

Erik Bartmann

Modulare Synthesizer mit

VCV-Rack 2

entdecken



Die Installation der
Entwicklungsumgebung für *Vult*

Inhaltsverzeichnis

Die Vult-Programmierung.....	2
Installationschritte für Linux.....	2
Vult-Compiler installieren.....	2
Rack-SDK installieren.....	2
Pfad zum Rack-SDK im Terminal festlegen.....	2
Einrichten der Entwicklungsumgebung.....	3
Vult-Template herunterladen.....	3
Der Build-Prozess.....	4
Die Überprüfung der Kompilierung.....	5
Das neue RackPlayground-Plugin in VCV-Rack 2.....	7
Installationschritte für Windows.....	8
Rack-SDK installieren.....	8
MSYS2 installieren.....	8
Vult-Compiler installieren.....	10
Vult-Template herunterladen.....	10
Der Build-Prozess.....	12
Die Überprüfung der Kompilierung.....	13
Das neue RackPlayground-Plugin in VCV-Rack 2.....	15
Abschließend.....	16
Weitere Informationen.....	16
Erik Bartmann.....	16
Das VCV-Rack.....	16
Vult.....	16

Autor	Erik Bartmann
Internet	https://erik-bartmann.de/
Thema	Einrichtung der Entwicklungsumgebung zur Programmierung von Plugins für das VCV-Rack 2 mithilfe der Vult-Programmiersprache
Version	1.01
Datum	19. Januar 2022

© 2022 by *Erik Bartmann*. All rights reserved.

Dieses Tutorial darf unangepasst frei kopiert, elektronisch verbreitet und für den persönlichen Gebrauch ausgedruckt werden.

Die Vult-Programmierung

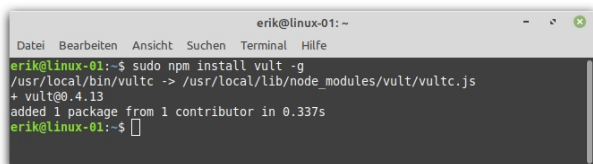
In diesem Tutorial geht es darum, eine Entwicklungsumgebung sowohl unter Linux als auch unter Windows für das VCV-Rack 2 einzurichten. Es werden alle erforderlichen Schritte genannt, damit im Anschluss über die Vult-Programmiersprache ein einfaches Entwickeln unterschiedlichster Plugins für das VCV-Rack 2 erreicht werden kann.

Installationschritte für Linux

Vult-Compiler installieren

Voraussetzungen:

- Installation von node.js
- Installation von npm (\$ sudo apt install npm)



```
erik@linux-01:~$ sudo npm install vult -g
/usr/local/bin/vultc -> /usr/local/lib/node_modules/vult/vultc.js
+ vult@0.4.13
added 1 package from 1 contributor in 0.337s
erik@linux-01:~$
```

Abbildung 1: Vult-Compiler installieren

Rack-SDK installieren

Unter der folgenden Internetadresse sind unter anderem alle Versionen des Rack-SDKs zu finden.

<https://vcvrack.com/downloads/>

Die letzte Version des Rack-SDK für Linux herunterladen.

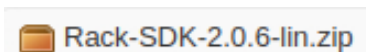


Abbildung 2: Rack-SDK

Entpacken und installieren. Den Pfad zum Rack-SDK notieren!

Pfad zum Rack-SDK im Terminal festlegen

Der Pfad zum Rack-SDK muss nun zum Beispiel in die ~/.bashrc oder einer anderen Shell-Umgebung hinzugefügt werden, so dass es nicht notwendig ist, diesen bei jedem Start eines Terminals neu einzugeben. Der allgemeine Eintrag lautet:

```
export RACK_DIR=<Rack SDK folder>
```

Bei mir lautet der Pfad:

```
/home/erik/Rack-SDK
```

so dass entsprechende Eintrag in die ~/.bashrc wie folgt aussieht:

```
export RACK_DIR=/home/erik/Rack-SDK
```

Diese Modifikation kann am besten mit einem Text-Editor wie zum Beispiel *nano* vorgenommen werden, wobei der Aufruf wie folgt aussieht:

```
$ nano ~/.bashrc
```

Der Eintrag wird am besten ganz am Ende der Datei platziert, mit *Strg-O* gespeichert und der Editor mit *Strg-X* verlassen. Für das Testen, ob diese hinzugefügte Variable auch erkannt worden ist, muss das Terminal-Fenster geschlossen und ein neues geöffnet werden. Dann gibt man den folgenden Befehl ein.

```
$ echo $RACK_DIR
```

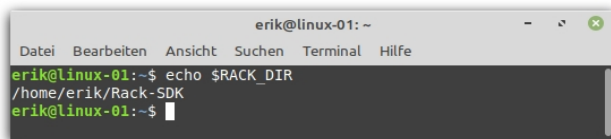


Abbildung 3: RACK-SDK-Pfad anzeigen lassen

Einrichten der Entwicklungsumgebung

Damit die Entwicklungsumgebung mit allen notwendigen Tools versorgt ist, sollte die folgende Befehlszeile aufgerufen werden. (ab Ubuntu 16.04):

```
$ sudo apt install unzip git gdb curl cmake libx11-dev libglu1-mesa-dev  
libxrandr-dev libxinerama-dev libxcursor-dev libxi-dev zlib1g-dev libasound2-dev  
libgtk2.0-dev libgtk-3-dev libjack-jackd2-dev jq zstd libpulse-dev
```

Vult-Template herunterladen

Um einen geeigneten Einstieg zu finden, kann das *RackPlayground*-Template genutzt werden, das unter der folgenden Internetadresse zu finden ist.

<https://github.com/vult-dsp/RackPlayground>

Für das Herunterzuladen, muss der folgende Befehl in ein Terminal-Fenster eingegeben werden, wobei ich dafür entsprechende Ordner im Home-Verzeichnis angelegt habe. Die Ordner-Struktur schaut wie folgt aus, wobei ich das Template in den untersten Ordner *VCV-Rack* speichern möchte.

