# Commandes DCC++ et leur structure

DCC++ lit les commandes à partir d'une source extérieure, voici quelques exemples :

Ligne de commande à partir du moniteur série ou d'un logiciel comme le contrôleur DCC++ ou JMRI

Si vous testez votre configuration ou écrivez votre propre programme de contrôle, ces commandes sont ce que la station de base DCC++ attend et, dans certains cas, vous répond.

Il est recommandé de télécharger la dernière version de l'IDE Arduino et du logiciel de station de base DCC++ à partir de GitHub. Si vous êtes nouveau sur DCC++, je vous suggère de consulter la page Premiers pas avec le matériel DCC++.

La station de base DCC++ écoute les commandes dans le(s) format(s) suivant(s) :

## ###COMMANDES À LETTRE UNIQUE###

• < s > s minuscule : état de la station de base DCC++

• Retourne : suivre l'état de l'alimentation, l'état de l'accélérateur, l'état de l'aiguillage et un numéro de version.

• < 0 > Numéro Zéro : met hors tension les pistes (principale et de programmation).

• Renvoie : < p0 > Mise hors tension des pistes.

• < 1 > Number One : Mettez sous tension les pistes (principales et de programmation).

• Renvoie : < p1 > Mise sous tension des pistes ON.

• < T > T majuscule : Liste tous les aiguillages définis.

• Retourne : < H ID ADDRESS SUBADDRESS THROW > pour chaque aiguillage défini ou < X > si aucun aiguillage n'est défini.

• < S > Majuscule S : Répertorie tous les capteurs définis.

• Renvoie : < Q ID PIN PULLUP > pour chaque capteur défini ou < X > si aucun capteur n'est défini.

• < Z > Z majuscule : répertorie toutes les broches de sortie définies

• Renvoie : < Y ID PIN IFLAG STATE > pour chaque broche de sortie définie ou < X > si aucune broche de sortie n'est définie

• < Q > Majuscule Q : Répertorie l'état de tous les capteurs.

• Renvoie : < Q ID > (actif) ou < q ID > (inactif)

• <E> E majuscule : commande pour stocker les définitions dans l'EEPROM

• Renvoie : < e nTurnouts nSensors >

• <e> Minuscule e : Commande pour effacer TOUTES les définitions (aiguillages, capteurs et sorties) de l'EEPROM

• Renvoie : < 0 > EEPROM vide

• (Il n'y a PAS d'annulation de suppression)

• < D > Majuscule D : Veuillez consulter Diagnostics-< D >-Page de commande

• Il existe quelques autres commandes de débogage dans SerialCommand.cpp qui ne doivent être utilisées que par des utilisateurs avancés (potentiellement dangereuses si elles ne sont pas utilisées correctement).

## ###Commandes de fonctionnement du décodeur de locomotive (CAB)###

### ####MANETTE DE VITESSE####

Le format des vitesses CAB est < t REGISTER CAB SPEED DIRECTION>.

La répartition pour cet exemple <t 1 03 20 1> est :

"<" = Écoutez-moi Je suis le début d'une commande DCC++. (Un espace après < n'est pas obligatoire mais acceptable)

"t" = (t minuscule) Cette commande est destinée à un décodeur installé dans une locomotive ou simplement une "cabine".

"1" = REGISTER : un numéro de registre interne, de 1 à MAX\_MAIN\_REGISTERS (la valeur par défaut est 12), pour stocker le paquet DCC utilisé pour contrôler ce paramètre de manette (le registre "0" est destiné à la programmation)

"03" = CAB : l'adresse courte (1-127) ou longue (128-10293) du décodeur de locomotive (elle doit déjà être programmée dans le décodeur) Voir les commandes de programmation ci-dessous.

