Microcontroleurs

- tinyAVR 0,1,2-series
- ESP32
    - Utilisation de Preferences.h avec ESP32 (Arduino)

tinyAVR 0,1,2-series

Pense-bête rapide

ATtinyXYZZ

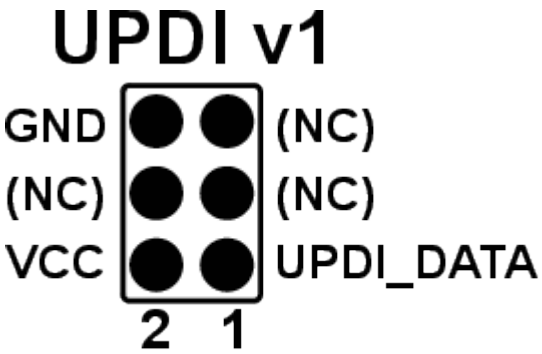
- **X** → génération (8 = ancien, 16 = plus récent)
- **ZZ** → taille Flash (04=4KB, 14=16KB, 16=16KB…)

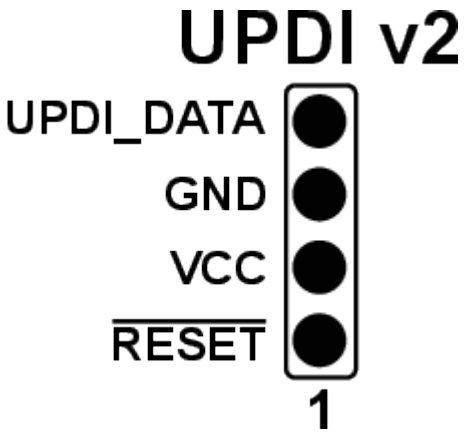
Résumé concret

Modèle	Flash	Remarque
ATtiny1604	4 KB	petit
ATtiny1614	16 KB	standard moderne
ATtiny1616	16 KB	plus de pins
ATtiny814	16 KB	ancienne génération
ATtiny816	16 KB	ancienne gen, plus de pins

Programmation par UPDI

La série tinyAVR 1 se programme via UPDI, voici les deux brochages recommandés / standard pour la programmation **in-circuit**.





Sources

- https://fabacademy.org/2025/labs/ilmenau/assignments/week04/
- https://www.programming-electronics-diy.xyz/2024/01/how-to-program-updi-avr-microcontroller.html

ESP32

ESP32

Utilisation de Preferences.h avec ESP32 (Arduino)

Introduction

La bibliothèque `Preferences.h` permet de stocker des données de manière persistante dans la mémoire flash de l'ESP32. Contrairement aux variables classiques stockées en RAM, les données enregistrées avec `Preferences` sont conservées après un redémarrage ou une coupure d'alimentation.

Elle repose sur le système interne NVS (Non-Volatile Storage) de l'ESP32 et offre une interface simple pour manipuler des paires clé-valeur.

Principe de fonctionnement

Les données sont organisées sous forme de :

- **namespace** (espace de stockage, comparable à un dossier)
- **clés** associées à des valeurs

Chaque namespace contient plusieurs variables identifiées par une clé unique.

Utilisation de base

1. Inclusion de la bibliothèque

```
#include <Preferences.h>
```

2. Déclaration de l'objet

```
Preferences preferences;
```

3. Ouverture d’un namespace

```
preferences.begin("mon-espace", false);
```

- `"mon-espace"` : nom du namespace
- `false` : mode lecture/écriture
- `true` : mode lecture seule

4. Écriture de données

```
preferences.putInt("compteur", 42);
preferences.putString("nom", "ESP32");
preferences.putBool("etat", true);
```

Types supportés :

- entiers (int, uint)
- flottants (float, double)
- booléens
- chaînes de caractères (String)
- tableaux de bytes

5. Lecture de données

```
int compteur = preferences.getInt("compteur", 0);
String nom = preferences.getString("nom", "inconnu");
bool etat = preferences.getBool("etat", false);
```

Le second paramètre correspond à la valeur par défaut si la clé n’existe pas.

6. Fermeture

```
preferences.end();
```

Cette étape libère les ressources associées.

Exemple complet

```
#include <Preferences.h>

Preferences preferences;

void setup() {
  Serial.begin(115200);

  preferences.begin("config", false);

  int bootCount = preferences.getInt("boot", 0);
  bootCount++;

  preferences.putInt("boot", bootCount);

  Serial.println(bootCount);

  preferences.end();
}

void loop() {}
```

Dans cet exemple, une valeur est stockée en mémoire flash et incrémentée à chaque redémarrage.

Opérations supplémentaires

Supprimer une clé

```
preferences.remove("cle");
```

Effacer tout un namespace

```
preferences.clear();
```

Cas d’utilisation

- Stockage de paramètres (WiFi, configuration utilisateur)
- Compteurs persistants
- Calibration de capteurs
- Sauvegarde d’état (ex : relais, LED)

Limitations et bonnes pratiques

- La mémoire flash a un nombre limité de cycles d’écriture
  → éviter les écritures trop fréquentes dans une boucle
- Les clés doivent être uniques dans un namespace
- Les données écrites avec une clé existante écrasent les précédentes

Conclusion

`Preferences.h` fournit une méthode simple et efficace pour gérer des données persistantes sur ESP32. Elle évite de manipuler directement la mémoire flash tout en offrant une interface claire et adaptée à la majorité des besoins embarqués.