ORTS File Checker

A command-line tool to check content for ORTS

1. Introduction

This document describes a tool for ORTS that checks whether files needed for ORTS can be loaded properly or not. This loading is done stand-alone in the sense that it is not needed to run the simulator. In this way you can check files to make sure they will not pose problems in ORTS or perhaps are simply neglected.

2. Usage

Note that since this is a console application all output goes to the console window. It can be re-directed to a file. In case the output is very long, the application does try to increase the buffer size of the console window such that no results are lost.

2.1. Single file usage

This is the basic usage mode:

ContentChecker.exe <file>

example:

ContentChecker.exe marias.tdb

The output will normally just be a line with the file that is being loaded and a line that says ‘OK’. It is possible that some warnings are shown as well. But if the file can at least be loaded, you will get an ‘OK’. In case the file cannot be loaded and an exception is raised in the code, the exception message will be shown instead

2.2. Multiple file usage

In many cases you want to check multiple files:

ContentChecker.exe <files>

examples:

ContentChecker.exe marias.tdb marias.rdb

ContentChecker.exe *.pat

ContentChecker.exe *.sms *.wav

ContentChecker.exe *

The files can either be actual files or they can contain the search character ‘*’ indicating multiple files. Do note that in case a ‘*’ is used also sub-directories will be searched. So calling ContentChecker.exe *.ace will find all *.ace files in or under the current directory.

When multiple files are loaded you will first get the same result as loading all files individually. But at the end there will also be a summary showing the amount of files loaded, the amount of files skipped (e.g. because the file is recognized but not used like a _n.raw file or because the file cannot be loaded independently), the amount of files not recognized, as well as a count of all errors, warnings and informations.

2.3. Help

The option /h or /help will show simple help

2.4. Verbose

The option /v or /verbose will not only show errors and warnings, but also informational trace output.

Possibly the advanced cross-checking (see below) should actually partially be implemented using these kind of messages. That needs to be seen.

2.5. Hierarchical loading

(partially implemented)

Obviously all the files that are used are not really independent. There are various relations between the files:

- Many files can be loaded independently
- A number of files cannot be loaded independently at all: they need information from another

file. For instance

  - signal script files cannot be loaded without information the signal config file
  - tile .raw files cannot be loaded without information from the .t terrain file

- Many files refer to other files. Although some or perhaps even all of these referred-to files can be loaded independently, it is nice to have an option where not only a file but also all directly referred files are loaded. For instance
  - .act files refer directly to .srv and .pat files
  - Signal config files refer directly to signal script files
  - the .trk file refers directly to a .ace and .sms file used during loading
- And then there are files that are loaded not directly, but only when needed in the simulator. The obvious example here are the worldfiles.

For all these relations we have separate options:

- Without any additional options we only load directly loadable files
- With the option /d or /dependent we also dependent files
- With the option /r or /referenced we also load directly referenced files
- With the option /a or /all we load all files that might be used during simulation

For clarity, only when no additional option is given will the files be loaded in alphabetical order. With any of the options given, additional files will be loaded as soon as they are identified. This is a depth-first way of loading the files which should keep related files as close as possible in the list.

Some files might be loaded referred to multiple times (e.g., from different activities or from different world files). These files will be loaded only once.

Examples:

ContentChecker.exe *.t /d

This will load not only the .t files, but also the dependent _y.raw and _f.raw files

ContentChecker.exe sigcfg.dat /d

This will also load possible signal scripts

ContentChecker.exe *.dat /d

This will load all .dat files. In case it finds a signal script file (which is not independently loadable and not even recognized as a script file) it will defer loading that file until a recognized signal config file is parsed and refers to that signal script file.

ContentChecker.exe eastbound.act /r

this will also load the referenced .srv and .pat files

ContentChecker.exe *.w /r

This will also load various .s files, hazards, …

ContentChecker.exe *.trk /a

This will load all relevant files belonging to that route, as well as a few engines and wagons

2.6. Advanced cross-checking

This is a more advanced feature that will be implemented later. This describes a little bit the vision of where it needs to go.

