



# Documentation de PolySpecteuR 4.0

B. Panneton

2022-03-31

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Fonctionnement général</b>	<b>2</b>
<b>3</b>	<b>Scripts R</b>	<b>7</b>
3.1	DoFluoSpecteuR.R . . . . .	7
3.2	DoRamanSpecteuR.R . . . . .	7
3.3	DoReflectSpecteuR.R . . . . .	8
3.4	DoTransmitSpecteuR.R . . . . .	8
3.5	GetPlanExp.R . . . . .	9
3.6	InitFluoSpecteuR.R . . . . .	10
3.7	InitRamanSpecteuR.R . . . . .	11
3.8	InitReflectSpecteuR.R . . . . .	11
3.9	InitTransmitSpecteuR.R . . . . .	12
3.10	PickFromPlan.R . . . . .	12
3.11	Plots_2_Shiny_MultiLevels.R . . . . .	13
3.12	writeData.R . . . . .	16
3.13	writeYFile.R . . . . .	16
<b>Annexe A</b>		<b>17</b>
	Script d'acquisition avec 3 instruments pour la fluorescence, la transmittance et la réflectance . . . . .	17

# 1 Introduction

**PolySpecteuR 4.0** est une suite de scripts R constituant des blocs à partir desquels une application dédiée d'acquisition de données spectroscopique est construite. Cette suite s'utilise conjointement avec le **SpectrAAC**, une plateforme matérielle d'acquisition de données spectroscopiques développée par Dr Alain Clément du Centre de R&D de St-Hyacinthe d'Agriculture et Agroalimentaire Canada.

La plateforme **SpectrAAC** est flexible et permet l'acquisition de données d'autofluorescence induite, de réflectance, de transmittance et de diffusion Raman. La particularité de cette plateforme est de permettre différentes mesures de spectroscopie sur un même échantillon. Par exemple, pour un projet en particulier, des mesures d'autofluorescence induite à plusieurs longueurs d'onde d'excitation peuvent être couplées à des mesures de réflectance. **PolySpecteuR 4.0** est l'outil logiciel polyvalent permettant d'interagir avec **SpectrAAC** pour piloter l'acquisition des données et constituer une base de données dont le format est compatible avec **InSpectoR**. **InSpectoR** est un outil logiciel d'analyse de données spectroscopiques dont la particularité est de permettre de façon fluide, l'analyse et le développement de modèles chimiométriques s'appuyant simultanément sur plusieurs spectres. Toute la partie logicielle (**PolySpecteuR** et **InSpectoR**) est programmée en R.

Ce document décrit de façon sommaire l'architecture de **PolySpecteuR 4.0**. Le but est de donner une vue d'ensemble facilitant la navigation à travers les différents scripts R qui composent la suite. Le fonctionnement de chaque script R peut être analysé à l'aide des commentaires inclus dans chaque script.

**PolySpecteuR 4.0** est assemblé dans un projet de RStudio. Ce projet regroupe tous les répertoires nécessaires à l'utilisation de suite logicielle. Cela inclut:

- des scripts R dans le sous-répertoire *R*,
- des fichiers de calibration d'instrument dans des sous-répertoires *Calib\_XXX*,
- du code en C dans le sous-répertoire *C*,
- des fichiers décrivant les instruments dans le sous-répertoire *Fichiers\_Instruments*,
- des fichiers décrivant des paramètres d'acquisition dans le sous-répertoire *Fichiers\_Parametres*,
- des fichiers décrivant des plans d'expérience (séquences d'échantillon à analyser) dans le sous-répertoire *Plan\_Exp*,
- différentes librairies (dll, script R, ...) dans le sous-répertoire *Lib\_n\_wrapper*,
- des notes de travail dans le sous-répertoire *Notes*,
- la documentation dans le sous-répertoire *Doc*.

**PolySpecteuR 4.0** a été conçu pour fonctionner dans l'environnement de travail RStudio. Cet environnement permet de créer une interface usager de base avec une console pour

l'interaction avec l'opérateur et un panneau d'affichage pour les graphiques et des interfaces graphiques basées sur [Shiny](#).

## 2 Fonctionnement général

Une session d'acquisition de données peut comporter cinq éléments:

1. Identifier les échantillons;
2. Initialiser les composantes matérielles (spectromètres, lampes, ...);
3. Acquérir les spectres;
4. Stocker les données;
5. Appliquer un ou des modèles aux nouvelles données.

Dans **PolySpecteuR 4.0**, ces cinq éléments sont mis en œuvre à l'aide de scripts dédiés. Pour les éléments 2 et 3, des scripts spécifiques à chaque type de données spectroscopiques (e.g. fluorescence, Raman, ...) sont disponibles. À partir de ces scripts, l'utilisateur peut facilement créer une session d'acquisition de données en écrivant un script R simple.

L'exemple dans la boîte de code plus bas illustre la démarche. Ce script permet l'acquisition de données de diffusion Raman sur un instrument. Les données sont simplement saisies et aucun modèle ne sera appliqué aux données. Dans la première partie du script (*#Initialisation*), on charge des *packages R* et tous les scripts de la suite **PolySpecteuR**, l'initialisation des instruments est lancée, un plan d'expérience est chargé et on définit un nom permettant l'identification du jeu de données. Ce nom permet de lier les fichiers de descripteurs des échantillons (fichiers Y) et les différents fichiers de données spectroscopiques, un fichier par type de mesure. Cette première partie se termine en demandant à l'utilisateur s'il veut modifier les paramètres d'acquisition à chaque itération, ce qui est utile en phase de rodage de protocoles de prise de données.

