TypeScript random maps for Age of Empires III

What is this?

This is an .XS/.XTS - limited TypeScript language - code transpiler. It allows you to write random maps using the advantages of TypeScript: syntax highlighting, autocomplete, code formatting and IntelliSense, as well as RM functions documentation and terrain enums. It features a converter for the scripts both ways, defining XML map info as a `const` straight from the script, type checking for RM functions/constants and JSDoc documentation for them.

How do I use it?

Installation & Configuration

1. Install Node.js. Chances are, if you want to use TypeScript you already have it.
2. Unpack the .ZIP and, if you like, move the contents of the included `workdir` to your desired location.
    - For example, if you're working on a mod you can move them straight to your game's random maps folder (i. e. *`Age of .../RM3`*). Note that this can change the game's CRC, rendering it incompatible for multiplayer, so you should only do this temporarily, for production.
    - If you're making TAD custom maps, you could move them to the *`Documents/My Games/Age of .../RM3`*.
    - Or you could just leave it as is and only change the `outDir` in the config file. Just make sure that the `sourceDir` has a `lib` folder as a "sibling" for the TypeScript imports to work properly.
3. Edit the `config.json` accordingly. Here are the options:
    - `sourceDir` - your .XTS map scripts folder. TypeScript code will be saved here after the conversion. Has to be a valid, existing path with the included `lib` folder as its "sibling".
    - `outDir` - your resulting .XS maps folder. Transpiled maps and their .XML info files will be saved here. Has to be a valid, existing path.
    - `ignoreWarnings` - do not show non-critical errors and warnings.
    - `ignoreErrors` - suppress even the critical errors, such as .XS syntax errors
    - `forceOpen` - if a file can't be found, look for it in the `sourceDir` and `outDir` folders.

4. To use the potential of XTS, use an IDE such as Visual Studio Code for syntax highlighting and autocomplete. If you *are* using VS Code, set .XTS file extensions to TypeScript language by default. Alternatively, select the language manually in the bottom right.

Converting to .XTS

In the command line, write `node [path to index.js] [your map name(s)]`.

▼ Examples (click to expand)

- With the command prompt in the XScript dir, convert the Great Plains from game's RM3 folder:

```
C:\Users\Jeremy\Saved Games\Tools\XScript> node index.js 'C:\Users\Jeremy\Saved Games\Age of Empires III\RM3\great plains.xs'
```

- Having moved the command prompt to the custom maps folder, convert two maps - My New Map and UnknownEnh:

```
C:\Users\Jeremy\Saved Games\Tools\XScript> cd 'C:\Users\Jeremy\Documents\My Games\Age of Empires III\RM3'
C:\Users\Jeremy\Documents\My Games\Age of Empires III\RM3> node 'C:\Users\Jeremy\Saved Games\Tools\XScript\index.js' 'My New Map.xs' 'unknownenh.xs'
```

- With the command prompt in the XScript dir, convert an included `mercenaries.xs` script:

```
C:\Users\Jeremy\Saved Games\Tools\XScript> node index.js 'C:\Users\Jeremy\Saved Games\Age of Empires III\RM3\mercenaries.xs'
```

The transpiler will generate .XTS files in TypeScript and save them to the specified `sourceDir`. It will also run through the dependencies, such as `mercenaries.xs`, trying to convert them as well if necessary (they are saved to the `lib\includes` folder), and look for the map's .XML data (with the name, description etc) to write this data in the beginning of the resulting .XTS.

Making your map

If you're unfamiliar with TypeScript, read the basics here.

Here's the brief structure of a valid .XTS script:

▼ .XTS example (click to expand)

```
/**
 * ------------------------------------------------------
 * THIS SCRIPT WAS GENERATED FROM AN .XS SCRIPT
 * Original script: **`**../great plains.xs`
 * ------------------------------------------------------
 */

import {int, float, double, vector, [...]} from "../lib/Types";
import {River, Ocean, Lake, Cliff, [...]} from "../lib/Terrain";
[...]

/**
 * XML map info. Don't use in the code, it will only be used
 * to generate the corresponding XML file.
 */
const mapinfo = {
    details: 28599,
    imagepath: "ui/random_map/great_plains",
    [...]
};

function min(a: float=-1.0, b: float=-1.0): float { if(a<=b) { return(a); } return(b); }

function max(a: float=-1.0, b: float=-1.0): float { if(a>=b) { return(a); } return(b); }

// GREAT PLAINS
import {chooseMercs} from "../lib/includes/mercenaries";
[...]

/**
 * Main entry point for random map script
 */
function main(): void {

    rmSetStatusText("", 0.01);

    // Chooses which natives appear on the map
    var subCiv0: int = -1;
    [...]

    // Choose which variation to use.  1=southeast trade route, 2=northwest trade route
```

```
    var whichMap: int = rmRandInt(1, 2);
    var map_type: string = "greatPlains";

    if (rmAllocateSubCivs(6) == true) {
        subCiv0 = rmGetCivID("Comanche");
        rmEchoInfo("subCiv0 is Comanche " + subCiv0);
        if (subCiv0 >= 0)
            rmSetSubCiv(0, "Comanche");
        [...]
    }

    // Picks the map size
    var playerTiles: int = 11000;
    if (cNumberNonGaiaPlayers > 4)
        playerTiles = 10000;
    if (cNumberNonGaiaPlayers > 6)
        playerTiles = 8500;

    var size: int = 2.0 * sqrt(cNumberNonGaiaPlayers * playerTiles);
    rmEchoInfo("Map size=" + size + "m x " + size + "m");
    rmSetMapSize(size, size);

    rmSetMapType(map_type as MapType);
    rmSetMapType("land");

    [...]
}
```

You can see imports at the top - they contain RM functions and terrain enums for the JSDoc and syntax highlighting. You don't have to change these.

