FT-891 Librairie d'émulation CAT
Version 1.1
John Price – WA2FZW

Table des Matières

Introduction

Cette bibliothèque fournit un moyen d'ajouter la fonctionnalité CAT (émetteur-récepteur assisté par ordinateur) aux applications basées sur Arduino (ou un autre microprocesseur). Je l'ai utilisé dans un VFO numérique basé sur ESP32 et mon interface simple pour l'audio numérique, qui fonctionne sur un Arduino Nano.

Cela a été développé à l'origine pour un projet VFO numérique par moi, Glenn (VK3PE) et Jim (G3ZQC).

Le langage de contrôle CAT utilisé est celui utilisé par le Yaesu FT-891 avec quelques aménagements. Pourquoi cette langue au lieu des variations linguistiques Kenwood les plus couramment utilisées?

Il y a plusieurs raisons :

- J'ai un FT-891 et je connais parfaitement les capacités de contrôle CAT.
- Le programme de contrôle CAT principal que j'utilise est *DxCommander* et lorsque j'ai reçu le FT-891 pour la première fois, j'ai découvert qu'il y avait un certain nombre de problèmes lors de l'utilisation des deux ensemble. Certains de ces problèmes étaient des malentendus de l'auteur de *DxCommander*, et d'autres étaient dus à des bogues dans le logiciel Yaesu. En travaillant avec Dave (AA6YQ, l'auteur de DxCommander), nous avons soit résolu tous les problèmes, soit inventé des solutions de contournement.
- Le logiciel de gestion de messages a été écrit à 90%, car je l’ai utilisé dans deux autres projets, donc pas besoin de réinventer la roue.

Historique des modifications

Dans la version 1.1, la bibliothèque a été modifiée de sorte que l'action de saisir ou de déverrouiller l'émetteur à la réception d'un «*TXn*»; La commande est désormais facultative. Cela a été fait en ajoutant une 2ème version de la fonction *begin* et est expliqué plus en détail ci-dessous.

Implémentation CAT

Bien que l'implémentation CAT utilise le langage Yaesu FT-891, seul un petit sous-ensemble du langage est implémenté. Certaines

fonctions ne sont pas implémentées simplement parce que les radios avec lesquelles moi et quelques autres prévoyons de les utiliser ne peuvent pas les prendre en charge; par exemple, des filtres réglables par logiciel.

D'autres sont omis simplement parce qu'ils le sont; nous ne fournissons pas la capacité de sauvegarder et de transmettre des messages vocaux ou CW, et un manipulateur intégré n’est pas mis en œuvre.

Architecture générale

Il existe deux composants principaux de la bibliothèque; une table d'état, qui contient l'état actuel de la radio (simulée) (fréquence de fonctionnement, mode, etc.) et le gestionnaire de messages lui-même.

Le logiciel est conçu pour être 100% compatible avec le programme *DxCommander*, mais devrait fonctionner avec d’autres applications compatibles avec le contrôle CAT telles que *N1MM* + ou *WSJT-X* par exemple à condition que ces programmes n’aient pas certains des problèmes que nous avons corrigés dans *DxCommander*.

Gestionnaire de messages

DxCommander utilise un schéma d'interrogation pour demander l'état de la radio et envoie également des commandes à la radio. L'utilisation de cette approche signifie que *DxCommander* peut initier des changements dans l'état de la radio, ou lorsque des changements dans l'état de la radio sont effectués par l'application utilisant cette bibliothèque ou par une opération manuelle des commandes de la radio *DxCommander* sera informé de ces changements.

Par exemple, dans le programme VFO pour lequel cela a été développé, on peut changer de fréquence non seulement via le contrôle CAT mais en utilisant un encodeur. Lorsque l'encodeur est déplacé, l'application utilise la méthode *SetFA()* dans la bibliothèque pour mettre à jour la table d'état interne de la bibliothèque, et la prochaine fois que *DxCommander* demandera la fréquence VFO-A, il obtiendra la valeur mise à jour.

Toutes les commandes et demandes d'état du FT-891 ne sont pas implémentées. L'objectif de ce projet a été d'utiliser le VFO pour mettre à niveau les radios héritées, donc beaucoup de commandes ne

peuvent tout simplement pas être implémentées comme elles pourraient l'être dans une radio moderne contrôlée par logiciel, cependant, certains des messages doivent être traités pour garder *DxCommander* heureux.