La deuxième partie du script (*#Phase d'acquisition de données*) est consacrée à l'acquisition et au stockage des données. À chaque itération de la boucle *while*, on traite un échantillon: on choisit un échantillon, l'acquisition des spectres est lancée, les données sont validées par l'opérateur puis les données valides sont enregistrées. Finalement, l'opérateur choisit d'interrompre ou pas la séquence d'acquisition. La dernière partie du script (*#Nettoyage et arrêt du script*) permet de terminer la session. Pour construire un nouveau protocole d'acquisition, l'opérateur n'a qu'à modifier les 2 lignes de codes identifiées par "*#options*". Tout le reste du script d'acquisition reste inchangé. Avec ce schéma d'acquisition, la priorité va aux échantillons dans le sens que pour un échantillon donné, on procède à toutes les mesures en séquence puis on passe à un autre échantillon. D'autres exemples des scripts sont donnés à l'[Annexe A](#).

```

mainRaman <- function()
*****
# INSTRUCTIONS ----
*****
# Script d'acquisition de données avec SpectrAAC-2 lorsqu'on ne fait que du
# Raman.
# Pour utiliser un autre instrument ou plusieurs instruments, il faut modifier
# les 2 lignes avec la mention "#options" à la fin. La première des deux peut
# être remplacée par une ou plusieurs lignes à raison d'une ligne par type
# de mesure désirée. Par exemple, pour faire des mesures de fluorescence
# suivie de mesures de Raman, la première ligne pourrait être remplacée par:
#       F_inst <- InitFluoSpecteurR()
#       R_inst <- InitRamanSpecteurR()
# Chacune des lignes appelant une routine d'initialisation doit commencer
# par un nom unique. Ensuite, il faut mettre à jour les éléments de la
# liste "lesInstruments" définie dans la deuxième ligne. Pour l'exemple
# ci-haut, cela donnerait:
#       lesInstruments <- list(F_inst,R_inst)
# Noter que l'ordre dans cette liste définit l'ordre d'acquisition des
# données.
#
*****
# AUTEUR: Bernard Panneton, Agriculture et Agroalimentaire Canada
# Février 2022
*****
{
  *****
  #Initialisation ----
  *****
  #Paramètres graphiques par défaut et logo comme variables globales.
  op <- par(no.readonly = TRUE)
  logo <- png::readPNG("PolySpecteurR_Logo.png")

  *****
  ## Enlève OOobj si existant----
  if (exists("OOobj", envir = .GlobalEnv)) rm(OOobj, envir=.GlobalEnv)
  *****

  *****
  ## Charger des packages R ----
  *****
  ok <- require("rlang")
  if (!ok) install.packages('rlang')

```

```

ok <- require("utils")
if (!ok) install.packages('utils')
ok <- require("here")
if (!ok) install.packages('here')

# Se placer dans le répertoire R du projet PolySpecteur.
# Pour que ça marche, il faut que le présent script
# soit dans le répertoire R du projet PolySpecteur.
# Cela permet d'utiliser efficacement des chemins
# relatifs pour se rendre dans les différents répertoires
# utilisés.
RPath=here::here()
setwd(RPath)

#####
## Charger les scripts du projet PolySpecteur ----
#####
setwd("R")
files.sources = list.files(pattern=glob2rx("*.*R"),full.names = TRUE)
dum <- sapply(files.sources, source, encoding="UTF-8")
setwd("..")

#####
## Définir les instruments nécessaires et les initialiser ----
#####

R_Inst <- InitRamanSpecteur() #instrument pour Raman #options
if (is.character(R_Inst)) return( "ABANDON") #options

lesInstruments <- list(R_Inst) #options

lestyles <- lapply(lesInstruments, function(I) I$type)

#####
## Définir le plan d'expérience et le chemin pour stocker les données ----
#####
Plan <- GetPlanExp()
dataSetID <-utils::winDialogString(
  "Entrer un identifiant pour les noms de fichier de données",
  as.character(Sys.Date()))
dataPath <- utils::choose.dir(default = "",
  caption = "Choisir un répertoire pour stocker les données.")

```

```

#####
## Permettre la modif des paramètres d'acquisition à chaque échantillon ----
#####
tuneParams <- FALSE
yesno <- utils::winDialog("yesno",
  "Permettre la modification des paramètres à chaque échantillon.")
if (yesno=="YES") tuneParams <- TRUE

#####
#Phase d'acquisition de données ----
#####
goOn <- TRUE
while(goOn){  ## Boucle sur les échantillons----
  letest <- PickFromPlan(Plan)
  if (letest == "OK"){
    k=0
    isValid <- TRUE
    for (t in lestypes){  ### Boucle sur les instruments----
      k <- k+1
      if (t=="Raman"){
        if (isValid) {
          DoRamanSpecteuR(lesInstruments[[k]],Plan, tuneParams)
          dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])
          isValid <- (dum=="OK") & isValid
        }
      }
      if (t=="Fluorescence"){
        if (isValid){
          DoFluoSpecteuR(lesInstruments[[k]], Plan, tuneParams)
          dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])
          isValid <- (dum=="OK") & isValid
        }
      }
      if (t=="Reflectance"){
        if (isValid){
          DoReflectSpecteuR(lesInstruments[[k]], Plan, tuneParams)
          dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])
          isValid <- (dum=="OK") & isValid
        }
      }
      if (t=="Transmittance"){
        if (isValid){
          DoTransmitSpecteuR(lesInstruments[[k]], Plan, tuneParams)

```

```
        dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])
        isValid <- (dum=="OK") & isValid
    }
}
}
## Écriture des données si valides----
if (isValid){
    writeYFile(Plan, dataPath,dataSetID)
    writeData(Plan,lesInstruments,dataPath,dataSetID) #options
}
}

###Option de continuer ou quitter ----
sel <- select.list(c("Oui","Non"), preselect = "Oui",
    title="CONTINUER?",graphics = T)
goOn <- ifelse(sel=="Oui",TRUE,FALSE)
}      #Fin de la boucle sur les échantillons

#####
#Nettoyage et arrêt du script ----
#####
setwd(RPath)
Clean_n_Close(lesInstruments)
}
```



### 3 Scripts R

Le projet **PolySpecteuR** comprend plusieurs scripts **R** qu'on utilise dans un script comme celui dans la boîte de code ci-dessus. Ce script charge tous les scripts de **PolySpecteuR** dans l'environnement R de base dans le bloc identifié *Charger les scripts du projet PolySpecteuR*. Cette section donne une présentation sommaire de ces scripts. Pour une compréhension détaillée, prière de consulter les scripts qui comportent suffisamment de commentaires pour permettre de suivre le déroulement.