"20" = SPEED : vitesse des gaz de 0 à 126, ou -1 pour l'arrêt d'urgence (réinitialise SPEED à 0)

"1" = DIRECTION : 1=avant, 0=arrière. Le réglage de la direction lorsque la vitesse = 0 ou la vitesse = -1 n'affecte que la directionnalité de l'éclairage de la cabine pour un train à l'arrêt

">" = Je suis la fin de cette commande

Si la commande a réussi, le moniteur série doit répondre par : <T 1 20 1> signifiant :

"<" = Commencer la commande DCC++

"T" = (T majuscule) La commande DCC++ Cab a été envoyée depuis DCC++ BaseStation

"1" = le registre 1 a été modifié

"20" = réglé sur la vitesse 20

"1" = sens avant

">" = Fin de la commande DCC++

### ####FONCTIONS DE LA CABINE####

**Le format des fonctions CAB est <f CAB BYTE1 [BYTE2]>**

• Cela active et désactive les fonctions du décodeur de la locomotive

• F0-F28 (F0 est parfois appelé FL)

• REMARQUE : les demandes de réglage sont transmises directement au décodeur de la locomotive

• l'état actuel des fonctions de la locomotive n'est pas enregistré par la DCC++ BaseStation

• Tous les groupes de fonctions sont définis en même temps selon les normes NMRA DCC

#### #####Pour activer les fonctions F0-F4 on=(1) ou off=(0) : <f CAB BYTE1 [BYTE2]>#####

• < = Commencer la commande DCC++

• f = (f minuscule) Cette commande est pour une fonction CAB, c'est-à-dire : Feux, klaxon, cloche

• CAB : l'adresse courte (1-127) ou longue (128-10293) du décodeur de locomotive

• OCTET1 : 128 + F1\*1 + F2\*2 + F3\*4 + F4\*8 + F0\*16

• AJOUTEZ ceux que vous voulez ACTIVER ensemble

• Ajouter 1 pour F1 ON

• Ajouter 2 pour F2 ON

• Ajouter 4 pour F3 ON

• Ajouter 8 pour F4 ON

• Ajouter 16 pour F0 ON

• 128 seul éteint F0-F4

• BYTE2 : omis

• > = Terminer la commande DCC++

Pour créer BYTE1 ajoutez les valeurs de ce que vous voulez ON ensemble,

ceux que vous souhaitez désactiver ne sont pas ajoutés à la valeur de base de 128.

F0 (Lumière)=16, F1 (Cloche)=1, F2 (Corne)=2, F3=4, F4=8

Tout éteint = 128

Allumé 128 + 16 = 144

Lumière et cloche sur 128 + 16 + 1 = 145

Feu et klaxon au 128 + 16 + 2 = 146

Corne juste 128 + 2 = 130

Si la lumière est allumée (144), alors vous allumez la cloche avec la lumière (145), la cloche s'éteint mais la lumière s'allume (144)

Répartition pour cet exemple <f 3265 144>

"<" = Commencer la commande DCC++

"f" = (minuscule f) Cette commande est pour une fonction CAB, c'est-à-dire : feux, klaxon, cloche

"3265" = CAB : l'adresse courte (1-127) ou longue (128-10293) du décodeur de locomotive

"144" = Allumer le phare

">" = Fin de la commande DCC++

#### #####Pour activer les fonctions F5-F8 on=(1) ou off=(0) : <f CAB BYTE1 [BYTE2]>#####

• < = Commencer la commande DCC++

• f = (f minuscule) Cette commande est destinée à une fonction CAB.

• OCTET1 : 176 + F5\*1 + F6\*2 + F7\*4 + F8\*8

• AJOUTER 176 + ceux que vous voulez ACTIVER ensemble

• Ajouter 1 pour F5 ON

• Ajouter 2 pour F6 ON

• Ajouter 4 pour F7 ON

• Ajouter 8 pour F8 activé

• 176 seul éteint F5-F8

• BYTE2 : omis

• = Fin de la commande DCC++

#### #####Pour activer les fonctions F9-F12 on=(1) ou off=(0) : <f CAB BYTE1 [BYTE2]>#####

• < = Commencer la commande DCC++

• f = (f minuscule) Cette commande est destinée à une fonction CAB.

• OCTET1 : 160 + F9\*1 +F10\*2 + F11\*4 + F12\*8

• AJOUTEZ 160 + ceux que vous voulez ACTIVER ensemble

• Ajouter 1 pour F9 ON

• Ajouter 2 pour F10 ON

• Ajouter 4 pour F11 ON

• Ajouter 8 pour F12 activé

• 160 seul éteint F9-F12

• BYTE2 : omis

• = Fin de la commande DCC++

#### #####Pour activer les fonctions F13-F20 on=(1) ou off=(0) : <f CAB BYTE1 [BYTE2]>#####

• < = Commencer la commande DCC++

• f = (f minuscule) Cette commande est destinée à une fonction CAB.