Les commandes/requêtes d'état reconnues (par ordre alphabétique) sont :

AB Copie VFO-A vers VFO-B
AI (Données 0 ou 1) Activer ou désactiver les informations automatiques
BA Copie VFO-B vers VFO-A
BS Sélection de bande BS (pas encore implémenté)
EX Commande du menu (ne fait rien mais doit être ici)
FA Régler ou demander la fréquence VFO-A
FB Définir ou demander la fréquence VFO-B
ID Request Radio's ID (0650 pour le FT-891)
IF Demande/réponse d'information
IS Définir ou demander un décalage IF
MD Set ou mode de demande (USB, LSB, CW, etc.)
NA Définir ou demander un décalage IF étroit
OI Demande/réponse d'information sur la bande opposée OI
RIC Demande alternative pour le statut de fractionnement
RM Compteur de lecture
SH Définir ou demander la bande passante IF
SM Lire S-mètre
ST (Données 0-2) Mode Split désactivé, activé ou activé + 5KHz
SV Swap VFO
TX Définir ou demander l'état de transmission/réception

Les descriptions de certains messages sont un peu trompeuses, mais le manuel CAT Yaesu FT891 clarifiera ce qu'ils font réellement.
Comme certains des messages de commande ne peuvent pas être réellement implémentés sur les radios héritées avec lesquelles nous avons joué, comme celles liées aux paramètres de filtre, cependant, DxCommander peut essayer de définir ces paramètres et il demande leur statut dans sa séquence d'interrogation de routine, ils doivent donc être manipulés bien qu'ils ne fassent rien.

Notez que si le réglage de quelque chose ne peut pas réellement changer quelque chose dans l'application utilisateur, la commande pour le faire sera ignorée. Cela se fait en commentant le code pour affecter la modification dans le fichier *FT891_CAT.cpp*. Par exemple, l'instruction case dans la fonction ProcessCmd pour définir le décalage IF comme distribué ressemble à ceci :

```
case MSG_IS: // Set IF shift & status
// strncpy ( tempBuff, &dataBuff[1], 5 ); // Extract the offset amount
// radioStatus.IS_VALUE = atoi ( tempBuff ); // Set value in the structure
// radioStatus.IS_STAT = dataBuff[0] - '0'; // Set on/off status break;
```

En laissant le code intact et en le commentant simplement, on peut facilement l'activer pour leur application.

L'architecture de base du gestionnaire de messages consiste en une table de consultation qui est utilisée pour traduire le message alphanumérique en un nombre et une paire d'instructions «switch» pour déterminer quoi faire avec le message; un pour les messages de type commande et un pour les demandes d'état.

Cette approche permet d'ajouter assez facilement des messages à la liste dont vous pourriez avoir besoin pour votre application particulière ou de modifier le code pour utiliser un langage de commande différent tel que l'une des variantes de Kenwood (qui sont très similaires au langage FT-891).

Tableau d'état

Comme mentionné précédemment, le gestionnaire de messages maintient une table contenant l'état actuel des paramètres qui peuvent être définis et/ou demandés par les messages répertoriés ci-dessus.

Chaque fois que *DxCommander* modifie la valeur de l'un des paramètres (uniquement ceux qui peuvent réellement être modifiés dans la radio), le tableau est mis à jour. Lorsque DxCommander demande l'état de quelque chose, la réponse est fournie en fonction du contenu du tableau.

Pour les paramètres qui ne peuvent pas réellement être modifiés, la table contient des valeurs qui rendront *DxCommander* heureux lorsqu'il les demandera.

L'application utilisant cette bibliothèque peut également modifier de nombreux paramètres via des appels de fonction à la bibliothèque de contrôle CAT. Par exemple, quand on déplace l'encodeur pour changer la fréquence VFO-A, le programme VFO appelle *CAT.SetFA()* pour changer la valeur VFO-A stockée dans la bibliothèque.

L'application utilisatrice devra vérifier régulièrement l'état des choses qui pourraient être modifiées à la suite d'une commande

reçue. Par exemple, il appelle la fonction *CAT.GetFA()* pour voir si la fréquence VFO-A a été modifiée via une commande CAT.

Il existe des méthodes d'obtention et de définition similaires pour le mode, la fréquence VFO-B, l'état de partage, etc.

Fonctions principales du logiciel

Il n'y a que quelques fonctions associées au traitement des commandes:

FT891_CAT	La fonction constructeur qui crée l'objet du module de contrôle CAT.
Begin	Cette fonction est surchargée (elle peut être appelée de plusieurs façons) et sera expliquée plus en détail ci-dessous.
CheckCAT	L'application doit appeler cette fonction périodiquement (dans la fonction *loop*) pour voir s'il y a une nouvelle commande à traiter. Il renvoie une indication *True* ou *False* si une nouvelle commande a été traitée.
GetMessage	Cette fonction vérifie les données entrantes sur le port USB et s'il y a des données entrantes, elle les lit simplement.
FindMsg	Cette fonction recherche le message ASCII dans la table de recherche et le convertit en représentation numérique interne.
ParseMsg	Si le message est accompagné de données jointes, cette fonction sépare les données du message.
ProcessCmd	Si le message est une commande, une instruction «switch» se charge de mettre à jour la table d'état et d'effectuer toute autre action qui pourrait être requise (par exemple, allumer ou éteindre l'émetteur).
ProcessStatus	Si le message est une demande d'état, cette fonction formate une réponse appropriée à la demande en fonction des informations de la table d'état et l'envoie à *DxCommander*.
Xtoi	Fonctionne comme la fonction standard atoi C, sauf qu'elle convertit les nombres hexadécimaux (dans divers formats) en nombres entiers.
Init	Cette fonction gère le traitement commun pour les variations de la fonction *begin*.