#### 3.1 DoFluoSpecteuR.R

1. Appel de la fonction: ***DoFluoSpecteuR(F\_Inst,leplan,tuneParams=FALSE)***
2. Paramètres d'entrée:
  - *F\_Inst* : un environnement d'instrument créé par *InitFluoSpecteuR()*;
  - *leplan* : plan d'expérience créé par *GetPlanExp.R* et modifié par *PickFromPlan.R* le cas échéant.
  - *tuneParams* : permet de modifier le fichier des paramètres à chaque lancement de la fonction.  
TRUE pour permettre la modification, FALSE (valeur par défaut) autrement.
3. Sortie: Aucune. L'environnement *F\_Inst* est modifié pour inclure *Spectres* qui est une liste contenant les spectres. La liste *Spectres* contient deux éléments correspondant aux spectres bruts et aux spectres interpolés-corrigés. Chaque élément de *Spectres* est une liste dont chaque élément correspond aux longueurs d'onde d'excitation. Chaque élément de cette dernière liste est une matrice dont la première ligne est l'axe des X et les lignes subséquentes, des spectres à raison d'une ligne (i.e. un spectre) pour chacune des répétitions de position d'échantillon. Ainsi, si l'échantillon est présenté à 3 positions différentes dans l'instrument, la matrice aura 4 lignes.
4. Détails: Guide l'opérateur à travers les différentes phases d'acquisition.

#### 3.2 DoRamanSpecteuR.R

1. Appel de la fonction: ***DoRamanSpecteuR(R\_Inst,leplan,tuneParams=FALSE)***
2. Paramètres d'entrée:
  - *R\_Inst* : un environnement d'instrument créé par *InitRamanSpecteuR()*;
  - *leplan* : plan d'expérience créé par *GetPlanExp.R* et modifié par *PickFromPlan.R* le cas échéant.
  - *tuneParams* : permet de modifier le fichier des paramètres à chaque lancement de la fonction.  
TRUE pour permettre la modification, FALSE (valeur par défaut) autrement.
3. Sortie: Aucune. L'environnement *R\_Inst* est modifié pour inclure *Spectres* qui est une liste contenant les spectres. La liste *Spectres* contient autant d'éléments que de types

de spectres (e.g interpolés, ligne de base, corrigés) et pour chaque type de spectres, on a une matrice dont la première ligne est l'axe des X et les lignes subséquentes, des spectres à raison d'une ligne (i.e. un spectre) pour chacune des répétitions de position d'échantillon. Ainsi, si l'échantillon est présenté à 3 positions différentes dans l'instrument, la matrice aura 4 lignes.

4. Détails: Guide l'opérateur à travers les différentes phases d'acquisition. N'affiche pas les résultats.

### 3.3 DoReflectSpecteuR.R

1. Appel de la fonction: ***DoReflectSpecteuR(Reflect\_Inst,leplan,tuneParams=FALSE)***

2. Paramètres d'entrée:

- *Reflect\_Inst* : un environnement d'instrument créé par *InitRamanSpecteuR()*;
- *leplan* : plan d'expérience créé par *GetPlanExp.R* et modifié par *PickFromPlan.R* le cas échéant.
- *tuneParams* : permet de modifier le fichier des paramètres à chaque lancement de la fonction.

TRUE pour permettre la modification, FALSE (valeur par défaut) autrement.

3. Sortie: Aucune. L'environnement *Reflect\_Inst* est modifié pour inclure *Spectres* qui est une liste contenant les spectres. La liste *Spectres* contient autant d'éléments que de types de spectres (e.g interpolés, ligne de base, corrigés) et pour chaque type de spectres, on a une matrice dont la première ligne est l'axe des X et les lignes subséquentes, des spectres à raison d'une ligne (i.e. un spectre) pour chacune des répétitions de position d'échantillon. Ainsi, si l'échantillon est présenté à 3 positions différentes dans l'instrument, la matrice aura 4 lignes.
4. Détails: Guide l'opérateur à travers les différentes phases d'acquisition. N'affiche pas les résultats.

### 3.4 DoTransmitSpecteuR.R

1. Appel de la fonction: ***DoTransmitSpecteuR(Trans\_Inst,leplan,tuneParams=FALSE)***

2. Paramètres d'entrée:

- *Trans\_Inst* : un environnement d'instrument créé par *InitRamanSpecteuR()*;
- *leplan* : plan d'expérience créé par *GetPlanExp.R* et modifié par *PickFromPlan.R* le cas échéant.
- *tuneParams* : permet de modifier le fichier des paramètres à chaque lancement de la fonction.

TRUE pour permettre la modification, FALSE (valeur par défaut) autrement.

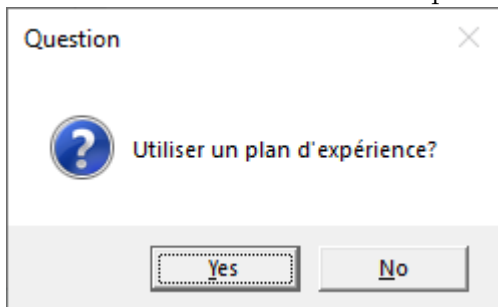
3. Sortie: Aucune. L'environnement *Reflect\_Inst* est modifié pour inclure *Spectres* qui est une liste contenant les spectres. La liste *Spectres* contient autant d'éléments que de types de spectres (e.g interpolés, ligne de base, corrigés) et pour chaque type de spectres,

on a une matrice dont la première ligne est l'axe des X et les lignes subséquentes, des spectres à raison d'une ligne (i.e. un spectre) pour chacune des répétitions de position d'échantillon. Ainsi, si l'échantillon est présenté à 3 positions différentes dans l'instrument, la matrice aura 4 lignes.

