Référence : SOU-PM-CRR-I-xxxx

**I**

Cahier des charges : SpectroHR460

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date de diffusion** | **Rédacteur** | **Vérificateur** | **Approbateur** | **Visa de l’approbateur** | **Modifications** | **Indice** |
| 24/04/2020 | S. MINOLLI | A. HEMMERLE |  |  | Création | V1 |
| 04/06/2020 | S. MINOLLI | A. HEMMERLE |  |  | Mise à jour suite tests équipement | V2 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Diffusion : Pour action / Pour information

Destinataires :

Device TANGO SpectroHR460

Table des Matières

[1. A propos 3](#_Toc42158856)

[2. Présentation générale du projet 3](#_Toc42158857)

[3. Architecture matérielle 3](#_Toc42158858)

[4. Architecture logicielle 4](#_Toc42158859)

[5. Interfaces du Device SpectroHR460 5](#_Toc42158860)

[5.1. Propriétés 5](#_Toc42158861)

[5.1. Attributs 6](#_Toc42158862)

[5.2. Commandes 10](#_Toc42158863)

[5.3. Gestion du State et du Status 13](#_Toc42158864)

[6. Proposition d’interface 15](#_Toc42158865)

[7. Aspects techniques 16](#_Toc42158866)

[7.1. Généralités 16](#_Toc42158867)

[7.2. Conception préliminaire 16](#_Toc42158868)

[7.2.1. Diagramme de classe macroscopique 16](#_Toc42158869)

[7.2.2. diagrammes de séquence macroscopiques 18](#_Toc42158870)

[8. Annexe 1 – Device Tango GPIBPrologix 24](#_Toc42158871)

[8.1. Principe du Device GPIBPrologix 24](#_Toc42158872)

[8.2. Déploiement pour le spectromètre de la ligne SIRIUS 24](#_Toc42158873)

[9. Annexe 2 – Mapping interface Device et commandes GPIB 26](#_Toc42158874)

[10. Annexe 3 – Références 28](#_Toc42158875)

# A propos

Le présent document détaille les spécifications du Device Tango **SpectroHR460**. Il constitue le document de référence dans le développement et le suivi du projet.

# Présentation générale du projet

L’objectif de ce projet est de réaliser un nouveau Device Tango nommé **SpectroHR460**, qui permettra de piloter un spectromètre imageur HR460 de chez Jobin-Yvon Horiba.

Ce spectromètre s’inscrit dans la réalisation du projet SFAX sur la ligne SIRIUS. Le SFAX est une machine de force (Surface Force Apparatus, SFA) adaptée aux expériences de diffusion de rayons X sur synchrotron (SFAX).

L’appareil de mesure des forces de surface permet de mesurer le profil d’interaction entre deux surfaces en fonction de leur séparation. Cette mesure est obtenue par interférométrie en lumière blanche. Le spectromètre permet de choisir la longueur utilisée et de récupérer, sur un imageur, l’interférogramme formé par les deux surfaces confinantes, pour analyse.

Cf. document Etude\_SFAX\_v1r8.docx pour la description complète et détaillée du projet SFAX sur la ligne SIRIUS. Le ticket jira **CTRLRFC-936** synthétise la demande et contient toutes les références du projet.

# Architecture matérielle

Le spectromètre est composé des éléments suivants :

* Un spectromètre HR460 de Jobin-Yvon Horiba, piloté par une interface GPIB implémentant le protocole IEEE-488, comportant les accessoires suivants :
  + Une tourelle motorisée supportant 2 réseaux de densité 600 traits/mm et 1200 traits/mm, permettant de choisir le type de réseau ainsi que la longueur d’onde centrale,
  + Une fente en entrée axiale motorisée
  + Une fente en entrée latérale motorisée
  + Une fente sortie axiale : supprimée (toujours ouverte)
  + Une fente sortie latérale : supprimée (toujours ouverte)
* Un contrôleur CCD-3500(V) de Jobin-Yvon Horiba, piloté par une interface GPIB implémentant le protocole IEEE-488,
* Une caméra CCD Spectrum One raccordée au contrôleur CCD pour son pilotage, et au spectromètre pour l’entrée TRIGGER.

Nota : dans l’utilisation SFAX, la CCD est utilisée en mode « continu » (non triggé).

# Architecture logicielle

Les éléments spectromètre et CCD n’étant pas couplés en terme d’accès ni en termes fonctionnels, on choisit d’effectuer le pilotage des 2 équipements par 2 Devices séparés.

