEyQbXgO/

OSZVD2IsiLlro+7Hf6Q18EJrKSEsdoMzKePKXct3gvD8oLcOQdIzGupr5Fj+EDe8gO/
lxc1BzfMpxvAEAAAEA/g/+AQAAlQAAAAAAAAAAAAAABGLBqXt1hyyvGWIuLVg4Nict3m0c9q16MA8Z
5XeG5EAdNHHP9WcPQFpwtIvc7QyYrYr7UL2iPSmi8iq3PoQVtMoBAAABAP4P/
gEAAJYAAAAAAAAAAAAAAATrk/uLPpfnISvVzByC9DFtsjDtSTeA7Ll0h21nisO94uqGt0k/4uL8fHq
3IrQ0Rv7YYLKd4IwmIaqsAMJI2TyxAQAAAQD+D/4BAACXAAAAAAAAAAAAAAAE41Cifkg6e8TylSpdt
TpeLVMqvSBEVzTttHvERD741+pnZ8ANv0004MRL43QKPDlK9cGvNp6NZWZUBlbGXYxxngEAAAEA/g/
+AQAAmAAAAAAAAAAAAAAABG+iJbrPnN0a3Q5PiYSylAEH884CxD8usHF6ku3/8XtRBE77Az+hjkRu0
hbjrOnCA+WJKnIVvy12W1+W3PI+2XEBAAABAP4P/
gEAAJkAAAAAAAAAAAAAAAQa7OJbd9JKailzHgpHKZAQlVtlR+qUKpyrt2+zpNjApFmJ6Ar/
Zu4zFBM+zowaU2/nTCazXr32pg7nhHxd9FfjAQAAAQD+D/4BAACaAAAAAAAAAAAAAAAEIXO6OCs9yg
8tMKzfPZ1YmheJbZCiEsnBdcB03l0OcfK9prKnJb96siuHCr5Fl37/
yo9DnKU+TLpxzTUspw9shgEAAAEA/g/+AQAAmwAAAAAAAAAAAAAABA9sDkqv3egILdvKu60iN44VOx
umors5YwxmdoU2RyJK0y1CPgBd1jHipBGfZzdn5AQJXJLU+r3GYwxYiITGBEMBAAABAP4P/
gEAAJwAAAAAAAAAAAAAAASEJlNK/BvNYa+GrGn9GpWrBT332kF/a+nuHMiOJg6yXzCo6ZTGRDp13DV
xXeXzzXX6R1QW6z+rh3VjcTdLISJ8AQAAAQD+D/4BAACdAAAAAAAAAAAAAAAEM1uQkMl8rQK/
szD0LNhtqxIPLpimGm8sOBwU7lLnCpSbTyY3yeU1Vc7l4KT5zT4s/
yOxHH5O7tIuuLOCnLADRwEAAAEA/g/+AQAAngAAAAAAAAAAAAAABI6030QuxTOpwob1O19iqnK4Krq
H61ixQ3mAtYSn8DVqJlb65+ySJLKn9+OFjdl/8vMa4KgXdnSQE2VeBm2rQX4BAAABAP4P/
gEAAJ8AAAAAAAAAAAAAAAS97Yo/p/8mds8EXKhsYe56sL2DUVgaf8XyfUtZPA3EITN3oh+pPBRBw1e
AS2aZDoOVGsLWGtKsGcJk+iRGsMePAQAAAQD+D/4BAACgAAAAAAAAAAAAAAAE1rkd7G70+o9KkTn5K
LmDYXihGoTaIGO9PIIN2ZB7UJxFrWw04CZHPYiMRjYsaDvVV7hP1dYNRLxSANLaBFGpogIAAAEABAA
QAAAAoQAAAAAAAAAAAAAABL+uuQrfVgkYxmvPpE69AswKVNPl17UYMJsyH9UnikhgL/
trHAJVuiUeTzBaOhc6qmeuD6O/XCjgyn9/wAJhAqgBAAABAAAIEAAAAKIAAAAAAAAAAAAAAASASGL+
NIGFnabrCNXnfRjLz582obw8CKczehXsLI77pzcISHXoSLObbW4dWzJUEQHpEgzwkciXoAZgK/
pDpD1OAQAAAQAEABAAAACjAAAAAAAAAAAAAAAEfJIW0+LYujdjUgJJuwesP4EjIBl/N/
TcOX3IvIHJQNsAqvV2CHIogsmA94BPG6jZATS4Hi+xv4SkBBQSt1N4/gEAAAEAAAgQAAAApAAAAAAA
AAAAAAAABJUQib/1Yt1ZcJm3eivqENCoSxzAMrDfJV6AiO13h5cWmo5NOruxvlLBUjtNTQLOYmLlip
lWY5ojsydDiZhrDMIBAAABAAQABAAAAKUAAAAAAAAAAAAAAARsUGw9as2+R0qs0Jh4Cn7GjYd5w4Y8
DL1bI3IH+Mqed52DOXhbgdv1kzNg1t/
x0wIaZR1Jq+VgeJHTVBphHnxzAQAAAQAACAQAAACmAAAAAAAAAAAAAAAEngYBh/
+bEedqkSevPVhLP4QfVPCpb+4BBe2p7Xs32dBgs7rh9nY2AIYUL6BgLw1JVXV8GlpKmb/hNiuIxfPf
ZgEAAAEABACAAAAApwAAAAAAAAAAAAAABLZW4S5Pll2ZoMKCCsByBxAWVMMZpe20SYJNF9rD8jUd0v
d3HUaul7pQGUUCdexWBhs1p3tH4K1wSqZe5eDPOx4BAAABAAAIgAAAAKgAAAAAAAAAAAAAAATxjnAy
FUg9pZ7qPDYf4qUSbjA/9/sRbfzXRkuYOWsQTLkq8Gxv0QIR+7+QxYPjziKN9Dz6ciLSCIyC72DVQC
64AQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAACWAAAADAAAgGNs
aS5qcwAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAgAAAGh1bWFuLWlkzgEAAAsAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAA
BAAAAAAAAAAAAAAABAAAAAQAAAAEAAAAAAAAAAAAAACAAAA1gQAAAkAAIBjbGkuanMAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAI
AAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
IAAABnbG9iAAAAAKwIAAAQAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAbQoAAAoAAIB3CgAAEA
AAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAbQoAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAACAAAAcm9sbHVwAACyFgAADwAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIAAABlc2J1aWxkACgjAAALAA
CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAYmlvbWUAAABh
MgAACQAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAyKcAAAAAAAAC4qAAAAAAAAAAo8c3JjLmluc3RhbGwubG9ja2
ZpbGUuVHJlZT4gMjAgc2l6ZW9mLCA0IGFsaWdub2YKAAAAAAD+/////////wAAAACgAAAAAQAAAA4A
AAAAAAAAoAAAAAEAAAACAAAApgAAAAAAAAChAAAAAQAAAAMAAACnAAAAAAAAAAKIAAAABAAAABAAAAA
8AAAABAAAAowAAAAIAAAAFAAAAOgAAAAAAAAAClAAAAAQAAAAYAAABKAAAAAAAAAAKYAAAACAAAABwAA
AEwAAAAAAAAAAqAAAAAEAAAAIAAAATgAAAAAAAAACpAAAAAwAAAAkAAABTAAAABgAAAKwAAAABAAAACg
AAAFAAAAAIAAAArQAAAAEAAAALAAAAUQAAAAgAAACuAAAAAQAAAOioAAAAAAAAApKsAAAAAAAAAKPHUz
Mj4gNCBzaXplb2YsIDQgYWxpZ25vZgoAAAAAAAAAAAAABAAAAAgAAAAMAAAAEAAAABQAAAAYAAAAHAA
AACAAAAAkAAAAKAAAAugAAALsAAAC8AAAAvQAAAL4AAAC/AAAAwAAAAMEAAAC5AAAAtwAAALgAAAAT
AAAAFAAAABUAAAAWAAAAFwAAABgAAAAZAAAAGgAAABsAAAAcAAAAHQAAAB4AAAAfAAAAIAAAACEAAA
AiAAAADgAAAA0AAAALAAAAtQAAAKQAAAClAAAApgAAAKcAAACoAAAAqQAAAKoAAACrAAAAowAAAIsA
AACMAAAAjQAAAI4AAACPAAAAkAAAAJEAAACSAAAAkwAAAJQAAACVAAAAlgAAAJcAAACYAAAAmQAAAJ
oAAACbAAAAnAAAAJ0AAACeAAAAnwAAAKAAAAChAAAAogAAAIAAAACBAAAAggAAAIMAAACEAAAAhQAA
AIYAAACHAAAAiAAAAHsAAABpAAAAawAAAGwAAABtAAAAbgAAAG8AAABwAAAAcQAAAHIAAABzAAAAdA
AAAHUAAAB2AAAAdwAAAHgAAAB5AAAAegAAAGQAAAAqAAAAKwAAACwAAAAtAAAALgAAAC8AAAAwAAAA
JwAAACgAAAAMAAAAsgAAAK4AAACsAAAAigAAAD4AAAA/AAAAQAAAAGUAAABmAAAAZwAAAF8AAABgAA
AAYQAAADYAAAA3AAAAOAAAADkAAAA6AAAAOwAAADEAAAAyAAAAMwAAALMAAABCAAAAQQAAAGgAAABi
AAAARwAAAEgAAABFAAAARAAAADQAAAC0AAAASQAAAEoAAABLAAAATAAAAE0AAABOAAAARgAAAF0AAA
BeAAAAVQAAAFsAAABYAAAAVgAAAFcAAAAPAAAAsAAAAK8AAAARAAAAEgAAAD0AAABTAAAAVAAAAFIA
AABPAAAAUAAAAFEAAABSAAAAVAAAAFIAAADQqwAAAAAAAANiuAAAAAAAAACjx1MzI+IDQgc2l6ZW9mLC
A0IGFsaWdub2YKAKAAAACeAAAAmwAAAA8AAAAOAAAADQAAAAgAAAAHAAAABgAAAAQAAAABAAAAAgAA
AAMAAAAFAAAACQAAAAoAAAD/////DAAAAAsAAACZAAAAmAAAAIoAAACJAAAAhwAAAG4AAABkAAAAYw
AAAGIAAABgAAAAXwAAAEwAAABGAAAAFAAAABEAAAAQAAAA/////////////////////xMAAAASAAAA
/////0EAAABAAAAAHQAAABwAAAAXAAAAFgAAABUAAAAaAAAAGwAAABgAAAAZAAAAGgAAACkAAAAtAA
AAKgAAACcAAAAfAAAAHgAAACEAAAAgAAAAJgAAACQAAAAiAAAAIwAAACUAAAApAAAAKAAAACsAAAAs
AAAALwAAAC4AAAA8AAAAMwAAADoAAAAxAAAAOQAAADAAAAA2AAAAMwAAADEAAAAyAAAAMQAAADUAAA
A0AAAANwAAADgAAAA/AAAAPAAAADoAAAA7AAAAOgAAAD4AAAA9AAAARQAAAEMAAABCAAAARAAAAEIA
AABHAAAASwAAAEkAAABIAAAASgAAAF4AAABdAAAAXAAAAFsAAABaAAAAWQAAAFgAAABXAAAAVgAAAF
UAAABUAAAAUwAAAFIAAABRAAAAUAAAAE8AAABOAAAATQAAAGEAAAD/////////////////////bAAA
AGsAAABqAAAAIQAAAGkAAABoAAAAZwAAAGYAAABlAAAAJgAAAG0AAACGAAAAhQAAAIQAAACDAAAAgg
AAAIEAAACAAAAAfwAAAH4AAAB9AAAAfAAAAHsAAAB6AAAAeQAAAHgAAAB3AAAAdgAAAHUAAAB0AAAA
cwAAAHIAAABxAAAAcAAAAG8AAACIAAAAlAAAAI8AAACTAAAAkQAAAI4AAACMAAAAiwAAAF0AAACNAA

AAjwAAAJAAAACSAAAAkgAAAIsAAACVAAAAlgAAAJcAAACaAAAAbgAAAJ0AAACcAAAAnwAAAKgAAACn
AAAApgAAAKUAAACkAAAAowAAAKIAAAChAAAACK8AAAAAAAC8wgAAAAAAAAo8WzI2XXU4PiAyNiBzaX
plb2YsIDEgYWxpZ25vZgoAFQAAAA4AAICinGyoccMmrggBXjEuOS40AAAjAAAACwAAgJEvWo0mtT05
CAFeMjIuNy45AC4AAAAMAACAGhUlEYEjU8wIAV4xOC4zLjEydHN1cAAAAACTysqKrAkm+AgBXjguMC
4yAAA6AAAACgAAgJ8lgCwrhWViCAFeNS42LjMAAEQAAAAJAACAV1KcAP7v+soCAV4xLjUuMQAATQAA
AAsAAICBOwfYujafTgIBXjQuMC4wAABlbWl0dGVyeajBoNZXVjlaAgFeMS4wLjMAAGh1bWFuLWlkMx
omrvasgOkCAV40LjEuMQAAWAAAAAsAAIBphiYTUlatUAIBXjEuMC4xAAByZWFjdAAAAJDn2N4ZkMSU
AgFeMTguMy4xAJYAAAAMAACAAtdxgJdBMisCAV4xLjEuMAAA8gAAAAkAAIDoqIoP1tYGcAIB4gAAAB
AAAIBzAQAAEQAAgBDTKB9tJFcSAgFeNi4wLjIAAIcCAAAKAACAkPaloLodA4ICAV4yLjYuMTIAzQIA
AAoAAIARGcWXGNA41gIBXjUuMC4wAABlbmNvZGluZ4SNlVTghZG5FAFeMC4xLjAAAHRyNDYAAAAAOm
bh2Jm1hEMCAX4wLjAuMwAAEwMAABIAAICkKt6LdUkpSgIBXjMuMC4wAAB/
BAAADgAAgB3Xg1uY38iIAgFeNS4wLjAAAGNhYwAAAAAA7rwG/
nXqDz4CAV42LjcuMTQAY2hva2lkYXJnwIpdGz8vugIBXjMuNi4wAABjb25zb2xhANw/fZ35hcE2AgF
eMy4yLjMAAGRlYnVnAAAAjoMANOEaysYCAV40LjMuNQAAZXNidWlsZAAQIz1DdwfztwIBXjAuMjMuM
ABleGVjYQAAAH1ZWR1g9JMcAgFeNS4xLjEAAGpveWNvbgAA8wt2hCAU+ZICAV4zLjEuMQAAjQQAAAo
AAIBziUj+tKUkfAIBXjEuMC4xAACXBAAAEwAAgE/5w7W8BGKaAgFeNi4wLjEAAKoEAAAMAACA7Phi3
V8TCDQCAV41LjAuMAAAcm9sbHVwAABQbNtID6VBkQIBXjQuMTkuMADCBAAACgAAgOTsuhrAIP3RAgG
2BAAADAAAgHN1Y3Jhc2UAtsVwC5tNFXICAV4zLjM1LjAAzAQAAAoAAIAzdLsU0RYn+QIBXjAuMi4xA
ADWBAAACQAAgCqarpCkemx7AgFeMS4yLjIAAN8EAAAYAACAzCgb4e1KjaoUAV43LjM2LjAA9wQAAAk
AAICFXMhggUwK6hQBXjEAAAAAAABwb3N0Y3NzAFxhzsrsTAiBFAFeOC40LjEyADoAAAAKAACAnyWAL
