# Nano Ardule Drum System – PC-side Toolchain Guide (ADT v2.2)

## 1. Overview

This document describes the full preprocessing workflow and Python tools used to prepare drum pattern data for the Nano Ardule Drum Player. It covers conversion from MIDI to ADT (Ardule Drum Text), generation of ADP binary cache files, index building, and arrangement (ARR) creation for use on the microSD card.

## 2. Directory Structure

PC/  
 ├─ MID/ # 2-bar MIDI segments  
 ├─ ADT/ # Converted ADT files (editable)  
 ├─ SD/  
 │ ├─ PATTERNS/ # Final ADP files (for device)  
 │ ├─ SONGS/ # ARR/APT song sequences  
 │ └─ SYSTEM/ # INDEX.TXT, SETTINGS.CFG, etc.  
 └─ tools/ # Python scripts (midi2adt.py, adt2adp.py, mkindex.py, arr\_make.py)

## 3. Processing Pipeline

1) Subdivision Analysis (Triplet/Straight detection)  
 python mid2report\_integrated.py 6ROCK.MID  
2) Split Type-0 MIDI into 2-bar segments  
 python split\_drums\_2bar\_save\_v4a.py 6ROCK.MID  
3) Convert MIDI → ADT  
 python mid2adt.py --in-dir ./MID --out-dir ./ADT --recursive  
 (Triplet sets: --grid 16T --length 48)  
4) Convert ADT → ADP  
 python adt2adp.py --in-dir ./ADT --out-dir ./SD/PATTERNS --recursive --overwrite  
5) Build INDEX.TXT  
 python mkindex.py --patterns ./SD/PATTERNS --out ./SD/SYSTEM/INDEX.TXT  
6) Make ARR sequences  
 python arr\_make.py --title 'Rock Demo' --tempo 120 --in ./lists/rock\_demo.csv --out ./SD/SONGS/ROCK\_DEMO.ARR

## 4. ADT v2.2 Specification

ADT (Ardule Drum Text) v2.2 defines a human-readable drum pattern format supporting multiple time subdivisions.  
Key parameters:  
 - LENGTH: 24, 32, or 48 steps (2 bars)  
 - GRID: 16 (straight), 8T (8th-triplet), or 16T (16th-triplet)  
 - SLOTS: up to 12 drum instruments per pattern  
 - ORIENTATION: STEP or SLOT (automatically normalized)  
  
Example Header:  
; ADT v2.2  
NAME=ROCK\_MAIN  
TIME\_SIG=4/4  
GRID=16T  
LENGTH=48  
SLOTS=12  
KIT=GM\_STD  
ORIENTATION=STEP

## 5. ADP v2.2 Binary Cache Format

ADP (Ardule Drum Pattern) files store pre-parsed binary representations of ADT patterns for fast loading on the device.  
Header structure (<little endian>):  
 magic[4]='ADP2', version=22, grid(0=16,1=8T,2=16T), length, slots, ppqn=96, swing, tempo, reserved, adt\_crc16, payload\_bytes  
Payload: For each step, count (1B) + (slot<<2|acc) \* count.

## 6. Tool Descriptions

### midi2adt.py

Converts 2-bar MIDI drum segments to ADT v2.2 format.  
Detects note hits, velocity, timing, and maps to 12-slot GM drum kit.  
Options: --grid, --length, --recursive, --out-dir.

### adt2adp.py

Converts ADT text files into binary ADP cache files.  
Supports automatic STEP/SLOT normalization, CRC16 validation, GRID 16/8T/16T, LENGTH 24/32/48.

### mkindex.py

Scans ADP files to create /SYSTEM/INDEX.TXT.  
Extracts header info (length, grid, CRC, size) and summarizes genres by filename prefix.

### arr\_make.py

Builds ARR song sequence files from a CSV or CLI list of pattern,repeat pairs.  
Example: RCK\_P001,4 → P01=RCK\_P001.ADP, REPEAT=4.

## 7. About '16T' Grid (Triplet)

'16T' stands for '16th-note triplet grid', meaning each beat (quarter note) is divided into 6 equal substeps.  
It corresponds to 16th-triplet resolution used in swing/shuffle or jazz grooves.  
For example, a 4/4 pattern with GRID=16T and LENGTH=48 represents two bars with 24 steps per bar.

## 8. Typical SD Card Layout

/PATTERNS/ — ADP pattern files  
/SONGS/ — ARR playlists  
/SYSTEM/ — INDEX.TXT, SETTINGS.CFG, TEMPO.CFG  
The device reads INDEX.TXT on boot for genre counts and file mapping, then loads ADP/ARR as needed.