Référence : SOU-PM-NT-1522

Matlab Middle Layer à SOLEIL : contrôle commande des installations via Matlab

Diffusion :

Liste de diffusion :

Groupe Physique Machine

Copie : J-M. Filhol, M-P. Level, Groupe Fonctionnement, groupe ICA

Date :	Rédacteur :	Vérificateur :	Approbateur :	Modifications :	Indice :
05/04/05	L. Nadolski				1
21/06/04	L. Nadolski				0

SYNCHROTRON SOLEIL – Société Civile au capital de 12.000 €
439 684 903 R.C.S. EVRY – N° TVA Intracommunautaire : FR 27 439 684 903 - NAF 731Z – SIRET 439 684 903 00016
Siège social : CEA SACLAY – L'Orme des Merisiers – F 91190 SAINT-AUBIN
Division Sources et Accélérateurs - Bât. A – L'Orme des Merisiers – Saint-Aubin - BP 48 – F 91192 Gif-sur-Yvette Cedex
Tél : 01 69 35 98 05 – Fax : 01 69 35 94 51 - Web http://www.synchrotron-soleil.fr/

Table des matières

1 Introduction

Ce document décrit la couche logicielle entre le binding Tango/Matlab écrit par Nicolas Leclercq et les interfaces Matlab de haut niveau pour le contrôle commande. Cette couche vise à définir un niveau d'abstraction entre les commandes Tango et les actions que les physiciens des accélérateurs veulent mettre en oeuvre. Elle a pour vocation de rendre plus transparent le contrôle des installations de SOLEIL en définissant un jeu de commandes haut niveau, riche, intuitif et facile d'utilisation pour un non expert du système de contrôle commande. L'ensemble de ce travail repose sur les développements faits par Gregory J. Portmann à l'Advanced Light Source (Berkeley) et à SPEAR3 (Stanford, USA).

L'essentiel du travail a consisté dans un premier temps à substituer la couche EPICS par la couche TANGO. Ensuite, un jeu de commandes complémentaires a été ajouté afin de permettre une utilisation plus aisée à SOLEIL.

IMPORTANT : Ce document est reprend en français une grande partie de la note technique intitulée « Middle Layer Sofware For Accelerator Control », Note SPEAR3, November 2003, Gregory J. Portmann, William J. Corbett, and Andrei Terebilo.

1.1 Historique

Hiroshi Nishimura (ALS) propose en 1988 d'utiliser Matlab [1] dans une salle de contrôle. Gregory Portmann décide d'écrire un jeu de fonctions pour l'ALS. A SSRL, Andrei Terebilo écrit l'Accelerator Toolbox (AT) comme simulateur de faisceau. Cette partie est effectivement une boite à outils (« toolbox ») de Matlab reprenant le coeur du code de simulation Tracy 2 développé à l'ALS au début des années 1990 par Johann Bengsston, Etienne Forest et Hiroshi Nishimura (J. Bengsston, E. Forest and H. Nishimura, Tracy User manual, unpublished (ALS, Berkeley)). Récemment (199?), James Safranek et Gregory Portmann ont également porté le programme LOCO sous Matlab. En vue de développer le système de contrôle de SPEAR3, la grande majorité des fonctions écrites à l'ALS a été portée à SPEAR3. Ces routines définissent le « middle layer » qui simplifie le développement d'applications de haut niveau tout en masquant les détails du système de contrôle (routines de bas niveau, nomenclature).

A l'ALS, Matlab est utilisé pour le contrôle de l'anneau de stockage depuis une dizaine d'années. Ce qui inclut la montée en énergie de tous les aimants de 1.5 GeV à 1.9 GeV, la sauvegarde et restauration des configurations de la machine, le feedback lent d'orbite, le beam based alignement, la correction des nombres d'ondes, des chromaticités, les

mesures de matrices réponse, les scripts et programmes dédiés au « run » de physique machine et à l'opération. La boîte à outils (toolbox) « Simple Channel Access » a été développée pour faire le pont logiciel en Matlab et le monde EPICS [2][3].

A SSRL, en vue de SPEAR3, une nouvelle toolbox, Matlab Channel Access (MCA) a été écrite pour communiquer de manière plus efficace avec EPICS. Matlab y est utilisé maintenant de manière similaire à l'ALS. Le concept est porteur et permet d'être rapidement adapté pour une utilisation dans une autre accélérateur. Par exemple, le source de troisième génération canadienne (CLS) a également adopté Matlab en salle de contrôle et la « toolbox » MCA.

L'utilisation de MML a été entendu au booster et aux lignes de transfert de SOLEIL. Sauf cas particulier, tout le document est écrit en prenant pour exemple l'anneau de stockage. Toutes les concepts généraux développés ici sont bien sur applicables à LT1, LT2 et au booster.

1.2 Présentation

Le « Matlab Middle Layer » (MML) est conçu de manière indépendante de l'accélérateur à piloter. Seule la couche bas niveau dépend du type de système de contrôle commande. Elle repose soit sur EPICS [4] (Spear3 [5], ALS [6], CLS [7]) soit sur TANGO [8] (SOLEIL). C'est cette partie qui a d'abord due être créée et écrite pour SOLEIL. De ce fait, tout le reste est commun entre les accélérateurs, ce qui permet d'utiliser les développements réalisés ailleurs et de partager de nouvelles fonctionnalités.

Concrètement, pour une machine donnée, il suffit de définir dans un fichier d'initialisation les paramètres spécifiques à l'accélérateur comme les paramètres physiques (énergie, émittance, modèle) et les paramètres de contrôle (« channel names » pour le monde EPICS, les noms des devices, attributs et commandes pour le monde TANGO).

Le « Matlab Middle Layer » est construit autour de deux structures Matlab contenant toute la configuration physique et contrôle de l'accélérateur.

1. La structure AO (Accelerator Object)

Cette structure contient la description de chaque élément et famille d'éléments (indice des éléments, nomenclature TANGO, etc.), les fonctions à appeler pour communiquer avec le monde TANGO, les fonctions de conversion entre les unités matérielles (« display unit » au sens TANGO) et les unités physiques (i.e. dans le système international ou encore « standard unit » au sens TANGO).

2. La structure AD (Accelerator Data)

Cette structure contient les noms de fichiers, répertoires pour les sauvegardes, configurations, les paramètres accélérateurs.

Les structures AO et AD sont stockées dans l'objet Matlab *ApplicationData(0)* (cf. *getappdata(0)*). Ces deux structures sont accessibles dans l'espace de travail Matlab à l'aide des commandes respectives *getao* et *getad*. La commande *aoinit* permet d'initialiser ces deux structures (voir aussi le fichier *soleilinit.m)*.

1.3 Schéma de principe

Le Matlab Middle Layer définit une bibliothèque de fonctions permettant d'accéder ou bien au monde Tango via le binding Tango/Matlab [9], ou bien au simulateur : l'Accelerator Toolbox [2]. Il y a également la possibilité de se connecter à un modèle virtuel de la machine tournant sous Tango et simulant les équipements (pseudo devices Tango) et/ou la machine via le serveur Tracy2 (cf. Illustration 1.1). Le serveur Tracy2 permet de simuler la machine sans passer par l'Accelerator Toolbox (AT). La boîte à outil AT manipule une structure (*THERING*) décrivant l'accélérateur défini localement sur l'ordinateur où tourne Matlab.

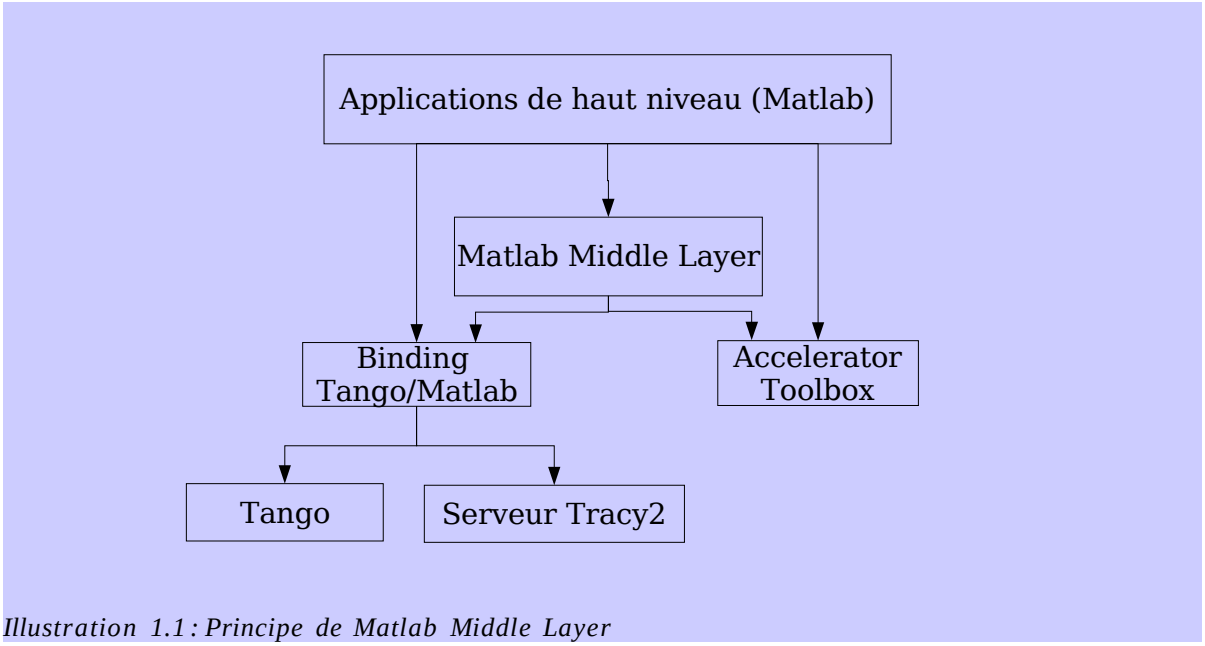


Illustration 1.1 : Principe de Matlab Middle Layer

Une des principales fonctions du MML est de traduire la nomenclature des attributs des équipements des accélérateurs en noms facilement maniables par des personnes non expertes du système de contrôle commande. Les équipements sont organisés en groupes, appelés *familles*, sous groupes (champs de ces familles) et « *devices*[1] » (équipements). Ainsi est-il possible de dialoguer avec les équipements de manière transparente en

[1] **Attention, cette notion de « device » n'est malheureusement pas identique à la notion de device TANGO.**

utilisant les mêmes noms, et ceci que la communication se fasse avec la machine en ligne ou avec le simulateur.