CuFZWIUAT49NC41LjAAZmRpcgAAAADNRzjjsNHjTAIBXjYuNC4wAAB2BQAACQAAgFVg7P/6kGH2AgF
eNC4wLjIAAHYFAAAJAACAVWDs//qQYfYUAV4zIHx8IF40RwYAABcAAICbFbfqGoW8awIBXjAuMy4yA
ABeBgAACQAAgNYOOW4SFUtJAgFeNC4wLjAAAGdsb2IAAAAAZleN7hpZTkUCAV4xMC4zLjEwZwYAABE
AAIB62x8QzGMteQIBXjEuMS42AABtegAAAAAANJS1IXxWHVDAgFeMi43LjAAAHBpcmF0ZXMAf+7re
TxKFp8CAV40LjAuMQAAeAYAABQAAIDqUWN7UIsofgIBXjAuMS45AAA/BwAACwAAgKX2eGRTDsYNAgF
eMS4wLjAAAEoHAAANAACAUonWrh5YSgoCAV40LjAuMQAAVwcAAAsAAIAF92Tk6uOJMAIBXjEuMC4wA
AB0aGVuaWZ5APLzrOmoX8GNAgGgBwAADAAAgD8HAAALAACApfZ4ZFMOxg0CAV4xLjAuMAAAbWluaXB
hc3NzBpB9Y5g6eAIBXjcuMS4yAADtCAAACQAAgMAaJQa7X2DgAgFeMy4xLjIAAPYIAAAJAACAHwLiX
lHu7nACAV45LjAuNAAA/wgAAAsAAICFEFxsbIQBnAIBXjEuMTEuMQAKCQAAEAAAgBkSEmxUIw/
UAgFeMy4xLjAAABoJAAAWAACAqgzW/
v9D1TQCAV4xLjAuMAAAzAkAAAsAAIBQfmy83LGzrAIBXjcuMC4wAADXCQAACwAAgL5SacMC/
tOQAgFeNC4wLjEAAHBhdGgta2V50nO//
tVSPAkCAV4zLjEuMAAAXgoAAA8AAIBAFd+YOr8IfQIBXjIuMC4wAAB3aGljaAAAAMxtjvejT/JGAgF
eMi4wLjEAAGlzZXhlAAAAqCxX2lMS6tcCAV4yLjAuMAAAMQsAAA0AAIAK6VcdP6CM+gIBXjMuMC4wA
ABtaW5pcGFzc3MGkH1jmDp4AgH3CwAAGgAAgBEMAAAJAACA8MR9cJzXzNMCAV4xMC4yLjAAxwwAAA8
AAIBZv+4FnN7tKAIBXjIuMC4xAAAcDQAADgAAgFdR1E1d8snhAgFeMS4wLjAAAKgNAAANAACAnqqS6
xYV0oUCAV44LjAuMgAAtQ0AABAAAICoef5vM6AbQQQBXjAuMTEuMABBDgAADAAAgP1sAbBwIkTaAgF
eNS4xLjIAAGQOAAAQAACA+Pp7aMJYlKUCAU0OAAAXAACAdA4AAAoAAIAdIuEjwAkiuQIBXjcuMC4xA
ACTDgAADgAAgGRZw8KeD6AFAgF+DgAAFQAAgKEOAAAJAACAVdPsJHpvEFkCAV44LjEuMAAAvg4AAA0
AAIAxCvHrKkYUBAIBqg4AABQAAIAFDwAACwAAgE4vwqIZysMUAgFeNC4wLjAAAEEOAAAMAACA/WwBs
HAiRNoCAV40LjEuMAAAdA4AAAoAAIAdIuEjwAkiuQIBXjYuMC4wAABMDwAACgAAgFoBnNtG959wAgF
eNS4wLjEAAHQOAAAKAACAHSLhI8AJIrkCAV42LjAuMQAA0g8AAAsAAIDsZFpV3TzARAIBXjguMC4wA
ADdDwAAFwAAgKxvMbRs7mOeAgFeMy4wLjAAAMYQAAANAACAYnEkPUALDzgCAV4yLjAuMQAAFREAAAo
AAIDRWKrQSfSf5QIBfjEuMS40AAAFDwAACwAAgE4vwqIZysMUAgFeNi4xLjAAAEEOAAAMAACA/WwBs

HAiRNoCAV41LjAuMQAAdA4AAAoAAIAdIuEjwAkiuQIBXjcuMC4xAABMDwAACgAAgFoBnNtG959wAgF
eNi4wLjEAAHQOAAAKAACAHSLhI8AJIrkCAV43LjAuMQAA0g8AAAsAAIDsZFpV3TzARAIBXjkuMi4yA
ABNEgAADgAAgFnGNlLqSvnsAgFeMC4yLjAAAJ8TAAAVAACA4f7Nrig0DZgCAV4xLjIuMQAAtBMAABk
AAIB58bXY2z+8fgIBXjAuMy4yNADNEwAAGwAAgPkRZWv+NM2ZAgFeMS40LjEwAIkUAAAXAACAYEljC
yKJVdoCAV4zLjEuMAAAzRMAABsAAID5EWVr/jTNmQIBXjEuNC4xNADNAgAACgAAgBEZxZcY0DjWAgF
eNy4wLjAAAK8VAAANAACAzq8iO5lUscECAV40LjcuMAAAdHI0NgAAAAA6ZuHYmbWEQwIBXjEuMC4xA
AATAwAAEgAAgKQq3ot1SSlKAgFeNC4wLjIAAHB1bnljb2RlKT5iiTcOBV0CAV4yLjEuMAAA9hYAAA0
AAIA+XgK81O8I3wIBMS4wLjYAAABmc2V2ZW50c7iQgheF7ojeBAF+Mi4zLjIAAAMXAAAbAACAuxQWE
l2PmYIEATQuMjQuMAAAHhcAABwAAIC1/+zyChrKJQQBNC4yNC4wAAA6FwAAHwAAgFGa7u5AoNH+BAE
0LjI0LjAAAFkXAAAeAACABnTSZGzO8wIEATQuMjQuMAAAdxcAAB8AAICzJQRdMOiFEwQBNC4yNC4wA
ACWFwAAHwAAgDjwe08OEIfRBAE0LjI0LjAAALUXAAAiAACAi4/9nLcjEeoEATQuMjQuMAAA1xcAACM
AAID4Xpl+NJzK/wQBNC4yNC4wAAD6FwAAHgAAgOZBJQj1Q0TEBAE0LjI0LjAAABgYAAAgAACANyetO
DL70EYEATQuMjQuMAAAOBgAACQAAICSgRxZfnb1KAQBNC4yNC4wAABcGAAAHgAAgFTev6GH9IEaBAE
0LjI0LjAAAHoYAAAZAACAPwdRZnKIyWQEATQuMjQuMAAAkxgAAB0AAIAdek8B5XzVgAQBNC4yNC4wA
ACwGAAAHAAAgA/i0TEs/Mv8BAE0LjI0LjAAAMwYAAAdAACAiHKbJ92ISQ0EATQuMjQuMAAAxh8AAAk
AAIB4HGnQ4Yhe8gIBXjMuMS4xAABqaXRpAAAAAImc1lcEVgzAFAE+PTEuMjEuMHBvc3Rjc3MAXGHOy
uxMCIEUAT49OC4wLjkAdHN4AAAAAAAAMSRSYjuoDvhQBXjQuOC4xAAB5YW1sAAAAADT6DL1XlbFeFAF
eMi40LjIAAG9uZXRpbWUAVzqrX40RAZcCAV41LjEuMgAAqyAAAAkAAIDWTkan64/
hNAIBXjIuMC4wAAC0IAAACgAAgCjx3u9zF/C3AgFeNi4wLjAAAMwJAAALAACAUH5svNyxs6wCAV43L
jAuMwAA1wkAAAsAAIC+UmnDAv7TkAIBXjMuMC4zAAC+IAAADAAAgMsI6SZKTsrMAgFeMi4wLjAAAMo
gAAAMAACAdlcyHY18TsoCAV40LjAuMQAA1iAAAA0AAIClIp7sBSBgQQIBXjIuMS4wAADjIAAAEwAAg
HBVz1jnV3J/AgFeMi4wLjAAAHBhdGgta2V50nO//tVSPAkCAV4zLjAuMAAAbWltaWMtZm6vXUQjC5e
c6gIBXjIuMS4wAABqIwAAEgAAgN7ev52rMr05BAEwLjIzLjEAAHwjAAASAACAabk8qDQGbeYEATAuM
jMuMQAAjiMAABIAAIDnlnTyoT/2uwQBMC4yMy4xAACgIwAAEgAAgMHlkagTF0f3BAEwLjIzLjEAALI
jAAASAACANkzMtqGStHoEATAuMjMuMQAAxCMAABMAAIAwQppftwYPAgQBMC4yMy4xAADXIwAAEwAAg
KKpfmruCCXSBAEwLjIzLjEAAOojAAATAACAiASsaF8ROngEATAuMjMuMQAA/SMAABMAAIAIET4ruAG
UzAQBMC4yMy4xAAAQJAAAFAAAgMgHUJdG6G3CBAEwLjIzLjEAACQkAAAUAACAGPr6MVLKNmUEATAuM
jMuMQAAOCQAABQAAIDCAO5HJukCQAQBMC4yMy4xAABMJAAAFAAAgNuDZkYxltrBBAEwLjIzLjEAAGA
kAAAUAACAKhIoMwGL46EEATAuMjMuMQAAdCQAABQAAIBiFvwJDx8U9gQBMC4yMy4xAACIJAAAFAAAg
JM77CNqjZqIBAEwLjIzLjEAAJwkAAAUAACAyPaAH3ykVZMEATAuMjMuMQAAsCQAABUAAICeGzC60+l
jWgQBMC4yMy4xAADFJAAAFgAAgFI+0CscHRGnBAEwLjIzLjEAANskAAAWAACADRdE6Z6udscEATAuM
jMuMQAA8SQAABYAAICrRmldcj8nnAQBMC4yMy4xAAAHJQAAFgAAgGxgov/WkblBBAEwLjIzLjEAAB0
lAAAWAACAMhF9VuG6UmIEATAuMjMuMQAAMyUAABcAAIDRoRAdxCjpTwQBMC4yMy4xAABtcwAAAAAAA
KS0xNW1149IAgFeMi4xLjMAAGJyYWNlcwAAwBTYBrrD5x8CAX4zLjAuMgAAaXMtZ2xvYgB3hGbmLk9
1NQIBfjQuMC4xAABhbnltYXRjaPh1+YRV65K1AgF+My4xLjIAAHJlYWRkaXJwIh+XGR0iUosCAX4zL
jYuMAAA1iwAAAsAAICehrtDqP0OpgIBfjUuMS4yAADhLAAADgAAgEQPsKd0jdCKAgF+Mi4xLjAAAO8
sAAAOAACAfAZdmz2MpW4CAX4zLjAuMAAAZnNldmVudHO4kIIXhe6I3gQBfjIuMy4yAACFLQAAEQAAg
GSMtlZp3J7uAgFeMi4wLjAAAGlzLWdsb2IAd4Rm5i5PdTUCAV40LjAuMQAAVC4AAAoAAIAKneHahAf
jMwIBXjIuMS4xAAB2BQAACQAAgFVg7P/6kGH2AgFeMi4yLjEAAHYFAAAJAACAVWDs//qQYfYCAV4yL
jAuNAAA7ywAAA4AAIB8Bl2bPYylbgIBXjMuMC4wAAB4LwAACgAAgD3zwLHKEDVoAgFeNy4xLjEAAL4
vAAAOAACABFgyVM0GqsUCAV41LjAuMQAAEDAAAAkAAIAqL0j8+FsU8gIBXjcuMC4wAADGMAAADQAAg
C4FA5pUphYmAgFeMC4yLjMAAGVzYnVpbGQAECM9Q3cH87cQAT49MC4xOAAAY3NzdHlwZQA59nQMC3m
TkQIBXjMuMC4yAABQMQAAEQAAgL2M9EYU73nGAgEqAAAAAAAAABQyAAAMAACACW88xupRmrsCAX42L
jE5LjIApTIAABYAAICs2hY1wfZO0QQBMS45LjQAAAC7MgAAGAAAgNE2s0K/

```
zvJeBAExLjkuNAAAANMyAAAXAACAD/OXlh8+z6AEATEuOS40AAAA6jIAABkAAIC9rEg/HFc4nwQBMS
45LjQAAAADMwAAFgAAgOX5M2vkIwMCBAExLjkuNAAAABkzAAAYAACAPy2pCGraHu8EATEuOS40AAAA
MTMAABsAAIC6mmUYR5RqwwQBMS45LjQAAABMMwAAHQAAgFWPQIKuSuIjBAExLjkuNAAAAAjDAAAAAA
AAyMMAAAAAAAAAKPHNyYy5pbnN0YWxsLnNlbXZlci5FeHRlcm5hbFN0cmluZz4gMTYgc2l6ZW9mLCA4
IGFsaWdub2YKAAB0c2MAAAAAAMD7FKVylJAUYmluL3RzYwAP3qM87h1xunRzc2VydmVyXdMggVJmS/
jbAwAADAAAgIQ4NcwBeA7+dHN1cAAAAACTysqKrAkm+CMEAAATAACAyvOXPX7Bee82BAAACQAAgPc3
kKYpphkzPwQAABAAAIDGbXQ3di+D2nN1Y3Jhc2UAtsVwC5tNFXLpBQAACwAAgNtVe6RXqyAZ9AUAAA
wAAICBF1UO/xQ03gAGAAAQAACAxTlfUz92Otf4wwAAAAAAAOH5AAAAAAAACjx1OD4gMSBzaXplb2Ys
IDEgYWxpZ25vZgoAAAAAAABAY2FydGVzaWEvY2FydGVzaWEtanNAYmlvbWVqcy9iaW9tZUB0eXBlcy
9ub2RlQHR5cGVzL3JlYWN0dHlwZXNjcmlwdGJhc2U2NC1qc2Nyb3NzLWZldGNocGFydHlzb2NrZXRo
dHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9yZWFjdC8tL3JlYWN0LTE4LjMuMS50Z3psb29zZS1lbn
ZpZnlodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9sb29zZS1lbnZpZnkvLS9sb29zZS1lbnZpZnkt
MS40LjAudGd6XjMuMC4wIHx8IF40LjAuMGpzLXRva2Vuc2h0dHBzOi8vcmVnaXN0cnkubnBtanMub3
JnL2pzLXRva2Vucy8tL2pzLXRva2Vucy00LjAuMC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9y
Zy9wYXJ0eXNvY2tldC8tL3BhcnR5c29ja2V0LTEuMC4yLnRnemV2ZW50LXRhcmdldC1zaGltaHR0cH
M6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvZXZlbnQtdGFyZ2V0LXNoaW0vLS9ldmVudC10YXJnZXQtc2hp
bS02LjAuMi50Z3pkaXN0L2NsaS5qc2h0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2h1bWFuLWlkLy
0vaHVtYW4taWQtNC4xLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvZW1pdHRlcnkvLS9l
bWl0dGVyeS0xLjAuMy50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9jcm9zcy1mZXRjaC8tL2
Nyb3NzLWZldGNoLTQuMC4wLnRnem5vZGUtZmV0Y2hodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9u
b2RlLWZldGNoLy0vbm9kZS1mZXRjaC0yLjcuMC50Z3p3aGF0d2ctdXJsaHR0cHM6Ly9yZWdpc3RyeS
5ucG1qcy5vcmcvd2hhdHdnLXVybC8tL3doYXR3Zy11cmwtNS4wLjAudGd6d2ViaWRsLWNvbnZlcnNp
b25zaHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvd2ViaWRsLWNvbnZlcnNpb25zLy0vd2ViaWRsLW
NvbnZlcnNpb25zLTMuMC4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3RyNDYvLS90cjQ2