• OCTET1 : 222

• OCTET2 : F13\*1 + F14\*2 + F15\*4 + F16\*8 + F17\*16 + F18\*32 + F19\*64 + F20\*128

• AJOUTEZ ceux que vous voulez ACTIVER ensemble

• Ajouter 1 pour F13 ON

• Ajouter 2 pour F14 ON

• Ajouter 4 pour F15 ON

• Ajouter 8 pour F16 ON

• Ajouter 16 pour F17 ON

• Ajouter 32 pour F18 ON

• Ajouter 64 pour F19 ON

• Ajouter 128 pour F20 ON

• 0 seul éteint F13-F20

• = Fin de la commande DCC

#### #####Pour activer les fonctions F21-F28 on=(1) ou off=(0) : <f CAB BYTE1 [BYTE2]>#####

• < = Commencer la commande DCC++

• f = (f minuscule) Cette commande est destinée à une fonction CAB.

• OCTET1 : 223

• OCTET2 : F21\*1 + F22\*2 + F23\*4 + F24\*8 + F25\*16 + F26\*32 + F27\*64 + F28\*128

• AJOUTEZ ceux que vous voulez ACTIVER ensemble

• Ajouter 1 pour F21 ON

• Ajouter 2 pour F22 ON

• Ajouter 4 pour F23 ON

• Ajouter 8 pour F24 ON

• Ajouter 16 pour F25 ON

• Ajouter 32 pour F26 ON

• Ajouter 64 pour F27 ON

• Ajouter 128 pour F28 ON

• 0 seul éteint F21-F28

• > = Fin de la commande DCC++

#### #####Renvoi : AUCUN#####

• Les fonctions CAB n'ont pas de retour

• Les fonctions CAB ne sont pas stockées dans la DCC++ BaseStation

• Chaque groupe n'affecte pas les autres groupes. Pour activer F0 et F22, vous devez envoyer deux commandes distinctes à la DCC++ BaseStation. Un pour F0 activé et un autre pour F22 activé.

### ####DECODEURS ACCESSOIRES STATIONNAIRES ET AIGUILLAGES####

La STATION DE BASE DCC++ peut suivre la direction de tout aiguillage contrôlé par un décodeur d'accessoire fixe DCC une fois qu'il est défini (configuré).

Tous les décodeurs qui ne sont pas dans une locomotive sont des décodeurs d’accessoires, y compris les aiguillages.

En plus d'être définis, tous les aiguillages, ainsi que tous les autres accessoires DCC connectés de cette manière, peuvent toujours être commandés à l'aide de la commande d'accessoire DCC BASE STATION :

#####Vous contrôlez un décodeur d'accessoire\*\* avec \*\*< a ADDRESS SUBADDRESS ACTIVATE >#####

• < : début de la commande DCC++

• a (minuscule a) cette commande est destinée à un décodeur d'accessoires

• ADRESSE : l'adresse primaire du décodeur contrôlant cet aiguillage (0-511)

• SOUS-ADRESSE : la sous-adresse du décodeur contrôlant cet aiguillage (0-3)

• ACTIVER : (0) (Désactiver, Off, Unthrown) ou (1) (Activer, On, Thrown)

• > : Terminer la commande DCC++

• Cependant, cette commande générale envoie simplement le paquet d'instructions DCC approprié à la voie principale pour faire fonctionner les accessoires connectés. Il ne stocke ni ne conserve aucune information concernant l'état actuel de cet accessoire.