Le Device SpectroH460 doit permettre de piloter le spectromètre uniquement. La caméra CCD sera pilotée par un Device Lima.

Le spectromètre implémentant le protocole IEEE-488, on choisit d’utiliser comme Device intermédiaire un Device GPIBPrologix qui interface un boîtier convertisseur GPIB/Ethernet Prologix. On aura donc l’architecture logicielle suivante :

* Un Device SpectroHR460 qui propose les fonctions de pilotage du spectromètre,
* Un Device GPIBPrologix qui propose les fonctions GPIB de base : Read, Write et WriteRead.



# Interfaces du Device SpectroHR460

Le mapping des interfaces du Device SpectroHR460 vers les commandes GPIB est donné dans le chapitre 9.

La description du Device GPIBPrologix est donnée dans le chapitre 8.

## Propriétés

**GpibDeviceUrl**

Nom du Device d’accès au spectromètre, au format xx/yy/zz.

*Description* : Name of the Tango device to access the power supply controller.

*Type* : Tango::DevString

*Valeur par défaut* : none

**ConnectionRetries**

Nombre de réessais lors des accès GPIB au spectromètre.

*Description* : Number of retries for spectrometer connection.

*Type* : Tango::DevUShort

*Valeur par défaut* : 3

**PollingPeriod**

Période de polling du spectromètre, en ms.

*Description* : Polling period of the spectrometer, in ms.

*Type* : Tango::DevUShort

*Valeur par défaut* : 1000

**InitMotorsTmo**

Délai maximal d'attente pour la commande InitMotors, en secondes.

*Description* : Timeout for InitMotors command, in s.

*Type* : Tango::DevUShort

*Valeur par défaut* : 200

**SlitDefinition**

Liste des fentes définies pour le spectromètre, au format index:name:description

Exemple :

0:axialInput:entrée axiale

1:lateralInput:entrée latérale

2:axialOutput:sortie axiale

3:lateralOutput:sortie latérale

Les champs "name" seront utilisés pour construire les attributs dynamiques slitPosition\_<name> et comme argument de la commande DefineSlitPosition().

Les champs "description" seront utilisés comme label des attributs dynamiques.

*Description* : Slit definition, with format: {index}:{name}:{description}

*Type* : array of string

*Valeur par défaut* :

0:axialInput:fente entree axiale

1:lateralInput:fente entree laterale

**GratingDefinition**

Liste des réseaux définis pour le spectromètre, au format index:name

Exemple :

0:600

1:1200

2:2400

Les champs "name" seront utilisés comme argument de la commande SelectGrating() et comme valeurs possibles de l'attribut selectedGrating.

*Description* : Grating definition, with format: {index}:{name}

*Type* : array of string

*Valeur par défaut* :

0:600

1:1200

**GratingMotorBacklash**

Valeur du backlash pour le moteur du réseau, en pas moteur.

*Description* : Backlash value for grating motor, in motor steps.

*Type* : Tango::DevUShort

*Valeur par défaut* : 320

## Attributs

**log**

Attribut contenant les traces du Device (généré automatiquement avec le logger de yat4tango).

**gratingPosition**

Position absolue du réseau, en pas moteurs. Cet attribut permet de déplacer le réseau courant pour choisir la longueur d'onde centrale. L’utilisateur pourra préférer l’utilisation de l’attribut centralWavelength pour bouger le réseau directement en nanomètres plutôt qu’en pas moteur.

*Nota* : calculer le mouvement relatif pour envoyer la commande bas-niveau. Utiliser le backlash quand le mouvement relatif est négatif car les longueurs d’onde doivent être scannées dans le sens positif : if delta steps < 0, delta steps = delta steps – GratingMotorBacklash.

*Nota2* : Quand une limite est atteinte lors du mouvement du réseau, arrêter le mouvement avec la commande Abort.

*Description* : Absolute position of the grating motor, in motor steps.

*Type* : SCALAR Tango::DevLong

*R/W* : READ/WRITE

*Niveau* : expert

*Mémorisation* : non

*Unité* : steps

*Min/Max* : 0 / 207820

**centralWavelength**

Longueur d’onde centrale du réseau, en nanomètres. Cet attribut permet à l’utilisateur de choisir la longueur d'onde centrale directement, plutôt qu’en pas moteur. Le Device effectue la conversion en pas moteur pour piloter le spectromètre.