LTAuMC4zLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2Jhc2U2NC1qcy8tL2Jhc2U2NC1qcy
0xLjUuMS50Z3piaW4vdHNzZXJ2ZXJodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy90eXBlc2NyaXB0
Ly0vdHlwZXNjcmlwdC01LjYuMy50Z3pkaXN0L2NsaS1kZWZhdWx0LmpzdHN1cC1ub2RlZGlzdC9jbG
ktbm9kZS5qc2h0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3RzdXAvLS90c3VwLTguMy4wLnRnemJ1
bmRsZS1yZXF1aXJlcGljb2NvbG9yc3Bvc3Rjc3MtbG9hZC1jb25maWdyZXNvbHZlLWZyb20wLjguMC
1iZXRhLjBzb3VyY2UtbWFwdGluaWdsb2JieXRyZWUta2lsbEBtaWNyb3NvZnQvYXBpLWV4dHJhY3Rv
ckBzd2MvY29yZWh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3RyZWUta2lsbC8tL3RyZWUta2lsbC
0xLjIuMi50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy90aW55Z2xvYmJ5Ly0vdGlueWdsb2Ji
eS0wLjIuOS50Z3pwaWNvbWF0Y2hodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9waWNvbWF0Y2gvLS
9waWNvbWF0Y2gtNC4wLjIudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvZmRpci8tL2ZkaXIt
Ni40LjIudGd6YmluL3N1Y3Jhc2VzdWNyYXNlLW5vZGViaW4vc3VjcmFzZS1ub2RlaHR0cHM6Ly9yZW
dpc3RyeS5ucG1qcy5vcmcvc3VjcmFzZS8tL3N1Y3Jhc2UtMy4zNS4wLnRnekBqcmlkZ2V3ZWxsL2dl
bi1tYXBwaW5nY29tbWFuZGVybGluZXMtYW5kLWNvbHVtbnN0cy1pbnRlcmZhY2UtY2hlY2tlcmh0dH
BzOi8vcmVnaXN0cnkubnBtanMub3JnL3RzLWludGVyZmFjZS1jaGVja2VyLy0vdHMtaW50ZXJmYWNl
LWNoZWNrZXItMC4xLjEzLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3BpcmF0ZXMvLS9waX
JhdGVzLTQuMC42LnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL216Ly0vbXotMi43LjAudGd6
YW55LXByb21pc2VvYmplY3QtYXNzaWdudGhlbmlmeS1hbGxodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm
9yZy90aGVuaWZ5LWFsbC8tL3RoZW5pZnktYWxsLTEuNi4wLnRnej49IDMuMS4wIDwgNGh0dHBzOi8v
cmVnaXN0cnkubnBtanMub3JnL3RoZW5pZnkvLS90aGVuaWZ5LTMuMy4xLnRnemh0dHBzOi8vcmVnaX
N0cnkubnBtanMub3JnL2FueS1wcm9taXNlLy0vYW55LXByb21pc2UtMS4zLjAudGd6aHR0cHM6Ly9y
```

ZWdpc3RyeS5ucG1qcy5vcmcvb2JqZWN0LWFzc2lnbi8tL29iamVjdC1hc3NpZ24tNC4xLjEudGd6aH
R0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvbGluZXMtYW5kLWNvbHVtbnMvLS9saW5lcy1hbmQtY29s
dW1ucy0xLjIuNC50Z3pkaXN0L2VzbS9iaW4ubWpzaHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvZ2
xvYi8tL2dsb2ItMTAuNC41LnRnemphY2tzcGVha21pbmltYXRjaHBhdGgtc2N1cnJ5Zm9yZWdyb3Vu
ZC1jaGlsZHBhY2thZ2UtanNvbi1mcm9tLWRpc3RodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9wYW
NrYWdlLWpzb24tZnJvbS1kaXN0Ly0vcGFja2FnZS1qc29uLWZyb20tZGlzdC0xLjAuMS50Z3podHRw
czovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9mb3JlZ3JvdW5kLWNoaWxkLy0vZm9yZWdyb3VuZC1jaGlsZC
0zLjMuMC50Z3pjcm9zcy1zcGF3bnNpZ25hbC1leGl0aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcv
c2lnbmFsLWV4aXQvLS9zaWduYWwtZXhpdC00LjEuMC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm
9yZy9jcm9zcy1zcGF3bi8tL2Nyb3NzLXNwYXduLTcuMC4zLnRnenNoZWJhbmctY29tbWFuZG5vZGUt
d2hpY2guL2Jpbi9ub2RlLXdoaWNoaHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvd2hpY2gvLS93aG
ljaC0yLjAuMi50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9pc2V4ZS8tL2lzZXhlLTIuMC4w
LnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3NoZWJhbmctY29tbWFuZC8tL3NoZWJhbmctY2
9tbWFuZC0yLjAuMC50Z3pzaGViYW5nLXJlZ2V4aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvc2hl
YmFuZy1yZWdleC8tL3NoZWJhbmctcmVnZXgtMy4wLjAudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy
5vcmcvcGF0aC1rZXkvLS9wYXRoLWtleS0zLjEuMS50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9y
Zy9wYXRoLXNjdXJyeS8tL3BhdGgtc2N1cnJ5LTEuMTEuMS50Z3peNS4wLjAgfHwgXjYuMC4yIHx8IF
43LjAuMGxydS1jYWNoZWh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2xydS1jYWNoZS8tL2xydS1j
YWNoZS0xMC40LjMudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvbWluaXBhc3MvLS9taW5pcG
Fzcy03LjEuMi50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9taW5pbWF0Y2gvLS9taW5pbWF0
Y2gtOS4wLjUudGd6YnJhY2UtZXhwYW5zaW9uaHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvYnJhY2
UtZXhwYW5zaW9uLy0vYnJhY2UtZXhwYW5zaW9uLTIuMC4xLnRnemJhbGFuY2VkLW1hdGNoaHR0cHM6
Ly9yZWdpc3RyeS5ucG1qcy5vcmcvYmFsYW5jZWQtbWF0Y2gvLS9iYWxhbmNlZC1tYXRjaC0xLjAuMi
50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9qYWNrc3BlYWsvLS9qYWNrc3BlYWstMy40LjMu
dGd6QGlzYWFjcy9jbGl1aUBwa2dqcy9wYXJzZWFyZ3NodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy
9AcGtnanMvcGFyc2VhcmdzLy0vcGFyc2VhcmdzLTAuMTEuMC50Z3podHRwczovL3JlZ2lzdHJ5Lm5w
bWpzLm9yZy9AaXNhYWNzL2NsaXVpLy0vY2xpdWktOC4wLjIudGd6c3RyaW5nLXdpZHRobnBtOnN0cm
luZy13aWR0aEBeNC4yLjBzdHJpbmctd2lkdGgtY2pzc3RyaXAtYW5zaW5wbTpzdHJpcC1hbnNpQF42
LjAuMXN0cmlwLWFuc2ktY2pzd3JhcC1hbnNpbnBtOndyYXAtYW5zaUBeNy4wLjB3cmFwLWFuc2ktY2
pzaHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvd3JhcC1hbnNpLy0vd3JhcC1hbnNpLTcuMC4wLnRn
emFuc2ktc3R5bGVzaHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvc3RyaXAtYW5zaS8tL3N0cmlwLW
Fuc2ktNi4wLjEudGd6YW5zaS1yZWdleGh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2Fuc2ktcmVn
ZXgvLS9hbnNpLXJlZ2V4LTUuMC4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3N0cmluZy
13aWR0aC8tL3N0cmluZy13aWR0aC00LjIuMy50Z3plbW9qaS1yZWdleGlzLWZ1bGx3aWR0aC1jb2Rl
LXBvaW50aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvaXMtZnVsbHdpZHRoLWNvZGUtcG9pbnQvLS
9pcy1mdWxsd2lkdGgtY29kZS1wb2ludC0zLjAuMC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9y
Zy9lbW9qaS1yZWdleC8tL2Vtb2ppLXJlZ2V4LTguMC4wLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtan
Mub3JnL2Fuc2ktc3R5bGVzLy0vYW5zaS1zdHlsZXMtNC4zLjAudGd6Y29sb3ItY29udmVydGh0dHBz
Oi8vcmVnaXN0cnkubnBtanMub3JnL2NvbG9yLWNvbnZlcnQvLS9jb2xvci1jb252ZXJ0LTIuMC4xLn
RnemNvbG9yLW5hbWVodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9jb2xvci1uYW1lLy0vY29sb3It
bmFtZS0xLjEuNC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy93cmFwLWFuc2kvLS93cmFwLW
Fuc2ktOC4xLjAudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvc3RyaXAtYW5zaS8tL3N0cmlw
LWFuc2ktNy4xLjAudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvYW5zaS1yZWdleC8tL2Fuc2
ktcmVnZXgtNi4xLjAudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvc3RyaW5nLXdpZHRoLy0v

c3RyaW5nLXdpZHRoLTUuMS4yLnRnemVhc3Rhc2lhbndpZHRoaHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy
5vcmcvZWFzdGFzaWFud2lkdGgvLS9lYXN0YXNpYW53aWR0aC0wLjIuMC50Z3podHRwczovL3JlZ2lz
dHJ5Lm5wbWpzLm9yZy9lbW9qaS1yZWdleC8tL2Vtb2ppLXJlZ2V4LTkuMi4yLnRnemh0dHBzOi8vcm
VnaXN0cnkubnBtanMub3JnL2Fuc2ktc3R5bGVzLy0vYW5zaS1zdHlsZXMtNi4yLjEudGd6aHR0cHM6
Ly9yZWdpc3RyeS5ucG1qcy5vcmcvY29tbWFuZGVyLy0vY29tbWFuZGVyLTQuMS4xLnRnemh0dHBzOi
8vcmVnaXN0cnkubnBtanMub3JnL0BqcmlkZ2V3ZWxsL2dlbi1tYXBwaW5nLy0vZ2VuLW1hcHBpbmct
MC4zLjUudGd6QGpyaWRnZXdlbGwvc2V0LWFycmF5QGpyaWRnZXdlbGwvdHJhY2UtbWFwcGluZ0Bqcm
lkZ2V3ZWxsL3NvdXJjZW1hcC1jb2RlY2h0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0BqcmlkZ2V3
ZWxsL3NvdXJjZW1hcC1jb2RlYy8tL3NvdXJjZW1hcC1jb2RlYy0xLjUuMC50Z3podHRwczovL3JlZ2
lzdHJ5Lm5wbWpzLm9yZy9AanJpZGdld2VsbC90cmFjZS1tYXBwaW5nLy0vdHJhY2UtbWFwcGluZy0w
LjMuMjUudGd6QGpyaWRnZXdlbGwvcmVzb2x2ZS11cmlodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy
9AanJpZGdld2VsbC9yZXNvbHZlLXVyaS8tL3Jlc29sdmUtdXJpLTMuMS4yLnRnemh0dHBzOi8vcmVn
aXN0cnkubnBtanMub3JnL0BqcmlkZ2V3ZWxsL3NldC1hcnJheS8tL3NldC1hcnJheS0xLjIuMS50Z3
podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9zb3VyY2UtbWFwLy0vc291cmNlLW1hcC0wLjguMC1i
ZXRhLjAudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvd2hhdHdnLXVybC8tL3doYXR3Zy11cm
wtNy4xLjAudGd6bG9kYXNoLnNvcnRieWh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3dlYmlkbC1j
b252ZXJzaW9ucy8tL3dlYmlkbC1jb252ZXJzaW9ucy00LjAuMi50Z3podHRwczovL3JlZ2lzdHJ5Lm
5wbWpzLm9yZy90cjQ2Ly0vdHI0Ni0xLjAuMS50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9w