#### #####Définir (mettre en place) un aiguillage#####

Pour que la Station de base DCC++ stocke et conserve la direction des aiguillages connectés au DCC, ainsi que pour invoquer automatiquement la commande <a> requise si nécessaire, commencez par définir/modifier/supprimer ces aiguillages en utilisant les variantes suivantes de la commande "T" :

• Commande pour définir un aiguillage **: < T ID ADDRESS SUBADDRESS >** :

• Crée un nouvel ID d'aiguillage, avec l'ADRESSE et la SOUS-ADRESSE spécifiées si l'ID d'aiguillage existe déjà, il est mis à jour (écrasé) avec la nouvelle ADRESSE et SOUS-ADRESSE spécifiées

• Renvoie : < O > en cas de succès et < X > en cas d'échec (par exemple, mémoire insuffisante)

• Commande pour Supprimer un aiguillage < T ID > :

• Supprime la définition d'un aiguillage avec cet ID

• Renvoie : **< O >** en cas de succès et **< X >** en cas d'échec (par exemple, l'ID n'existe pas)

• Commande pour lister tous les aiguillages définis : < T > :

• Liste tous les aiguillages définis.

• Retourne : **< H ID ADDRESS SUBADDRESS THROW >** pour chaque aiguillage défini ou < X > si aucun aiguillage n'a été défini ou enregistré.

• ID : L'ID numérique (0-32767) de l'aiguillage à contrôler.

• (Vous choisissez l'ID et ils sont partagés entre les aiguillages, les capteurs et les sorties)

• ADRESSE : l'adresse primaire du décodeur contrôlant cet aiguillage (0-511)

• SOUS-ADRESSE : la sous-adresse du décodeur contrôlant cet aiguillage (0-3)

Une fois que tous les aiguillages ont été correctement définis, utilisez la commande <E> pour stocker leurs définitions dans l'EEPROM.

Si vous faites plus tard des modifications/ajouts/suppressions aux définitions d'aiguillage, vous devez invoquer la commande <E> si vous voulez que ces nouvelles définitions soient mises à jour dans l'EEPROM.

Vous pouvez également EFFACER tout (aiguillages, capteurs et sorties) stocké dans l'EEPROM en invoquant la commande <e> (e minuscule).

**(Il n'y a pas d'annulation de suppression)**

Exemple : Vous avez un aiguillage sur votre ligne principale allant à l'industrie des entrepôts. L'aiguillage est contrôlé par un décodeur d'accessoire avec une adresse de 123 et est câblé à la sortie 3. Vous voulez qu'il ait l'ID de 10.

Vous enverriez la commande suivante à la DCC++ BaseStation

**< T 10 123 3 >**

• Cette commande signifie :

• < : début de la commande DCC++

• T : (T majuscule) Définir un aiguillage

• 10 : numéro d'identification que je suis en train de configurer pour utiliser cet aiguillage

• 123 : L'adresse des décodeurs d'accessoires

• 3 : L'aiguillage est câblé à la sortie 3

• : Terminer la commande DCC++

DCC++ doit renvoyer < O > Signification Commande réussie

Ensuite, vous enverriez la commande suivante à la DCC++ BaseStation

< E >

• Cette commande signifie :

• < : début de la commande DCC++

• E : (E majuscule) Stocke (sauvegarde) cette définition dans l'EEPROM

• : Terminer la commande DCC++

DCC++ doit renvoyer < O > Signification Commande réussie

#### #####Contrôle d'un aiguillage défini#####

• Définit l'ID d'aiguillage sur la position « lancé » (dévié) ou « non lancé » (droit)

• Le format de l'aiguillage est < T ID THROW >

• ID : L'ID numérique (0-32767) que vous avez donné à l'aiguillage à contrôler lorsque vous l'avez défini.

• THROW : 0 (non lancé) ou 1 (lancé)

• Retourne : < H ID THROW > ou < X > si l'ID d'aiguillage n'existe pas

Exemple Suite d'en haut :

Pour lancer l'aiguillage 10 afin qu'une locomotive puisse se diriger vers la voie d'évitement de l'entrepôt, vous devez envoyer la commande suivante.

**< T 10 1 >**

• Cette commande signifie :

• < : début de la commande DCC++

• T : (T majuscule) Lancer un aiguillage.

• 10 : Numéro d'identification de l'aiguillage défini que je veux contrôler.

• 1 : Réglez l'aiguillage sur la position Lancer (allumé, activé).