*Nota* : La conversion pas moteur ⬄ longueur d'onde (en nm) est la suivante :

En lecture :

lambda relue = position moteur / 160 \* 1200 / densité

lambda (tango) = lambda relue + offset

En écriture :

delta lambda = lambda consigne (tango) - lambda actuel (tango)

delta position moteur = delta lambda x 160 x densité / 1200

si delta position < 0, retrancher le backlash (propriété GratingMotorBacklash)

*Description* : Current central wavelength, in nm.

*Type* : SCALAR Tango::DevDouble

*R/W* : READ/WRITE

*Niveau* : operateur

*Mémorisation* : non

*Unité* : nm

*Min/Max* : 0 / 1300

**wavelengthOffset**

Offset à appliquer sur la longueur d’onde déterminé à la calibration, en nm. L’offset est saisi par l'opérateur quand la calibration est terminée.

*Nota* : Pour une calibration avec lampe à Mercure :

lambda théorique = 546.074 nm

lambda mesurée = xxxxx

offset = lambda théorique - lambda mesurée

*Description* : Wavelength offset, in nm.

*Type* : SCALAR Tango::DevDouble

*R/W* : READ/WRITE

*Niveau* : operateur

*Mémorisation* : oui, réappliqué à l’init sauf si InitMotors nécessaire. Dans ce cas, réinitialisé à 0.

*Unité* : nm

*Min/Max* : -10 / 10

**limitSwitch**

Indicateur de limite atteinte pour le moteur des mouvements du réseau.

*Description* : Flag for grating motor limit reached.

*Type* : SCALAR Tango::DevBoolean

*R/W* : READ

*Niveau* : operateur

*Mémorisation* : --

*Unité* : --

*Min/Max* : -- / --

**slitPosition\_<slit>**

Ouverture de la fente <slit> en microns. <slit> est défini par le champ « name » décrit dans la propriété SlitDefinition. On aura un attribut slitPosition par fente défini dans cette propriété.

*Nota* : Le device effectuera le calcul du mouvement relatif et la conversion microns ⬄ pas moteurs : slit opening in microns = 1.8333333 \* slit opening in steps.

L’index de la fente à utiliser pour l’interface GPIB est définie dans la propriété SlitDefinition.

*Description* : Slit opening, in microns.

*Type* : SCALAR Tango::DevDouble

*R/W* : READ/WRITE

*Niveau* : operateur

*Mémorisation* : non

*Unité* : µm

*Min/Max* : 0 / 2000

**selectedGrating**

Densité du réseau sélectionné, en traits/mm. Les valeurs possibles de l’attribut sont définies par les champs « name » décrits dans la propriété GratingDefinition. Cet attribut reflète le réseau sélectionné par la dernière commande SelectGrating.

*Nota* : on n’a pas de commande GPIB pour récupérer le réseau courant. Il faudra mémoriser la dernière commande SelectGrating passée et réappliquer cette valeur à l’initialisation, sauf si celle-ci nécessite un InitMotors (redémarrage spectro). Cf. § 7.2 pour les détails sur l’initialisation du Device.

*Description* : Selected grating density, in grooves/mm.

*Type* : SCALAR Tango::DevString

*R/W* : READ

*Niveau* : operateur

*Mémorisation* : --

*Unité* : grooves/mm

*Min/Max* : -- / --

**entranceMirrorPosition**

Position courante du miroir d’entrée (AXIAL ou LATERAL). Cet attribut reflète la position demandée par la dernière commande MoveEntranceMirrorXXXX.

*Nota* : on n’a pas de commande GPIB pour récupérer la position courante du miroir. Il faudra mémoriser la dernière commande MoveEntranceMirrorXXXX passée et réappliquer cette valeur à l’initialisation, sauf si celle-ci nécessite un InitMotors (redémarrage spectro). Cf. § 7.2 pour les détails sur l’initialisation du Device.

*Description* : Entrance mirror position.

*Type* : SCALAR Tango::DevString

*R/W* : READ

*Niveau* : operateur

*Mémorisation* : --

*Unité* : --

*Min/Max* : AXIAL / LATERAL

**exitMirrorPosition**

Position courante du miroir de sortie (AXIAL ou LATERAL). Cet attribut reflète la position demandée par la dernière commande MoveExitMirrorXXXX.

*Nota* : on n’a pas de commande GPIB pour récupérer la position courante du miroir. Il faudra mémoriser la dernière commande MoveExitMirrorXXXX passée et réappliquer cette valeur à l’initialisation, sauf si celle-ci nécessite un InitMotors (redémarrage spectro). Cf. § 7.2 pour les détails sur l’initialisation du Device.