dW55Y29kZS8tL3B1bnljb2RlLTIuMy4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2xvZG
FzaC5zb3J0YnkvLS9sb2Rhc2guc29ydGJ5LTQuNy4wLnRnemRpc3QvYmluL3JvbGx1cGh0dHBzOi8v
cmVnaXN0cnkubnBtanMub3JnL3JvbGx1cC8tL3JvbGx1cC00LjI0LjAudGd6QHR5cGVzL2VzdHJlZU
Byb2xsdXAvcm9sbHVwLWRhcndpbi1hcm02NEByb2xsdXAvcm9sbHVwLWFuZHJvaWQtYXJtNjRAcm9s
bHVwL3JvbGx1cC13aW4zMi1hcm02NC1tc3ZjQHJvbGx1cC9yb2xsdXAtbGludXgtYXJtNjQtZ251QH
JvbGx1cC9yb2xsdXAtbGludXgtYXJtNjQtbXVzbEByb2xsdXAvcm9sbHVwLWFuZHJvaWQtYXJtLWVh
YmlAcm9sbHVwL3JvbGx1cC1saW51eC1hcm0tZ251ZWFiaWhmQHJvbGx1cC9yb2xsdXAtbGludXgtYX
JtLW11c2xlYWJpaGZAcm9sbHVwL3JvbGx1cC13aW4zMi1pYTMyLW1zdmNAcm9sbHVwL3JvbGx1cC1s
aW51eC1yaXNjdjY0LWdudUByb2xsdXAvcm9sbHVwLWxpbnV4LXBvd2VycGM2NGxlLWdudUByb2xsdX
Avcm9sbHVwLWxpbnV4LXMzOTB4LWdudUByb2xsdXAvcm9sbHVwLWRhcndpbi14NjRAcm9sbHVwL3Jv
bGx1cC13aW4zMi14NjQtbXN2Y0Byb2xsdXAvcm9sbHVwLWxpbnV4LXg2NC1nbnVAcm9sbHVwL3JvbG
x1cC1saW51eC14NjQtbXVzbGh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Byb2xsdXAvcm9sbHVw
LWxpbnV4LXg2NC1tdXNsLy0vcm9sbHVwLWxpbnV4LXg2NC1tdXNsLTQuMjQuMC50Z3podHRwczovL3
JlZ2lzdHJ5Lm5wbWpzLm9yZy9Acm9sbHVwL3JvbGx1cC1saW51eC14NjQtZ251Ly0vcm9sbHVwLWxp
bnV4LXg2NC1nbnUtNC4yNC4wLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Byb2xsdXAvcm
9sbHVwLXdpbjMyLXg2NC1tc3ZjLy0vcm9sbHVwLXdpbjMyLXg2NC1tc3ZjLTQuMjQuMC50Z3podHRw
czovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9Acm9sbHVwL3JvbGx1cC1kYXJ3aW4teDY0Ly0vcm9sbHVwLW
Rhcndpbi14NjQtNC4yNC4wLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Byb2xsdXAvcm9s
bHVwLWxpbnV4LXMzOTB4LWdudS8tL3JvbGx1cC1saW51eC1zMzkweC1nbnUtNC4yNC4wLnRnemh0dH
BzOi8vcmVnaXN0cnkubnBtanMub3JnL0Byb2xsdXAvcm9sbHVwLWxpbnV4LXBvd2VycGM2NGxlLWdu
dS8tL3JvbGx1cC1saW51eC1wb3dlcnBjNjRsZS1nbnUtNC4yNC4wLnRnemh0dHBzOi8vcmVnaXN0cn
kubnBtanMub3JnL0Byb2xsdXAvcm9sbHVwLWxpbnV4LXJpc2N2NjQtZ251Ly0vcm9sbHVwLWxpbnV4
LXJpc2N2NjQtZ251LTQuMjQuMC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9Acm9sbHVwL3
JvbGx1cC13aW4zMi1pYTMyLW1zdmMvLS9yb2xsdXAtd2luMzItaWEzMi1tc3ZjLTQuMjQuMC50Z3po
dHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9Acm9sbHVwL3JvbGx1cC1saW51eC1hcm0tbXVzbGVhYm
loZi8tL3JvbGx1cC1saW51eC1hcm0tbXVzbGVhYmloZi00LjI0LjAudGd6aHR0cHM6Ly9yZWdpc3Ry

eS5ucG1qcy5vcmcvQHJvbGx1cC9yb2xsdXAtbGludXgtYXJtLWdudWVhYmloZi8tL3JvbGx1cC1saW
51eC1hcm0tZ251ZWFiaWhmLTQuMjQuMC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9Acm9s
bHVwL3JvbGx1cC1hbmRyb2lkLWFybS1lYWJpLy0vcm9sbHVwLWFuZHJvaWQtYXJtLWVhYmktNC4yNC
4wLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Byb2xsdXAvcm9sbHVwLWxpbnV4LWFybTY0
LW11c2wvLS9yb2xsdXAtbGludXgtYXJtNjQtbXVzbC00LjI0LjAudGd6aHR0cHM6Ly9yZWdpc3RyeeS
5ucG1qcy5vcmcvQHJvbGx1cC9yb2xsdXAtbGludXgtYXJtNjQtZ251Ly0vcm9sbHVwLWxpbnV4LWFy
bTY0LWdudS00LjI0LjAudGd6aHR0cHM6Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvQHJvbGx1cC9yb2xsdX
Atd2luMzItYXJtNjQtbXN2Yy8tL3JvbGx1cC13aW4zMi1hcm02NC1tc3ZjLTQuMjQuMC50Z3podHRw
czovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9Acm9sbHVwL3JvbGx1cC1hbmRyb2lkLWFybTY0Ly0vcm9sbH
VwLWFuZHJvaWQtYXJtNjQtNC4yNC4wLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Byb2xs
dXAvcm9sbHVwLWRhcndpbi1hcm02NC8tL3JvbGx1cC1kYXJ3aW4tYXJtNjQtNC4yNC4wLnRnemh0dH
BzOi8vcmVnaXN0cnkubnBtanMub3JnL2ZzZXZlbnRzLy0vZnNldmVudHMtMi4zLjMudGd6aHR0cHM6
Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvQHR5cGVzL2VzdHJlZS8tL2VzdHJlZS0xLjAuNi50Z3podHRwcz
ovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9yZXNvbHZlLWZyb20vLS9yZXNvbHZlLWZyb20tNS4wLjAudGd6
aHR0cHM6Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvcG9zdGNzcy1sb2FkLWNvbmZpZy8tL3Bvc3Rjc3MtbG
9hZC1jb25maWctNi4wLjEudGd6bGlsY29uZmlnaHR0cHM6Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvbGls
Y29uZmlnLy0vbGlsY29uZmlnLTMuMS4yLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3BpY2
9jb2xvcnMvLS9waWNvY29sb3JzLTEuMS4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2pv
eWNvbi8tL2pveWNvbi0zLjEuMS50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9leGVjYS8tL2
V4ZWNhLTUuMS4xLnRnemlzLXN0cmVhbWdldC1zdHJlYW1tZXJnZS1zdHJlYW1ucG0tcnVuLXBhdGho
dW1hbi1zaWduYWxzc3RyaXAtZmluYWwtbmV3bGluZWh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3
N0cmlwLWZpbmFsLW5ld2xpbmUvLS9zdHJpcC1maW5hbC1uZXdsaW5lLTIuMC4wLnRnemh0dHBzOi8v
cmVnaXN0cnkubnBtanMub3JnL2h1bWFuLXNpZ25hbHMvLS9odW1hbi1zaWduYWxzLTIuMS4wLnRnem
h0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL25wbS1ydW4tcGF0aC8tL25wbS1ydW4tcGF0aC00LjAu
MS50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9tZXJnZS1zdHJlYW0vLS9tZXJnZS1zdHJlYW
0tMi4wLjAudGd6aHR0cHM6Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvc2lnbmFsLWV4aXQvLS9zaWduYWwt
ZXhpdC0zLjAuNy50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9nZXQtc3RyZWFtLy0vZ2V0LX
N0cmVhbS02LjAuMS50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9pcy1zdHJlYW0vLS9pcy1z
dHJlYW0tMi4wLjEudGd6aHR0cHM6Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvb25ldGltZS8tL29uZXRpbW
UtNS4xLjIudGd6aHR0cHM6Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvbWltaWMtZm4vLS9taW1pYy1mbi0y
LjEuMC50Z3piaW4vZXNidWlsZGh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2VzYnVpbGQvLS9lc2
J1aWxkLTAuMjMuMS50Z3pAZXNidWlsZC9haXgtcHBjNjRAZXNidWlsZC9saW51eC1hcm1AZXNidWls
ZC9saW51eC14NjRAZXNidWlsZC9zdW5vcy14NjRAZXNidWlsZC93aW4zMi14NjRAZXNidWlsZC9kYX
J3aW4teDY0QGVzYnVpbGQvbGludXgtaWEzMkBlc2J1aWxkL25ldGJzZC14NjRAZXNidWlsZC93aW4z
Mi1pYTMyQGVzYnVpbGQvYW5kcm9pZC1hcm1AZXNidWlsZC9hbmRyb2lkLXg2NEBlc2J1aWxkL2ZyZW
Vic2QteDY0QGVzYnVpbGQvbGludXgtYXJtNjRAZXNidWlsZC9saW51eC1wcGM2NEBlc2J1aWxkL2xp
bnV4LXMzOTB4QGVzYnVpbGQvb3BlbmJzZC14NjRAZXNidWlsZC93aW4zMi1hcm02NEBlc2J1aWxkL2
Rhcndpbi1hcm02NEBlc2J1aWxkL2FuZHJvaWQtYXJtNjRAZXNidWlsZC9mcmVlYnNkLWFybTY0QGVz
YnVpbGQvbGludXgtbG9vbmc2NEBlc2J1aWxkL2xpbnV4LXJpc2N2NjRAZXNidWlsZC9vcGVuYnNkLW
FybTY0QGVzYnVpbGQvbGludXgtbWlwczY0ZWxodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9AZXNi
dWlsZC9saW51eC1taXBzNjRlbC8tL2xpbnV4LW1pcHM2NGVsLTAuMjMuMS50Z3podHRwczovL3JlZ2
lzdHJ5Lm5wbWpzLm9yZy9AZXNidWlsZC9vcGVuYnNkLWFybTY0Ly0vb3BlbmJzZC1hcm02NC0wLjIz
LjEudGd6aHR0cHM6Ly9yZWdpc3RyeeS5ucG1qcy5vcmcvQGVzYnVpbGQvbGludXgtcmlzY3Y2NC8tL2
xpbnV4LXJpc2N2NjQtMC4yMy4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Blc2J1aWxk

L2xpbnV4LWxvb25nNjQvLS9saW51eC1sb29uZzY0LTAuMjMuMS50Z3podHRwczovL3JlZ2lzdHJ5Lm
5wbWpzLm9yZy9AZXNidWlsZC9mcmVlYnNkLWFybTY0Ly0vZnJlZWJzZC1hcm02NC0wLjIzLjEudGd6
aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvYW5kcm9pZC1hcm02NC8tL2FuZHJvaW
QtYXJtNjQtMC4yMy4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Blc2J1aWxkL2Rhcndp
bi1hcm02NC8tL2Rhcndpbi1hcm02NC0wLjIzLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcm
cvQGVzYnVpbGQvd2luMzItYXJtNjQvLS93aW4zMi1hcm02NC0wLjIzLjEudGd6aHR0cHM6Ly9yZWdp
c3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvb3BlbmJzZC14NjQvLS9vcGVuYnNkLXg2NC0wLjIzLjEudG
d6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvbGludXgtczM5MHgvLS9saW51eC1z
MzkweC0wLjIzLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvbGludXgtcH
BjNjQvLS9saW51eC1wcGM2NC0wLjIzLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVz
YnVpbGQvbGludXgtYXJtNjQvLS9saW51eC1hcm02NC0wLjIzLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS
5ucG1qcy5vcmcvQGVzYnVpbGQvZnJlZWJzZC14NjQvLS9mcmVlYnNkLXg2NC0wLjIzLjEudGd6aHR0
cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvYW5kcm9pZC14NjQvLS9hbmRyb2lkLXg2NC
0wLjIzLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvYW5kcm9pZC1hcm0v
LS9hbmRyb2lkLWFybS0wLjIzLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbG
Qvd2luMzItaWEzMi8tL3dpbjMyLWlhMzItMC4yMy4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMu
b3JnL0Blc2J1aWxkL25ldGJzZC14NjQvLS9uZXRic2QteDY0LTAuMjMuMS50Z3podHRwczovL3JlZ2
lzdHJ5Lm5wbWpzLm9yZy9AZXNidWlsZC9saW51eC1pYTMyLy0vbGludXgtaWEzMi0wLjIzLjEudGd6
aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvZGFyd2luLXg2NC8tL2Rhcndpbi14Nj
QtMC4yMy4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0Blc2J1aWxkL3dpbjMyLXg2NC8t
L3dpbjMyLXg2NC0wLjIzLjEudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvc3
Vub3MteDY0Ly0vc3Vub3MteDY0LTAuMjMuMS50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9A
ZXNidWlsZC9saW51eC14NjQvLS9saW51eC14NjQtMC4yMy4xLnRnemh0dHBzOi8vcmVnaXN0cnkubn
BtanMub3JnL0Blc2J1aWxkL2xpbnV4LWFybS8tL2xpbnV4LWFybS0wLjIzLjEudGd6aHR0cHM6Ly9y
ZWdpc3RyeS5ucG1qcy5vcmcvQGVzYnVpbGQvYWl4LXBwYzY0Ly0vYWl4LXBwYzY0LTAuMjMuMS50Z3
podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9kZWJ1Zy8tL2RlYnVnLTQuMy43LnRnemh0dHBzOi8v
cmVnaXN0cnkubnBtanMub3JnL21zLy0vbXMtMi4xLjMudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy
5vcmcvY29uc29sYS8tL2NvbnNvbGEtMy4yLjMudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcv
Y2hva2lkYXIvLS9jaG9raWRhci0zLjYuMC50Z3pnbG9iLXBhcmVudGlzLWJpbmFyeS1wYXRobm9ybW
FsaXplLXBhdGhodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9ub3JtYWxpemUtcGF0aC8tL25vcm1h
bGl6ZS1wYXRoLTMuMC4wLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2lzLWJpbmFyeS1wYX
RoLy0vaXMtYmluYXJ5LXBhdGgtMi4xLjAudGd6YmluYXJ5LWV4dGVuc2lvbnNodHRwczovL3JlZ2lz
dHJ5Lm5wbWpzLm9yZy9iaW5hcnktZXh0ZW5zaW9ucy8tL2JpbmFyeS1leHRlbnNpb25zLTIuMy4wLn
Rnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2dsb2ItcGFyZW50Ly0vZ2xvYi1wYXJlbnQtNS4x
LjIudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvaXMtZ2xvYi8tL2lzLWdsb2ItNC4wLjMudG
d6aXMtZXh0Z2xvYmh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2lzLWV4dGdsb2IvLS9pcy1leHRn
bG9iLTIuMS4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3JlYWRkaXJwLy0vcmVhZGRpcn
AtMy42LjAudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvcGljb21hdGNoLy0vcGljb21hdGNo
LTIuMy4xLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL2FueW1hdGNoLy0vYW55bWF0Y2gtMy
4xLjMudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvYnJhY2VzLy0vYnJhY2VzLTMuMC4zLnRn
emZpbGwtcmFuZ2VodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9maWxsLXJhbmdlLy0vZmlsbC1yYW
5nZS03LjEuMS50Z3p0by1yZWdleC1yYW5nZWh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL3RvLXJl
Z2V4LXJhbmdlLy0vdG8tcmVnZXgtcmFuZ2UtNS4wLjEudGd6aXMtbnVtYmVyaHR0cHM6Ly9yZWdpc3
RyeS5ucG1qcy5vcmcvaXMtbnVtYmVyLy0vaXMtbnVtYmVyLTcuMC4wLnRnemh0dHBzOi8vcmVnaXN0

cnkubnBtanMub3JnL2NhYy8tL2NhYy02LjcuMTQudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcm
cvYnVuZGxlLXJlcXVpcmUvLS9idW5kbGUtcmVxdWlyZS01LjAuMC50Z3psb2FkLXRzY29uZmlnaHR0
cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvbG9hZC10c2NvbmZpZy8tL2xvYWQtdHNjb25maWctMC4yLj
UudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQHR5cGVzL3JlYWN0Ly0vcmVhY3QtMTguMy4x
Mi50Z3pAdHlwZXMvcHJvcC10eXBlc2h0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0B0eXBlcy9wcm
9wLXR5cGVzLy0vcHJvcC10eXBlcy0xNS43LjEzLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3Jn
L2Nzc3R5cGUvLS9jc3N0eXBlLTMuMS4zLnRnemh0dHBzOi8vcmVnaXN0cnkubnBtanMub3JnL0B0eX
Blcy9ub2RlLy0vbm9kZS0yMi43LjkudGd6dW5kaWNpLXR5cGVzaHR0cHM6Ly9yZWdpc3RyeS5ucG1q
cy5vcmcvdW5kaWNpLXR5cGVzLy0vdW5kaWNpLXR5cGVzLTYuMTkuOC50Z3piaW4vYmlvbWVodHRwcz
ovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9AYmlvbWVqcy9iaW9tZS8tL2Jpb21lLTEuOS40LnRnekBiaW9t
ZWpzL2NsaS13aW4zMi14NjRAYmlvbWVqcy9jbGktd2luMzItYXJtNjRAYmlvbWVqcy9jbGktZGFyd2
luLXg2NEBiaW9tZWpzL2NsaS1kYXJ3aW4tYXJtNjRAYmlvbWVqcy9jbGktbGludXgteDY0QGJpb21l
anMvY2xpLWxpbnV4LWFybTY0QGJpb21lanMvY2xpLWxpbnV4LXg2NC1tdXNsQGJpb21lanMvY2xpLW
xpbnV4LWFybTY0LW11c2xodHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9AYmlvbWVqcy9jbGktbGlu
dXgtYXJtNjQtbXVzbC8tL2NsaS1saW51eC1hcm02NC1tdXNsLTEuOS40LnRnemh0dHBzOi8vcmVnaX
N0cnkubnBtanMub3JnL0BiaW9tZWpzL2NsaS1saW51eC14NjQtbXVzbC8tL2NsaS1saW51eC14NjQt
bXVzbC0xLjkuNC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9yZy9AYmlvbWVqcy9jbGktbGludX
gtYXJtNjQvLS9jbGktbGludXgtYXJtNjQtMS45LjQudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5v
cmcvQGJpb21lanMvY2xpLWxpbnV4LXg2NC8tL2NsaS1saW51eC14NjQtMS45LjQudGd6aHR0cHM6Ly
9yZWdpc3RyeS5ucG1qcy5vcmcvQGJpb21lanMvY2xpLWRhcndpbi1hcm02NC8tL2NsaS1kYXJ3aW4t
YXJtNjQtMS45LjQudGd6aHR0cHM6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGJpb21lanMvY2xpLWRhcn
dpbi14NjQvLS9jbGktZGFyd2luLXg2NC0xLjkuNC50Z3podHRwczovL3JlZ2lzdHJ5Lm5wbWpzLm9y
Zy9AYmlvbWVqcy9jbGktd2luMzItYXJtNjQvLS9jbGktd2luMzItYXJtNjQtMS45LjQudGd6aHR0cH
M6Ly9yZWdpc3RyeS5ucG1qcy5vcmcvQGJpb21lanMvY2xpLXdpbjMyLXg2NC8tL2NsaS13aW4zMi14
NjQtMS45LjQudGd6AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
```
{
  "name": "@cartesia/cartesia-js",
  "author": {
    "name": "Cartesia",
    "url": "https://cartesia.ai"
  },
  "version": "1.3.0",
  "description": "Client for the Cartesia API.",
  "type": "module",
  "module": "./dist/index.js",
  "types": "./dist/index.d.ts",
  "exports": {
    ".": {
      "import": "./dist/index.js",
      "require": "./dist/index.cjs"
    },
    "./react": {
      "import": "./dist/react/index.js",
      "require": "./dist/react/index.cjs"
    }
  },
  "engines": {
    "node": ">=18"
  },
  "dependencies": {
    "base64-js": "^1.5.1",
    "cross-fetch": "^4.0.0",
    "emittery": "^1.0.3",
    "human-id": "^4.1.1",
    "partysocket": "^1.0.1",
    "react": "^18.3.1"
  },
  "publishConfig": {
    "access": "public"
  },
  "scripts": {
    "build": "tsup src/ --format cjs,esm --dts",
    "dev": "bun run build -- --watch",
    "release": "rm -rf dist && bun run build && npm publish"
  },
  "devDependencies": {
    "@biomejs/biome": "^1.9.4",
    "@types/node": "^22.7.9",
    "@types/react": "^18.3.12",
    "tsup": "^8.0.2",
    "typescript": "^5.6.3"
  }
}
export { Cartesia as default } from "./lib";
export * from "./lib";
export * from "./types";
export { default as WebPlayer } from "./tts/player";
export { default as Source } from "./tts/source";
export { default as WebSocket } from "./tts/websocket";
import fetch from "cross-fetch";
import type { ClientOptions } from "../types";
import { BASE_URL, CARTESIA_VERSION, constructApiUrl } from "./constants";

export class Client {
  apiKey: () => Promise<string>;
  baseUrl: string;

  constructor(options: ClientOptions = {}) {
    const apiKey = options.apiKey || process.env.CARTESIA_API_KEY;
    if (!apiKey) {
      throw new Error("Missing Cartesia API key.");
    }

    this.apiKey = typeof apiKey === "function" ? apiKey : async () => apiKey;
    this.baseUrl = options.baseUrl || BASE_URL;
  }

  protected async _fetch(path: string, options: RequestInit = {}) {
    const url = constructApiUrl(this.baseUrl, path);
    const headers = new Headers(options.headers);

    headers.set("X-API-Key", await this.apiKey());
    headers.set("Cartesia-Version", CARTESIA_VERSION);

    return fetch(url.toString(), {
      ...options,
      headers,
    });
  }
}
```

```typescript
export const BASE_URL = "https://api.cartesia.ai";
export const CARTESIA_VERSION = "2024-06-10";

/**
 * Construct a URL for the Cartesia API.