Le coeur du MML est la structure Matlab appelée *Accelerator Object* (AO). Elle contient les définitions de tous les éléments des différentes familles, la nomenclature Tango des équipements et de leurs attributs, les facteurs de conversion entre les unités physique et matérielle, etc. Cette structure est facilement éditable et configurable puisque que c'est un simple fichier texte (soleilinit.m, pour l'anneau de stockage de SOLEIL, LT1init, LT2init et boosterinit pour respectivement les lignes de transfert LT1, LT2 et le booster). L'AO réside en mémoire local dans la structure ApplicationData de l 'espace de travail de Matlab. Une structure complémentaire appelée *Accelerator Data* (AD) contient l'arborescence des fichiers et répertoires de sauvegardes, les paramètres de l'accélérateur. Comme l'AO, cette structure réside dans l'espace de travail de Matlab. La commande *aoinit* permet de configurer ces deux structures. L'annexe page 70 détaille comment faire l'initialisation.

Notons qu'il n'est pas possible d'avoir simultanément le MML de plusieurs accélérateurs chargés en mémoire Matlab. C'est à l'utilisateur de choisir l'anneau, LT2, LT2 ou le booster. L'environnement Matlab ne permet pas actuellement à l'uatilisateur de distinguer visuellement la machine chargée en mémoire. La commande *getfamilydata('Machine')* renverra cependant le nom de la machine.

Champ	**Description**
addaoprefix	– Ajout un prefixeà la strucutre AO (Non utilisé à SOLEIL)
aoinit	– Recherche la structure par défaut (soleilinit)
aokeep	– Modifie la structure AO en ne conservant que les familles spécifiées
cap	– Vide les structuresAO et AD
checkforao	– Verifie que la structure AO existe sinon appelle la fonction aoinit
getad	– Renvoie la structure AD
getao	– Renvoie la structure AO

Tableau 1.1 : Fonctions manipulant les structures AO et AD

2 Nomenclature du « Matlab Middle Layer »

La convention pour la nomenclature du MML se décline (en anglais) de la façon suivante :

2.1 Familles/devices

Le Tableau 2.1 contient les champ principaux usuels requis pour chaque familles du MML.

Champ	Description	Type Matlab
Family	Descripteur de groupe	chaîne de caractères
Field	Champ associé au descripteur	chaîne de caractères
DeviceList	[Cellule Elément_dans_cellule]	matrice deux colonnes
ElementList	Numéro d'élément dans la famille	vecteur colonne
CommonName	Nom commun	chaîne de caractères (facultatif)
DeviceName	Nom du device TANGO	chaîne de caractères

Tableau 2.1 Principaux champs pour une famille de l'Accelerator Object.

La nomenclature TANGO est définie dans la note « Nomenclature TANGO » [10].

2 Fonctions

Le nom des fonctions suit une nomenclature qui dépend du type d'action effectuée (cf. Tableau 2.2).

Préfixe	Rôle de la fonction
1. anal…	– Analyse d'un jeu de données
2. calc...	– Calcul ou conversion à partir d'un jeu préexistant de données
3. get…	– Lecture d'un paramètre, d'un attribut, d'un signal (valeurs de consigne non modifiées)
4. meas …	– Effectue une mesure et retourne un résultat (la valeurs de consigne son en général modifiées)
5. mon...	– Surveille un groupe d'attribut
6. ramp...	– Monte un énergie un groupe d'attributs
7. set...	– Assigne la nouvelle valeur de consigne
8. step...	– Incrémente une valeur de consigne
9. model...	– Action spécifique au modèle AT en ligne

Tableau 2.2 Nomenclature des fonctions

1 Familles définies dans le middle layer

Du point du vue du contrôle commande, chaque signal ou commande relative à un équipement a une nomenclature unique (nom de device, nom de l'attribut). Cependant cette nomenclature n'est jamais simple à manipuler. Le physicien des accélérateurs a, de plus, l'habitude de raisonner en termes de familles d'équipements, en nombre d'équipements d'une famille donnée, dans une cellule donnée. En effet, à chaque aimant sont, par exemple, associées une position, une force multipôlaire, une longueur, etc.

La liste des familles d'équipements pour l'anneau de stockage de SOLEIL estt donnée par le Tableau 2.3.

Famille	Fonction
BEND	Dipôle
Q1 à Q10	Quadrupôles de la famille 1 à 10
S1 à S10	Sextupôles de la famille 1 à 10
QT	Quadrupôles tournés
HCOR, VCOR	Correcteurs lents H et V
FHCOR, FVCOR	Correcteurs rapides H et V
BPMx, BPMz	BPM H et V
TUNE	Nombres d'ondes
RF	Fréquence RF
DCCT	Courant
MachineParameters	mode energy current lifetime

Tableau 2.3 Familles définies pour l'anneau

De manière similaire à la majorité des codes de simulation, la MML utilise ce type de convention mais associe pour chaque équipement un index pour l'élément d'une famille (*element index)* ainsi qu'un couple de valeur [cellule élément_dans_la_cellule] pour le « device » (*device index pair).* L'indexation des éléments correspond à leur position physique dans l'anneau (voir Tableau 2.4). Par exemple, le quatrième correcteur lent horizontal de l'anneau est désigné selon cette convention par (HCOR, 4). De manière équivalente, en utilisant la paire [cellule, élément], il est désigné par (HCOR, [1

4]). Les deux manières pour accéder à un équipement sont équivalentes. Suivant l'application, l'une ou l'autre sera privilégiée.

Par exemple, il y a 56 correcteurs lents dans chaque plan pour corriger l'orbite fermée. L'anneau possède 16 cellules. Le Tableau 2.4 illustre comment ces deux méthodes fonctionnent. En général, il est bien plus aisé d'utiliser et de se souvenir des équipements en termes de familles et localisation qu'en termes de nomenclature TANGO.

Par exemple, la fonction *getam* est utilisée pour obtenir la valeur de relecture d'un attribut; *getam('HCOR',4)* retourne la valeur de relecture du quatrième correcteur lent horizontal de l'anneau. De manière équivalente, le même résultat peut être obtenu par la commande *getam('HCOR',[1 4]).* Toutes les fonctions acceptent des entrées sous forme de vecteurs. Autre exemple, la commande *getam('HCOR',[1 2;1 3;7 1])* retourne les valeurs de relecture des deuxième et troisième correcteurs de la cellule 1 ainsi que celle du premier correcteur de la cellule 7; *getam('HCOR') ou getam('HCOR', [])* retourne tous les éléments de la famille HCOR.

Méthode Famille index élément	Méthode Famille [cellule élément]	Nom complet de l'attribut TANGO
HCOR, 1	HCOR, [1,1]	ANS-C01/AEsim/S1-CH/current
HCOR, 2	HCOR, [1,2]	Non existant aujourd'hui (réservé)
HCOR, 4	HCOR, [1,4]	ANS-C01/AEsim/S6-CH/current
HCOR, 7	HCOR, [1,7]	ANS-C01/AEsim/S4-CH/current
HCOR, 8	HCOR, [2,1]	ANS-C02/AEsim/S8.1-CH/current
HCOR, 11	HCOR, [2,4]	ANS-C02/AEsim/S10.1-CH/current
HCOR, 12	HCOR, [2,5]	ANS-C02/AEsim/S10.2-CH/current
HCOR, 15	HCOR, [2,8]	ANS-C02/AEsim/S8.2-CH/current
- - -	- - -	
HCOR, 117	HCOR, [16,5]	ANS-C16/AEsim/S4-CH/current
HCOR, 120	HCOR, [16,7]	ANS-C16/AEsim/S1-CH/current

Tableau 2.4 Famille/ index élément, Famille/{cellule élément], nomenclature TANGO pour les correcteurs lents horizontaux de SOLEIL.

Il est important de ne pas modifier la nomenclature en cours du fonctionnement de SOLEIL, ce qui implique que la liste [cellule élément] et l'index de l'élément dans l'anneau ne doivent pas changer même si de nouveaux équipements sont introduits dans l'accélérateur au fil des années. Par exemple, chacun des 120 sextupôles de l'anneau est équipés de bobines secondaires pour réaliser les fonctions de correcteurs horizontaux, verticaux

et de quadrupôles tournés. Bien qu'au démarrage de SOLEIL, seuls 56 (sur 120) correcteurs dans les deux plans et 32 (sur 120) quadrupôles tournés soient prévus, l'emplacement des autres est réservé. Les fonctions *dev2elem* et *elem2dev* permettent de passer d'une liste indexée « element » à une liste « device » ([cellule élément]). Toutes les fonctions du MML utilisent ces deux fonctions. Il est également possible de référencer un attribut d'un équipement par sa nomenclature TANGO et par une nom commun (« commonname »). Pour plus de détails, le lecteur est prié de se reporter en annexe page 70.

2 Les modes

Chaque commande de base peut être appelée en spécifiant comme argument un « mode » pouvant être l'un de ceux du Tableau 2.5.

« Online »	– Accès à Tango (machine en ligne)
« Simulator »	– Accès au simulateur AT
« Model »	– Idem simulateur ?
« Manual »	– L'utilisateur entre à la main la valeur
« Special »	– Fonction spéciale

Tableau 2.5 Différents modes de fonctionnement des commandes du MML.