Additional option: /c or /crosscheck

In the future, this option is used to do more extensive cross-checking between files (e.g. checking that locations defined in a .pat file are in fact present in the .tdb, …).

2.7. Starting from windows explorer

Primarily this tool is a command-line tool. However, windows explorer also allows you to open a file with an application of choice. For this tool that does not make sense: the output will not be saved anywhere and the output window will be visible only until the tool is finished, which is way too short.

3. File recognition

Files that are loaded directly (so not from another file) will be recognized mainly by their extension.

If the extension is not unique (especially .dat files), the SIMIS "JINX" header at the start identifies the file type; e.g. "JINX0G0t" is the signal configuration (sigcfg.dat), "JINX0g0t" is the gantry file (gantry.dat), and "JINX0v1t" is a car spawner file (carspawn.dat). However, the same header is sometimes the same for multiple file types; e.g. "JINX0T0t" is used for the four files containing the track and road databases and item tables. As a last resort hard-coded names are used.

4. File relations

Below we describe the relations between different files. This is more of a documentation of the implementation than a description of how it should be, at least for now.

4.1. sigcfg.dat signal configuration

4.1.1. Dependent files

All scriptfiles (e.g. sigscr.dat) that are defined in the sigcfg.dat file need the information from the signal config file and are therefore dependent.

4.2. .sms sound management

4.2.1. Referenced files

The .sms files directly reference one or more .wav files. These .wav files will be search in either the same directory as the .sms file (possibly via a relative path) or in the <route>/SOUND or <install_dir>/SOUND directories. Note that it might be a bit tricky to find these latter directories from only the full name of the .sms file.

4.3. .t terrain tiles

4.3.1. Dependent files

The _y.raw and if present the _f.raw (flags) files need to know the samplecount that is defined in the .t file and are therefore dependent on the .t file.

4.4. .tdb track database

4.4.1. Dependent files

The world-sound files (.ws) need the track database that is defined in the .tdb files, and therefore the .ws files depend on the .tdb file.

4.5. .trk main route entry

4.5.1. Referenced files

- Loading screen and thumbnail .ace files
- 6 types of default .sms files
- 12 .env environment files.

4.5.2. All files

Note all of the files mentioned below might subsequently load dependent or referenced files.

- The global/tsection.dat track sections
- The track (.tdb) and road (.rdb) databases.
- The sigcfg.dat, possibly in the openrails directory.
- All activity (.act) files in the Activities directory.
- All .timetable_or, .timetable-or, .timetablelist-or and .timetablelist-or files in the Activities/Openrails directory.
- All tiles (.t) files in the Tiles and Lo_Tiles directories
- All .pat files in the Paths directory. Note that some of these might already be loaded via the activities.
- All world (.w) files in the World directory.

4.6. Tsection.dat track sections

4.6.1. Dependent files

This one is a bit complicated. There is a global/tsection.dat file. This file can be loaded independently.

There seems to be a route-specific openrails/tsection.dat that possibly can be loaded independently as well. However, due to a lack of an example the implementation might not be good yet.

But the route-specific MSTS tsection.dat files is an addition to the global/tsection.dat file and therefore depends on the global one. Unfortunately, the global one does not know by itself which route is needed.

This leads to the following implementation

- The <route>/tsection.dat cannot be loaded independently. Currently as a result also the <route>/openrails/tsection.dat will not be loaded independently.
- The global/tsection.dat will be loaded if so requested. But only if it is loaded via the .trk file (using the /all switch) the loader will know about the route directory and only

then the <route>/tsection.dat will be loaded.

5. Limitations and known issues

- Not all referenced files are implemented yet
- Due to lack of examples many openrails-specific files might not yet be loaded or recognized properly.
- Currently only .wav files are handled as referenced files in the .sms files. Possibly this should be extended.