*Description* : Exit mirror position.

*Type* : SCALAR Tango::DevString

*R/W* : READ

*Niveau* : operateur

*Mémorisation* : --

*Unité* : --

*Min/Max* : AXIAL / LATERAL

## Commandes

**Init**

Cette commande permet de réinitialiser le Device de pilotage du spectromètre.

*Description :* Device initialization.

*Arguments*: -

*Retour*: -

*Niveau* : operateur

*Nota :* cette commande réalise les fonctions bas-niveau « WhereAmI » et « StartMainProg » si nécessaire. Etat pendant la commande : INIT. Etat en sortie de commande : DISABLE si InitMotors nécessaire, STANDBY sinon. Cf. § 7.2 pour les détails sur l’initialisation.

**InitMotors**

Cette commande lance l’initialisation des moteurs du spectromètre. Cette action peut prendre jusqu’à 2 minutes.

*Description :* Launch motors initialization.

*Arguments*: -

*Retour*: -

*Niveau* : expert

*Nota :* cette commande réalise la fonction bas-niveau « MotorsInit ». Etat pendant la commande : RUNNING. Etat en sortie de commande : STANDBY. Cf. § 7.2 pour les détails sur la commande.

**Reset**

Cette commande lance la remise à zéro du spectromètre.

*Description :* Reset hardware.

*Arguments*: -

*Retour*: -

*Niveau* : expert

*Nota :* cette commande réalise la fonction bas-niveau « RebootIfHung ». Etat pendant la commande : FAULT puis INIT. Etat en sortie de commande : DISABLE.

**Abort**

Cette commande permet l’arrêt du mouvement en cours.

*Description :* Stop motors.

*Arguments*: -

*Retour*: -

*Niveau* : operateur

*Nota :* cette commande réalise la fonction bas-niveau « GratingSlitsStop ». Etat pendant la commande : MOVING. Etat en sortie de commande : STANDBY.

**SelectGrating**

Cette commande permet de sélectionner le réseau parmi ceux définis dans la propriété GratingDefinition (champ « name »). Elle met à jour l’attribut selectedGrating qui reflète le nouveau réseau sélectionné.

*Description :* Select new grating.

*Arguments*: DevString : grating label

*Retour*: -

*Niveau* : operateur

*Nota :* cette commande réalise la fonction bas-niveau « TurretMoveToPositionX». Etat pendant la commande : MOVING. Etat en sortie de commande : STANDBY.

*Nota* : on n’a pas de commande GPIB pour récupérer le réseau courant. Il faudra mémoriser la dernière commande SelectGrating passée et réappliquer cette valeur à l’initialisation, sauf si celle-ci nécessite un InitMotors (redémarrage spectro). Cf. § 7.2 pour les détails sur l’initialisation du Device.

*Nota* : sur changement de réseau, l’offset de la longueur d’onde (attribut wavelengthOffset) ne sera pas modifié.

**DefineGratingPosition**

Cette commande permet de définir la position courante du réseau, en pas moteur. Cette commande sera utile après un InitMotors, quand le réseau courant se trouve à une position connue.

*Description :* Define grating position.

*Arguments*: DevLong : motor position in steps

*Retour*: -

*Niveau* : expert

*Nota :* cette commande réalise la fonction bas-niveau « GratingSetPosition ». Etat pendant la commande : STANDBY. Etat en sortie de commande : STANDBY.

**DefineSlitPosition**

Cette commande permet de définir l’ouverture courante d’une fente, en microns. Cette commande sera utile après un InitMotors, quand les fentes se trouvent à une position connue. La fente est définie par son label, décrit dans la propriété SlitDefinition (champ « name »).

*Description :* Define slit position.

*Arguments*: DevVarDoubleStringArray : slit name, slit opening in microns

*Retour*: -

*Niveau* : expert

*Nota :* cette commande réalise la fonction bas-niveau « SlitSetPosition ». Conversion steps <-> microns : microns = 1.8333333 \* slit opening in steps. Cf. propriété SlitDefinition pour la correspondance fente <-> index.

Etat pendant la commande : STANDBY. Etat en sortie de commande : STANDBY.

**MoveEntranceMirrorAXIAL**

Cette commande permet de changer la position courante du miroir d’entrée en AXIAL. Cette commande met à jour l’attribut entranceMirrorPosition.