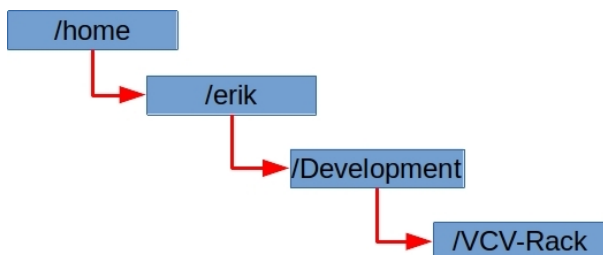


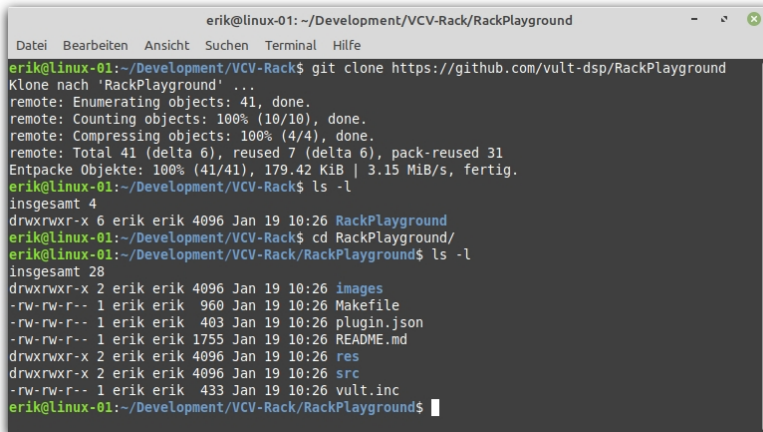
Abbildung 4: Pfad zum Template

Aus dem Git-Repository werden nun über den Befehl

```
$ git clone https://github.com/vult-dsp/RackPlayground
```

alle erforderlichen Dateien in den folgenden Ordner heruntergeladen und gespeichert.

```
/home/erik/Development/VCV-Rack
```



```
erik@linux-01: ~/Development/VCV-Rack/RackPlayground
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
erik@linux-01:~/Development/VCV-Rack$ git clone https://github.com/vult-dsp/RackPlayground
Klone nach 'RackPlayground' ...
remote: Enumerating objects: 41, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 41 (delta 6), reused 7 (delta 6), pack-reused 31
Entpackte Objekte: 100% (41/41), 179.42 KiB | 3.15 MiB/s, fertig.
erik@linux-01:~/Development/VCV-Rack$ ls -l
insgesamt 4
drwxrwxr-x 6 erik erik 4096 Jan 19 10:26 RackPlayground
erik@linux-01:~/Development/VCV-Rack$ cd RackPlayground/
erik@linux-01:~/Development/VCV-Rack/RackPlayground$ ls -l
insgesamt 28
drwxrwxr-x 2 erik erik 4096 Jan 19 10:26 images
-rw-rw-r-- 1 erik erik 960 Jan 19 10:26 Makefile
-rw-rw-r-- 1 erik erik 403 Jan 19 10:26 plugin.json
-rw-rw-r-- 1 erik erik 1755 Jan 19 10:26 README.md
drwxrwxr-x 2 erik erik 4096 Jan 19 10:26 res
drwxrwxr-x 2 erik erik 4096 Jan 19 10:26 src
-rw-rw-r-- 1 erik erik 433 Jan 19 10:26 vult.inc
erik@linux-01:~/Development/VCV-Rack/RackPlayground$
```

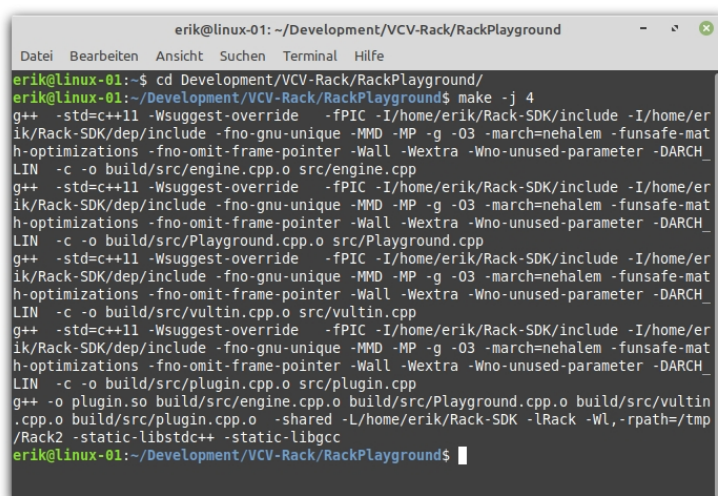
Abbildung 5: Herunterladen des Templates über Git

Der Build-Prozess

Im nächsten Schritt kann der sogenannte Build-Prozess gestartet werden, der dafür sorgt, dass gewisse Abhängigkeiten bei der Kompilierung berücksichtigt werden. Das Make-Kommando *make* ist ein Dienstprogramm für die Erstellung und Verwaltung von Gruppen von Programmen aus einem Quellcode. Dieses Kommando setzt ein sogenanntes *Makefile* voraus, in dem alle erforderlichen Informationen gespeichert sind. Ich wechsle also jetzt in das *RackPlayground*-Verzeichnis, in dem sich unter anderem die genannte Datei befindet, wie das in dem letzten Screenshot zu sehen ist. Es muss nun das folgende Kommando abgesetzt werden.

```
$ make -j 4
```

Über die Option *-j* kann die Anzahl der Aufträge (Befehle) angegeben werden, die gleichzeitig zur Ausführung kommen sollen. Das Ganze schaut dann wie folgt aus.



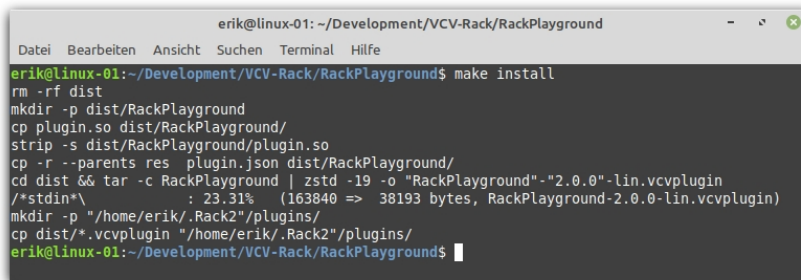
```
erik@linux-01: ~/Development/VCV-Rack/RackPlayground
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
erik@linux-01:~/Development/VCV-Rack/RackPlayground$ cd Development/VCV-Rack/RackPlayground/
erik@linux-01:~/Development/VCV-Rack/RackPlayground$ make -j 4
g++ -std=c++11 -Wsuggest-override -fPIC -I/home/erik/Rack-SDK/include -I/home/erik/Rack-SDK/dep/include -fno-gnu-unique -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_LIN -c -o build/src/engine.cpp.o src/engine.cpp
g++ -std=c++11 -Wsuggest-override -fPIC -I/home/erik/Rack-SDK/include -I/home/erik/Rack-SDK/dep/include -fno-gnu-unique -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_LIN -c -o build/src/Playground.cpp.o src/Playground.cpp
g++ -std=c++11 -Wsuggest-override -fPIC -I/home/erik/Rack-SDK/include -I/home/erik/Rack-SDK/dep/include -fno-gnu-unique -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_LIN -c -o build/src/vultin.cpp.o src/vultin.cpp
g++ -std=c++11 -Wsuggest-override -fPIC -I/home/erik/Rack-SDK/include -I/home/erik/Rack-SDK/dep/include -fno-gnu-unique -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_LIN -c -o build/src/plugin.cpp.o src/plugin.cpp
g++ -o plugin.so build/src/engine.cpp.o build/src/Playground.cpp.o build/src/vultin.cpp.o build/src/plugin.cpp.o -shared -L/home/erik/Rack-SDK -lRack -lWl,-rpath=/tmp/Rack2 -static-libstdc++ -static-libgcc
erik@linux-01:~/Development/VCV-Rack/RackPlayground$
```