4. Détails: Guide l'opérateur à travers les différentes phases d'acquisition. N'affiche pas les résultats.

### 3.5 GetPlanExp.R

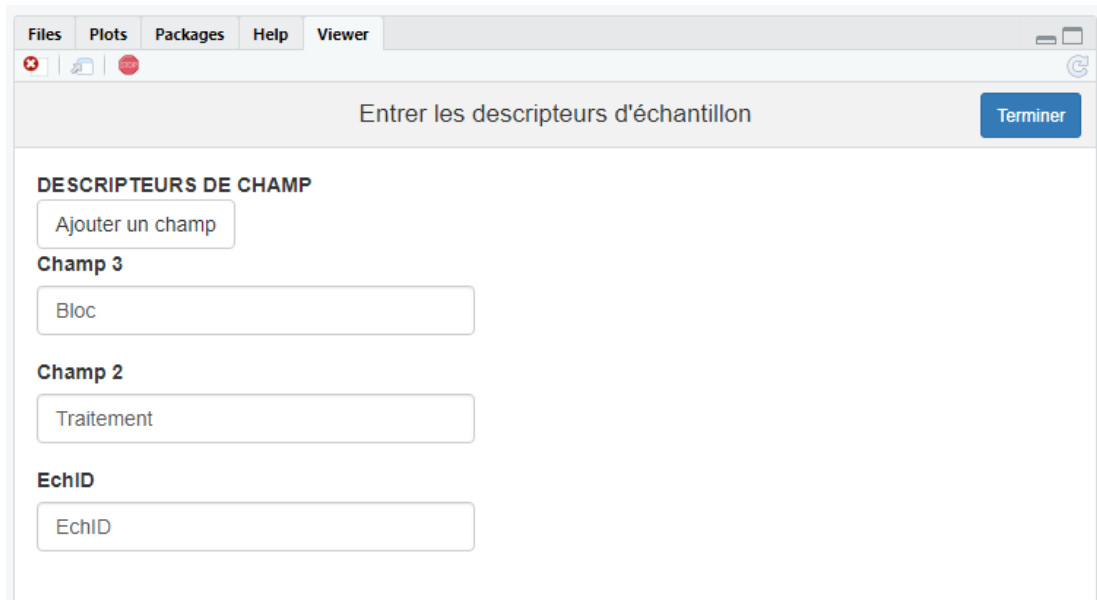
1. Appel de la fonction: ***Plan <- GetPlanExp()***
2. Paramètres d'entrée: Aucun.
3. Sortie: un environnement contenant le variables suivantes
  - *EchID* : chr "" - identificateur unique d'échantillon. Vide au départ;
  - *leplan* : 'data.frame': 0 obs. of 0 variables - tableau décrivant les échantillons;
  - *leType* : chr "Manuel" - type de plan d'expérience: "Manuel" ou "Fichier";
  - *liste\_ids* : chr [1:3, 1] "EchID" "Traitement" "Bloc" - nom des descripteurs d'échantillon, ici 3 descripteurs;
  - *selected* : num 0 - ligne où se retrouve l'échantillon courant. 0 au départ.
4. Détails: Cette fonction offre 2 options présentées à l'utilisateur avec la fenêtre suivante.



Selon la réponse, une des 2 options est choisie:

1. "Yes": lit un fichier de plan d'expérience et retourne un environnement contenant les informations au sujet de ce plan. Dans ce cas, le reste du script s'exécute de façon automatique.
2. "No": prépare pour que les descripteurs des échantillons soient entrés à la main par l'utilisateur à chaque nouvel échantillon. Retourne un environnement contenant les informations pertinentes. Dans ce cas, un éditeur de nom de descripteurs apparaît dans le panneau "Viewer", de **RStudio** ce qui permet à l'utilisateur de définir le nom des champs (voir ci-dessous). Le champ EchID est obligatoire. Pour ajouter un champ, on clique sur le bouton "Ajouter un champ" et on entre un nom dans le tableau dans le nouveau champ créé. On répète pour ajouter d'autres champs. Quand tous les noms de champ sont bien définis, on clique sur le bouton "Terminer" pour compléter. Dans l'exemple plus bas, on a défini 2 champs (*Traitement* et *Bloc*) en plus du champ *EchID*.

Il n'est pas obligatoire de définir des champs supplémentaires. Dans ce cas, seul l'*EchID* devra être complété lors de la prise de données. La fonction générant l'interface **Shiny** pour définir les descripteurs est *Define\_Descript.R*.



### 3.6 InitFluoSpecteuR.R

1. Appel de la fonction: ***F\_Inst <- InitFluoSpecteuR()***
2. Paramètres d'entrée: aucun.
3. Sortie: Un environnement R contenant toutes les informations concernant l'instrument: contenu des fichiers de configuration, des fichiers de définition de l'instrument. Cet environnement s'apparente à un objet en terme de programmation.
4. Détails: Cette fonction prépare un instrument pour la mesure de la fluorescence induite. En plus de définir un environnement d'instrument, ce script:
  - charge les libraires pour d'**Ocean Optics** et définit le "*wrapper*" donnant accès aux fonctions pour contrôler les spectromètres si nécessaire;
  - charge le script *OOInterface.R* contenant les fonctions **R** pour contrôler les spectromètres;
  - initialise le spectromètre;
  - charge le script *MCDAQ.R* et les librairies *MC\_cbw64\_CWrapper.dll* et *cbw64.dll* pour contrôler l'interface permettant le contrôle des DELs excitant la fluorescence. *MC\_cbw64\_CWrapper.dll* est un "*wrapper*" en C compilé pour accéder aux fonctions de la librairie *cbw64.dll* fournie par **Measurement Computing** avec un format de sortie permettant des appels directs depuis R avec les fonctions contenus dans *MCDAQ.R* qui utilise la fonction ".C" de **R** pour appeler les fonctions de *MC\_cbw64\_CWrapper.dll*.