À l'exception du constructeur, de la fonction begin et de CheckCAT, toutes celles répertoriées ci-dessus sont des fonctions privées qui ne sont pas accessibles depuis l'application utilisateur.

Il existe également un certain nombre de fonctions publiques que l'application peut utiliser pour mettre à jour l'état des choses dans la table d'état. Ceux-ci comprennent actuellement :

```
void SetFA ( uint32_t freq ) // Régler la fréquence du VFO-A
void SetFB ( uint32_t freq ) // Régler la fréquence du VFO-B
void SetMDA ( uint8_t mode ) // Régler le mode VFO-A
void SetMDB ( uint8_t mode ) // Régler le mode VFO-B
void SetTX ( uint8_t tx ) // Régler l'état Emission/réception
void SetST ( uint8_t tx ) // Régler le mode split on ou off
```

Il existe également un certain nombre de fonctions publiques que l'application peut utiliser pour obtenir l'état actuel des choses. Ceux-ci inclus :

```
uint32_t GetFA (); // Obtenir la fréquence du VFO-A
uint32_t GetFB (); // Obtenir la fréquence du VFO-B
uint8_t GetMDA (); // Obtenir le mode du VFO-A
uint8_t GetMDB (); // Obtenir le mode du VFO-B
bool GetTX (); // Obtenir l'état d'émission/réception
bool GetST (); // Obtenir l'état du mode split
```

La fonction *begin*

Dans la version 1.1 de la bibliothèque, j'ai ajouté une deuxième version de la fonction *begin*. Les deux appels de fonction disponibles sont :

```
begin (int txPin, débogage booléen);
begin (débogage booléen);
```

Dans la première version, l'application peut spécifier une broche GPIO qui sera utilisée pour saisir l'émetteur lors d'un «TX1»; ou "TX2;" la commande est reçue du programme de contrôle CAT.

Si la deuxième forme de l'appel de fonction est utilisée, la bibliothèque ne fera pas fonctionner l'émetteur lorsqu'une commande d'émission est reçue, ce sera plutôt à l'application.

Dans les deux cas, l’argument de débogage est facultatif et vaut par défaut «*false*».

Mais que fait l’argument de débogage lorsqu'il est défini sur «*true*», on pourrait se demander.

Les commandes FT-891 ne se terminent pas par un caractère de retour à la ligne ou de retour chariot; le point-virgule est le terminateur. Cependant,

si l'on essaie de déboguer une application à l'aide de la fonction de moniteur série de l'IDE Arduino, les réponses aux commandes et demandes d'état envoyées à l'application via le moniteur série sont beaucoup plus faciles à lire si elles sont terminées par une nouvelle ligne.

Si vous définissez l'argument de débogage sur «*true*», une nouvelle ligne sera ajoutée aux réponses. Notez cependant que le retour à la ligne va confondre *DxCommander*, donc l'argument doit être défini sur «*false*» lorsque l'application est en cours d'utilisation.

Clé de l'émetteur

Comme mentionné dans la section précédente, si une broche GPIO est spécifiée pour être utilisée pour activer ou désactiver la radio qui sera effectuée par la bibliothèque. Dans le fichier d'en-tête de la bibliothèque, vous devrez peut-être modifier deux définitions :

```
#define XMIT_ON HIGH
#define XMIT_OFF LOW
```

Si votre implémentation matérielle touche l'émetteur en définissant une condition LOW sur la broche spécifiée, retournez simplement les définitions.

Définitions requises dans l'application

Il n'y a qu'une ou deux choses que l'application doit fournir à la bibliothèque; le débit à utiliser sur le port série et (en option) le numéro de broche à utiliser pour saisir l'émetteur.

Ceux-ci peuvent être définis comme des symboles ou des variables ou simplement codés en dur dans les appels aux fonctions *Serial.begin()* et *CAT.begin()* (une pratique de codage très médiocre)

Table des messages

Le *msgTable* est un tableau avec une entrée pour chaque message que nous traitons. Chaque entrée de ce tableau est une structure de *msg* qui contient le texte du message ASCII (*Name*), la représentation numérique interne (*ID*) et un octet indiquant si le message est une commande, une demande d'état, ou s'il peut être en fonction de s'il est venu avec des données jointes ou non (*Type*). Les indicateurs de *Type* sont des bits individuels définis comme *MSG_STS* pour le message d'état uniquement, *MSG_CMD* pour un message de commande uniquement, ou *MSG_BOTH* indiquant qu'il peut s'agir de l'un ou l'autre (note, *MSG_BOTH = MSG_STS | MSG_CMD*).

Boîte à suggestion

J'apprécie toute suggestion d'améliorations supplémentaires. N'hésitez pas à m'envoyer un e-mail à WA2FZW@ARRL.net.