Abbildung 6: Das make-Kommando

Um das Projekt letztendlich noch zu installieren, ist das folgende Kommando erforderlich.

```
$ make install
```

Das Ergebnis gestaltet wie folgt.



```
erik@linux-01:~/Development/VCV-Rack/RackPlayground$ make install
rm -rf dist
mkdir -p dist/RackPlayground
cp plugin.so dist/RackPlayground/
strip -s dist/RackPlayground/plugin.so
cp -r --parents res plugin.json dist/RackPlayground/
cd dist && tar -c RackPlayground | zstd -19 -o "RackPlayground"-2.0.0-lin.vcvplugin
/*stdin* : 23.31% (163840 => 38193 bytes, RackPlayground-2.0.0-lin.vcvplugin)
mkdir -p "/home/erik/.Rack2"/plugins/
cp dist/*.vcvplugin "/home/erik/.Rack2"/plugins/
erik@linux-01:~/Development/VCV-Rack/RackPlayground$
```

Abbildung 7: Das make install-Kommando

Die Überprüfung der Kompilierung

Um nun zu überprüfen, ob sich auch wirklich etwas im Dateisystem getan hat, das als VCV-Rack-Plugin genutzt werden kann, muss ein Blick in ein bestimmtes Verzeichnis geworfen werden. Im zuletzt gezeigten Terminal-Fenster ist dieser Pfad in der vorletzten Zeile von unten sogar zu sehen. Er lautet:

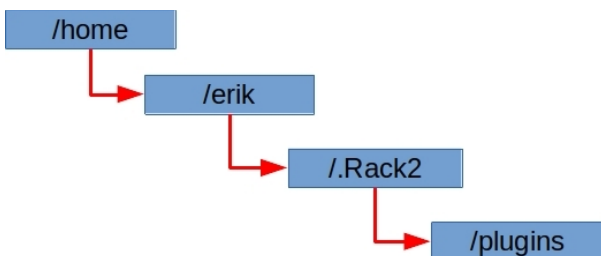


Abbildung 8: Pfad zu den VCV-Rack-Plugins

Es handelt sich dabei um ein spezielles Verzeichnis, das von der VCV-Rack-Installation genutzt wird. Nach der Installation von VCV-Rack 2 unter Linux werden dort alle Plugins, also Erweiterungen, gespeichert. Werfen wir dort einen Blick hinein.

Es befinden sich an dieser Stelle sehr viele Erweiterungen, die ich meinem VCV-Rack schon durch mehrere Abonnements hinzugefügt habe. Unter anderem ist auch eine spezielle Datei zu sehen, die ich rechts unten in der Abbildung rot markiert habe.

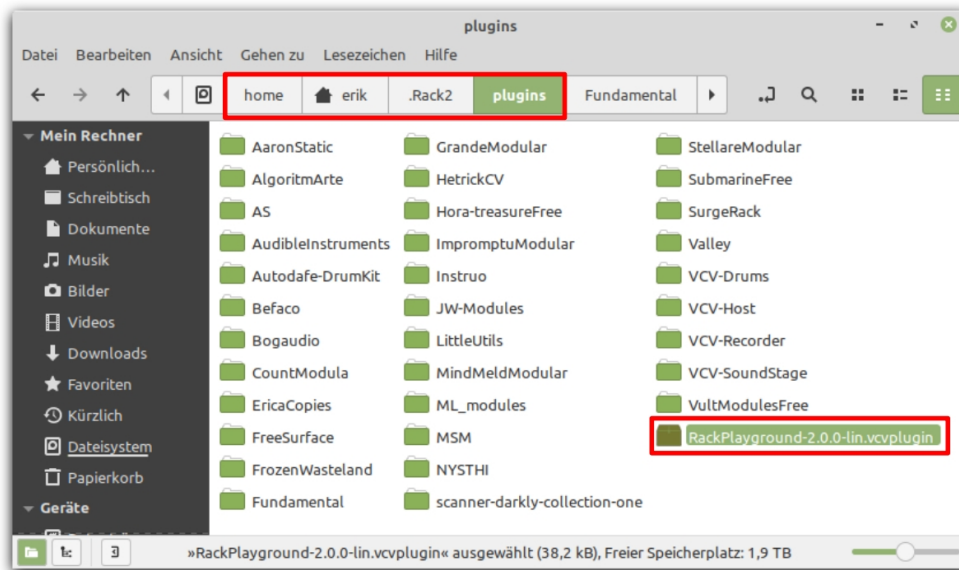


Abbildung 9: Das Plugin-Verzeichnis von VCV-Rack 2

Es handelt sich dabei genau um das gerade erzeugte RackPlayground-Plugin, das durch die Dateierweiterung `vcvplugin` gekennzeichnet ist. Doch diese Erweiterung unterscheidet sich von den anderen Erweiterungen, die durch einen Ordner im Dateisystem gekennzeichnet sind. Warum ist das so? Ganz einfach, denn nach einer Installation einer derartigen VCV-Rack-Erweiterung erfolgt erst bei einem erneuten Start von VCV-Rack die eigentliche Installation und die Konvertierung in einen entsprechenden Unterordner. Ich starte also mein VCV-Rack einmal neu und werfe einen erneuten Blick in das Plugin-Verzeichnis.

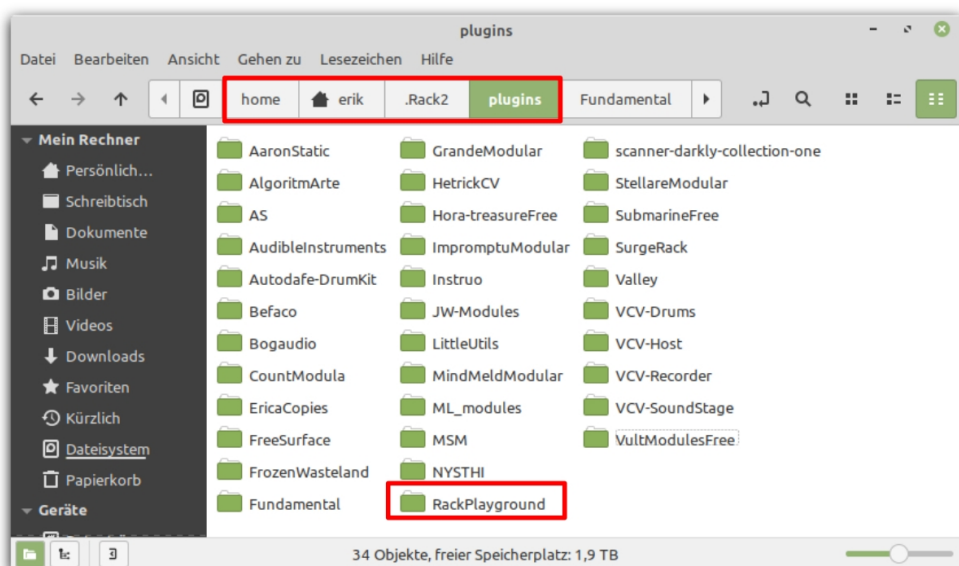


Abbildung 10: Das Plugin-Verzeichnis von VCV-Rack 2

Es ist nun zu erkennen, dass jetzt ein neuer Unterordner entstanden ist, der genau den Namen des RackPlayground-Templates trägt. Darin sind nun alle erforderlichen Dateien zur Nutzung des Plugins enthalten. Aber ist das neue Plugin denn auch im VCV-Rack zu finden? Sehen wir nach.