- Fait les mesures de normalisation sur les cibles standards et stocke le coefficient de normalisation dans l’environnement d’instrument créé par la fonction.

### 3.7 InitRamanSpecteuR.R

1. Appel de la fonction: ***R\_Inst <- InitRamanSpecteuR()***
2. Paramètres d’entrée: aucun.
3. Sortie: Un environnement R contenant toutes les informations concernant l’instrument: contenu des fichiers de configuration, des fichiers de définition de l’instrument. Cet environnement s’apparente à un objet en terme de programmation.
4. Détails: Cette fonction prépare un instrument pour la mesure de la diffusion Raman. En plus de définir un environnement d’instrument, ce script:
  - charge les libraires pour d’**Ocean Optics** et définit le “*wrapper*” donnant accès aux fonctions pour contrôler les spectromètres si nécessaire;
  - charge le script *OOInterface.R* contenant les fonctions **R** pour contrôler les spectromètres;
  - initialise le spectromètre;
  - charge le script *Newport\_LS\_2.R* pour contrôler la source laser LS2 de **Newport** ou charge le script... pour contrôler le laser ...;
  - Fait une vérification du fonctionnement du laser.

### 3.8 InitReflectSpecteuR.R

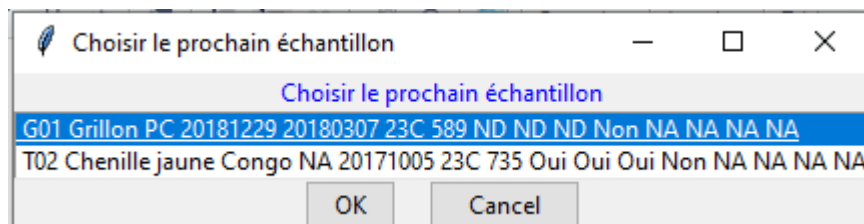
1. Appel de la fonction: ***Reflect\_Inst <- InitReflectSpecteuR()***
2. Paramètres d’entrée: aucun.
3. Sortie: Un environnement R contenant toutes les informations concernant l’instrument: contenu des fichiers de configuration, des fichiers de définition de l’instrument. Cet environnement s’apparente à un objet en terme de programmation.
4. Détails: Cette fonction prépare un instrument pour la mesure de la fluorescence induite. En plus de définir un environnement d’instrument, ce script:
  - charge les libraires pour d’**Ocean Optics** et définit le “*wrapper*” donnant accès aux fonctions pour contrôler les spectromètres si nécessaire;
  - charge le script *OOInterface.R* contenant les fonctions **R** pour contrôler les spectromètres;
  - initialise le spectromètre;
  - charge le script *MCDAQ.R* pour charger des définitions de variables pouvant être présentes dans le fichier de configuration d’instrument..
  - Fait des mesures pour valider la qualité de la lampe blanche.

### 3.9 InitTransmitSpecteuR.R

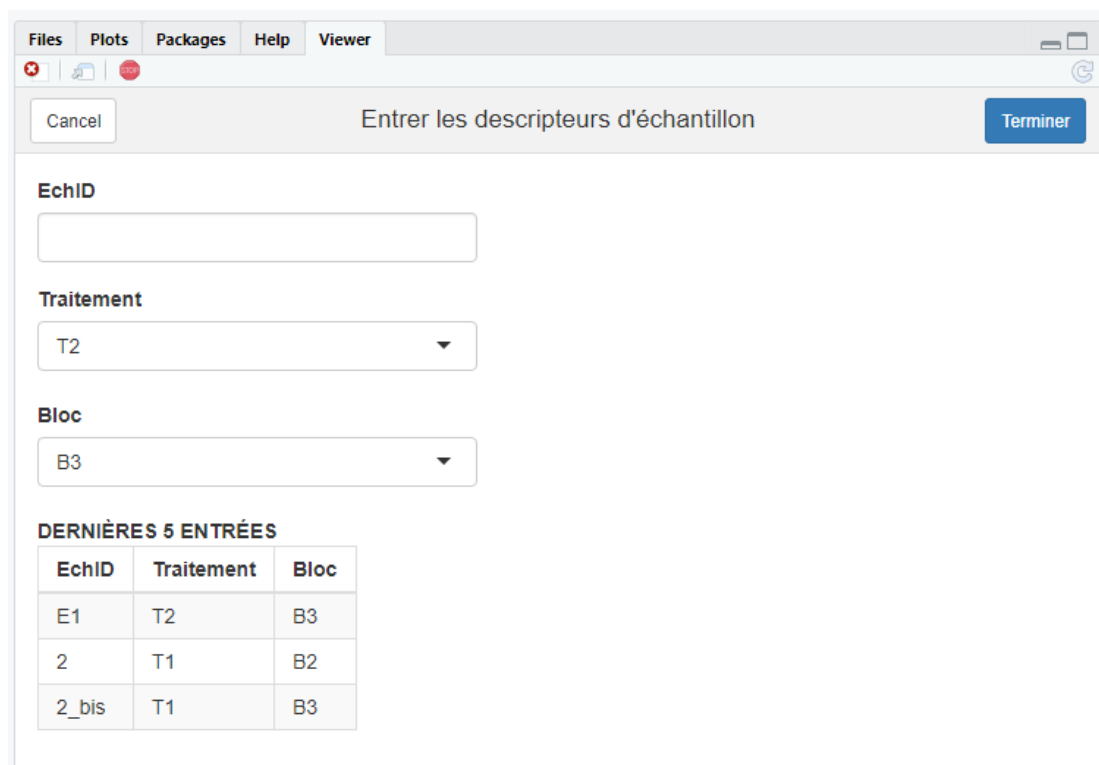
1. Appel de la fonction: ***Transmit\_Inst <-InitTransmitSpecteuR()***
2. Paramètres d'entrée: aucun.
3. Sortie: Un environnement R contenant toutes les informations concernant l'instrument: contenu des fichiers de configuration, des fichiers de définition de l'instrument. Cet environnement s'apparente à un objet en terme de programmation.
4. Détails: Cette fonction prépare un instrument pour la mesure de la fluorescence induite. En plus de définir un environnement d'instrument, ce script:
  - charge les libraires pour d'**Ocean Optics** et définit le “*wrapper*” donnant accès aux fonctions pour contrôler les spectromètres si nécessaire;
  - charge le script *OOInterface.R* contenant les fonctions **R** pour contrôler les spectromètres;
  - initialise le spectromètre;
  - charge le script *MCDAQ.R* pour charger des définitions de variables pouvant être présentes dans le fichier de configuration d'instrument..
  - Fait des mesures pour valider la qualité de la lampe blanche.