*Description :* Move entrance mirror to AXIAL position.

*Arguments*: -

*Retour*: -

*Niveau* : operateur

*Nota :* cette commande réalise la fonction bas-niveau « EntranceMirrorMoveToAxial ». Etat pendant la commande : MOVING. Etat en sortie de commande : STANDBY.

**MoveEntranceMirrorLATERAL**

Cette commande permet de changer la position courante du miroir d’entrée en LATERAL. Cette commande met à jour l’attribut entranceMirrorPosition.

*Description :* Move entrance mirror to LATERAL position.

*Arguments*: -

*Retour*: -

*Niveau* : operateur

*Nota :* cette commande réalise la fonction bas-niveau « EntranceMirrorMoveToLateral ». Etat pendant la commande : MOVING. Etat en sortie de commande : STANDBY.

*Nota* : on n’a pas de commande GPIB pour récupérer la position courante du miroir. Il faudra mémoriser la dernière commande MoveEntranceMirrorXXXX passée et réappliquer cette valeur à l’initialisation, sauf si celle-ci nécessite un InitMotors (redémarrage spectro). Cf. § 7.2 pour les détails sur l’initialisation du Device.

**MoveExitMirrorAXIAL**

Cette commande permet de changer la position courante du miroir de sortie en AXIAL. Cette commande met à jour l’attribut exitMirrorPosition.

*Description :* Move exit mirror to AXIAL position.

*Arguments*-

*Retour*: -

*Niveau* : operateur

*Nota :* cette commande réalise la fonction bas-niveau « ExitMirrorMoveToAXIAL ». Etat pendant la commande : MOVING. Etat en sortie de commande : STANDBY.

**MoveExitMirrorLATERAL**

Cette commande permet de changer la position courante du miroir de sortie en LATERAL. Cette commande met à jour l’attribut exitMirrorPosition.

*Description :* Move exit mirror to LATERAL position.

*Arguments*-

*Retour*: -

*Niveau* : operateur

*Nota :* cette commande réalise la fonction bas-niveau « ExitMirrorMoveToLATERAL ». Etat pendant la commande : MOVING. Etat en sortie de commande : STANDBY.

*Nota* : on n’a pas de commande GPIB pour récupérer la position courante du miroir. Il faudra mémoriser la dernière commande MoveExitMirrorXXXX passée et réappliquer cette valeur à l’initialisation, sauf si celle-ci nécessite un InitMotors (redémarrage spectro). Cf. § 7.2 pour les détails sur l’initialisation du Device.

## Gestion du State et du Status

Les différents états du Device sont :

* **INIT** : Initialization in progress.
* **STANDBY** : Spectrometer is up and ready.
* **MOVING** : At least one motor is moving.
* **RUNNING** : Init motors in progress.
* **DISABLE :** Init motors to be done.
* **FAULT :** Device configuration error or spectrometer unreachable or hardware fault.

L’ordre de priorité des états est :

* FAULT
* INIT ou RUNNING
* DISABLE
* MOVING
* STANDBY

Actions autorisées en fonction de l’état :

* FAULT : commandes Init et Reset
* INIT : aucune action
* DISABLE : commandes Init, Reset, InitMotors
* RUNNING : aucune action
* MOVING : commande Abort
* STANDBY : toutes commandes, toutes écritures sur attributs

Dans le status du device, on notera :

* L’état du Device d’accès au spectromètre,
* La dernière commande MoveExitMirrorXXXX, MoveEntranceMirrorXXXXX et SelectGrating passée : date et valeur utilisée,
* La raison de l’état FAULT.

Exemple de status :