Das neue RackPlayground-Plugin in VCV-Rack 2

Wird der Browser im VCV-Rack geöffnet, dann ist das neue RackPlayground-Plugin direkt links oben zu sehen und kann durch einen Mausklick in das VCV-Rack übernommen und eingefügt werden.



Abbildung 11: Das neue RackPlayground-Plugin im VCV-Rack 2

Installationschritte für Windows

Rack-SDK installieren

Unter der folgenden Internetadresse sind unter anderem alle Versionen des Rack-SDKs zu finden.

<https://vcvrack.com/downloads/>

Die letzte Version des Rack-SDK für Windows herunterladen.

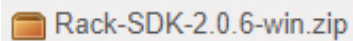


Abbildung 12: Rack-SDK

Diese Datei muss im Dateisystem entpackt werden. Der Pfad dorthin wird gleich benötigt, damit der Kompilierungsprozess das Rack-SDK auch findet. Dazu muss die Umgebungsvariable `RACK_SDK` angelegt werden. Ich habe das Rack-SDK bei mir unter [D:\Rack-SDK](#) entpackt.

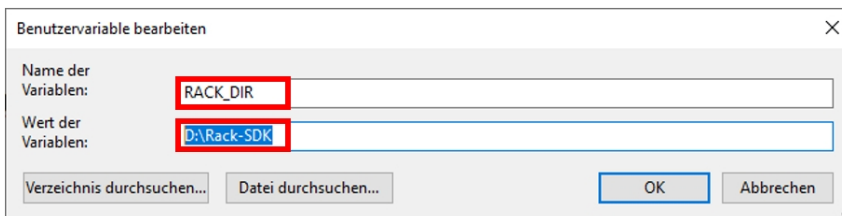


Abbildung 13: Die Umgebungsvariable für das RACK-SDK

MSYS2 installieren

Unter Windows starten wir mit der Installation von `MSYS2`. Bei `MSYS2` handelt es sich um eine Sammlung von Werkzeugen und Bibliotheken, die eine einfach zu bedienende Umgebung für das Erstellen, Installieren und Ausführen nativer Windows-Software liefert. Die Software ist unter der folgenden Internetadresse zu finden, wobei die 64-Bit-Version zu installieren ist.

<https://www.msys2.org/>

Nach der Installation stehen unter Windows verschiedene Programme zur Verfügung, wobei die rot markierte Anwendung gestartet verwendet werden muss.



Abbildung 14: Die `MSYS2`-Programme

Nach dem Aufruf öffnet sich ein Terminal-Fenster, in dem für ein erforderliches Update noch der Befehl

```
$ pacman -Su
```

einggegeben werden muss, was dann wie folgt aussieht.

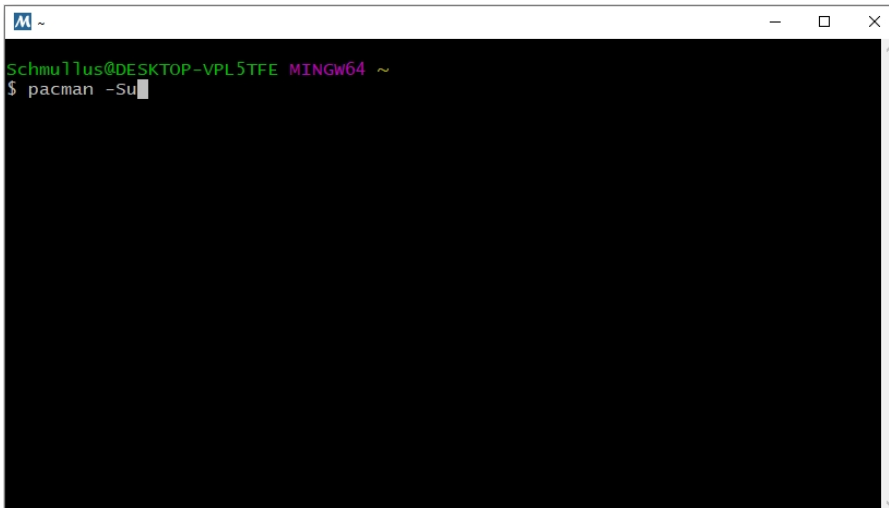


Abbildung 15: Das erforderliche Update wird durchgeführt

Im Anschluss werden noch einige Sicherheitsabfragen gestellt, die alle mit Y beantwortet werden sollten. Danach beginnt die Update-Prozedur, wobei die einzelnen Schritte hinsichtlich ihres Fortschrittes angezeigt werden.

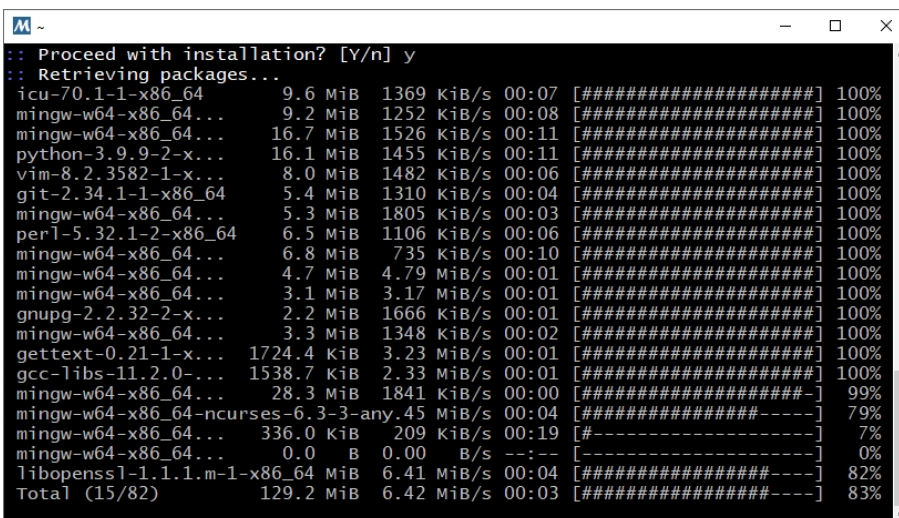


Abbildung 16: Die Anzeige des Update-Fortschrittes

Ist das Update erfolgreich abgeschlossen worden, muss das Terminal-Fenster geschlossen und erneut geöffnet werden. Die folgende Befehlszeile ist nun einzugeben.

```
$ pacman -Su git wget make tar unzip zip mingw-w64-x86_64-gcc mingw-w64-x86_64-gdb mingw-w64-x86_64-cmake autoconf automake mingw-w64-x86_64-libtool mingw-w64-x86_64-jq python zstd
```

Nach einigen erneuten Sicherheitsabfragen beginnt der Installationsprozess.