Exemples :

1. *getam('Q1',1, 'Online')*
   Lecture de l'attribut TANGO « courant » du premier quadrupôle de la famille Q1
2. *getam('Q1',1, 'Simulator')*
   Lecture du courant du premier quadrupôle de la famille Q1 du modèle AT
3. *getam('Q1',1, Manual)*
   %Matlab demande la valeur de relecture de Q1 [1 1] à l'utilisateur
   >> Manual input: Q1(1,1) [ampere] =

Fonctions de bases de la couche intermédiaire

1 Introduction

Bien que la bibliothèque de fonctions de la MML soit bien établie, elle ne va pas cessée d'être complétée au cours des ans. A chaque fois que possible, toute nouvelle fonction doit être écrite de manière indépendante de la machine. Bien sur, cette règle est parfois difficile à suivre. La présente section décrit les fonctions de bases nécessaires au bon fonctionnement du MML.

2 Fonctions de base du MML

2.1 Fonctions de base lecture/écriture

Ces fonctions (cf. Tableau 3.1) permettent de communiquer avec les équipements en ligne, les « devices serveurs » TANGO, ou l'Accelerator Toolbox. Les trois fonctions « mère » de cette classe de fonctions sont *getpv* (lecture d'une valeur de contrôle – *Process variable*), *setpv* (assignation d'une valeur de contrôle – *Process variable*) et *stepv* (assignation d'une valeur de contrôle par incrément– *Process variable*). Ces trois fonctions peuvent être appelées avec une multitude d'arguments (familles, ensembles d'équipements hétérogènes, information temporelle, etc.). Pour plus d'information sur ces fonctions clefs, le lecteur voudra bien se reporter aux annexes page 52. Les suffixes des différentes fonctions accédant aux données (« database access functions ») dérive historiquement du monde Epics :

- **pv** – Process Variable : valeur de contrôle
- **am** – Analog Monitor : valeur de relecture
- **sp** – Setpoint : valeur de consigne

Nom	**Fonctionnalité**
getpv	– Lit un groupe de variables de contrôle
setpv	– Écrit un groupe de variables de contrôle
steppv	– Écrit par incrément un groupe de variables de contrôle
getam	– Lit la valeur de relecture d'un groupe d'attributs
getsp	– Lit la valeur de consigne d'un groupe d'attributs
setsp	– Assigne un groupe de valeurs de consigne
stepsp	– Assigne par incrément la valeur de consigne d'un groupe d'attributs
ramppv	– « Rampe » un groupe de valeurs de contrôle

Tableau 3.1 Fonctions pour écrire ou lire une valeur de contrôle

2.2 Fonctions pour changer de mode de la session Matlab

Le Tableau 3.2 contient les noms des fonctions à appeler pour changer le comportement global du MML pour passer en unités physiques (« Physics »), matérielles (« Hardware ») et pour passer du simulateur à la machine en ligne.

Version 2

Nom	**Fonctionnalité**
switch2sim	– Change le mode courant en mode simulateur
switch2online	– Change le mode courant en mode machine en ligne
switch2physics	– Change l'unité courante en unité physique (SI)
switch2hw	– Change l'unité courante en unité matérielle

Tableau 3.2 Fonctions pour modifier le comportement du MML

3 Fonctions de conversion entre les différentes nomenclatures

Les principales fonctions pour passer d'une nomenclature à une autre sont données par le Tableau 3.3.

Nom	**Fonctionnalité**
builddevlist	– Construit une liste [cellule équipement]
common2dev	– Conversion nom commun/liste [cellule équipement]
common2family	– Conversion nom commun/famille
common2handle	– Conversion nom commun/référence
common2tango	– Conversion nom commun/TANGO
dev2elem	– Conversion liste équipement/liste index éléments
dev2tango	– Conversion liste équipement/TANGO raccourci de family2dev
elem2dev	– Conversion liste élément/liste [cellule équipement]
family2atindex	– Conversion famille/ index dans AT
family2common	– Conversion famille/nom commun
family2dev[2] ou *getlist*	– Conversion famille/liste [cellule équipement]
family2elem	– Conversion famille/list elements
family2handle	– Conversion famille/référence
family2status	– Donne le statut d'un équipement (1 - en opération, 0 - retiré de la liste)
family2tango	– Conversion famille/TANGO
tango2common	– Conversion TANGO/nom commun
tango2dev	– Conversion TANGO/liste [cellule équipement]
tango2family	– Conversion TANGO/famille
tango2handle	– Conversion TANGO/référence

Tableau 3.3 Fonctions de passage d'une nomenclature à une autre.

4 Fonctions pour obtenir des données du MML

[2] Le comportement par défaut de la fonction family2dev est crucial car de nombreuses fonctions dont *getpv*/*setpv* utilisent cette fonction pour obtenir la liste par défaut [cellule élément] si elle n'est donnée par l'utilisateur comme argument de la commande.

Version 2

Les principales fonctions servant à obtenir des données provenant de diverses sources (fichiers, configuration, etc.) sont :

1. La fonction *getfamilydata* fournit des paramètres sur une famille d'équipements et sur les paramètres du système de contrôle commande.
2. La fonction *getphysdata* fournit des données physiques.
3. La fonction *getdata* permet de charger des données à partir d'un fichier.

La plupart des fonctions énumérées ci-après (Tableau 3.4) ne sont que des alias vers ces trois fonctions.

Nom	**Fonctionnalité**
checklimits	– Verifie qu'une valeur de consigne n'exède pas ses limites (butées logicielles min et max)
family2datastruct	– Renvoie la struutre associée à une famille
family2mode	– Retourne le mode associé au champ d'une famille
family2struct	– Retourne la structure associée à une famille
family2tol	– Retourne la tolérance valeur de (relecture-consigne) pour un champ d'une famille
family2units	– Retourne l'unité associée au champ d'une famille
findkeyword	– Cherche un mot clef dans une structure
findmemberof	– Affiche tous les membres d'un groupe de familles
findrowindex(list1,list 2)	– Rentourne les indices éléments de la liste2 se trouvant dans la liste1
getdata	– Recherche dans un fichier une structure correspondant à la famille spécifiée.
getfamilydata	– Retourne un champ donné pour une famille
getfamilylist	– Retourne la liste des familles
getfamilytype	– Retourne la liste des types de familles
getgolden	– Retourne les valeurs de référence pour une famille
getmaxsp	– Retourne la valeur maximum de consigne
getminsp	– Retourne la valeur minimum de consigne
getmode	– Retourne le mode d'une famille
getoffset	– Retourne les offsets d'une famille
getphysdata	– Retourne les valeurs de calibration (gain, offset, etc.)

Nom	**Fonctionnalité**
getsigma	– Retourne l'écart type qui a été précédemment mesuré
getspos	– Retourne la position « s » d'un élément dans l'anneau
gettol	– Retourne la tolérance acceptée entre valeurs de consigne et de relecture
getunits	– Retourne les unités d'une famille
isfamily	– Vérifie que la famille est définie
ismemberof	– Vérifie que la famille est membre d'un groupe donné
setfamilydata	– Assigne un champ donné pour une famille
setphysdata	– Assigne les valeurs de calibration
setrange	– Lit et assigne les valeurs max and min lues dans la base de données statique TANGO
showao	– Affiche tous les champs d'une structure AO contenant des champs TANGO
showfamily	– Affiche les champs AO d'une famille

Tableau 3.4 Fonctions générales et génériques pour accéder aux données du MML

Exemples :

1. *getfamilydata('BPMx')*
   Renvoie le champ AO.BPMx
2. *getfamilydata('BPMx','Monitor')*
   Renvoie le champ AO.BPMx.Monitor
3. *getfamilydata('BPMx','Monitor','Units')*
   Renvoie le champ AO.BPMx.Units
4. *getfamilydata('HCOR', 'Status')* équivalent à *getstatus('HCOR')*
   Renvoie le champ AO.HCOR.Status
5. *getfamilydata('Machine')*
   Renvoie le nom de l'accélérateur chargé dans le MML.
6. *getphysdata(Family, 'Golden')*
   Renvoie les valeurs de référence de la famille *Family*.

5 Sauvegarde/restauration d'une configuration machine

5.1 La fonction *getmachineconfig*

La fonction *getmachineconfig* lit les valeurs de consignes des aimants et l'orbite puis les sauvegarde dans un fichier ou une variable. Voir l'aide en ligne (*help getmachinconfig*) pour plus de détails.

Notes :

1. *getmachineconfig* sauvegarde toutes les valeurs d'alimentation (si elles sont membres de 'MachineConfig' ou de 'RF' et l'orbite 'BPM') dans le fichier *GoldenLattice.mat*
2. Utiliser *setmachineconfig* pour sauvegarder dans un fichier la sortie de la commande précédente
3. Utiliser *getmachineconfig('Archive')* pour archiver la configuration dans un fichier
4. Les familles inconnues sont ignorées
5. Utiliser *getmachineconfig('Golden')* pour faire de la configuration actuelle la configuration de référence
6. *getmachineconfig('Q1','monfichier')* archive la configuration actuelle pour la famille 'Q1' dans le fichier *monfichier*
7. *Q1conf =getmachineconfig('Q1')* fait la même chose que précédemment sans archiver.

5.2 La fonction *setmachineconfig*

La fonction *setmachineconfig* assigne à l'accélérateur les valeurs consignes lues dans un fichier de configuration ou depuis la structure produite par la commande *getmachineconfig*. Voir l'aide en ligne (*help setmachinconfig*) pour plus de détails.

Notes :

1. *setmachineconfig* restaure toutes les valeurs de consignes pour les familles membres de 'MachineConfig', 'RF' ou 'BPM' à partir d'un fichier de configuration (fenêtre interactive)
2. *setmachineconfig* restaure toutes les valeurs de consignes à partir du fichier de référence *GoldenLattice.mat*
3. *setmachineconfig({'HCOR','VCOR'}, 'Golden')* ne restaure que les valeurs de consignes des correcteurs lents (valeurs de référence)
4. *setmachineconfig({'HCOR','VCOR'}, 'Golden','Simulator')* ne restaure que les valeurs de consignes des correcteurs lents (valeurs de référence) dans le simulateur

5.3 IHM configgui

L'IHM *configui* (Illustration 3.1) permet de réaliser ces deux fonctions graphiquement que ce soit pour une famille ou un ensemble de familles d' équipements.