 *
 * @param baseUrl The base URL for the API.
 * @param path The path to append to the base URL.
 * @param options Options for the URL.
 * @param options.websocket Whether to use the WebSocket protocol.
 * @returns A URL object.
 */
export const constructApiUrl = (
  baseUrl: string,
  path: string,
  { websocket = false } = {},
) => {
  const url = new URL(path, baseUrl);
  if (websocket) {
    // Using find-and-replace ensures that if the base URL uses TLS, the
    // new protocol does too.
    url.protocol = baseUrl.replace(/^http/, "ws");
  }
  return url;
};
import TTS from "../tts";
import type { ClientOptions } from "../types";
import VoiceChanger from "../voice-changer";
import Voices from "../voices";
import { Client } from "./client";

export class Cartesia extends Client {
  tts: TTS;
  voices: Voices;
  voiceChanger: VoiceChanger;

  constructor(options: ClientOptions = {}) {
    super(options);

    this.tts = new TTS(options);
    this.voices = new Voices(options);
    this.voiceChanger = new VoiceChanger(options);
  }
}
import type { UnsubscribeFunction } from "emittery";
import { useCallback, useEffect, useMemo, useRef, useState } from "react";
import { Cartesia } from "../lib";
import Player from "../tts/player";
import type Source from "../tts/source";
import type WebSocket from "../tts/websocket";
import type { StreamRequest } from "../types";
import { pingServer } from "./utils";

export type UseTTSOptions = {
  apiKey: string | (() => Promise<string>) | null;
  baseUrl?: string;
  sampleRate: number;
  onError?: (error: Error) => void;
};

export type PlaybackStatus = "inactive" | "playing" | "paused" | "finished";
export type BufferStatus = "inactive" | "buffering" | "buffered";

export type Metrics = {
  modelLatency: number | null;
};

export interface UseTTSReturn {
  buffer: (options: StreamRequest) => Promise<void>;
  play: (bufferDuration?: number) => Promise<void>;
  pause: () => Promise<void>;
  resume: () => Promise<void>;
  toggle: () => Promise<void>;
  source: Source | null;
  playbackStatus: PlaybackStatus;
  bufferStatus: BufferStatus;
  isWaiting: boolean;
  isConnected: boolean;
  metrics: Metrics;
}

const PING_INTERVAL = 5000;
const DEFAULT_BUFFER_DURATION = 0.01;

type Message = {
  step_time: number;
};

/**
 * React hook to use the Cartesia audio API.
 */
export function useTTS({
  apiKey,
  baseUrl,
  sampleRate,
  onError,
}: UseTTSOptions): UseTTSReturn {
  if (typeof window === "undefined") {
    return {
      buffer: async () => {},
      play: async () => {},
      pause: async () => {},
      resume: async () => {},
      toggle: async () => {},
      playbackStatus: "inactive",
      bufferStatus: "inactive",
      isWaiting: false,
      source: null,
      isConnected: false,
      metrics: {
        modelLatency: null,
      },
    };
  }

  const websocket = useMemo(() => {
    if (!apiKey) {
      return null;
    }
    const cartesia = new Cartesia({ apiKey, baseUrl });
    baseUrl = baseUrl ?? cartesia.baseUrl;
    return cartesia.tts.websocket({
      container: "raw",
      encoding: "pcm_f32le",
      sampleRate,
    });
  }, [apiKey, baseUrl, sampleRate]);
  const websocketReturn = useRef<ReturnType<WebSocket["send"]> | null>(null);
  const player = useRef<Player | null>(null);
  const [playbackStatus, setPlaybackStatus] =
    useState<PlaybackStatus>("inactive");
  const [bufferStatus, setBufferStatus] = useState<BufferStatus>("inactive");
  const [isWaiting, setIsWaiting] = useState(false);
  const [isConnected, setIsConnected] = useState(false);
  const [bufferDuration, setBufferDuration] = useState<number | null>(null);
  const [messages, setMessages] = useState<Message[]>([]);

  const buffer = useCallback(
    async (options: StreamRequest) => {
      websocketReturn.current?.stop(); // Abort the previous request if it
exists.

      try {
        setMessages([]);
        setBufferStatus("buffering");
        websocketReturn.current = websocket?.send(options) ?? null;
        if (!websocketReturn.current) {
          return;
        }
        const unsubscribe = websocketReturn.current.on("message", (message)
=> {
          const parsedMessage = JSON.parse(message);
          setMessages((messages) => [...messages, parsedMessage]);
          if (parsedMessage.error) {
            onError?.(new Error(parsedMessage.error));
          }
        });
        await websocketReturn.current.source.once("close");
        setBufferStatus("buffered");
        unsubscribe();
      } catch (error) {
        if (error instanceof Error) {
          onError?.(error);
        } else {
          console.error(error);
        }
      }
    },
    [websocket, onError],
  );

  const metrics = useMemo(() => {
    // Model Latency is the first step time
    if (messages.length === 0) {
      return {
        modelLatency: null,
      };
    }
    const modelLatency = messages[0].step_time ?? null;
    return {
      modelLatency: Math.trunc(modelLatency),
    };
  }, [messages]);
  useEffect(() => {
    let cleanup: (() => void) | undefined = () => {};
    async function setupConnection() {
      try {
        const connection = await websocket?.connect();
        if (!connection) {
          return;
        }
        const unsubscribes = <UnsubscribeFunction[]>[];
        // The await ensures that the connection is open, so we already know
that we are connected.
        setIsConnected(true);
        // If the WebSocket is the kind that automatically reconnects, we
need an additional
        // listener for the open event to update the connection status.
        unsubscribes.push(
          connection.on("open", () => {
            setIsConnected(true);
          }),
        );
        unsubscribes.push(
          connection.on("close", () => {
            setIsConnected(false);
          }),
        );
        const intervalId = setInterval(() => {
          if (baseUrl) {
            pingServer(new URL(baseUrl).origin).then((ping) => {
              let bufferDuration: number;
              if (ping < 300) {
                bufferDuration = 0.01; // No buffering for very low latency
              } else if (ping > 1500) {
                bufferDuration = 6; // Max buffering for very high latency (6
seconds)
              } else {
                bufferDuration = (ping / 1000) * 4; // Adjust buffer duration
based on ping
              }
              setBufferDuration(bufferDuration);
            });
          }
        }, PING_INTERVAL);
        return () => {
          for (const unsubscribe of unsubscribes) {
            unsubscribe();
          }
          clearInterval(intervalId);
          websocket?.disconnect();
        };
      } catch (e) {
        console.error(e);
      }
    }
    setupConnection().then((cleanupConnection) => {
      cleanup = cleanupConnection;
    });
    return () => cleanup?.();
  }, [websocket, baseUrl]);

  const play = useCallback(async () => {
    try {
      if (playbackStatus === "playing" || !websocketReturn.current) {
        return;
      }
      if (player.current) {
        // Stop the current player if it exists.
        await player.current.stop();
      }

      if (playbackStatus === "finished") {
        websocketReturn.current.source.seek(0, "start");
      }

      setPlaybackStatus("playing");

      const unsubscribes = [];
      unsubscribes.push(
        websocketReturn.current.source.on("wait", () => {
          setIsWaiting(true);
        }),
      );
      unsubscribes.push(
        websocketReturn.current.source.on("read", () => {
          setIsWaiting(false);
        }),
      );

      player.current = new Player({
        bufferDuration: bufferDuration ?? DEFAULT_BUFFER_DURATION,
      });
      // Wait for the playback to finish before setting isPlaying to false.
      await player.current.play(websocketReturn.current.source);

      for (const unsubscribe of unsubscribes) {
        // Deregister the event listeners (.on()) that we registered above to
avoid memory leaks.
        unsubscribe();
      }

      setPlaybackStatus("finished");
    } catch (error) {
      if (error instanceof Error) {
        onError?.(error);
      } else {
        console.error(error);
      }
    }
  }, [playbackStatus, bufferDuration, onError]);

  const pause = useCallback(async () => {
    try {
      await player.current?.pause();
      setPlaybackStatus("paused");
    } catch (error) {
      if (error instanceof Error) {
        onError?.(error);
      } else {
        console.error(error);
      }
    }
  }, [onError]);
  const resume = useCallback(async () => {
    try {
      await player.current?.resume();
      setPlaybackStatus("playing");
    } catch (error) {
      if (error instanceof Error) {
        onError?.(error);
      } else {
        console.error(error);
      }
    }
  }, [onError]);

  const toggle = useCallback(async () => {
    try {
      await player.current?.toggle();
      setPlaybackStatus((status) => {
        if (status === "playing") {
          return "paused";
        }
        if (status === "paused") {
          return "playing";
        }
        return status;
      });
    } catch (error) {
      if (error instanceof Error) {
        onError?.(error);
      } else {
        console.error(error);
      }
    }
  }, [onError]);

  return {
    buffer,
    play,
    pause,
    source: websocketReturn.current?.source ?? null,
    resume,
    toggle,
    playbackStatus,
    bufferStatus,
    isWaiting,
    isConnected,
    metrics,
  };
}
/**
 * Ping the server to calculate the round-trip time. This is useful for
buffering audio in high-latency environments.
 * @param url The URL to ping.
 */

export async function pingServer(url: string): Promise<number> {
  const start = new Date().getTime();
  await fetch(url);
  const end = new Date().getTime();
  return end - start;
}
```

```
import { Client } from "../lib/client";
import type { BytesRequest, WebSocketOptions } from "../types";
import WebSocket from "./websocket";

export default class TTS extends Client {
  /**
   * Get a WebSocket client for streaming audio from the TTS API.
   *
   * @returns {WebSocket} A Cartesia WebSocket client.
   */
  websocket(options: WebSocketOptions): WebSocket {
    return new WebSocket(options, {
      apiKey: this.apiKey,
      baseUrl: this.baseUrl,
    });
  }

  /**
   * Generate audio bytes from text.
   *
   * @param options - The options for the request.
   * @returns {Promise<ArrayBuffer>} A promise that resolves to an
ArrayBuffer containing the audio bytes.
   */
  async bytes(options: BytesRequest): Promise<ArrayBuffer> {
    const response = await this._fetch("/tts/bytes", {
      method: "POST",
      body: JSON.stringify(options),
    });

    return response.arrayBuffer();
  }
}
import type Source from "./source";
import { playAudioBuffer } from "./utils";

export default class Player {
  #context: AudioContext | null = null;
  #startNextPlaybackAt = 0;
  #bufferDuration: number;

  /**
   * Create a new Player.
   *
   * @param options - Options for the Player.
   * @param options.bufferDuration - The duration of the audio buffer to play.
   */
  constructor({ bufferDuration }: { bufferDuration: number }) {
    this.#bufferDuration = bufferDuration;
  }

  async #playBuffer(buf: Float32Array, sampleRate: number) {
    if (!this.#context) {
      throw new Error("AudioContext not initialized.");
    }
    if (buf.length === 0) {
      return;
    }

    const startAt = this.#startNextPlaybackAt;
    const duration = buf.length / sampleRate;
    this.#startNextPlaybackAt =
      duration + Math.max(this.#context.currentTime,
this.#startNextPlaybackAt);

    await playAudioBuffer(buf, this.#context, startAt, sampleRate);
  }

  /**
   * Play audio from a source.
   *
   * @param source The source to play audio from.
   * @returns A promise that resolves when the audio has finished playing.
   */
  async play(source: Source) {
    this.#startNextPlaybackAt = 0;
    this.#context = new AudioContext({ sampleRate: source.sampleRate });
    const buffer = new Float32Array(
      source.durationToSampleCount(this.#bufferDuration),
    );

    const plays: Promise<void>[] = [];
    while (true) {
      const read = await source.read(buffer);
      // If we've reached the end of the source, then read < buffer.length.
      // In that case, we don't want to play the entire buffer, as that
      // will cause repeated audio.
      // So we set the buffer to the correct length.
      const playableAudio = buffer.subarray(0, read);
      plays.push(this.#playBuffer(playableAudio, source.sampleRate));

      if (read < buffer.length) {
        // No more audio to read.
        break;
      }
    }
    await Promise.all(plays);
  }

  /**
   * Pause the audio.
   *
   * @returns A promise that resolves when the audio has been paused.
   */
  async pause() {
    if (!this.#context) {
      throw new Error("AudioContext not initialized.");
    }
    await this.#context.suspend();
  }

  /**
   * Resume the audio.
   *
   * @returns A promise that resolves when the audio has been resumed.
   */
  async resume() {
    if (!this.#context) {
      throw new Error("AudioContext not initialized.");
    }
    await this.#context.resume();
  }
  /**
   * Toggle the audio.
   *
   * @returns A promise that resolves when the audio has been toggled.
   */
  async toggle() {
    if (!this.#context) {
      throw new Error("AudioContext not initialized.");
    }
    if (this.#context.state === "running") {
      await this.pause();
    } else {
      await this.resume();
    }
  }

  /**
   * Stop the audio.