Vult-Compiler installieren

Der Vult-Compiler kann unter der folgenden Internetadresse heruntergeladen werden.

<https://github.com/vult-dsp/vult/releases>

Die Datei *vultc.exe* muss natürlich dem System bekanntgemacht werden. Dazu muss die Datei in das folgende Verzeichnis der MSYS2-Installation kopiert werden.

```
C:\msys64\mingw64\bin
```

Vult-Template herunterladen

Um auch unter Windows das *RackPlayground*-Template herunterzuladen, kann wieder Git bemüht werden, das unter der folgenden Internetadresse zu finden ist.

<https://github.com/vult-dsp/RackPlayground>

Um die Daten aus dem Git-Repository herunterzuladen, muss der folgende Befehl in ein Terminal-Fenster eingegeben werden.

```
$ git clone https://github.com/vult-dsp/RackPlayground
```

In MSYS2 schaut das dann wie folgt aus.

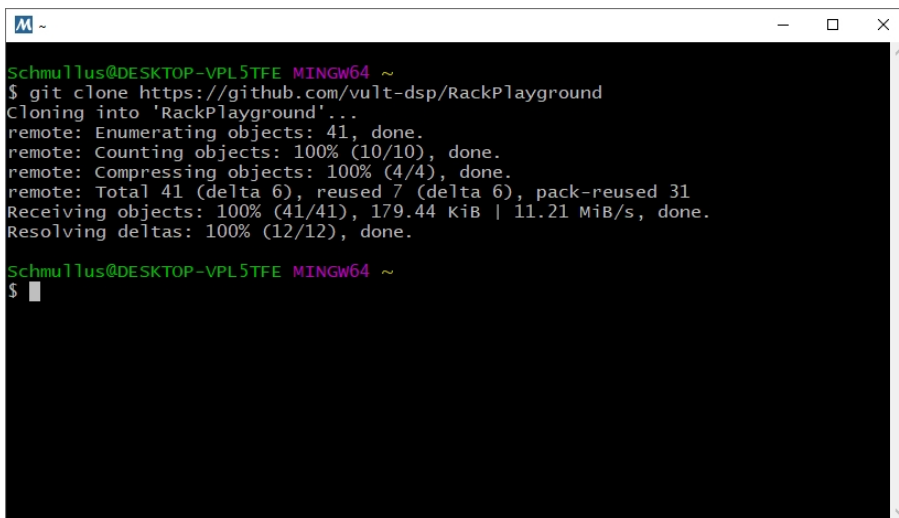
A screenshot of a terminal window with a black background and green text. The window title bar shows a blue icon and the text '~'. The terminal content shows a user named 'Schmullus' at a desktop named 'DESKTOP-VPL5TFE' in a 'MINGW64' environment. The user enters the command '\$ git clone https://github.com/vult-dsp/RackPlayground'. The terminal output shows the cloning process: 'Cloning into 'RackPlayground'...', 'remote: Enumerating objects: 41, done.', 'remote: Counting objects: 100% (10/10), done.', 'remote: Compressing objects: 100% (4/4), done.', 'remote: Total 41 (delta 6), reused 7 (delta 6), pack-reused 31', 'Receiving objects: 100% (41/41), 179.44 KiB | 11.21 MiB/s, done.', and 'Resolving deltas: 100% (12/12), done.'. The prompt '\$' is shown again at the bottom of the terminal.

Abbildung 17: Herunterladen des Templates über Git

Nun besteht die berechtigte Frage, wo sich denn nun die heruntergeladenen Dateien im Windows-Dateisystem befinden.

Ein Blick in den folgenden Ordner gibt Aufschluss.

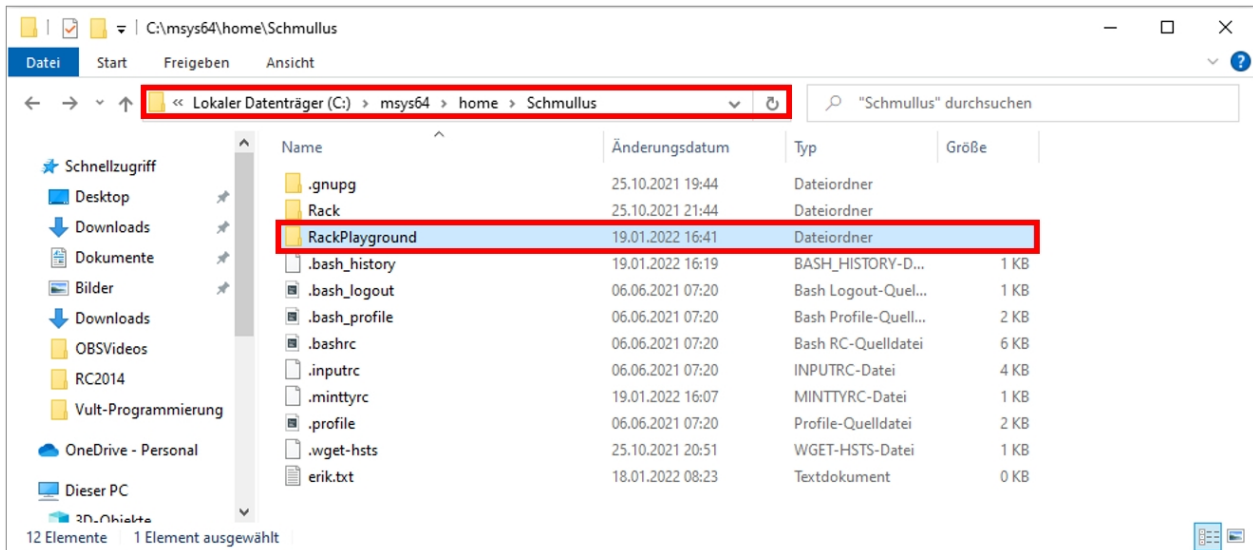


Abbildung 18: Das Download-Verzeichnis des Templates

Es ist zu sehen, dass im Verzeichnis

```
c:\msys64\home\
```

sich ein Ordner befindet, der sich *RackPlayground* nennt. Es handelt sich um den Ordner aus dem Git-Repository. Im nächsten Schritt erfolgt eine Aktualisierung etwaiger Submodule über Git. Das wird über die folgende Befehlszeile im MSYS2-Terminal-Fenster erreicht, wobei zuvor in das Template-Verzeichnis gewechselt werden muss.

```
$ cd RackPlayground  
$ git submodule update --init --recursive
```