• : Terminer la commande DCC++

DCC++ devrait retourner < H 10 1 > Signification Commande Lancer l'aiguillage 10 a réussi REMARQUE : Ce retour peut répertorier tous les aiguillages et leurs directions

### ####CAPTEURS####

DCC++ BaseStation prend en charge les entrées de capteur qui peuvent être connectées à n'importe quelle broche Aruidno non utilisée par ce programme. Les capteurs peuvent être de tout type (infrarouge, magnétique, mécanique...). La seule exigence est que lorsqu'il est "activé", le capteur doit forcer la broche Arduino spécifiée LOW (c'est-à-dire à la terre), et lorsqu'elle n'est pas activée, cette broche doit rester HIGH (c'est-à-dire 5V), ou être autorisée à flotter HIGH si l'Arduino est utilisé. La résistance pull-up interne de la broche est spécifiée.

Pour garantir des niveaux de tension appropriés, une partie du circuit du capteur DOIT être reliée à la même terre que celle utilisée par l'Arduino.

Le code du capteur utilise un lissage exponentiel pour « anti-rebondir » les pointes générées par les commutateurs mécaniques et les transistors. Cela évite d'avoir à créer un circuit de lissage pour chaque capteur. Vous devrez peut-être modifier les paramètres dans Sensor.cpp par essais et erreurs pour vos capteurs spécifiques.

Pour que cette esquisse surveille une ou plusieurs broches Arduino pour les déclencheurs de capteur, commencez par définir/modifier/supprimer les définitions de capteur à l'aide de la variation suivante de la commande "S":

• < S ID PIN PULLUP > : Crée un nouvel ID de capteur, avec le PIN et le PULLUP spécifiés si l'ID de capteur existe déjà, il est mis à jour avec le PIN et le PULLUP spécifiés (Vous choisissez le numéro).

• Renvoie : < O > en cas de succès et < X > en cas d'échec (par exemple, mémoire insuffisante)

• < S ID > : Supprime la définition de l'ID du capteur

• Renvoie : < O > en cas de succès et < X > en cas d'échec (par exemple, l'ID n'existe pas)

• < S > : répertorie tous les capteurs définis

• Renvoie : < Q ID PIN PULLUP > pour chaque capteur défini ou si aucun capteur n'est défini

ID : L'ID numérique (0-32767) du capteur

(Vous choisissez l'ID et ils sont partagés entre les aiguillages, les capteurs et les sorties)

PIN : le numéro de broche Arduino auquel le capteur est connecté sur la carte Arduino.

PULLUP : 1 = utiliser la résistance pull-up interne pour le PIN, 0 = ne pas utiliser la résistance pull-up interne pour le PIN

Une fois que tous les capteurs ont été correctement définis, utilisez la commande <E> (E majuscule) pour stocker leurs définitions dans l'EEPROM.

Si vous apportez ultérieurement des modifications/ajouts/suppressions aux définitions de capteur, vous devez appeler la commande <E> (E majuscule) si vous souhaitez que ces nouvelles définitions soient mises à jour dans l'EEPROM.

Vous pouvez également effacer tout (aiguillages, capteurs et sorties) stockés dans l'EEPROM en invoquant la commande <e> (e minuscule).

(Il n'y a PAS de UN Delete)

Tous les capteurs définis comme ci-dessus sont vérifiés de manière répétée et séquentielle dans la boucle principale de cette esquisse. Si une broche de capteur est passée d'un état à un autre, l'un des messages série suivants est généré :

• < Q ID > - pour la transition de l'ID du capteur de l'état HIGH à l'état LOW (c'est-à-dire que le capteur est déclenché)

• < q ID > - pour la transition de l'ID du capteur de l'état BAS à l'état HAUT (c'est-à-dire que le capteur n'est plus déclenché)

Selon que le capteur physique agit comme un "déclencheur d'événement" ou un "capteur de détection", vous pouvez décider d'ignorer le retour < q ID > et de ne réagir qu'aux déclencheurs < Q ID >.

### ####BROCHES DE SORTIE ARDUINO####

DCC++ BaseStation prend en charge le contrôle OUTPUT facultatif de toutes les broches Arduino inutilisées à des fins personnalisées. Les broches peuvent être activées ou désactivées. La valeur par défaut est de définir les broches ACTIVE HIGH et les broches INACTIVE LOW. Cependant, ce comportement par défaut peut être inversé pour n'importe quelle broche, auquel cas ACTIVE=LOW et INACTIVE=HIGH.

Les définitions et l'état (ACTIF/INACTIF) des broches sont conservés dans l'EEPROM et restaurés à la mise sous tension.