GPIB Device state: ON

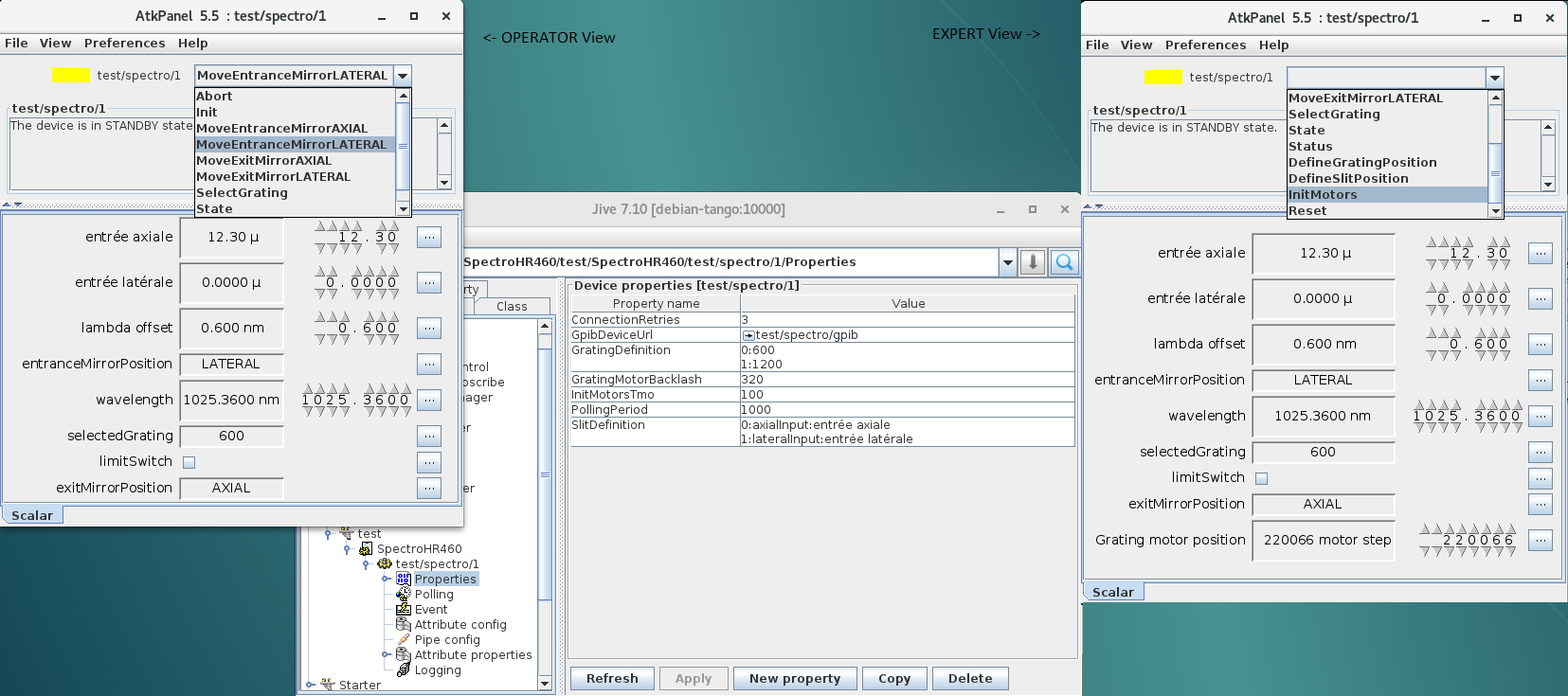
Last Entrance Mirror Command: AXIAL - 11/05/2020 12:50:03

Last Exit Mirror Command: none

Last Select Grating Command: 1200 - 11/05/2020 11:53:50

# Proposition d’interface

Voici une proposition d’interface du Device SpectroHR460 pour illustration :



# Aspects techniques

## Généralités

* Le Device Tango sera implémenté en C++.
* Le développement devra suivre le document “Guidelines” des Devices Tango.
* Le développement devra suivre les règles de codage et de nommage en vigueur à SOLEIL (cf. Confluence).
* On utilisera les librairies YAT et YAT4Tango pour implémenter les logs et la tâche périodique de surveillance du spectromètre.
* On s’inspirera fortement du code initial de pilotage du spectromètre pour le passage des commandes bas-niveau (cf. § 10).

## Conception préliminaire

Dans ce chapitre sont donnés quelques points de conception préliminaire pour aider à la compréhension de ce que doit faire le Device.

### Diagramme de classe macroscopique

Le Device comportera 3 classes principales :

* La classe du device : elle implémente l’interface du device telle que spécifiée dans les chapitres précédents.
* Un manager : c’est une DeviceTask qui gère les demandes du device dans sa file de messages et transmets les requêtes vers la classe d’interface du spectro.
* Une classe d’interface du spectromètre : elle implémente les commandes GPIB spécifiques au spectromètre via le Device GPIBPrologix.