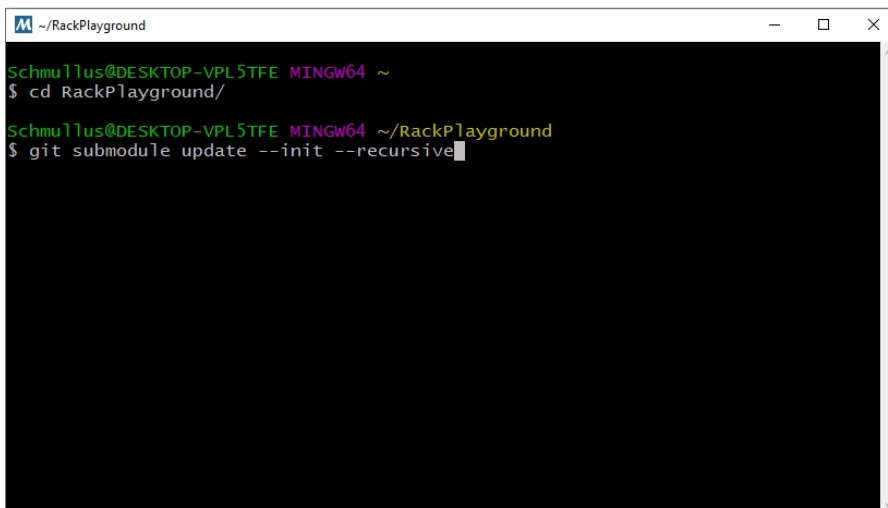


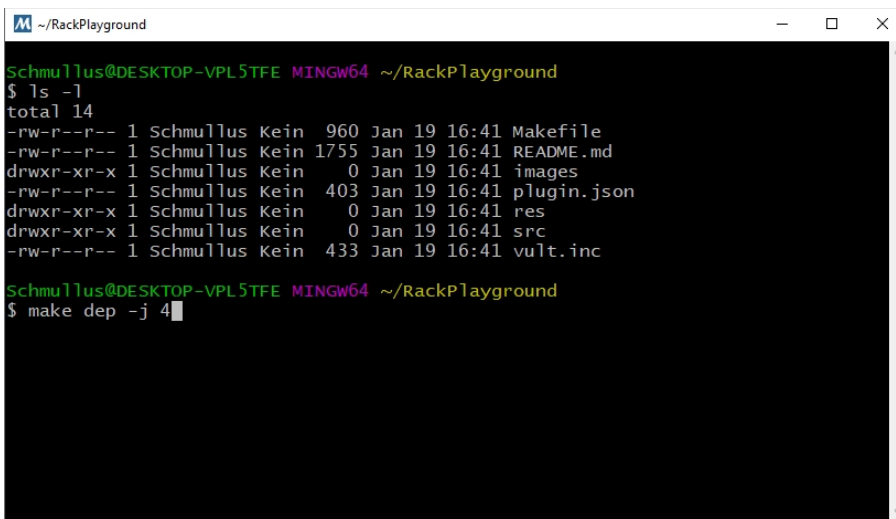
Abbildung 19: Ein Update der Submodule

Der Build-Prozess

Im nächsten Schritt kann der sogenannte Build-Prozess gestartet werden, der dafür sorgt, dass gewisse Abhängigkeiten bei der Kompilierung berücksichtigt werden. Das Make-Kommando *make* ist ein Dienstprogramm für die Erstellung und Verwaltung von Gruppen von Programmen aus einem Quellcode. Dieses Kommando setzt ein sogenanntes *Makefile* voraus, in dem alle erforderlichen Informationen gespeichert sind. Es muss das folgende Kommando abgesetzt werden.

```
$ make dep -j 4
```

Über die Option *-j* kann die Anzahl der Aufträge (Befehle) angegeben werden, die gleichzeitig zur Ausführung kommen sollen. Das Ganze schaut dann wie folgt aus.



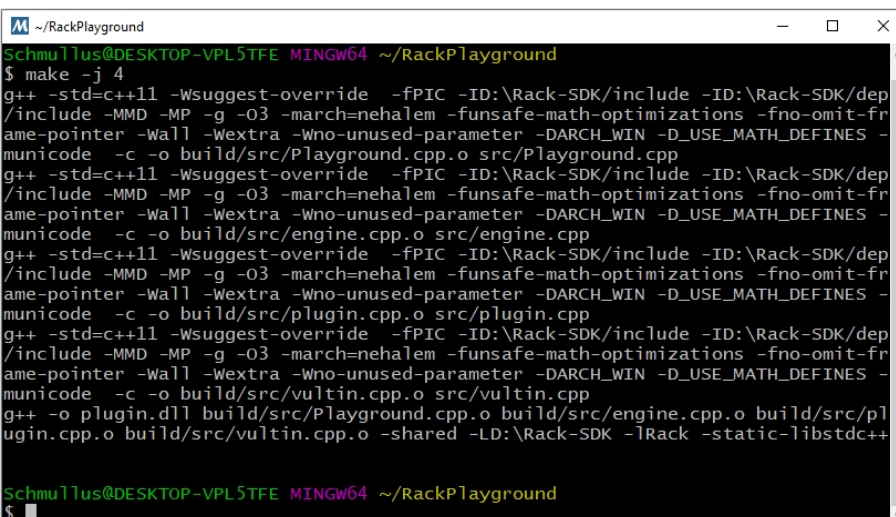
```
~/RackPlayground
Schmullus@DESKTOP-VPL5TFE MINGW64 ~/RackPlayground
$ ls -l
total 14
-rw-r--r-- 1 Schmullus Kein 960 Jan 19 16:41 Makefile
-rw-r--r-- 1 Schmullus Kein 1755 Jan 19 16:41 README.md
drwxr-xr-x 1 Schmullus Kein 0 Jan 19 16:41 images
-rw-r--r-- 1 Schmullus Kein 403 Jan 19 16:41 plugin.json
drwxr-xr-x 1 Schmullus Kein 0 Jan 19 16:41 res
drwxr-xr-x 1 Schmullus Kein 0 Jan 19 16:41 src
-rw-r--r-- 1 Schmullus Kein 433 Jan 19 16:41 vult.inc

Schmullus@DESKTOP-VPL5TFE MINGW64 ~/RackPlayground
$ make dep -j 4
```