After that, there's the `const mapinfo` which describes the map's name, icon etc. You can edit these directly here - this info will be included into the resulting .XML file. You may remove it if you want to edit the .XML manually.

From this point on, your actual random map script starts.

Then there are two functions declared. Often you don't need user-defined functions, but nothing stops you from using them, just follow these rules:

- Explicitly define the parameters' type and default value.
- Explicitly define the function's return type (including `void`).

- Rather than arrow functions or `const` notation, use the standard

```
function name(param: int = 0): int { ... }
```

Most of the following script doesn't need much explanation. Keep these .XTS limitations in mind:

- Explicitly define any variable's type and initial value - as you would with the .XS script.
- Instead of the TypeScrpit's `number`, use `int`, `float`, `double` or `long` types.
- `var` variable declaration is preferred. You may use `let` or `const`, but they will all be transpiled to the .XS's `var`-like (function scope) variable declarations, without runtime `const` behavior.
- Keep in mind that all you write should have an XS equivalent. Don't use advanced TypeScript stuff such as:
  - arrays, objects, classes
  - string methods (such as `**.replace`), string templates
  - `for ... of`, `for ... in`
  - `yield`, `async`, etc
  - `Math`, `Object`, etc
  - `===`, `!==`, `![...]`
  - `undefined`, `null`, `NaN`
  - `console.log()`, etc
  - `declare`, `export`, `require`, etc
  - custom types/interfaces
  - composite types, like `[type1]|[type2]`
- Always write `for` cycles with the following notation:

```
for (var i = 0; i < num; i++) { ... }
```

so as to allow conversion into the .XS equivalent:

```
for (i = 0; < num)
```

XTS is **not** just TypeScript – XS is a very simple language, so a very limited subset of TypeScript functionality is available in XTS.

Sometimes using `as` is necessary - you can notice in the example above that `as` is supported, just don't combine types like `[type1][type2]`:

```
rmSetMapType(map_type as MapType);
```

Transpiling to XS

Again, just run the following command: `node [path to index.js] [your map name(s)]`.

Example:

- With the command prompt in the XScript dir, transpile the Great Plains from the specified `sourceDir` and My New Map from the desktop:

```
C:\Users\Jeremy\Saved Games\Tools\XScript> node index.js 'great plains.xts' 'C:\Users\Jeremy\Desktop\My New Map.xts'
```

If your code is alright, this command will strip it of the unnecessary bits and transpile to an XS script. If the `const mapinfo` is present, it will read it to generate an XML map data file, too. Both will be saved to the specified `outDir`, possibly replacing an older file.

FAQ

Why do I get "Cannot find name" errors?

Most likely an RM function has not been declared in the `lib` files. It is difficult to find all the RM functions and documentation on them, so it is very likely that I overlooked some. If this is the case, please, contact me so that I can fix this. Meanwhile, you can add the declaration for this function to the corresponding `.d.ts` file in the `lib` folder and manually add the import in the beginning of your .XTS script.

Why do I get type errors?

Some RM functions are hardcoded to accept a limited set of possible argument values. For example,

```
rmSetAreaMix(areaID: int, mixName: Mix): void
```

as you see, only accepts a `Mix`-type second argument (such as `bayou_grass`) rather than any strings. Thus, any variable you want to pass as an argument either has to be of the corresponding type or has to be forced to this type like this:

▼ Types example (click to expand)

```
rmSetAreaMix(areaID, "bayou_grass");           // Works
var proper_mix: Mix = "bayou_grass";
rmSetAreaMix(areaID, proper_mix);              // Works
var another_mix: string = "bayou_forest";
rmSetAreaMix(areaID, another_mix as Mix);      // Works, but not recommended
var non_existent_mix: string = "no such mix actually exists";
rmSetAreaMix(areaID, non_existent_mix);        // Doesn't work
rmSetAreaMix(areaID, non_existent_mix as Mix); // Works, but only do this if you're absolutely sure
```

Here are all the types for which this is the case: `River, Ocean, Lake, Water (=River|Ocean|Lake), Cliff, MapType, Mix, Forest`

Got it, but I'm writing maps for a mod that has new terrain.

Then you can either use `as` or, to get proper autocomplete, edit the `lib/Terrain.d.ts`, adding your new terrain to the corresponding types. In the future I plan to implement some sort of extensions functionality to swap enums and RM functions based on the mod you're scripting for.

What are these new terrain enums?

Apart from the usual TAD terrain enums, you could have noticed new ones, such as a `african_prairie` mix, or `Korea Coast` ocean, etc. These are from my The Native Empires mod, and you should just ignore them. In the future I plan to implement some sort of extensions functionality and move these TNE-specific enums to a separate file.

Will this work with DE?

There are no guarantees, but provided DE's RMS structure stays the same, it should work.

Why can't the converter save the files?

Make sure the paths specified in your `config.json` exist and comply with the location of your `workdir`.

Does the resulting XS keep comments?

Yes, they are kept intact (unless there are some bugs).

Can I simultaneously convert .XS and .XTS maps?

Yes, for example, you can write something like

```
C:\Users\Jeremy\Documents\My Games\Age of Empires III\RM3> node 'C:\Users\Jeremy\Saved Games\Tools\XScript\index.js' 'unknownenh.xs' 'unknownenh.xts'
```

to first convert to XTS and then immediately back to XS. The reason you may want this is to tidy up an existing map with the included *Prettier* code formatter.

I've found a bug. How can I report it?

Please contact me on Discord, via e-mail or on moddb.