Ceylan's HOWTOs

HOW-TO

Contact: about (dash) howtos (at) esperide (dot) com

Creation date: Wednesday, November 17, 2021

Lastly updated: Sunday, November 21, 2021

Version: 0.0.1

Status: In progress

Dedication: Users of these HOWTOs

Abstract: The role of these `HOW-TOs` is, akin to a cookbook, to share a collection of (technical) recipes ("how-to do this task?) regarding various topics.

These elements are part of the Ceylan umbrella project.

The latest version of this documentation is to be found at the official Ceylan-HOWTOs website (`http://howtos.esperide.org`).

Table of Contents

Build Tools

Organisation: Copyright (C) 2021-2021 Olivier Boudeville
Contact: about (dash) howtos (at) esperide (dot) com
Creation date: Saturday, November 20, 2021
Lastly updated: Sunday, November 21, 2021

Table of Contents

Purpose of Build Tools

A build tool allows to automate all kinds of tasks, by **applying rules and tracking task dependencies**: not only compiling, linking, etc. applications, but also testing them, performing checks on them, generating their documentation, etc.

Choice

Often build tools are tied to some programming languages (ex: Maven for Java, Rebar3 for Erlang, etc.).
Some tools are more generic by nature, like late GNU autotools, or Cmake or GNU make.
For most uses, our personal preference goes to the latter. Notably all our Erlang-based developments, starting from Ceylan-Myriad, are based on GNU make.

GNU make

We recommend the reading of this source for reference purpose, especially the section about The Two Flavors of Variables.
Taking our Erlang developments as an example, its first layer, Ceylan-Myriad, relies on build facilities that are also reused and further adapted/specialised/parametrised by the upper layers (ex: Ceylan-WOOPER).
For that Myriad defined three top-level make files:

- base build-related variables in GNUmakevars.inc, providing defaults that can be overridden by upper layers
- automatic rules, in GNUmakerules-automatic.inc, which operate typically depending on file extensions
- explicit rules, in GNUmakerules-explicit.inc, for named make targets (ex: `all`, `clean`)

Each layer gathers these three (and the ones of all layers below) in its GNUmakesettings.inc file, the only element that the per-directory `GNUmakefile` files have to include.
Such a system allows defining settings and rules only once, while remaining flexible and enabling individual makefiles to be minimalistic: beside said include, they just have to list their subdirectories the build should traverse (example).

See Also

The Ceylan-HOWTOs main page.

3D HOWTO

As usual, these information relate to a GNU/Linux perspective.

Cross-Platform Game Engines

Godot Godot is our personal favorite engine.

Installation On Arch Linux: `pacman -Sy godot`.

Unreal Engine

Installation

Unity3D Unity may be installed at least in order to access its asset store, knowing that apparently an asset purchased in this store may be used with any game engine of choice.
Indeed, for the standard licence, it is stipulated in the EULA legal terms that: *Licensor grants to the END-USER a non-exclusive, worldwide, and perpetual license to the Asset to integrate Assets only as incorporated and embedded components of electronic games and interactive media and distribute such electronic game and interactive media.*
So, in legal terms, an asset could be bought in the Unity Asset Store and used in Godot, for example - provided that its content can be used there without too much effort/constraint.

Installation Unity shall now be obtained thanks to the Unity Hub.
On Arch Linux it is available through the AUR, as an AppImage; one may thus use: `yay -Sy unityhub`.
A Unity account will be needed, then a licence, then a Unity release will have to be added in order to have it downloaded and installed for good, covering the selected target platforms (ex: Linux and Windows "Build Supports").
Additional information: Unity3D on Arch.

Configuration It does not seem easy to configure Unity so that its interface (mouse, keyboard bindings) behave like, for example, the one of Blender.

Assets They can be downloaded through the `Window -> Package Manager -> Packages:My Assets` menu option.
To access the assets provided by such packages, of course the simplest approach is to use the Unity editor; this is done by creating a project (ex: `MyProject`), selecting the aforementioned menu option (just above), and then clicking on `Import` and selecting the relevant content that will end up in clear form in your project, in `MyProject/Assets/CorrespondingAssetProvider/AssetName`.