La valeur par défaut consiste à définir chaque broche définie sur active ou inactive en fonction de son état restauré. Cependant, le comportement par défaut peut être modifié afin que n'importe quelle broche puisse être forcée d'être active ou inactive lors de la mise sous tension, quel que soit son état précédent avant la mise hors tension.

Pour que DCC++ BaseStation utilise une ou plusieurs broches Arduino en tant que sorties personnalisées, commencez par définir/modifier/supprimer les définitions de sortie à l'aide de la variante suivante de la commande « Z » :

• < Z ID PIN IFLAG > : Crée un nouvel ID de sortie, avec les valeurs PIN et IFLAG spécifiées.

o si l'ID de sortie existe déjà, il est mis à jour avec le code PIN et l'IFLAG spécifiques.

o Remarque : l'état de la sortie sera immédiatement défini sur ACTIVE/INACTIVE et la broche sera définie sur HIGH/LOW en fonction de la valeur IFLAG spécifiée (voir ci-dessous).

o Renvoie : < O > en cas de succès et < X > en cas d'échec (par exemple, mémoire insuffisante).

• < Z ID > : Supprime la définition de l'ID de sortie

• Renvoie : < O > en cas de succès et < X > en cas d'échec (par exemple, l'ID n'existe pas)

• < Z > : répertorie toutes les broches de sortie définies

o Renvoie : < Y ID PIN IFLAG STATE > pour chaque broche de sortie définie ou < X > si aucune broche de sortie n'est définie.

ID : L'ID numérique (0-32767) de la sortie

(Vous choisissez l'ID et ils sont partagés entre les aiguillages, les capteurs et les sorties)

PIN : Le numéro de broche Arduino à utiliser pour la sortie.

ETAT : L'état de la sortie (0=INACTIVE / 1=ACTIVE)

IFLAG : définit le comportement opérationnel de la sortie en fonction des bits 0, 1 et 2 comme suit :

IFLAG, bit 0 : 0 = opération vers l'avant (ACTIVE=HIGH / INACTIVE=LOW)

 1 = fonctionnement inversé (ACTIF=BAS / INACTIF=HAUT)

IFLAG, bit 1 : 0 = état de la broche restauré à la mise sous tension sur ACTIF ou INACTIF

 en fonction de l'état avant la mise hors tension.

 1 = état de la broche définie à la mise sous tension ou lors de sa première création,

 à ACTIF ou INACTIF selon IFLAG, bit 2

IFLAG, bit 2 : 0 = état de la broche définie sur INACTIVE lors de la mise sous tension ou lors de la première création

 1 = état de la broche définie sur ACTIVE lors de la mise sous tension ou lors de la première création

Une fois que toutes les sorties ont été correctement définies, utilisez la commande <E> E majuscule pour stocker leurs définitions dans l'EEPROM.

Si vous apportez ultérieurement des modifications/ajouts/suppressions aux définitions de sortie, vous devez appeler la commande <E> si vous souhaitez que ces nouvelles définitions soient mises à jour dans l'EEPROM.

Vous pouvez également EFFACER tout (aiguillages, capteurs et sorties) stocké dans l'EEPROM en invoquant la commande <e> (e minuscule).

(Il n'y a pas d'annulation de suppression)

Pour modifier l'état des sorties définies, utilisez :

• < Z ID STATE > : définit l'ID de sortie sur l'état ACTIF ou INACTIF

• Renvoie : < Y ID STATE >, ou < X > si l'ID de sortie n'existe pas

o ID : ID numérique (0-32767) de la sortie définie à contrôler

o ETAT : L'état de la sortie (0=INACTIVE / 1=ACTIVE)

Lorsqu'il est contrôlé en tant que tel, l'Arduino met à jour et stocke la direction de chaque sortie dans l'EEPROM afin qu'elle soit conservée même sans alimentation. Une liste des états actuels de chaque sortie sous la forme <Y ID STATE> est générée par DCC++ BaseStation chaque fois que la commande d'état <s> est invoquée. Cela fournit un moyen efficace d'initialiser l'état de toutes les sorties surveillées ou contrôlées par une interface distincte ou un programme GUI.