   *
   * @returns A promise that resolves when the audio has been stopped.
   */
  async stop() {
    if (!this.#context) {
      throw new Error("AudioContext not initialized.");
    }
    await this.#context?.close();
  }
}
import Emittery from "emittery";
import type { Encoding, SourceEventData, TypedArray } from "../types";

type EncodingInfo = {
  arrayType:
    | Float32ArrayConstructor
    | Int16ArrayConstructor
    | Uint8ArrayConstructor;
  bytesPerElement: number;
};

export const ENCODING_MAP: Record<Encoding, EncodingInfo> = {
  pcm_f32le: { arrayType: Float32Array, bytesPerElement: 4 },
  pcm_s16le: { arrayType: Int16Array, bytesPerElement: 2 },
  pcm_alaw: { arrayType: Uint8Array, bytesPerElement: 1 },
  pcm_mulaw: { arrayType: Uint8Array, bytesPerElement: 1 },
};

export default class Source {
  #emitter = new Emittery<SourceEventData>();
  #buffer: TypedArray;
  #readIndex = 0;
  #writeIndex = 0;
  #closed = false;
  #sampleRate: number;
  #encoding: Encoding;
  #container: string;

  on = this.#emitter.on.bind(this.#emitter);
  once = this.#emitter.once.bind(this.#emitter);
  events = this.#emitter.events.bind(this.#emitter);
  off = this.#emitter.off.bind(this.#emitter);
```

```
/**
 * Create a new Source.
 *
 * @param options - Options for the Source.
 * @param options.sampleRate - The sample rate of the audio.
 */
constructor({
  sampleRate,
  encoding,
  container,
}: {
  sampleRate: number;
  encoding: string;
  container: string;
}) {
  this.#sampleRate = sampleRate;
  this.#encoding = encoding as Encoding;
  this.#container = container;
  this.#buffer = this.#createBuffer(1024); // Initial size, can be adjusted
}

get sampleRate() {
  return this.#sampleRate;
}

get encoding() {
  return this.#encoding;
}

get container() {
  return this.#container;
}

/**
 * Create a new buffer for the source.
 *
 * @param size - The size of the buffer to create.
 * @returns The new buffer as a TypedArray based on the encoding.
 */
#createBuffer(size: number): TypedArray {
  const { arrayType: ArrayType } = ENCODING_MAP[this.#encoding];
  return new ArrayType(size);
}

/**
 * Append audio to the buffer.
 *
 * @param src The audio to append.
 */
async enqueue(src: TypedArray) {
  const requiredCapacity = this.#writeIndex + src.length;

  // Resize buffer if necessary
  if (requiredCapacity > this.#buffer.length) {
    let newCapacity = this.#buffer.length;
    while (newCapacity < requiredCapacity) {
      newCapacity *= 2; // Double the buffer size
    }

    const newBuffer = this.#createBuffer(newCapacity);
```

```
        newBuffer.set(this.#buffer);
        this.#buffer = newBuffer;
      }

      // Append the audio to the buffer.
      this.#buffer.set(src, this.#writeIndex);
      this.#writeIndex += src.length;
      await this.#emitter.emit("enqueue");
    }

    /**
     * Read audio from the buffer.
     *
     * @param dst The buffer to read the audio into.
     * @returns The number of samples read. If the source is closed, this will
  be
     * less than the length of the provided buffer.
     */
    async read(dst: TypedArray): Promise<number> {
      // Read the buffer into the provided buffer.
      const targetReadIndex = this.#readIndex + dst.length;

      while (!this.#closed && targetReadIndex > this.#writeIndex) {
        // Wait for more audio to be enqueued.
        await this.#emitter.emit("wait");
        await Promise.race([
          this.#emitter.once("enqueue"),
          this.#emitter.once("close"),
        ]);
        await this.#emitter.emit("read");
      }

      const read = Math.min(dst.length, this.#writeIndex - this.#readIndex);
      dst.set(this.#buffer.subarray(this.#readIndex, this.#readIndex + read));
      this.#readIndex += read;
      return read;
    }

    /**
     * Seek in the buffer.
     *
     * @param offset The offset to seek to.
     * @param whence The position to seek from.
     * @returns The new position in the buffer.
     * @throws {Error} If the seek is invalid.
     */
    async seek(
      offset: number,
      whence: "start" | "current" | "end",
    ): Promise<number> {
      let position = this.#readIndex;
      switch (whence) {
        case "start":
          position = offset;
          break;
        case "current":
          position += offset;
          break;
        case "end":
          position = this.#writeIndex + offset;
          break;
        default:
          throw new Error(`Invalid seek mode: ${whence}`);
      }

      if (position < 0 || position > this.#writeIndex) {
        throw new Error("Seek out of bounds");
      }

      this.#readIndex = position;
      return position;
    }

    /**
     * Get the number of samples in a given duration.
     *
     * @param durationSecs The duration in seconds.
     * @returns The number of samples.
     */
    durationToSampleCount(durationSecs: number) {
      return Math.trunc(durationSecs * this.#sampleRate);
    }

    get buffer() {
      return this.#buffer;
    }

    get readIndex() {
      return this.#readIndex;
    }

    get writeIndex() {
      return this.#writeIndex;
    }

    /**
     * Close the source. This signals that no more audio will be enqueued.
     *
     * This will emit a "close" event.
     *
     * @returns A promise that resolves when the source is closed.
     */
    async close() {
      this.#closed = true;
      await this.#emitter.emit("close");
      this.#emitter.clearListeners();
    }
  }
  import base64 from "base64-js";
  import type Emittery from "emittery";
  import type {
    Chunk,
    EmitteryCallbacks,
    Encoding,
    Sentinel,
    TypedArray,
    WebSocketResponse,
  } from "../types";
  import { ENCODING_MAP } from "./source";

  /**
   * Convert base64-encoded audio buffer(s) to a TypedArray.
   *
   * @param b64 The base64-encoded audio buffer, or an array of base64-encoded
   * audio buffers.
   * @param encoding The encoding of the audio buffer(s).
   * @returns The audio buffer(s) as a TypedArray.
   */
  export function base64ToArray(b64: Chunk[], encoding: string): TypedArray {
    const byteArrays = filterSentinel(b64).map((b) => base64.toByteArray(b));

    const { arrayType: ArrayType, bytesPerElement } =
      ENCODING_MAP[encoding as Encoding];

    const totalLength = byteArrays.reduce(
      (acc, arr) => acc + arr.length / bytesPerElement,
      0,
    );
    const result = new ArrayType(totalLength);

    let offset = 0;
    for (const arr of byteArrays) {
      const floats = new ArrayType(arr.buffer);
      result.set(floats, offset);
      offset += floats.length;
    }

    return result;
  }

  /**
   * Schedule an audio buffer to play at a given time in the passed context.
   *
   * @param floats The audio buffer to play.
   * @param context The audio context to play the buffer in.
   * @param startAt The time to start playing the buffer at.
   * @param sampleRate The sample rate of the audio.
   * @returns A promise that resolves when the audio has finished playing.
   */
  export function playAudioBuffer(
    floats: Float32Array,
    context: AudioContext,
    startAt: number,
    sampleRate: number,
  ) {
    const source = context.createBufferSource();
    const buffer = context.createBuffer(1, floats.length, sampleRate);
    buffer.getChannelData(0).set(floats);
    source.buffer = buffer;
    source.connect(context.destination);
    source.start(startAt);

    return new Promise<void>((resolve) => {
      source.onended = () => {
        resolve();
      };
    });
  }

  /**
   * Unwraps a chunk of audio data from a message event and calls the
   * handler with it if the context ID matches.
   *
   * @param contextId The context ID to listen for.
   * @param handler The handler to call with the chunk of audio data.
   * @returns A message event handler.
   */
  export function createMessageHandlerForContextId(
    contextId: string,
    handler: ({
      chunk,
      message,
    }: {
      chunk?: Chunk;
      message: string;
      data: WebSocketResponse;
    }) => void,
  ) {
    return (event: MessageEvent) => {
      if (typeof event.data !== "string") {
        return; // Ignore non-string messages.
      }
      const message: WebSocketResponse = JSON.parse(event.data);
      if (message.context_id !== contextId) {
        return; // Ignore messages for other contexts.
      }
      let chunk: Chunk | undefined;
      if (message.done) {
        // Convert the done message to a sentinel value.
        chunk = getSentinel();
      } else if (message.type === "chunk") {
        chunk = message.data;
      }
      handler({ chunk, message: event.data, data: message });
    };
  }

  /**
   * Get a sentinel value that indicates the end of a stream.
   * @returns A sentinel value to indicate the end of a stream.
   */
  export function getSentinel(): Sentinel {
    return null;
  }

  /**
   * Check if a chunk is a sentinel value (i.e. null).
   *
   * @param chunk
   * @returns Whether the chunk is a sentinel value.
   */
  export function isSentinel(x: unknown): x is Sentinel {
    return x === getSentinel();
  }

  /**
   * Filter out null values from a collection.
   *
   * @param collection The collection to filter.
   * @returns The collection with null values removed.
   */
  export function filterSentinel<T>(collection: T[]): Exclude<T, Sentinel>[] {
    return collection.filter(
      (x): x is Exclude<T, ReturnType<typeof getSentinel>> => !isSentinel(x),
    );
  }

  /**
   * Check if an array of chunks is complete by testing if the last chunk is a
  sentinel
   * value (i.e. null).
   * @param chunk
   * @returns Whether the array of chunks is complete.
   */
  export function isComplete(chunks: Chunk[]) {
    return isSentinel(chunks[chunks.length - 1]);
  }

  /**
   * Get user-facing emitter callbacks for an Emittery instance.
   * @param emitter The Emittery instance to get callbacks for.
   * @returns User-facing emitter callbacks.
   */
  export function getEmitteryCallbacks<T>(
    emitter: Emittery<T>,
  ): EmitteryCallbacks<T> {
    return {
      on: emitter.on.bind(emitter),
      off: emitter.off.bind(emitter),
      once: emitter.once.bind(emitter),
      events: emitter.events.bind(emitter),
    };
  }
  import Emittery from "emittery";
  import { humanId } from "human-id";
  import { WebSocket as PartySocketWebSocket } from "partysocket";
  import { Client } from "../lib/client";
  import { CARTESIA_VERSION, constructApiUrl } from "../lib/constants";
  import type {
    ConnectionEventData,
    ConnectOptions,
    ContinueRequest,
    EmitteryCallbacks,
    StreamOptions,
    StreamRequest,
    WebSocketOptions,
    WordTimestamps,
  } from "../types";
  import Source from "./source";
  import {
    base64ToArray,
    createMessageHandlerForContextId,
    getEmitteryCallbacks,
    isSentinel,
  } from "./utils";

  export default class WebSocket extends Client {
    socket?: PartySocketWebSocket;
    #isConnected = false;
    #sampleRate: number;
    #container: string;
    #encoding: string;

    /**
     * Create a new WebSocket client.
     *
     * @param args - Arguments to pass to the Client constructor.
     */
    constructor(
      { sampleRate, container, encoding }: WebSocketOptions,
      ...args: ConstructorParameters<typeof Client>
    ) {
      super(...args);

      this.#sampleRate = sampleRate;
      this.#container = container ?? "raw"; // Default to raw audio for
  backwards compatibility.
      this.#encoding = encoding ?? "pcm_f32le"; // Default to 32-bit floating
  point PCM for backwards compatibility.
    }

    /**
     * Send a message over the WebSocket to start a stream.
     *
     * @param inputs - Generation parameters. Defined in the StreamRequest type.
     * @param options - Options for the stream.
     * @param options.timeout - The maximum time to wait for a chunk before
  cancelling the stream.
     *                          If set to `0`, the stream will not time out.
     * @returns A Source object that can be passed to a Player to play the
  audio.
     * @returns An Emittery instance that emits messages from the WebSocket.
     * @returns An abort function that can be called to cancel the stream.