### 3.10 PickFromPlan.R

1. Appel de la fonction: ***PickFromPlan(Plan, monDelai = 1000)***
2. Paramètres d'entrée:
  - *Plan*: un environnement généré par **GetPlanExp.R**.
  - *monDelai*: temps en msec avant de mettre à jour EchID avec l'extension *\_bis* quand l'EchID entré par l'utilisateur a déjà été utilisé (voir Option 2 plus bas).
3. Sortie: Aucune. L'environnement *Plan* est modifié par le script pour mettre à jour les informations.
4. Détails: Selon que le plan d'expérience origine d'un fichier (option 1) ou qu'il est construit au fur et à mesure de l'acquisition de données (option 2).
  - Option 1: fait apparaître une fenêtre montrant la liste de tous les échantillons. Chaque ligne est construite par concaténation de tous les descripteurs de l'échantillon. L'opérateur clique sur un des échantillons dans la liste puis sur le bouton “OK” pour sélectionner un échantillon. En cliquant sur “Cancel”, aucun échantillon n'est choisi et cela va permettre de quitter la session de travail si désiré (bloc if (!is\_empty(Plan\$EchID)) dans l'exemple de script principal à la section 2



- Option 2: fait apparaître une fenêtre dans le panneau *Viewer* de **RStudio** permettant de compléter tous les champs décrivant un échantillon (voir ci-dessous). Le champ *EchID* est obligatoire et doit contenir un identifiant unique. Si l'opérateur entre un identifiant déjà utilisé, les caractères “\_bis” sont automatiquement ajoutés. Pour les autres champs (ici *Traitement* et *Bloc*), il s'agit de choisir dans la liste. On peut ajouter à la liste en faisant une entrée à la main. Cette nouvelle entrée sera disponible pour la définition des échantillons subséquents. Le tableau au bas présente les 5 dernières entrées faites par l'utilisateur comme aide-mémoire (3 dans l'exemple plus bas car seules 3 entrées ont été faites). Quand les champs sont bien complétés, on clique sur le bouton “Terminer” pour compléter l'opération. Si on appuie sur *Cancel*, la fonction retourne *NULL* qui est utilisé pour sauter la prise de mesure. Si on appuie sur *Terminer*, la fonction retourne “OK”.



Cancel Entrer les descripteurs d'échantillon Terminer

**EchID**

**Traitement**

T2

**Bloc**

B3

**DERNIÈRES 5 ENTRÉES**

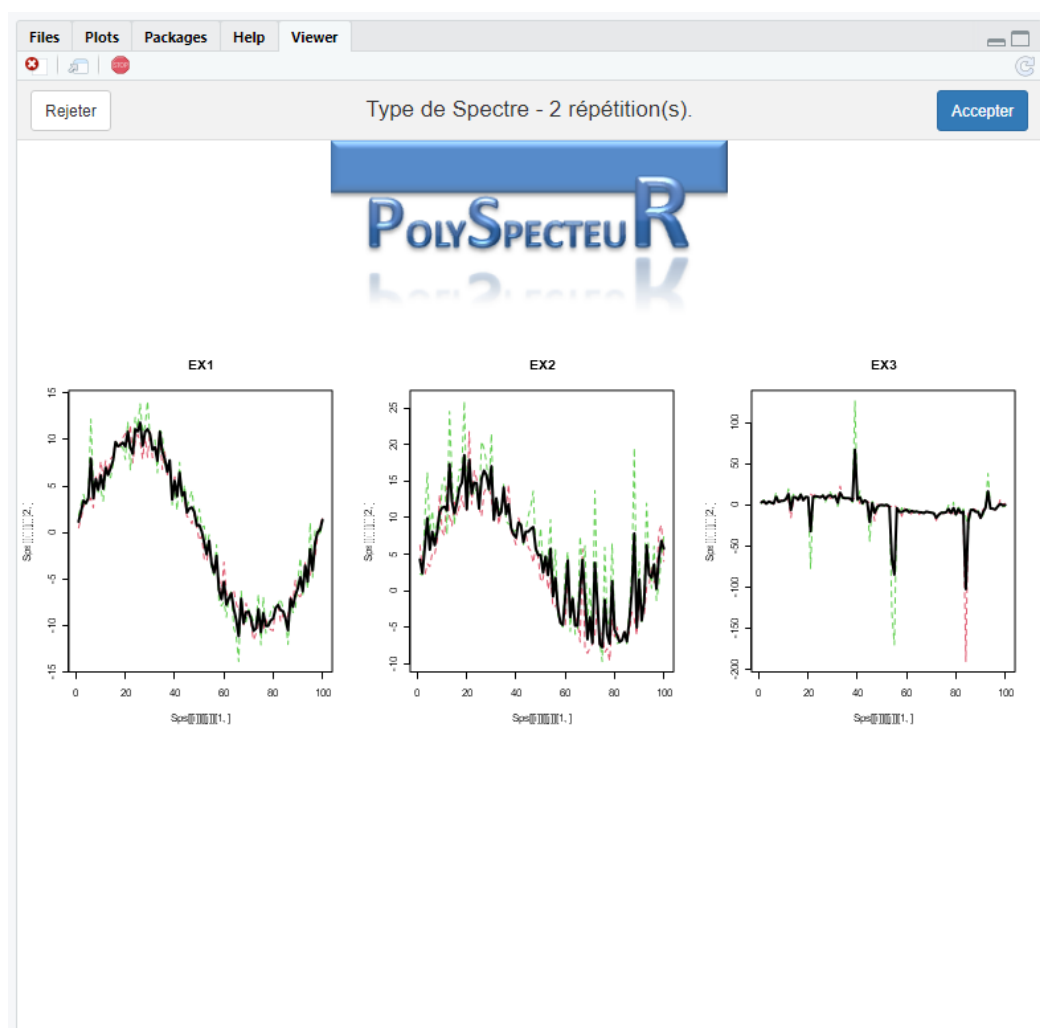
EchID	Traitement	Bloc
E1	T2	B3
2	T1	B2
2_bis	T1	B3