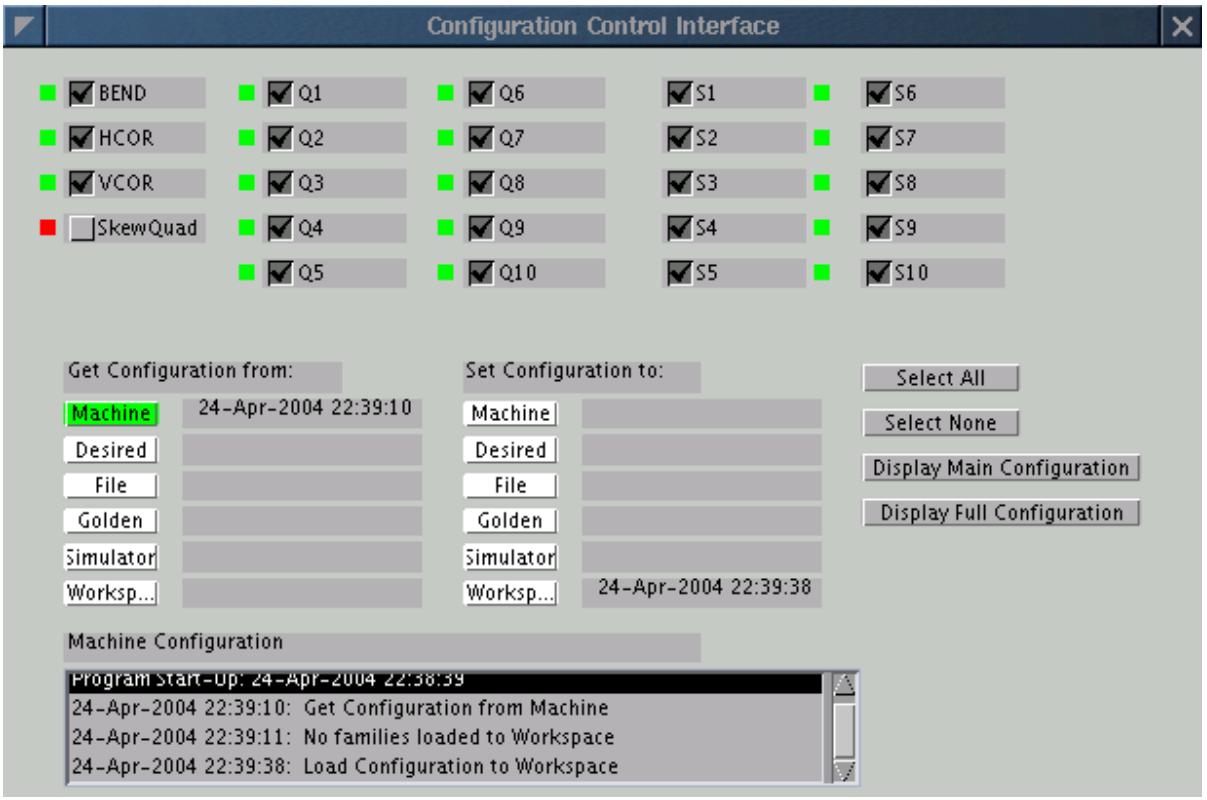


Illustration 3.1 Interface *graphique* configui pour sauvegarder et restaurer une configuration machine.

Alias de fonctions courantes

1 Définition

Pour les fonctions les plus utilisées, il est agréable de ne pas avoir à chaque fois à entrer une multitude d'arguments. C'est pour cette raison que des fonctions dites raccourcies (alias) ont été définies. Par exemple, les fonctions *getam* et *getsp* sont toutes deux des alias de la fonction générique *getpv* avec l'argument 'Monitor' et 'Setpoint' respectivement. Il en est de même pour les fonctions *setsp* et *stepsp*.

2 Quelques alias de la fonction getpv

Quelques alias usuels de fonctions sont données par le Tableau 4.1.

Nom	**Fonctionnalité**
getbpm	– Retourne l'orbite horizontale et verticale
getdcct	– Retourne le courant stocké
getrf/setrf	– Retourne/assigne la fréquence RF
gettune	– Retourne les nombres d'ondes
getx	– Retourne l'orbite horizontale
getz	– Retourne l'orbite verticale

Tableau 4.1 Exemples d'alias de fonction

Notes :

1. Certains de ces raccourcis appartiennent aux fonctions qui sont introduites dans la section suivante. Par exemple, si 'TUNE' est une famille, alors la fonction *gettune* est juste un alias vers la fonction *getam* ('TUNE'). Cependant, faire de 'TUNE' une famille peut ne pas avoir de sens dans certains accélérateurs. C'est pourquoi il a été choisi de faire une catégorie de fonctions spéciales.
2. L'utilisation de fonctions comme alias d'autres fonctions permet d'écrire plus facilement des fonctions de haut niveau tout en conservant une indépendance par rapport à une machine spécifique.

Fonctions spéciales

1 Introduction

Certains équipements ne peuvent pas facilement être introduits dans l'Accelerator Object. Des fonctions particulières ont ainsi été écrites pour un accès direct. Ceci peut être le cas pour accéder à des variables ne dépendant pas expressément de TANGO. Le fichier de configuration de l'Accelerator Object peut souvent être encore utilisé pour définir ces cas particuliers. Par exemple, les nombres d'ondes sont obtenus via une fonction spéciale et constituent encore une famille de l'AO. Les fonctions spéciales qui ne se réfèrent pas à l'AO ne sont plus indépendantes de la machine. Le Tableau 5.1 donne quelques exemples.

Nom	Fonctionnalité
getid/setid	– Lit/assigne la valeur d'entrefer et la vitesse d'un élément d'insertion
getepu/setepu	– Lit/assigne la phase longitudinale d'un Apple II
getlifetime	– Lit la durée de vie (si l'attribut n'existe pas, utiliser la fonction *measlifetime*)
getrfcavitytemperature/setrfcavity temperature	
getrfpower / setrfpower	
getscrap/setscrap	– Lit/assigne la position des scrapers
getbpmv	
power supply functions	
temperature monitors	
vacuum pressure functions	

Tableau 5.1Exemples de fonctions spéciales

Fonctions pour la Physique Machine

1 Introduction

Le premier objectif des fonctions du MML est de fournir un support pour accéder (lecture/écriture) à la fois aux équipements des accélérateurs et au simulateur de faisceau. L'étape suivante consiste à compléter ces bibliothèques de fonctions pour faire de la Physique des Accélérateurs. Cette section devrait s'enrichir au fur et à mesure de la vie de l'accélérateur et en fonction des besoins des utilisateurs du Matlab Middle Layer.

2 Fonctions générales pour la Physique Machine

Tableau 6.1 contient une liste de fonctions générales relatives à la Physiques Machine. Ce tableau sera complété si besoin.

Version 2

Nom	**Fonctionnalité**
amps2mm	– Pour un courant donner, donne le maximum d'orbite crée. Alglo: mm(i,j) = max(R(:,j).Amps(i) où R est la matrice réponse et Amps un vecteur de courant.
bend2gev	– Convertit un courant dipolaire en variation d'énergie (possibilité d'introduire la contribution des correcteurs)
bpm2orbit	– Calcule l'angle et la position au centre d'une insertion à partir des lectures de IDBPM.
buildlocoinput	– Construit un fichier d'entrée pour LOCO
bumpinj	– Crée un bump d'injection
getbrho	– Retourne le Brho de la machine
getcircumference	– Retourne la circonférence de la machine
getenergy	– Retourne l'énergie de la machine
getmcf	– Retourne le momentum compaction factor
gev2bend	– Convertit l'énergie du faisceau en courant des dipôles
hw2physics	– Conversion d'unité matérielle vers unité physique
measbpmsigma	– Mesure l'écart type des orbites H et V
measchro	– Mesure les chromaticités
measdisp	– Mesure les fonctions dispersion
measlifetime	– Calcul la durée de vie du faisceau à partir de la lecture de courant par un algorithme des moindres carrés
mm2amps	– Convertit un changement en un changement en courant d'un correcteur (utilise une matrice réponse)
monbpm	– Surveille, dessine et calcule les moments statistiques des BPM
monmags	– Surveille, dessine et calcule les moments statistiques des courant des aimants
physics2hw	– Conversion d'unité physique vers unité matérielle
plotdisp	– Affiche la fonction dispersion mesurée
raw2real	– Convertit données brutes (raw) en données calibrées (real) (LOCO)
real2raw	– Convertit données calibrées (real) en données

Nom	**Fonctionnalité**
	brutes (raw) (LOCO)
setangle4	– Bump d'angle avec 4 correcteurs
setbump4	– Bump d'orbite avec 4 correcteurs
setchro	– Change les valeurs valeurs de chromaticités
settune	– Change les valeurs des nombres d'ondes
stepchro	– Incrémente les valeurs de chromaticités
steptune	– Incrémente les valeurs des nombres d'ondes

Tableau 6.1 Fonctions générales pour la Physique Machine

3 Matrices réponse

Les fonctions permettant de lire et mesurer les matrices réponses sont données par le Tableau 6.2. Les fichiers de référence pour les matrices réponses sont obtenus par la commande *getfamilydata('OpsData','RespFiles')* :

>> 'GoldenBPMResp' 'GoldenTuneResp' 'GoldenChroResp' 'GoldenDispResp'

Voir l'annexe page 57 pour savoir comment faire d'une matrice réponse mesurée la matrice de référence.