     */
    send(inputs: StreamRequest, { timeout = 0 }: StreamOptions = {}) {
      if (!this.#isConnected) {
        throw new Error("Not connected to WebSocket. Call .connect() first.");
      }

      if (!inputs.context_id) {
        inputs.context_id = this.#generateId();
      }
      if (!inputs.output_format) {
        inputs.output_format = {
          container: this.#container,
          encoding: this.#encoding,
          sample_rate: this.#sampleRate,
        };
      }

      // Send audio request.
      this.socket?.send(
        JSON.stringify({
          ...inputs,
        }),
      );

      const emitter = new Emittery<{
        message: string;
        timestamps: WordTimestamps;
      }>();
      const source = new Source({
        sampleRate: this.#sampleRate,
        encoding: this.#encoding,
        container: this.#container,
      });
```

```
// Used to signal that the stream is complete, either because the
// WebSocket has closed, or because the stream has finished.
const streamCompleteController = new AbortController();
// Set a timeout.
let timeoutId: ReturnType<typeof setTimeout> | null = null;
if (timeout > 0) {
  timeoutId = setTimeout(streamCompleteController.abort, timeout);
}
const handleMessage = createMessageHandlerForContextId(
  inputs.context_id,
  async ({ chunk, message, data }) => {
    emitter.emit("message", message);
    if (data.type === "timestamps") {
      emitter.emit("timestamps", data.word_timestamps);
      return;
    }
    if (isSentinel(chunk)) {
      await source.close();
      streamCompleteController.abort();
      return;
    }
    if (timeoutId) {
      clearTimeout(timeoutId);
      timeoutId = setTimeout(streamCompleteController.abort, timeout);
    }
    if (!chunk) {
      return;
    }
    await source.enqueue(base64ToArray([chunk], this.#encoding));
  },
);
this.socket?.addEventListener("message", handleMessage, {
  signal: streamCompleteController.signal,
});
this.socket?.addEventListener(
  "close",
  () => {
    streamCompleteController.abort();
  },
  {
    once: true,
    signal: streamCompleteController.signal,
  },
);
this.socket?.addEventListener(
  "error",
  () => {
    streamCompleteController.abort();
  },
  {
    once: true,
    signal: streamCompleteController.signal,
  },
);
streamCompleteController.signal.addEventListener("abort", () => {
  source.close();
  if (timeoutId) {
    clearTimeout(timeoutId);
  }
  emitter.clearListeners();
});
```

```
    return {
      source,
      ...getEmitteryCallbacks(emitter),
      stop: streamCompleteController.abort.bind(streamCompleteController),
    };
  }

  /**
   * Continue a stream.
   *
   * @param inputs - Generation parameters. Defined in the StreamRequest
type, but must include a `context_id` field. `continue` is set to true by
default.
   */
  continue(inputs: ContinueRequest) {
    if (!this.#isConnected) {
      throw new Error("Not connected to WebSocket. Call .connect() first.");
    }

    if (!inputs.context_id) {
      throw new Error("context_id is required to continue a context.");
    }
    if (!inputs.output_format) {
      inputs.output_format = {
        container: this.#container,
        encoding: this.#encoding,
        sample_rate: this.#sampleRate,
      };
    }

    // Send continue request.
    this.socket?.send(
      JSON.stringify({
        continue: true,
        ...inputs,
      }),
    );
  }

  /**
   * Generate a unique ID suitable for a streaming context.
   *
   * Not suitable for security purposes or as a primary key, since
   * it lacks the amount of entropy required for those use cases.
   *
   * @returns A unique ID.
   */
  #generateId() {
    return humanId({
      separator: "-",
      capitalize: false,
    });
  }

  /**
   * Authenticate and connect to a Cartesia streaming WebSocket.
   *
   * @returns A promise that resolves when the WebSocket is connected.
   * @throws {Error} If the WebSocket fails to connect.
   */
```

```
async connect(options: ConnectOptions = {}) {
  if (this.#isConnected) {
    throw new Error("WebSocket is already connected.");
  }

  const emitter = new Emittery<ConnectionEventData>();
  this.socket = new PartySocketWebSocket(
    async () => {
      const url = constructApiUrl(this.baseUrl, "/tts/websocket", {
        websocket: true,
      });
      url.searchParams.set("api_key", await this.apiKey());
      url.searchParams.set("cartesia_version", CARTESIA_VERSION);
      return url.toString();
    },
    undefined,
    options,
  );
  this.socket.binaryType = "arraybuffer";

  this.socket.onopen = () => {
    this.#isConnected = true;
    emitter.emit("open");
  };
  this.socket.onclose = () => {
    this.#isConnected = false;
    emitter.emit("close");
  };

  return new Promise<EmitteryCallbacks<ConnectionEventData>>(
    (resolve, reject) => {
      this.socket?.addEventListener(
        "open",
        () => {
          resolve(getEmitteryCallbacks(emitter));
        },
        {
          once: true,
        },
      );

      const aborter = new AbortController();
      this.socket?.addEventListener(
        "error",
        () => {
          aborter.abort();
          reject(new Error("WebSocket failed to connect."));
        },
        {
          signal: aborter.signal,
        },
      );

      this.socket?.addEventListener(
        "close",
        () => {
          aborter.abort();
          reject(new Error("WebSocket closed before it could connect."));
        },
        {
          signal: aborter.signal,
          },
        );
      },
    );
  }

  /**
   * Disconnect from the Cartesia streaming WebSocket.
   */
  disconnect() {
    this.socket?.close();
  }
}
import type Emittery from "emittery";
import type { Options } from "partysocket/ws";

export interface ClientOptions {
  apiKey?: string | (() => Promise<string>);
  baseUrl?: string;
}

export type Sentinel = null;

export type Chunk = string | Sentinel;

export type ConnectionEventData = {
  open: never;
  close: never;
};

export type VoiceSpecifier =
  | {
      mode?: "id";
      id: string;
    }
  | {
      mode?: "embedding";
      embedding: number[];
    };

export type Emotion =
  | "anger"
  | "sadness"
  | "positivity"
  | "curiosity"
  | "surprise";
export type Intensity = "lowest" | "low" | "high" | "highest";
export type EmotionControl = Emotion | `${Emotion}:${Intensity}`;

export type VoiceOptions = VoiceSpecifier & {
  __experimental_controls?: {
    speed?: "slowest" | "slow" | "normal" | "fast" | "fastest" | number;
    emotion?: EmotionControl[];
  };
};

export type StreamRequest = {
  model_id: string;
  transcript: string;
  voice: VoiceOptions;
  output_format?: {
    container: string;
    encoding: string;
    sample_rate: number;
  };
  context_id?: string;
  continue?: boolean;
  duration?: number;
  language?: Language;
  add_timestamps?: boolean;
};

export type BytesRequest = Omit<
  StreamRequest,
  "continue" | "add_timestamps" | "context_id"
>;

export type ContinueRequest = StreamRequest & {
  context_id: string;
};

export type Language =
  | "en"
  | "es"
  | "fr"
  | "de"
  | "ja"
  | "zh"
  | "pt"
  | (string & {});

export type StreamOptions = {
  timeout?: number;
};

export type WebSocketBaseResponse = {
  context_id: string;
  status_code: number;
  done: boolean;
};

export type WordTimestamps = {
  words: string[];
  start: number[];
  end: number[];
};

export type WebSocketTimestampsResponse = WebSocketBaseResponse & {
  type: "timestamps";
  word_timestamps: WordTimestamps;
};

export type WebSocketChunkResponse = WebSocketBaseResponse & {
  type: "chunk";
  data: string;
  step_time: number;
};

export type WebSocketErrorResponse = WebSocketBaseResponse & {
  type: "error";
  error: string;
};
```

```
export type WebSocketResponse =
  | WebSocketTimestampsResponse
  | WebSocketChunkResponse
  | WebSocketErrorResponse;

export type EmitteryCallbacks<T> = {
  on: Emittery<T>["on"];
  off: Emittery<T>["off"];
  once: Emittery<T>["once"];
  events: Emittery<T>["events"];
};

export type CloneOptions =
  | {
      mode: "url";
      link: string;
      enhance?: boolean;
    }
  | {
      mode: "clip";
      clip: Blob;
      enhance?: boolean;
    };

export type VoiceChangerOptions = {
  clip: File;
  voice: { id: string }; // match VoiceSpecifier shape, but only id is
supported for now
  output_format:
    | {
        container: "mp3";
        bit_rate: number;
        sample_rate: number;
      }
    | {
        container: "wav";
        encoding: Encoding;
        sample_rate: number;
        bit_rate: number;
      }
    | {
        container: "raw";
        encoding: Encoding;
        sample_rate: number;
      };
};

export type LocalizeOptions = {
  mode: "embedding";
  embedding: number[];
} & {
  language: Language;
  dialect: string & {};
  original_speaker_gender: "male" | "female" | (string & {});
};

export interface VoiceToMix {
  id?: string;
  embedding?: number[];
  weight: number;
}

export interface MixVoicesOptions {
  voices: VoiceToMix[];
}

export type Voice = {
  id: string;
  name: string;
  description: string;
  embedding: number[];
  is_public: boolean;
  user_id: string;
  created_at: string;
  language: Language;
};

export type CreateVoice = Pick<Voice, "name" | "description" | "embedding"> &
  Partial<Omit<Voice, "name" | "description" | "embedding">>;

export type UpdateVoice = Partial<
  Pick<Voice, "name" | "description" | "embedding">
>;

export type CloneResponse = {
  embedding: number[];
};

export type VoiceChangerBytesResponse = {
  buffer: ArrayBuffer;
};

export type LocalizeResponse = {
  embedding: number[];
};

export type MixVoicesResponse = {
  embedding: number[];
};

export type WebSocketOptions = {
  container?: string;
  encoding?: string;
  sampleRate: number;
};

export type ConnectOptions = Pick<Options, "WebSocket">;

export type SourceEventData = {
  enqueue: never;
  close: never;
  wait: never;
  read: never;
};

export type TypedArray = Float32Array | Int16Array | Uint8Array;

export type Encoding = "pcm_f32le" | "pcm_s16le" | "pcm_alaw" | "pcm_mulaw";
import { Client } from "../lib/client";
import type { VoiceChangerBytesResponse, VoiceChangerOptions } from "../
types";
```

```
export default class VoiceChanger extends Client {
  async bytes(
    options: VoiceChangerOptions,
  ): Promise<VoiceChangerBytesResponse> {
    const formData = new FormData();
    formData.append("clip", options.clip); // TODO: handle Blobs that are not
Files
    formData.append("voice[id]", options.voice.id);

    const fmt = options.output_format;
    formData.append("output_format[container]", fmt.container);
    if ("encoding" in fmt) {
      formData.append("output_format[encoding]", fmt.encoding);
    }
    if ("bit_rate" in fmt) {
      formData.append("output_format[bit_rate]", fmt.bit_rate.toString());
    }
    if ("sample_rate" in fmt) {
      formData.append("output_format[sample_rate]",
fmt.sample_rate.toString());
    }

    const response = await this._fetch("/voice-changer/bytes", {
      method: "POST",
      body: formData,
    });

    if (!response.ok) {
      throw new Error(
        `Voice changer error! status: ${
          response.status
        }, message: ${await response.text()}`,
      );
    }

    return { buffer: await response.arrayBuffer() };
  }
}
import { Client } from "../lib/client";
import type {
  CloneOptions,
  CloneResponse,
  CreateVoice,
  LocalizeOptions,
  LocalizeResponse,
  MixVoicesOptions,
  MixVoicesResponse,
  UpdateVoice,
  Voice,
} from "../types";

export default class Voices extends Client {
  async list(): Promise<Voice[]> {
    const response = await this._fetch("/voices");
    return response.json();
  }

  async get(voiceId: string): Promise<Voice> {
    const response = await this._fetch(`/voices/${voiceId}`);
    return response.json();
  }

  async create(voice: CreateVoice): Promise<Voice> {
    const response = await this._fetch("/voices", {
      method: "POST",
      body: JSON.stringify(voice),
    });
    return response.json() as Promise<Voice>;
  }

  async update(id: string, voice: UpdateVoice): Promise<Voice> {
    const response = await this._fetch(`/voices/${id}`, {
      method: "PATCH",
      body: JSON.stringify(voice),
    });
    return response.json() as Promise<Voice>;
  }

  async clone(options: CloneOptions): Promise<CloneResponse> {
    if (options.mode === "clip") {
      const formData = new FormData();
      formData.append("clip", options.clip);
      if (options.enhance !== undefined) {
        formData.append("enhance", options.enhance.toString());
      }

      const response = await this._fetch("/voices/clone/clip", {
        method: "POST",
        body: formData,
      });
      return response.json();
    }

    throw new Error("Invalid mode for clone()");
  }

  async mix(options: MixVoicesOptions): Promise<MixVoicesResponse> {
    const response = await this._fetch("/voices/mix", {
      method: "POST",
      body: JSON.stringify(options),
    });

    return response.json() as Promise<MixVoicesResponse>;
  }

  async localize(options: LocalizeOptions): Promise<LocalizeResponse> {
    const response = await this._fetch("/voices/localize", {
      method: "POST",
      body: JSON.stringify(options),
    });

    return response.json() as Promise<LocalizeResponse>;
  }
}
{
  "$schema": "https://json.schemastore.org/tsconfig",
  "compilerOptions": {
    "lib": ["ES2015", "DOM"],
    "module": "ESNext",
    "jsx": "react-jsx",
    "composite": false,
    "declaration": true,
    "declarationMap": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
    "allowImportingTsExtensions": true,
    "noEmit": true,
    "inlineSources": false,
    "isolatedModules": true,
    "moduleResolution": "Bundler",
    "noUnusedLocals": false,
    "noUnusedParameters": false,
    "preserveWatchOutput": true,
    "skipLibCheck": true,
    "strict": true,
    "strictNullChecks": true,
    "target": "ES6"
  },
  "exclude": ["node_modules"]
}
```