Yet such Unity packages, once downloaded (whether or not they have been imported in projects afterwards) are files stored typically in the `~/.local/share/unity3d/Asset Store-5.x` directory, whose extension is `.unitypackage`.
Such files are actually `.tar.gz` archives, and thus their content can be listed thanks to:

```
$ tar tvzf Foobar.unitypackage
```

Inside such archives, each individual package resource is located in a directory whose name is probably akin to the checksum of this resource (ex: `167e85f3d750117459ff6199b79166fd`)[1]; such directory generally contains at least 3 files:

- `asset`: the resource itself, renamed to that unique name yet containing its exact original content (ex: the one of a Targa image)
- `asset.meta`: the metadata about that asset (file format, identifier, timestamp, type-specific settings, etc.)
- `pathname`: the path of that asset in the package "virtual" tree (ex: `Assets/Foo/Textures/baz.tga`)

When applicable, a `preview.png` file may also exist.
Some types of content are Unity-specific and thus may not transpose (at least directly) to another game engine. This is the case for example for materials or prefabs (whose file format is relatively simple, based on YAML 1.1).
Tools like AssetStudio (probably Windows-only) strive to automate most of the process of exploring, extracting and exporting assets.
Meshes are typically in the FBX (proprietary) file format, that can nevertheless be imported in Blender; one may use for that our blender-import.sh script.

3D Data

File Formats They are designed to store 3D content (scenes, nodes, vertices, normals, meshes, textures, materials, animations, skins, cameras, lights, etc.).
We prefer to rely on the open, well-specified, modern glTF 2.0 format in order to perform import/export operations.
It comes in two forms:

- `*.gltf` when JSON-based, possibly embedding the actual data (vertices, normals, textures, etc.) as ASCII base64-encoded content
- `*.glb` when binary

See also the glTF 2.0 quick reference guide.
The second best choice we see is Collada (`*.dae` files), an XML-based counterpart (open as well) to glTF.
Often, assets can be found as FBX of OBJ files and thus may have to be converted (typically to glTF), which is never a riskless task.

[1] Yet no checksum tool among `md5sum`, `sha1sum`, `sha256sum`, `sha512sum`, `shasum`, `sha224sum`, `sha384sum` seems to correspond; it must a be a different, possibly custom, checksum.

Samples For:

- glTF, notably glTF 2.0; also a simple cube
- DAE; also a simple cube
- FBX

Modelling Software

Blender Blender is a very powerful open-source 3D toolset.

Wings3D

OpenGL Corner

Hints OpenGL allows the main program running on the CPU to communicate with typically a graphic card. As such most of the operations are asynchronous: through OpenGL they are triggered by the program but they return whereas not performed yet; they have just be queued. Indeed OpenGL implementations are almost always pipelined, the rendering must be thought as taking place in a background process.

Information

- FAQ for OpenGL and GLUT
- OpenGL Performance

[[_TOC_]]

Knowing that in functional languages such as Erlang terms are immutable, why could not they be shared between local processes when sent through messages, instead of being copied in the heap of each of them, as it is the case in the Erlang VM?

Because, beyond of the constness of these terms, their life-cycle has also to be managed. If they are copied, each process can very easily perform its (concurrent, autonomous) garbage collections. If terms were shared, then reference counting would be needed, which is bound to require locks in a concurrent context.

A trade-off between memory (due to data duplication) and processing (due to lock contention) has to be found and at least for most terms (expect larger binaries), the sweet spot shall be sacrificing a bit of memory in favour of CPU load.

> The language that's part of massive systems that underpin pretty much the entire internet?*

And now a bit of Shameless Advertisement for Erlang

What makes Elixir StackOverflow's #4 most-loved language? What makes Erlang and Elixir StackOverflow's #3 and #4 best-paid languages? How did WhatsApp scale to billions of users with just dozens of Erlang engineers? What's so special about Erlang that it powers CouchDB and RabbitMQ? Why

are multi-billion-dollar corporations like Bet365 and Klarna built on Erlang? Why do PepsiCo, Cars.com, Change.org, Boston's MBTA, and Discord all rely on Elixir? Why was Elixir chosen to power a bank? Why does Cisco ship 2 million Erlang devices each year? Why is Erlang used to control 90% of Internet traffic?
Source : https://erlangforums.com/t/erlang-101-processes-parallelization/594
Resources:

- The Erlang Runtime System, a.k.a. "the BEAM book", by Erik Stenman

Installation
Nous recommandons fortement l'installation depuis les sources de la dernière version stable d'Erlang telle que réalisée par notre script `conf/install-erlang.sh`, présent sous Ceylan-Myriad (et d'emblée intégré dans la distribution Sim-Diasca, sous le répertoire `myriad` présent à sa racine, cf. [la page d'installation de Sim-Diasca](https://gitlab.pleiade.edf.fr/Capitalisation/espace-principal/-/wikis/Usage-de-Sim-Diasca#installation)), qu'il est donc préférable de cloner auparavant.
Pour une activité de développement, il est préférable d'ajouter les options suivantes pour l'exécution de ce script d'installation :

- `--doc-install` afin de disposer de la documentation de référence en local (sous `~/Software/Erlang/Erlang-current-documentation/` ; à placer en signet dans son navigateur préféré)
- `--generate-plt` afin de générer un fichier PLT permettant la vérification statique de types (cela entraîne après la phase de compilation une phase relativement longue et exigeante en ressources, mais ces opérations ne sont à faire que lors de l'installation d'Erlang, une fois pour toutes pour une version donnée)

En pratique, il faut :

- soit permettre le téléchargement des sources et de la documentation via le proxy EDF (par exemple via [l'alias set-proxy](https://gitlab.pleiade.edf.fr/Capitalisation/espace-principal/-/wikis/Calibre-&-compagnie#proxy)) puis exécuter : *install-erlang.sh --doc-install --generate-plt*
- soit (solution recommandée, car la plus fiable) les télécharger directement depuis son navigateur, depuis [cette page](https://www.erlang.org/downloads) (télécharger `OTP x.y Source File` et `OTP x.y HTML Documentation File`) et placer les deux archives correspondantes dans le même répertoire que `install-erlang.sh`, puis exécuter *./install-erlang.sh --doc-install --generate-plt --no-download*

Exécuter `./install-erlang.sh --help` pour plus d'informations sur les modalités d'installation d'Erlang.
Vérification de bon fonctionnement
Après avoir vérifié que *~/Software/Erlang/Erlang-current-install/bin/* se trouve bien dans votre PATH (cela devrait être le cas si vous avez suivi [cette

procédure](https://gitlab.pleiade.edf.fr/Capitalisation/espace-principal/-/wikis/Configuration-du-shell)), il suffit d'exécuter (depuis un répertoire quelconque) `erl` (l'interpréteur Erlang), résultant en une invite du type :
“‘ $ erl Erlang/OTP 23 [erts-11.0] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [hipe]
Eshell V11.0 (abort with ^G) 1> “‘
Entrer alors deux fois `CTRL-C` pour revenir au shell (UNIX).
Bravo, vous disposez d'Erlang maintenant !
Ressources mobilisables
Consulter [cette page](https://web.pleiade.edf.fr/simdiasca/index.php/GunsForHire).
Sources d'information

- la référence est le [site officiel d'Erlang](http://erlang.org), y compris pour :
  - la [page d'accueil de la documentation](https://erlang.org/docs/)
  - la [liste des modules standard](https://erlang.org/doc/search/), qui est la page la plus utile au quotidien ; il est donc préférable de placer dans votre navigateur un signet sur sa version locale : `~/Software/Erlang/Erlang-current-docu` c'est-à-dire par exemple `file:///home/E22850/Software/Erlang/Erlang-current-documentat`
- pour se référer à la documentation : [Erldocs](https://erldocs.com/) fournit aussi une vue complète
- pour apprendre : nous conseillons la lecture du tutoriel [Learn You Some Erlang for Great Good!](https://learnyousomeerlang.com/content) ; à noter qu'EDF dispose aussi de quelques livres sur Erlang (bibliothèque R&D)

Voir aussi [cette page d'aide](https://web.pleiade.edf.fr/simdiasca/index.php/UpToSpeedWithErlang).

Please React!

If you have information more detailed or more recent than those presented in this document, if you noticed errors, neglects or points insufficiently discussed, drop us a line! (for that, use the contact address at the top of this document).

Ending Word

Hoping that these Ceylan-HOWTOs may be of help!

HOW-TO