De façon synthétique :

* La classe du Device :
  + récupère et vérifie les propriétés,
  + crée le proxy vers le device GPIB,
  + récupère les valeurs mémorisées,
  + lance le manager.
* Le manager :
  + Gère les demandes du Device sous forme de messages,
  + Lance une tâche périodique pour évaluer l’état courant du device,
  + Lance un thread si nécessaire pour les actions longues, pour ne pas bloquer le device.
* La classe d’interface Spectro :
  + Gère les réessais pour chaque Read (propriété ConnectionRetries),
  + Gère les réponses attendues et transforme la réponse brute en ok, ko, …
  + Gère les éventuels délais d’attente spécifiques après chaque commande (cf. code existant).

### diagrammes de séquence macroscopiques

Séquence d’initialisation du Device :





Nota : Mapping init\_state et DevState :

E\_InitStatus =

spectro\_init ---> INIT

spectro\_main\_prog

spectro\_hung ---> FAULT

spectro\_init\_error

init\_final ---> INIT

init\_motors\_needed ---> DISABLE

init\_motors ---> RUNNING

init\_done ---> STANDBY

Tâche périodique :



Commande InitMotors :



Nota : La commande InitMotors pose la problématique suivante : la réponse à la requête ne vient que quand les déplacements sont terminés. On ne peut pas scruter à la fois cette réponse ET les autres questions/réponses de la tâche périodique !

Pendant cette commande, il faut bloquer les autres requêtes de la tâche périodique, n’interroger le spectro qu’en lecture dans l’attente de la réponse, gérer un temps maximal d’attente (propriété InitMotorsTmo) et, au bout de ce temps, si la réponse correcte n’est pas venue, alors on passe en FAULT.

# Annexe 1 – Device Tango GPIBPrologix

## Principe du Device GPIBPrologix

Ce Device permet de piloter un boîtier de conversion GPIB / Ethernet de marque Prologix.

Propriétés à configurer :

* Url : @IP du boîtier Prologix
* Port : port TCP du boîtier Prologix (par défaut : 1234)
* ReadTermination : type de terminaison de la lecture, parmi :
  + TIMEOUT : lecture jusqu’au timeout défini par ReadTimeout
  + EOI : lecture jusqu’au marqueur EOI
  + TERMINATOR : lecture jusqu’au marqueur défini par ReadTerminator.
* ReadTimeout : timeout de lecture, en ms.
* ReadTerminator : marqueur de fin de message en lecture, parmi : 10 (LF), 13 (CR)
* WriteTerminator : marqueur de fin de message en écriture, parmi : 0 (CRLF), 1 (CR), 2 (LF), 3 (NONE)
* GpibAddress : @gpib (de 0 à 30)

Commandes disponibles :

* DevVoid Write(DevString)
* DevString Read()
* DevString WriteRead(DevString)

Etats possibles du Device :

* FAULT / UNKNOWN : erreur d’initialisation ou de communication
* ON : tout va bien

NOTA : Il n’y a pas de configuration spécifique du boîtier à effectuer, le Device GPIBPrologix s’occupe de tout.

## Déploiement pour le spectromètre de la ligne SIRIUS

Pour spectromètre HR460 :



Pour caméra CCD (pour information) :



# Annexe 2 – Mapping interface Device et commandes GPIB

Les commandes du Device GPIBPrologix à utiliser sont : Write et Read.

La commande WriteRead est tentante mais au vu du code existant, le spectromètre semble assez lent à répondre sur GPIB, donc plutôt privilégier un Write puis Read, pour pouvoir gérer des temps d’attente entre le Write et le Read plus facilement.

Ci-dessous le tableau de mapping de l’interface du Device avec les commandes GPIB à utiliser, très largement inspiré du document Liste\_Commandes\_GPIB\_SFAX\_v2r0.docx.

La fonction indiquée dans le tableau correspond au nom du chapitre décrivant la commande GPIB dans ce même document. On se reportera au code existant pour les détails d’implémentation en C++.