Nom	**Fonctionnalité**
getrespmat	– Lit une matrice réponse à partir d'un fichier (fonction générale)
1. *getbpmresp*	– Lit une matrice réponse des BPM
2. *gettuneresp*	– Lit une matrice réponse des nombres d'ondes
3. *getchroresp*	– Lit une matrice réponse des chromaticités
4. *getdispresp*	– Lit une matrice réponse de la dispersion (non écrit)
measrespmat	– Mesure une matrice réponse (fonction générale)
1. *measbpmresp*	– Mesure une matrice réponse des BPM
2. *measdispresp*	– Mesure une matrice réponse des BPM (facteur d'Amman?)
3. *measchroresp*	– Mesure une matrice réponse des chromaticités
4. *measTuneresp*	– Mesure une matrice réponse des nombres d'ondes
plotbpmresp	– Affiche la matrice réponse des BPM (à écrire)

Tableau 6.2 Fonctions manipulant les matrices réponse

4 Feedforward pour les insertions

A faire plus tard ... Ces fonctions n'ont pas encore été testées à SOLEIL

Nom	**Fonctionnalité**
ffgettbl	– Lit une nouvelle table de feedforward
fftest	– Teste la table actuelle de feedforward
ffanal	– Analyse la table actuelle de feedforward

Tableau 6.3 Fonctions relatives au feedforward des insertions.

5 Vérification du système

A faire plus tard ... Ces fonctions n'ont pas encore été testées à SOLEIL

Nom	**Fonctionnalité**
getrate	– measures the data rate for a channel (channel must be noisy, ie, changes every update)

Nom	**Fonctionnalité**
checkbpms	– checks if the BPMs are functioning (based on response matrix)
checkmags	– checks the magnets (setpoint, tolerance, on/off, etc)
checkorbit	– checks the orbit (based on golden orbit)
magstep	– checks the step response of a corrector magnet
checkmachine	– look for errors in the storage ring
Power supply problems	
Orbit errors	
Temperatures	
Vacuum	
…	

Tableau 6.4Fonctions pour vérifier le bon fonctionnement du système de contrôle commande et des équipements.

6 Fonctions propre au modèle en ligne

Le Matlab Middle Layer peut fonctionner indépendamment de l'Accelerator Toolbox (AT). Cependant, il est très pratique de pouvoir passer rapidement du modèle à la machine en ligne, et vice-versa. Les deux fonctions *switch2sim* et *switch2online* permettent de passer d'un mode à l'autre de manière aisée. Il est également possible de spécifier l'un des deux modes en argument d'une fonction donnée. Il est néanmoins utile de pouvoir disposer d'un jeu de fonctions dédiées au modèle AT. Le Tableau 6.5 contient ces fonctions propres au modèle en ligne.

Version 2

Nom	**Fonctionnalité**
getcircumference	– Retourne la circonférence de la machine
getenergymodel	– Retourne l'énergie de la machine
modelbeta	– Calcule les fonctions bêtatrons
modelchro	– Calcule les chromaticités
modelcurlh	– Calcule les fonctions H rondes
modeldisp	– Calcule les fonctions dispersion
modelmcf	– Calcul le momentum compaction factor
modelphase	– Calcule les avances de phase
modeltune	– Calcul les nombres d'ondes
modeltwiss	– Calcule mes fonctions de Twiss
machine2sim	– Copie les valeurs de consigne de la machine dans le modèle
sim2machine	– Copie les valeurs de consigne du modèle dans la machine
modelsextu_on SETSEXTU('on'/'OFF')	– Active les sextupôles dans le modèle AT
modelsextu_off	– Desactive les sextupôles dans le modèle AT
atsummary	– Retourne les principaux paramètres physique du modèle
buildatindex	– Retourne les indices d'une famille dans la structure THERING
getatfield	– Retourne le champ AT d'une famille
getharmonicnumber	– Retourne le nombre d'harmoniques
getkleff	– Retourne la force intégrée d'un aimant
getleff	– Retourne la longueur effective d'un aimant
getnusympmat	– Retourne les nombres d'ondes calculés de manière symplectique
setatfield	– Ecrit le champ AT d'une famille
setcavity	– Eteint/allume les cavités RF
setenergymodel	– Modifie l'energie du modèle.
plotcod	– Affiche l'orbite fermée en utilisant findorbit4 (pas de cavité ou de radiation pris en compte).
plotmodelorbit	– Affiche l'orbite fermée en utilisant modeltwiss.

Version 2

Nom	**Fonctionnalité**
ploteta	– Affiche la fonction dispersion
plotbeta	– Affiche les fonctions béta avec la maille (masque celle venant avec AT).

Tableau 6.5 Fonctions propres au modèle AT

7 Fonctions diverses

Afin de faciliter l'utilisation de Matlab, la bibliothèque de fonctions de Matlab a été enrichie (Tableau 6.6).

Nom	**Fonctionnalité**
addlabel	– Ajoute une légende à une figure
appendtimestamp	– Ajoute la date et l'heure à la fin d'une chaîne de caractères.
gettime	– Temps en secondes (t0 différents sous Windows et Linux)
prependtimestamp	– Ajoute la date et l'heure au début d'une chaîne de caractères.
popplot	– Extrait un système d'axe dans une nouvelle figure
sleep	– Suspend une fonction pendant « n » secondes
xaxis	– Modifie uniquement l'axe horizontal d'un système d'axes
xaxiss	– Modifie tous les axes horizontaux d'une figure
yaxis	– Modifie uniquement l'axe vertical d'un système d'axes
yaxiss	– Modifie tous les axes verticaux d'une figure
zaxis	– Modifie uniquement l'axe des cotes d'un système d'axes
maxn	– Maximum d'une matrice
minn	– Minimum d'une matrice
editlist	– Editeur graphique de la liste de statuts
rload	– à détailler
gotodirectory	– Va dans un répertoire donné et crée l'arborescence si nécessaire
gotoat	– Va dans le répertoire racine de AT
where	– Recherche toutes les occurrences d'un fichier (cf . *which -all*)
locate	– Recherche toutes les occurrences d'un fichier dans le chemin MatMatlab
cddataroot	– Va dans le répertoire de mesures
cdopsdata	– Va dans le répertoire de fichiers de référence

Tableau 6.6 Fonctions diverses enrichissant les bibliothèques de Matlab.

Gestion de données

1 Introduction

Gérer l'ensemble des données nécessaires au démarrage, réglage et fonctionnement des différents accélérateurs exige un travail considérable. Pour centraliser les données, l'utilisation de bases de données simplifie grandement cette tâche. Cependant, une telle solution requiert la collaboration et la coordination de personnes de différents groupes (Physique Machine, Groupe Diagnostics, Groupe ICA, etc.). A SOLEIL, il est prévu d'avoir à terme un système de gestion de données centralisées. En attendant, pour éviter que le chaos ne s'établisse, une partie de cette gestion est prise en compte par le MML. Cette solution, fondée sur l'utilisation de fichiers, est temporaire. Notons que dès aujourd'hui une partie des données de configuration est déjà obtenue en interrogeant la base de données statique de TANGO. Un outil générique de gestion des fichiers de consigne est également en cours de développement au sein de la division informatique.

Les données précédemment citées peuvent être classées sous plusieurs catégories :

1. Des données quasi-statiques
2. Des données nécessitant une mise à jour régulière.
3. Des données relatives archivables

2 Données machine quasi-statiques

Ces données sont par exemple celles :

1. Permettant de faire la conversion entre les unités physiques et matérielles.
2. Définissant les valeurs maximales et minimales pour les valeurs de consignes.
3. Définissant la position des équipements dans l'anneau
4. Relatives au cyclage des aimants (hystérésis)
5. Relatives à la calibration des aimants

Autant que possible, nous chercherons à centraliser ces données physiques quasi-statiques dans une base de données (à définir). Une partie de ces données est déjà obtenue à partir de la base de données statique de TANGO.

3 Données physiques nécessitant une mise à jour régulière

Ces données sont obtenues par exemples :

1. lors d'un ré-alignement de la machine
2. lors du « beam based alignement » (recherche du centre des quadrupôles)
3. lors de la calibration du modèle (utilisation de LOCO)

   Paramètres de référence (« Golden parameters »)
   - Orbite
   - Nombres d'ondes
   - Chromaticités
   - Paramètres RF
   - Feedbacks
   - Injection
   - Couplage
   - etc.
4. Fichiers de consignes : sauvegarde/restauration partielle ou totale d'un état des accélérateurs.
5. Matrices réponse
   - des correcteurs aux BPM
   - des quadrupôles aux nombres d'ondes
   - des sextupôles aux chromaticités
   - des fonctions dispersion aux quadrupôles
6. Les offsets des BPM, aimants
7. Les tables de feedforward pour compenser l'effet des insertions sur la dynamique faisceau
8. etc.

4 Sauvegarde et restauration de paramètres machine

Bien que la plupart des attributs et paramètres soient sauvegardés dans la base de données historique et la base de données intermédiaire selon différents modes, il est nécessaire, actuellement, d'avoir également un archivage propre au MML pour les raisons suivantes. Tout d'abord, il est souvent plus pratique de sauvegarder directement des données sans passer par une base de données ; il est souvent difficile de retrouver simplement et rapidement des données dans une base données (granularité, difficulté à parcourir les données archivées ...). Ensuite, un certain nombres de données physiques ne sont pas associées à un équipement, mais sont dérivées de mesures indirectes plus ou moins complexes, de calculs physique (la dispersion, la chromaticité, ...).