Abbildung 20: Das *make-dep*-Kommando

Im nächsten Schritt wird das *make*-Kommando alleine aufgerufen.

```
$ make -j 4
```



```
~/RackPlayground
Schmullus@DESKTOP-VPL5TFE MINGW64 ~/RackPlayground
$ make -j 4
g++ -std=c++11 -wsuggest-override -fPIC -ID:\Rack-SDK/include -ID:\Rack-SDK/dep/include -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_WIN -D_USE_MATH_DEFINES -municode -c -o build/src/Playground.cpp.o src/Playground.cpp
g++ -std=c++11 -wsuggest-override -fPIC -ID:\Rack-SDK/include -ID:\Rack-SDK/dep/include -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_WIN -D_USE_MATH_DEFINES -municode -c -o build/src/engine.cpp.o src/engine.cpp
g++ -std=c++11 -wsuggest-override -fPIC -ID:\Rack-SDK/include -ID:\Rack-SDK/dep/include -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_WIN -D_USE_MATH_DEFINES -municode -c -o build/src/plugin.cpp.o src/plugin.cpp
g++ -std=c++11 -wsuggest-override -fPIC -ID:\Rack-SDK/include -ID:\Rack-SDK/dep/include -MMD -MP -g -O3 -march=nehalem -funsafe-math-optimizations -fno-omit-frame-pointer -Wall -Wextra -Wno-unused-parameter -DARCH_WIN -D_USE_MATH_DEFINES -municode -c -o build/src/vultin.cpp.o src/vultin.cpp
g++ -o plugin.dll build/src/Playground.cpp.o build/src/engine.cpp.o build/src/plugin.cpp.o build/src/vultin.cpp.o -shared -LD:\Rack-SDK -lRack -static-libstdc++

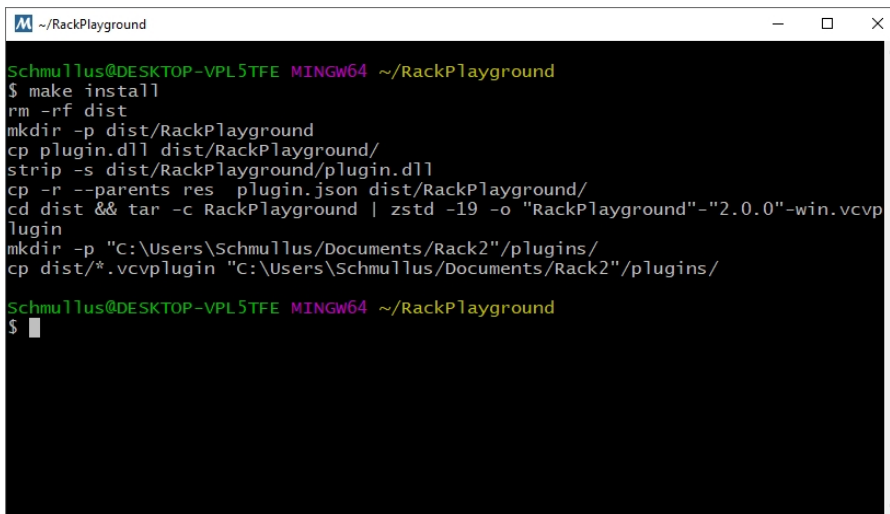
Schmullus@DESKTOP-VPL5TFE MINGW64 ~/RackPlayground
$
```

Abbildung 21: Das *make*-Kommando

Um das Projekt letztendlich noch zu installieren, ist das folgende Kommando erforderlich.

```
$ make install
```

Das Ergebnis schaut dann wie folgt aus.



```
~/RackPlayground
Schmu11us@DESKTOP-VPL5TFE MINGW64 ~/RackPlayground
$ make install
rm -rf dist
mkdir -p dist/RackPlayground
cp plugin.dll dist/RackPlayground/
strip -s dist/RackPlayground/plugin.dll
cp -r --parents res plugin.json dist/RackPlayground/
cd dist && tar -c RackPlayground | zstd -19 -o "RackPlayground"-2.0.0-win.vcplugin
mkdir -p "C:\Users\Schmu11us/Documents/Rack2"/plugins/
cp dist/*.vcplugin "C:\Users\Schmu11us/Documents/Rack2"/plugins/

Schmu11us@DESKTOP-VPL5TFE MINGW64 ~/RackPlayground
$
```

Abbildung 22: Das make-install-Kommando

Die Überprüfung der Kompilierung

Um jetzt zu überprüfen, ob sich auch wirklich etwas im Dateisystem getan hat, das als VCV-Rack-Plugin genutzt werden kann, muss ein Blick in ein bestimmtes Verzeichnis geworfen werden. Im zuletzt gezeigten Terminal-Fenster ist dieser Pfad in der vorletzten Zeile von unten sogar zu sehen. Er lautet:

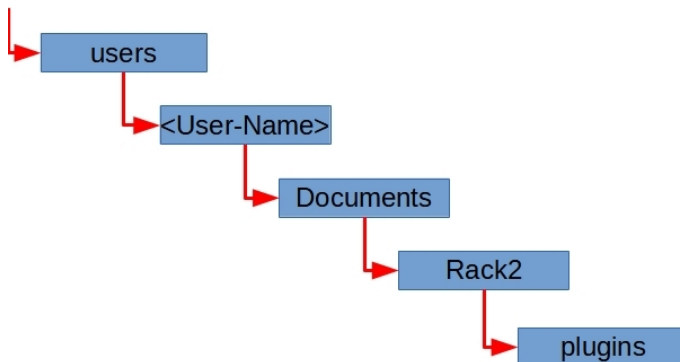


Abbildung 23: Pfad zu den VCV-Rack-Plugins

Es handelt sich dabei um ein spezielles Verzeichnis, das von der VCV-Rack-Installation genutzt wird. Nach der Installation von VCV-Rack 2 unter Windows werden dort alle Plugins, also Erweiterungen, gespeichert. Werfen wir dort einen Blick hinein. Es befinden sich dort sehr viele Erweiterungen, die ich meinem VCV-Rack durch mehrere zuvor durchgeführte Abonnements hinzugefügt habe.

Unter anderem ist auch eine spezielle Datei zu sehen, die im unteren Bereich der Abbildung rot markiert wurde.