| Interface du Device | Fonction spectro | Commentaires |
| --- | --- | --- |
| Commande Init | WhereAmI | Faire des ré-essais au démarrage car l’appareil peut être lent à répondre (polling sur le read), avec une tempo de 100ms par exemple.  Selon la première réponse au WhereAmI, on fera le StartMainProgram ou pas.  Cf. § 7.2 pour les détails sur la phase d’init. |
| StartMainProgram |
| Commande InitMotors | MotorsInit | Action à threader car prend beaucoup de temps (2 min).  La commande est terminée quand la réponse ‘o’ est reçu. Avant = rien. Ceci implique de ne pas faire d’autre requête pendant ce temps pour ne pas mélanger les réponses. Cf. § 7.2.2. |
| Commande Reset | RebootIfHung | Cette commande ne renvoie pas de réponse. Attente de 300ms dans le code existant. |
| Commande Abort | GratingSlitsStop |  |
| Commande SelectGrating | TurretMoveToPosition0 | Durée de l'action : quelques secondes mais la commande renvoie la réponse tout de suite.  => poller le status pour savoir quand le mouvement est terminé |
| TurretMoveToPosition1 |
| Attribut selectedGrating | -- | Pas de commande GPIB pour récupérer le réseau courant. Pour mettre à jour l’attribut, utiliser la valeur passée par la dernière commande SelectGrating.  A l’init du device, récupérer la valeur mémorisée pour initialiser l’attribut.  En cas d’initialisation de zéro, on a :   * grating = 1 à l’allumage * grating = 1 en sortie de StartMainProgram * grating= 1 en sortie de MotorsInit |
| Attribut gratingPosition | GratingRelativeMove | En lecture : position absolue en pas moteur récupérée directement  En écriture : calculer le mouvement relatif pour envoyer la commande gpib. Utiliser le backlash quand le mouvement relatif est négatif : if delta steps < 0, delta steps = delta steps - backlash |
| GratingGetPosition |
| Commande DefineGratingPosition | GratingSetPosition | Définition de la position moteur en pas moteur. Cette commande ne bouge pas les moteurs. |
| Attribut centralWavelength | -- | Cet attribut est une facilité utilisateur pour piloter le spectromètre en longueur d’onde plutôt qu’en pas moteur. Il fait donc appel à une conversion lambda <-> pas moteur. |
| Attribut wavelengthOffset | -- | Cet attribut est une facilité utilisateur pour piloter le spectromètre en longueur d’onde plutôt qu’en pas moteur. Il entre dans le calcul de l’attribut centralWavelength. |
| Attributs slitPosition\_<label> | SlitRelativeMove | En lecture : ouverture absolue en pas moteur récupérée directement  En écriture : calculer le mouvement relatif pour envoyer la commande gpib.  Conversion en microns : microns = 1.8333333 \* pas moteur |
| SlitGetPosition |
| Commande DefineSlitPosition | SlitSetPosition | Définition de la position de la fente en pas moteur. Cette commande ne bouge pas les moteurs.  Conversion en microns : microns = 1.8333333 \* pas moteur |
| Attribut limitSwitch | GratingGetLimitStatus | Pas besoin de connaître le détail des limites atteintes. Activer le booléen dès que la réponse est différente de 0. |
| Attribut state | GratingSlitsGetStatus | Interdire tout nouveau mouvement si un des accessoire, le réseau ou une fente est déjà en mouvement. |
| AccessoriesGetStatus |
| Commande MoveEntranceMirrorAXIAL | EntranceMirrorMoveToAxial | Durée de l'action : de l’ordre de la seconde mais la commande renvoie la réponse tout de suite.  => poller le status pour savoir quand le mouvement est terminé |
| Commande MoveEntranceMirrorLATERAL | EntranceMirrorMoveToLateral |
| Attribut entranceMirrorPosition | -- | Pas de commande GPIB pour récupérer la position courante du miroir. Pour mettre à jour l’attribut, utiliser la valeur passée par la dernière commande MoveEntranceMirrorXXXX.  A l’init du device, récupérer la valeur mémorisée pour initialiser l’attribut.  En cas d’initialisation de zéro, on a :   * position = AXIAL à l’allumage * position = AXIAL en sortie de StartMainProgram * position = AXIAL en sortie de MotorsInit |
| Commande MoveExitMirrorAXIAL | ExitMirrorMoveToAxial | Durée de l'action : de l’ordre de la seconde mais la commande renvoie la réponse tout de suite.  => poller le status pour savoir quand le mouvement est terminé |
| Commande MoveExitMirrorLATERAL | ExitMirrorMoveToLateral |
| Attribut exitMirrorPosition | -- | Pas de commande GPIB pour récupérer la position courante du miroir. Pour mettre à jour l’attribut, utiliser la valeur passée par la dernière commande MoveExitMirrorXXXX.  A l’init du device, récupérer la valeur mémorisée pour initialiser l’attribut.  En cas d’initialisation de zéro, on a :   * position = AXIAL à l’allumage * position = AXIAL en sortie de StartMainProgram * position = AXIAL en sortie de MotorsInit |

# Annexe 3 – Références



Code de pilotage initial :