Actuellement l'ensemble de ces données est sauvegardé dans des fichiers dont l'arborescence est définie dans l'objet « Accelerator Data » (*getad).*

```
controlroom
|-- measdata
|   |-- Boosterdata
|   |   |-- BPM
```

```
|  |  |-- Bumps
|  |  |-- Chromaticity
|  |  |-- Dispersion
|  |  |-- MachineConfig
|  |  |-- PhysData
|  |  |-- Response
|  |  |  |-- BPM
|  |  |  |-- Chrom
|  |  |  |-- Disp
|  |  |  `-- Tune
|  |  `-- Tune
|  |-- LT1data
|  |  |-- MachineConfig
|  |  |-- emittance
|  |  `-- fae
|  |-- LT2data
|  `-- Ringdata
|     |-- BPM
|     |-- Backup
|     |-- Bumps
|     |-- Dispersion
|     |-- Loco
|     |-- MachineConfig
|     |-- Magnets
|     |-- PhysData
|     |-- Response
|     |  |-- BPM
|     |  |-- Chrom
|     |  |-- Disp
|     |  |-- Skew
|     |  `-- Tune
|     `-- Tune
controlroom/mmlcontrol/
|-- opsdata
|  |-- Booster
|  |-- LT1
|  |-- LT2
|  `-- Ring
```

Nom	**Fonctionnalité**
cddataroot	Aller au répertoire racine de

Version 2

Nom	**Fonctionnalité**
	données(measdata/nom_machine)
cdopdata	Aller au répertoire de fichier de consignes (opsdata/nom_machine)

En attente d'un outil générique de génération et de gestion des fichiers de consignes(fourniture groupe ICA), l'application IHM suivant a été écrite pour gérer les fichiers de consignes des électro- aimants.

Elle permet de lire/écrire des fichiers de consignes sur :

- MACHINE : la machine (via TANGO)
- FILE : un fichier de consignes
- GOLDEN : le fichier nominal de consignes
- SIMULATOR : le modèle en ligne (via AT)
- WORKSPACE : depuis l'espace de travail Matlab.

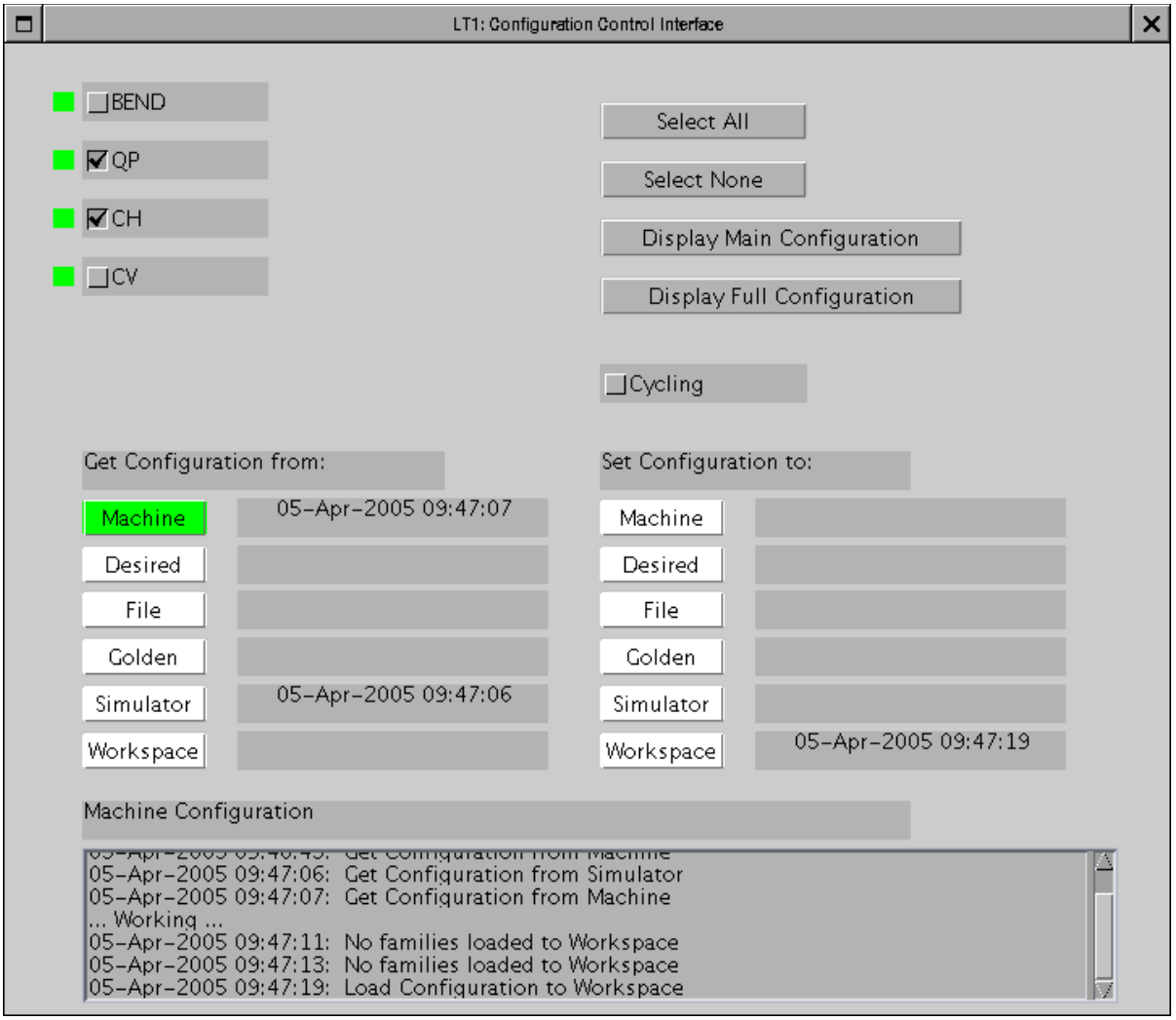
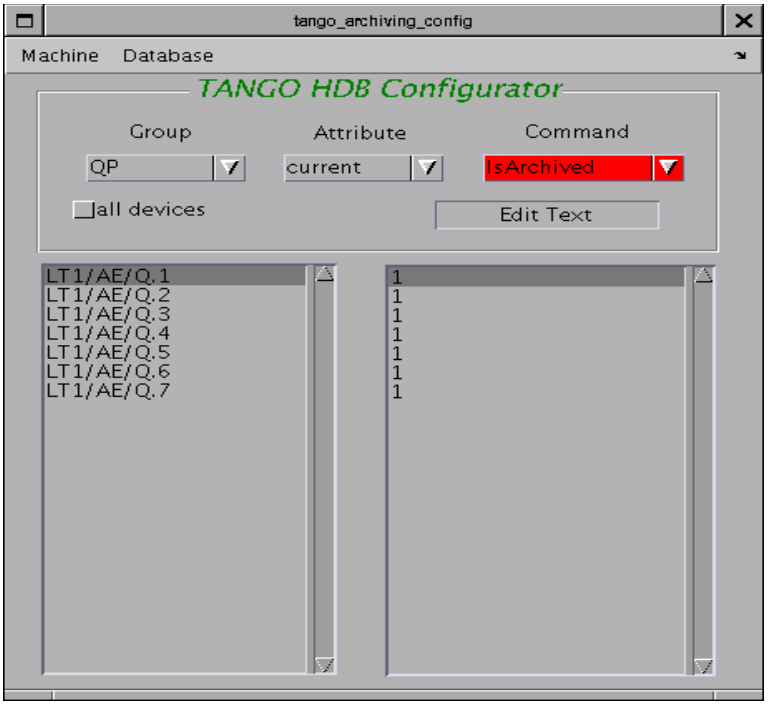


Illustration 7.1 Interface pour gérer les fichiers de consignes

5 Archivage

Fonctions propres aux bases de données historiques et temporaires :

En attente d'un outil générique écrit par le groupe ICA, une interface homme/machine (IHM) a été rapidement écrite pour configurer l'archivage TANGO : *tango_archiving_config*



Fonctions permettant de lire et d'afficher des attributs archivés dans la base de données historiques ou intermédiaires :

Champ	**Contenu**
arread	Lit une attribut archivé
arrplot	Affiche le resultat d'une commande arread

Structuration des données

Par soucis d'homogénéité, les données sont sauvegardées dans un format consistant. Les commandes *getpv*, *getam*, *getsp*, *getx*, *gety*, *getrf*, *getdisp*, *getchro*, etc utilisées avec le paramètre facultatif 'Struct', retournent la structure suivante définie par le Tableau 8.1.

Version 2

Champ	Contenu	Type Matlab
Data	Données	vecteur
FamilyName	Nom de la famille	string
Field	Field to set of get (string)	string
DeviceList	Device list (2-column matrix)	Matrice colonne
Mode	Online' ou 'Simulator'	string
Status	1-device ok	entier
	0-device bad	
t	Temps de début de mesure (Matlab)	vecteur
tout	Temps de fin de mesure (Matlab)	vecteur
TimeStamp	Première horodatage (TANGO) matriel au format Matlab	double
TimeStampMatlab	Horloge Matlab quand la mesure débute	double
GeV	Energie (GeV	double
Units	'Physics'	string
	'Hardware'	
UnitsString	unité de mesure	string
DataDescriptor	Description ('Horizontal Orbit', 'Vertical Dispersion)	string
CreatedBy	Nom de l'application générant les données	string

Tableau 8.1 Format type pour une mesure dans le MML

A chaque fois que possible, il est recommandé de définir le format de cette structure tel que défini par le Tableau 8.1. Un grand nombre de fonctions ont une option 'Archivage' qui sauvegarde automatiquement des données sous cette forme dans un sous répertoire du répertoire <DataRoot> (Voir la commande *DataRoot=getfamilydata('Directory', 'DataRoot')* pour connaître la localisation des fichiers de sauvegarde).

Les matrices réponses retournées par les commandes *measrespmat, measbpmresp, etc,* ont un format de structure légèrement différent. Voir *help measrespmat* et la section suivante pour plus de détails. La chromaticité et la fonction dispersion ont une structure de retour proche de celle des matrices réponse avec des champs supplémentaires nécessaires pour des mesures spécifiques (voir *help measdisp* et *meachro* pour plus de détails).