### 3.11 Plots\_2\_Shiny\_MultiLevels.R

- Appel de la fonction: `dum <- Plots_2_Shiny_MultiLevels(unInstrument)`

## 2. Paramètres d'entrée:

- *unInstrument* : environnement d'instrument créé par *Init...SpecteuR*. Cet environnement comprend la configuration de l'instrument, le lien avec le spectro pour *OmniDriver*. Le *wrapper* a été créé dans l'environnement global de R pour être accessible par tous les instruments. L'environnement comprend aussi les paramètres d'acquisition. Il doit y avoir un élément nommé *Spectres* contenant les spectres à afficher. *Spectres* est créé par les scripts d'acquisition comme *DoRamanSpecteuR.R* par exemple. *Spectres* peut être:
  - a. une liste qui contient autant d'éléments que de types de spectres (e.g interpolés, ligne de base, corrigés). Pour chaque type de spectres, on a une matrice dont la première ligne est l'axe des X et les lignes subséquentes, des spectres à raison d'une ligne (i.e. un spectre) pour chacune des répétitions de position d'échantillon. Ainsi, si l'échantillon est présenté à 3 positions différentes dans l'instrument, la matrice aura 4 lignes. Dans ce cas, l'affichage montre autant de graphiques que d'éléments de la liste comme montré dans la figure plus bas.





- b. une liste de listes comme celle décrite au point a plus haut. Le deuxième niveau est utile pour les mesures de fluorescence car on a plusieurs longueurs d'onde d'excitation. Typiquement, le niveau supérieur dans la structure correspond à des types de spectre (brut, interpolé...) et le deuxième niveau correspond aux longueurs d'onde d'excitation. Dans un tel cas, la présentation des graphiques ressemblent à celle sur la figure plus bas. Les boutons au bas de la figure permettent de naviguer les différents types de spectre. Il y a autant de graphiques que de longueur d'ondes d'excitation.



3. Sortie: “OK” si l'utilisateur a “Accepter” les données illustrées  
“REJET” si l'utilisateur a “Rejeter” les données illustrées
4. Détails: Programme pour l'affichage des spectres et la validation des données dans un *gadget Shiny* affiché dans le panneau “Viewer” de **Rstudio**. Une fois les données montrées, l'opérateur clique sur le bouton “Accepter” ou le bouton “Rejeter” pour accepter ou rejeter les données.

### 3.12 writeData.R

1. Appel de la fonction: ***writeData(Plan, lesInstruments, dataPath, dataSetID)***
2. Paramètres d'entrée:
  - *Plan* : un environnement généré par **GetPlanExp.R** et/ou modifié par **Pick-FromPlan.R**;
  - *lesInstruments* : liste créée par une routine mainXXXX.R (e.g. mainFluo.R)
  - *dataPath* : chemin vers le répertoire pour stocker les données;
  - *dataSetID* : identifiant pour former le nom du fichier. Le même identifiant doit être utilisé pour les fichiers des spectres;
3. Sortie: Aucune. Des fichiers contenant les données spectrales sont créés si nécessaire et la dernière mesure est enregistré.
4. Détails: Les données spectrales brutes sont stockées dans un sous-répertoire de “dataPath” nommé “Brutes”. Les données spectrales interpolées et corrigées sont stockées dans le répertoire “dataPath”. Pour un répertoire “dataPath”, il faut s’assurer de n’utiliser le “dataSetID” qu’une seule fois. Quand il y a des répétitions sur la position de l’échantillon, seul la moyenne sur les échantillons est stockée.

### 3.13 writeYFile.R

1. Appel de la fonction: ***writeYFile(Plan, dataPath, dataSetID)***
2. Paramètres d'entrée:
  - *Plan* : un environnement généré par **GetPlanExp.R** et/ou modifié par **Pick-FromPlan.R**;
  - *dataPath* : chemin vers le répertoire pour stocker les données;
  - *dataSetID* : identifiant pour former le nom du fichier. Le même identifiant doit être utilisé pour les fichiers des spectres;
3. Sortie: Aucune. Le fichier des Y est créé et/ou modifié.
4. Détails: le fichier créé par cette fonction ne contient qu’une seule ligne pour tous les types de spectres acquis sur un même échantillon. Les données de chaque couple (type de données, instrument) sont stockées dans des fichiers séparés. La variable *dataSetID* et l’identificateur *EchID* contenu dans le *Plan* permettent de lier toutes ces données.