## ###Commandes de programmation du décodeur de locomotive###

### ####PROGRAMMATION sur la PISTE PRINCIPALE####

#### #####ÉCRIRE L'OCTET DE CV AU DÉCODEUR DE LOCOMOTIVE SUR LA PISTE PRINCIPALE#####

Écrit, sans aucune vérification, une variable de configuration BYTE dans le décodeur de locomotive sur la piste d'opérations principale.

• Le format d'écriture CV BYTE est : **< w CAB CV VALUE >**

• CAB : L'adresse courte (1-127) ou longue (128-10293) du décodeur de locomotive

• CV : le numéro de l'emplacement de mémoire de la variable de configuration dans le décodeur dans lequel écrire (1-1024)

• VALUE : La valeur à écrire dans l'emplacement mémoire de la variable de configuration (0-255)

• Retours : AUCUN

#### #####ÉCRIRE LE BIT DE CV AU DÉCODEUR DE LOCOMOTIVE SUR LA PISTE PRINCIPALE#####

Écrit, sans aucune vérification, un seul bit dans une variable de configuration BIT dans le décodeur d'une locomotive sur la piste d'opérations principale.

• Le format d'écriture de CV BIT est : **< b CAB CV BIT VALUE >**

• CAB : l'adresse courte (1-127) ou longue (128-10293) du décodeur de locomotive

• CV : le numéro de l'emplacement mémoire de la variable de configuration dans le décodeur dans lequel écrire (1-1024)

• BIT : le numéro de bit du registre de variables de configuration à écrire (0-7)

• VALUE : la valeur du bit à écrire (0-1)

• Retours : AUCUN

### #### PROGRAMMATION sur la PISTE DE PROGRAMMATION- ####

#### #####ÉCRIRE L'OCTET DE CV AU DÉCODEUR DE LOCOMOTIVE SUR LA PISTE DE PROGRAMMATION#####

Écrit, puis vérifie, une variable de configuration BYTE dans le décodeur d'une locomotive sur la voie de programmation

• Le format d'écriture CV BYTE est : **< W CV VALUE CALLBACKNUM CALLBACKSUB >**

• **CV** : Le numéro de l'emplacement mémoire de la variable de configuration dans le décodeur dans lequel écrire (1-1024 ).

• **VALEUR** : La valeur à écrire dans l'emplacement mémoire de la variable de configuration (0-255).

• **CALLBACKNUM** : Un entier arbitraire (0-32767) qui est ignoré par la station de base et est simplement renvoyé en écho dans la sortie - utile pour les programmes externes qui appellent cette fonction.

• **CALLBACKSUB** : un deuxième entier arbitraire (0-32767) qui est ignoré par la station de base et qui est simplement renvoyé en écho dans la sortie - utile pour les programmes externes (par exemple, l'interface DCC++) qui appellent cette fonction.

• Renvoie : **< r CALLBACKNUM|CALLBACKSUB|Valeur CV >**

• **VALEUR CV** : est un nombre compris entre 0 et 255 tel que lu à partir du CV demandé, ou -1 si la lecture de vérification échoue.

#### #####ÉCRIRE LE BIT DE CV AU DÉCODEUR DE LOCOMOTIVE SUR LA PISTE DE PROGRAMMATION#####

Écrit, puis vérifie, une variable de configuration BIT dans le décodeur d'une locomotive sur la voie de programmation

• Le format d'écriture de CV BIT est : < B CV BIT VALUE CALLBACKNUM CALLBACKSUB >

• CV : Le numéro de l'emplacement mémoire de la variable de configuration dans le décodeur dans lequel écrire (1-1024).

• BIT : Le numéro de bit de l'emplacement mémoire de la variable de configuration à écrire (0-7).

• VALEUR : La valeur du bit à écrire (0-1).

• CALLBACKNUM : Un entier arbitraire (0-32767) qui est ignoré par la station de base et est simplement renvoyé en écho dans la sortie - utile pour les programmes externes qui appellent cette fonction.

• CALLBACKSUB : Un deuxième entier arbitraire (0-32767) qui est ignoré par la station de base et qui est simplement renvoyé en écho dans la sortie - utile pour les programmes externes (par exemple, l'interface DCC++) qui appellent cette fonction.