Mesure de matrices réponse

La fonction *measrespmat* est la fonction utilisée pour toute mesure de matrice réponse entre familles d'actionneurs (correcteur, quadrupôles, sextupôles) et familles d'observables (BPM, nombres d'ondes, chromaticités).

```
>> help measrespmat

  For family name, device list inputs:
  S = measrespmat(MonitorFamily, MonitorDeviceList, ActuatorFamily, ActuatorDeviceList,
                  ActuatorDelta, ModulationMethod, WaitFlag, ExtraDelay)

  For data structure inputs:
  S = measrespmat(MonitorStruct, ActuatorStruct, ActuatorDelta, ModulationMethod,
                  WaitFlag, ExtraDelay)

  Inputs:
    MonitorFamily        - AcceleratorObjects family name for monitors
    MonitorDeviceList    - AcceleratorObjects device list for monitors (element or device)
    or MonitorStruct can replace MonitorFamily and MonitorDeviceList

    ActuatorFamily       - AcceleratorObjects family name for actuators
    ActuatorDeviceList   - AcceleratorObjects device list for actuators
                           (element or device) or ActuatorStruct can replace
                           ActuatorFamily and ActuatorDeviceList

    ActuatorDelta     - AcceleratorObjects family name for monitors
    ModulationMethod  - Method for changing the ActuatorFamily
                        bipolar'  changes the ActuatorFamily by +/- ActuatorDelta
                                  on each step {default}
                        unipolar' changes the ActuatorFamily from 0 to ActuatorDelta
                                  on each step
    WaitFlag          - (see setpv for WaitFlag definitions) {default: -1}
    ExtraDelay        - extra time delay [seconds] after a setpoint change to wait before
                        reading the MonitorFamily {default: 0}

  Output:
    S = the response matrix.

    If 'struct'  is an input, the output will be a response matrix structure
                              {default for data structure inputs}
    If 'numeric' is an input, the output will be a numeric matrix
                              {default for non-data structure inputs}

  Notes:
  1. If MonitorFamily and MonitorDeviceList are cell arrrays, then S is a cell array
     of response matrices.
  2. ActuatorFamily, ActuatorDeviceList, ActuatorDelta, ModulationMethod, WaitFlag are
     not cell arrrays.
  3. If ActuatorDeviceList is empty, then the entire family is change together.
  4. Bipolar mode changes the actuator by +/- ActuatorDelta/2
  5. Unipolar mode changes the actuator by ActuatorDelta
  6. Return values are MonitorChange/ActuatorDelta (normalized)
  7. When using cell array inputs don't mix structure data input with non-structure data

  Examples:
  1. 2x2 tune response matrix for QF and QD families (delay for tune matrix will need
     to be addjusted):
         TuneRmatrix = [measrespmat('TUNE',[1;2],'QF',[],2,'unipolar') ...
                        measrespmat('TUNE',[1;2],'QD',[],2,'unipolar')];

  2. Orbit response matrix for all the horizontal correctors (+/- 1 amp kick size):
     Smat = measrespmat({'BPMx','BPMy'}, {getlist('BPMx'),getlist('BPMy')}, 'HCM', ...
                                     getlist('HCM'),ones(size(getlist('HCM'),1),1), ...
                                     'bipolar',- 2);
     The output is stored in a cell array.  Smat{1} is the horizontal plane and Smat{2}
     is the vertical cross plane.
     For struct outputs,
     Smat = measrespmat({'BPMx','BPMy'}, {getlist('BPMx'),getlist('BPMy')}, 'HCM',
            getlist('HCM'),ones(size(getlist('HCM'),1),1),'bipolar',- 2,'struct');
```

La matrice réponse, Rmat, est sauvegardée en respectant le format défini dans le Tableau 9.1.

Data:	[Response matrix]
Monitor:	[Data Structure for the Monitor]
Actuator:	[Data Structure for the Actuator]
ActuatorDelta:	[Delta change in the Actuator]
GeV:	Energy
TimeStamp:	Output of *clock*, e.g. [2003 6 17 20 27 25.0770]
DCCT:	Beam current
ModulationMethod:	'bipolar' or 'unipolar'
WaitFlag:	WaitFlag
ExtraDelay:	ExtraDelay
DataType:	'Response Matrix'
CreatedBy:	'measrespmat'

Tableau 9.1 Format de sauvegarde d'une matrice réponse

Pour l'opération de tous les accélérateurs, les matrices réponses sont utilisées quotidiennement. Comme ces matrices sont souvent utilisées et générées plusieurs fois au cours d'une année, une fonction spécifique a été écrite pour forcer un format consistant, prenant en compte qu'un équipement ne fonctionne pas, et permettant un archivage aisé. Les principales fonctions pour accéder à une matrice réponse déjà mesurée et archivée sont maintenant rappelées :

- getbpmresp – matrice réponse des correcteurs
- gettuneresp – matrice réponse des nombres d'ondes
- getchromresp – matrice réponse des chromaticités
- getdispresp – matrice réponse pour corriger la dispersion
- getrespmat – fonction générique

La fonction générique *getrespmat* peut s'utiliser comme suit :

S = getrespmat(BPMFamily, BPMDevList, CorrFamily, CorrDevList, FileName, GeV)

Cette fonction est très polyvalente : la matrice réponse peut être lue depuis un fichier donné, si spécifié en argument (*FileName,*); sinon la matrice réponse est recherchée parmi les fichiers de référence dont les noms peuvent être obtenu en utilisant la commande *getfamilydata ('OpsData','RespFiles')*, e.g. {*'GoldenBPMResp', 'GoldenTuneResp'* }. La fonction *getrespmat* recherche

alors parmi toutes les variables du fichier (et à travers chaque cellule et tableau de structure s'ils sont définis) si la matrice recherchée avec les bons champs « Monitor » (observables) et « Actuator » (actionneur) existe. L'exemple suivant permet de lire la matrice réponse horizontale pour faire la correction d'orbite de l'anneau.

```
>> HCORsp = getsp('HCOR', 'Struct');
>> BPMam = getam('BPMx', 'Struct');
>> R = getrespmat(BPMam, HCORsp);
```

Fonctions de contrôle commande de haut niveau

1 Introduction

Une des raisons principales qui ont poussé le développement du Matlab Middle Layer est de pouvoir écrire plus facilement des scripts pour contrôler les accélérateurs et pour écrire des application de contrôle de haut niveau. Comme illustration, nous pouvons prendre le script permettant de faire fonctionner le feedback lent de l'ALS. L'agorithme utilisé est la décomposition en valeurs singulières (SVD). Ici seules les 24 premières valeurs sont utilisées pour faire la correction.

```
                                                    %ALS orbit correction example
Sx=getrespmat('BPMx',[ ],'HCM',[ ]);                %Get the proper response matrix
X = getx;                                           %Gets all 96 horizontal BPMs (96x1 vector)
Ivec = 1:24;                                        %Use singular vectors 1 thru 24
[U, S, V] = svd(Sx);                                % Computes the SVD of the response matrix,
Sx(96x94)
DeltaAmps = - V(:,Ivec)*((U(:,Ivec)*S(Ivec,Ivec))\X) ;  %Find the corrector changes (94x1 vector)
stepsp('HCM', DeltaAmps);                           %Changes the current in all 94 horizontal corrector
magnets
plot(1:96, X, 'b', 1:96, getx, 'r');                %Plot new orbit
```

2 Liste de fonctions et applications

Archivage:
arplot
arread

Cyclage
setcycleramp
getcycleramp

fonction tango indicée de 2 :
Contrôle d'erreur TANGO

Version 2

Nom	Fonctionnalité
bpm2quad	– Retourn le quadrupôle le plus proche d'un BPM – – – –
findrf	– one method of finding an “optimal” RF frequency based on dispersion – A voir resultat différent de la correction d'orbite !!!
findqfa	– optimizes the setpoint of the quadrupole in the center of arcs.
goldenpage	– displays the important settings and setpoints (like tune, chromaticity, etc)
plotfamily	– general plotting GUI for families (see section 11 for details)
rmdisp	– adjusts the RF frequency to remove the dispersion component of the orbit by fitting the orbit to the dispersion orbit (fitting the mean is optional).
setorbit	– general purpose global orbit correction function
srcycle	– cycles the storage ring magnets
srramp	– energy ramping (with beam) of the storage ring
setlocalbump	– general purpose local bump function
quadcenter, quadplot, quaderror	– finds the quadrupole center of one magnet at a time
setorbitquadcenter	– corrects the orbit to the quadrupole centers
srsetup	– launch pad for setup applications
srcontrol	– GUI for storage ring operations
scanbpms	– local bump scan in the straight sections checking the linearity of the BPMs
scanpipe	– used for scanning the electron beam

Nom	Fonctionnalité
	in the straight sections and checking the lifetime (physical aperture)
scantune	– scan in tune space and record the lifetime (or loss monitors)
plotlattice	– Dessine les icônes représentant les éléments de la maille de la machine

Tableau 10.1 Applications de contrôle de haut niveau.

3 Correction de l'orbite fermée (SETORBIT)

3.1.1 Correction sans ajustement de la fréquence RF

1. SVD (choix du nombre de valeurs singulières)
2. Pondération des BPM
3. Nombre d'itération ajustable
4. Changement absolu ou incrémental de l'orbite

3.1.2 Correction avec ajustement de la fréquence RF

1. La fonction dispersion est ajoutée dans la matrice réponse comme une colonne supplémentaire. Plusieurs choix de fonctions dispersion sont possibles :

   a. dispersion entrée par l'utilisateur
   b. dispersion mesurée
   c. dispersion du modèle
   d. dispersion de référence (Golden dispersion)

2. correction utilisant l'algorithme SVD

Voir aussi la note « IHM pour la correction de l'orbite fermée de l'anneau », note SOU-PM-CR-1644.

Note :

« Instead of fitting the RF frequency one could also remove the dispersion component of the orbit use *rmdisp* or *findrf* before calling *setorbit*. At the ALS the RF is manually adjusted to fix energy shift of the horizontal correctors to zero by iterating orbit correction and RF frequency adjustment. Accomplishing this as an energy constraint within the orbit correction algorithm is a planned future improvement. »

4 Bump local d'orbite fermée

Fonction *setorbitbum:*

Cette fonction permet de déplacer localement d'orbite fermée en position ou en angle.