## Annexe A

### Script d'acquisition avec 3 instruments pour la fluorescence, la transmittance et la réflectance

```
mainTest_Fl_Tr_Re <- function()

#####
# INSTRUCTIONS ----
#####
# Script d'acquisition de données avec SpectrAAC-2 lorsqu'on ne fait que de la
# fluorescence.
# Pour utiliser un autre instrument ou plusieurs instruments, il faut modifier
# les 2 lignes avec la mention "#options" à la fin. La première des deux peut
# être remplacée par une ou plusieurs lignes à raison d'une ligne par type
# de mesure désirée. Par exemple, pour faire des mesures de fluorescence
# suivie de mesures de Raman, la première ligne pourrait être remplacée par:
#     F_inst <- InitFluoSpecteur()
#     R_inst <- InitRamanSpecteur()
# Chacune des lignes appelant une routine d'initialisation doit commencer
# par un nom unique. Ensuite, il faut mettre à jour les éléments de la
# liste "lesInstruments" définie dans la deuxième ligne. Pour l'exemple
# ci-haut, cela donnerait:
#     lesInstruments <- list(F_inst,R_inst)
# Noter que l'ordre dans cette liste définit l'ordre d'acquisition des
# données.
#
#####
# AUTEUR: Bernard Panneton, Agriculture et Agroalimentaire Canada
# Mars 2022
#####
{
  #####
  #Initialisation ----
  #####
  ##Paramètres graphiques par défaut et logo comme variables globales----
  op <- par(no.readonly = TRUE)
  logo <- png::readPNG("PolySpecteuR_Logo.png")
  #####

  #####
  ## Enlève OOobj si existant----
  if (exists("OOobj", envir = .GlobalEnv)) rm(OOobj, envir=.GlobalEnv)
```

```

#####

#####
## Charger des packages R ----
#####
ok <- require("rlang")
if (!ok) install.packages('rlang')
ok <- require("utils")
if (!ok) install.packages('utils')
ok <- require("here")
if (!ok) install.packages('here')

# Se placer dans le répertoire R du projet PolySpecteur.
# Pour que ça marche, il faut que le présent script
# soit dans le répertoire R du projet PolySpecteur.
# Cela permet d'utiliser efficacement des chemins
# relatifs pour se rendre dans les différents répertoires
# utilisés.
RPath=here::here()
setwd(RPath)

#####
## Charger les scripts du projet PolySpecteur ----
#####
setwd("R")
files.sources = list.files(pattern=glob2rx("*.R"),full.names = TRUE)
dum <- sapply(files.sources, source, encoding="UTF-8")
setwd("..")

#####
##Définir les instruments nécessaires et les initialiser ----
#####

F_Inst <- InitFluoSpecteur()                                #options
if (is.character(F_Inst)) return( "ABANDON")                #options

R_Inst <- InitReflectSpecteur()                              #options
if (is.character(R_Inst)) return( "ABANDON")                #options

T_Inst <- InitTransmitSpecteur()                             #options
if (is.character(T_Inst)) return( "ABANDON")                #options

lesInstruments <- list(F_Inst, R_Inst, T_Inst)              #options

```

```

lestypes <- lapply(lesInstruments, function(I) I$type)

#####
## Définir le plan d'expérience et le chemin pour stocker les données ----
#####
Plan <- GetPlanExp()
dataSetID <- utils::winDialogString(
  "Entrer un identifiant pour les noms de fichier de données",
  as.character(Sys.Date()))
dataPath <- utils::choose.dir(default = "",
  caption = "Choisir un répertoire pour stocker les données.")

#####
## Permettre la modif des paramètres d'acquisition à chaque échantillon ----
#####
tuneParams <- FALSE
yesno <- utils::winDialog("yesno",
  "Permettre la modification des paramètres à chaque échantillon.")
if (yesno=="YES") tuneParams <- TRUE

#####
#Phase d'acquisition de données ----
#####
goOn <- TRUE
while(goOn){  ## Boucle sur les échantillons----
  letest <- PickFromPlan(Plan)
  if (letest == "OK"){
    k=0
    isValid <- TRUE
    ### Boucle sur les instruments----
    for (t in lestypes){
      k <- k+1
      if (t=="Raman"){
        if (isValid) {
          DoRamanSpecteuR(lesInstruments[[k]],Plan, tuneParams)
          dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])
          isValid <- (dum=="OK") & isValid
        }
      }
      if (t=="Fluorescence"){
        if (isValid){
          DoFluoSpecteuR(lesInstruments[[k]], Plan, tuneParams)
          dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])

```

```

        isValid <- (dum=="OK") & isValid
    }
}
if (t=="Reflectance"){
    if (isValid){
        DoReflectSpecteuR(lesInstruments[[k]], Plan, tuneParams)
        dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])
        isValid <- (dum=="OK") & isValid
    }
}
if (t=="Transmittance"){
    if (isValid){
        DoTransmitSpecteuR(lesInstruments[[k]], Plan, tuneParams)
        dum <- Plots_2_Shiny_MultiLevels(lesInstruments[[k]])
        isValid <- (dum=="OK") & isValid
    }
}
}

### Écriture des données si valides----
if (isValid){
    writeYFile(Plan, dataPath,dataSetID)
    writeData(Plan,lesInstruments,dataPath,dataSetID)
}
}

### Option de continuer ou quitter ----
sel <- select.list(c("Oui","Non"), preselect = "Oui",
    title="CONTINUER?",graphics = T)
goOn <- ifelse(sel=="Oui",TRUE,FALSE)
}      #Fin de la boucle sur les échantillons

#####
#Nettoyage et arrêt du script ----
#####
setwd(RPath)
Clean_n_Close(list(lesInstruments))
}

```