• Renvoie : < r CALLBACKNUM|CALLBACKSUB|CV BIT VALUE>

• CV VALUE est un nombre compris entre 0 et 1 tel qu'il est lu à partir du bit CV demandé, ou -1 si la lecture de vérification échoue.

#### #####LIRE L'OCTET DE CONFIGURATION DE VARIABLE DU DÉCODEUR DE LOCOMOTIVE SUR LA PISTE DE PROGRAMMATION#####

Lit une variable de configuration à partir du décodeur d'une locomotive sur la voie de programmation.

• Lire CV BYTE Le format est :< R CV CALLBACKNUM CALLBACKSUB >

• CV : Le numéro de l'emplacement mémoire de la variable de configuration dans le décodeur à lire (1-1024).

• CALLBACKNUM : Un entier arbitraire (0-32767) qui est ignoré par la station de base et est simplement renvoyé en écho dans la sortie - utile pour les programmes externes qui appellent cette fonction.

• CALLBACKSUB : Un deuxième entier arbitraire (0-32767) qui est ignoré par la station de base et qui est simplement renvoyé en écho dans la sortie - utile pour les programmes externes (par exemple, l'interface DCC++) qui appellent cette fonction.

• Renvoie : \*\*< r CALLBACKNUM|CALLBACKSUB|CV VALUE>

• CV VALUE est un nombre compris entre 0 et 255 tel que lu à partir du CV demandé, ou -1 si la lecture n'a pas pu être vérifiée.

# REMARQUES:

• En raison du fonctionnement du langage MarkDown, j'ai dû mettre un espace entre le < et la lettre de commande, sinon le texte est lu comme du code HTML. Un espace après < , ou avant > dans la commande est correct mais pas obligatoire. Les espaces entre les valeurs dans la commande sont obligatoires.

• Toutes les commandes et tous les formats ont été tirés des commentaires des fichiers DCCpp\_Uno.ino et Related \*.cpp créés par Gregg E. Berman et les fichiers peuvent être trouvés ici Station de base DCC++

https://github.com/DccPlusPlus/BaseStation/wiki/Commands-for-DCCpp-BaseStation

**Liens vers le document original**

* **[Home](https://github.com/DccPlusPlus/BaseStation/wiki)**
* [**Commands for DCCpp BaseStation**](https://github.com/DccPlusPlus/BaseStation/wiki/Commands-for-DCCpp-BaseStation)
* [**DCC Introduction**](https://github.com/DccPlusPlus/BaseStation/wiki/DCC-Introduction)
* [**Diagnostics D Command**](https://github.com/DccPlusPlus/BaseStation/wiki/Diagnostics---D---Command)
* [**Getting Started With DCC Hardware**](https://github.com/DccPlusPlus/BaseStation/wiki/Getting-Started-With-DCC---Hardware)
* [**What is DCC Plus Plus**](https://github.com/DccPlusPlus/BaseStation/wiki/What-is-DCC--Plus-Plus)
* [**What is DCC Plus Plus BaseStation**](https://github.com/DccPlusPlus/BaseStation/wiki/What-is-DCC--Plus-Plus-BaseStation)

Table des matières

[Commandes DCC++ et leur structure 1](#_Toc94101152)

[###COMMANDES À LETTRE UNIQUE### 1](#_Toc94101153)

[###Commandes de fonctionnement du décodeur de locomotive (CAB)### 2](#_Toc94101154)

[####MANETTE DE VITESSE#### 2](#_Toc94101155)

[####FONCTIONS DE LA CABINE#### 2](#_Toc94101156)

[####DECODEURS ACCESSOIRES STATIONNAIRES ET AIGUILLAGES#### 6](#_Toc94101157)

[####CAPTEURS#### 8](#_Toc94101158)

[####BROCHES DE SORTIE ARDUINO#### 9](#_Toc94101159)

[###Commandes de programmation du décodeur de locomotive### 11](#_Toc94101160)

[####PROGRAMMATION sur la PISTE PRINCIPALE#### 11](#_Toc94101161)

[#### PROGRAMMATION sur la PISTE DE PROGRAMMATION- #### 11](#_Toc94101162)

[REMARQUES: 13](#_Toc94101163)

[**Liens vers le document original** 13](#_Toc94101164)