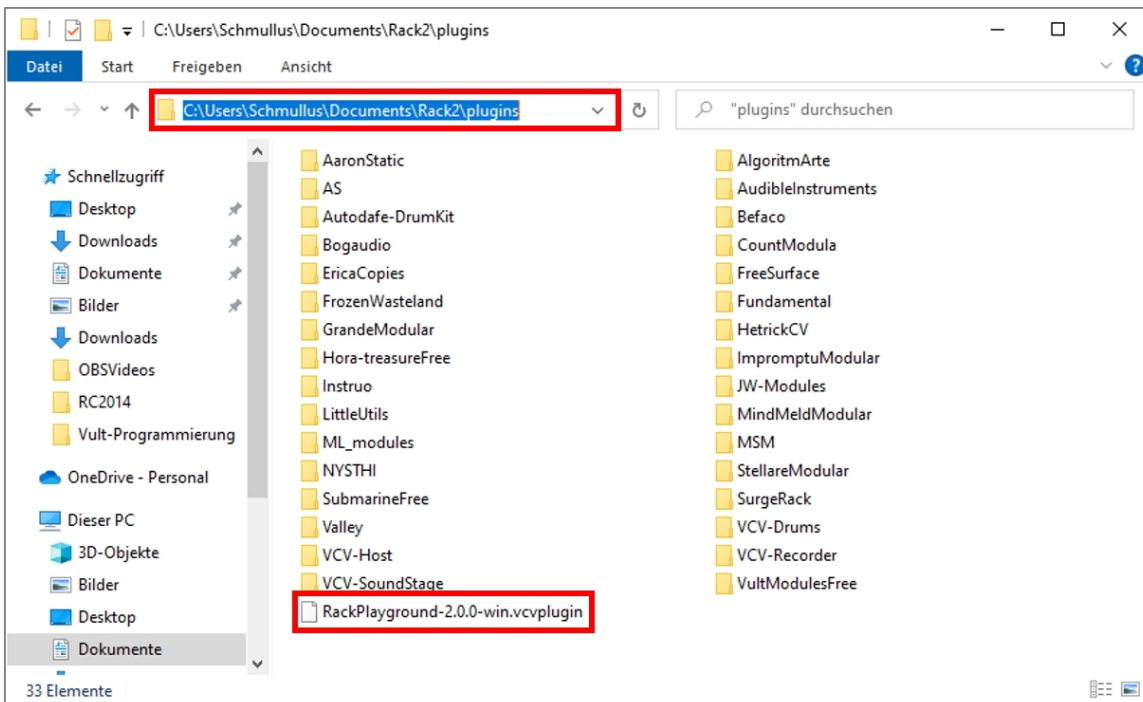


Abbildung 24: Das Plugin-Verzeichnis von VCV-Rack 2

Es handelt sich dabei genau um das gerade erzeugte RackPlayground-Plugin, das durch die Dateierweiterung *vcvplugin* gekennzeichnet ist. Doch diese Erweiterung unterscheidet sich von den anderen Erweiterungen, die durch einen Ordner im Dateisystem gekennzeichnet sind. Warum ist das so? Ganz einfach, denn nach einer Installation einer derartigen VCV-Rack-Erweiterung erfolgt erst bei einem erneuten Start von VCV-Rack die eigentliche Installation und die Konvertierung in einen entsprechenden Unterordner.

Ich starte also mein VCV-Rack einmal neu und werfe einen Blick in das Plugin-Verzeichnis.

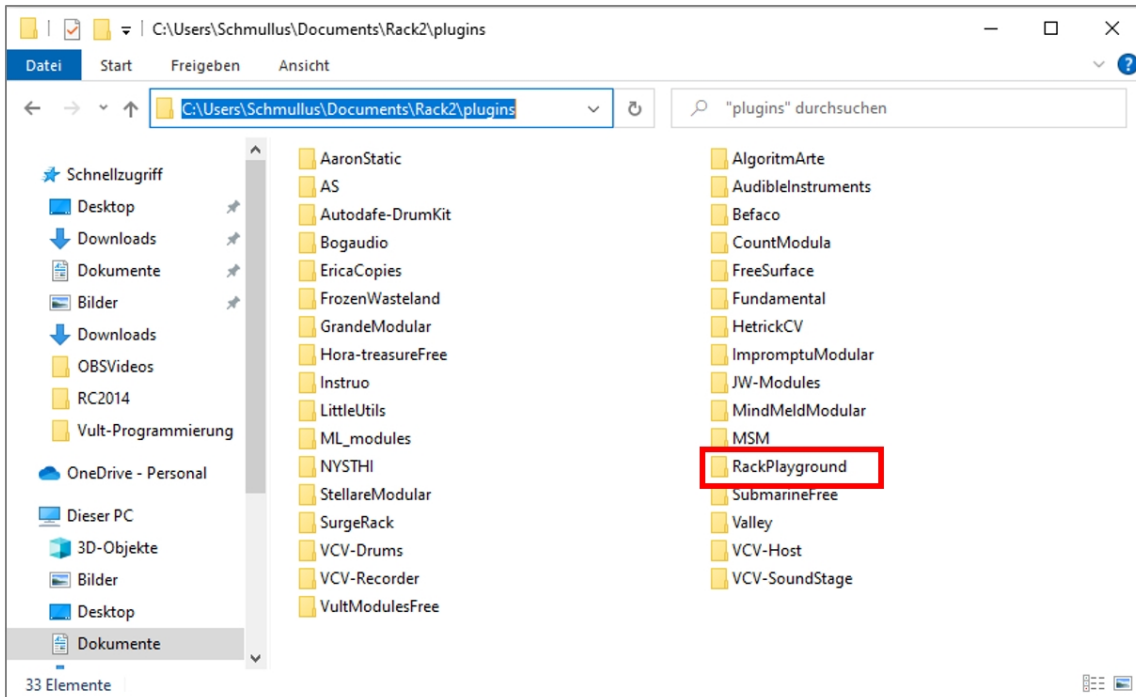


Abbildung 25: Das Plugin-Verzeichnis von VCV-Rack 2

Es ist nun zu erkennen, dass nun ein neuer Unterordner entstanden ist, der genau den Namen des RackPlayground-Templates trägt. Darin sind nun alle erforderlichen Dateien zur Nutzung des Plugins enthalten. Aber ist das neue Plugin denn auch im VCV-Rack zu finden? Sehen wir nach.

Das neue RackPlayground-Plugin in VCV-Rack 2

Wird der Browser im VCV-Rack geöffnet, dann ist das neue RackPlayground-Plugin direkt links oben zu sehen und kann durch einen Mausklick in das VCV-Rack übernommen und eingefügt werden.



Abbildung 26: Das neue RackPlayground-Plugin im VCV-Rack 2

Abschließend

Dieses Plugin besitzt noch keine rechte Funktionalität und sollte im ersten Schritt nur als Einstieg dienen, um die Entwicklungsumgebung entsprechend zu installieren und die einzelnen erforderlichen Schritte aufzuzeigen. Das sollte als Einführung in die Thematik erst einmal reichen, wenn es darum geht, die Entwicklungsumgebung für *Vult* respektive für die *Vult-Programmiersprache* vorzubereiten. Über den folgenden Link sind alle erforderlichen Informationen zu diesem Thema zu erhalten. Ebenfalls gibt es zahlreiche Videos, die das Arbeiten mit der Programmiersprache detailliert zeigen.

<https://modlfo.github.io/vult/tutorials/>

Weitere Informationen

Natürlich sind alle Informationen auf den folgenden Internetseiten noch im Detail zu finden.

Erik Bartmann

<https://erik-bartmann.de/>

https://erik-bartmann.de/?Musik_VCV-Rack

Das VCV-Rack

<https://vcvrack.com/>

<https://vcvrack.com/manual/Building#Setting-up-your-development-environment>

Vult

<https://modlfo.github.io/vult/tutorials/>

<https://github.com/vult-dsp/RackPlayground>

Viel Spaß beim Kodieren!

Erik Bartmann