Exemples d'utilisation:

1. Déplacement de l'orbite de 1 mm par rapport à l'orbite actuelle dans les premier et deuxième BPM de la cellule 10 (méthode dite incrémentale).
   Commande : *setorbitbump('BPMx', [10 1 ;10 2],[1 1],'HCOR',[-2 –1 1 2]);*

2. Déplacement de l'orbite de –1 mm et 1 mm par rapport à l'orbite actuelle respectivement dans les premier et deuxième BPM de la cellule 5 (méthode dite incrémentale).
   Commande : setorbitbump('BPMx', [5 1 ;5 2],[-1 1],'HCOR',[-2 –1 1 2]);

3. Déplacement de l'orbite de 1 mm en absolue (quelque soit l'orbite de référence) dans les premier et deuxième BPM de la cellule 10 (méthode dite absolue).
   Commande : *setorbitbump('BPMx', [10 1 ;10 2],[1 1],'HCOR',[-2 –1 1] ,'Absolute');*

4. Utiliser une interface graphique minimale
   Commande : *setorbitbump ;*

5 Interfaces graphiques

5.1 Affichage (*PLOTFAMILY*)

Version 2

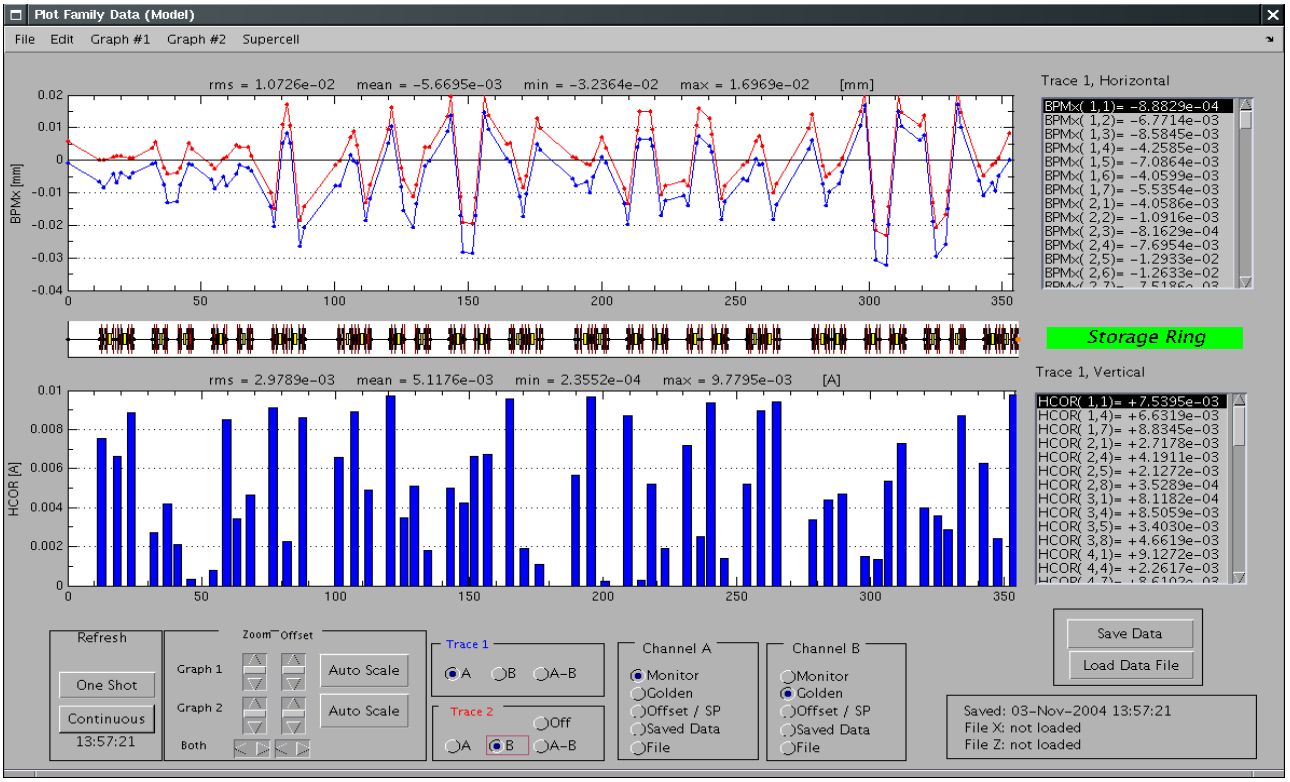


Illustration 10.1 Application plotfamily.

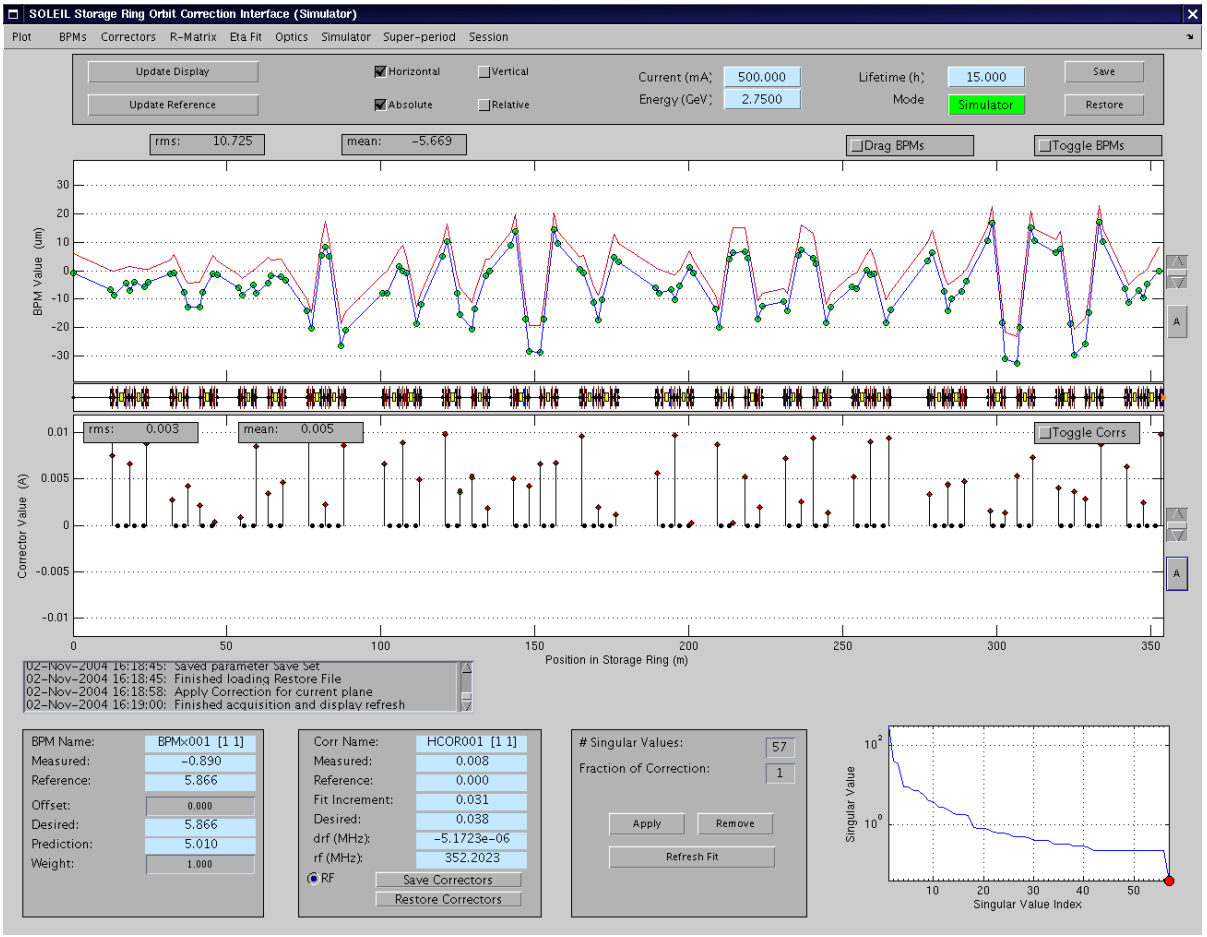
L'application plotfamily permet de tracer simplement l'orbite fermée de la machine et les valeurs de consignes de toutes les familles d'équipements définies dans le MML

L'interface est intuitive et documentation plus détaillée reste à écrire. Succinctement, l'application est compsée:

- D'une barre de menu pour choisir la famille à afficher, l'échelle d'affichage, le mode simulateur ou en ligne, l'affichage pour tout l'anneau ou une super- période.
- De deux panneaux graphiques : diagramme en barres pour les aimants, courbes pour l'orbite fermée.
- Sur la droite, deux panneaux permettent d'avoir accès aux informations détaillées.
- La partie inférieure de l'application permet de choisir
  - le mode d'affichage : à la demande ou rafraichi
  - le zoom et l'offset sur les graphes
  - les canneaux à afficher (A et/ou B) parmis:
    - les valeurs actuelles
    - les valeurs de références (*golden parameters*)
    - les offsets
    - les valeurs sauvées dans l'application
    - les valeurs lues depuis un fichier.

5.2 Correction de l'orbite : solorbit

Le lecteur est prié de se reporter à la note intitulée « IHM pour la correction de l'orbite fermée de l'anneau » (SOU-PM-CR-1644) pour plus de détails.



5.3 BEAM BASED ALIGNMENT

1. Un quadrupôles
2. Tous les quadrupôles

A FAIRE

5.4 MML pour le démarrage et l'opération des installations

Version 2

A FAIRE

3 Fonctions propres à LT1

Le MML a été entendu pour prendre en compte LT1.

6 Familles de LT1

Champ	Contenu
BEND	2 dipôles en série
QP	7 quadrupôles équivalents
CH	correcteurs horizontaux
CV	Correcteurs verticaux
MC	2 mesureurs de charges
PI	Pompes ioniques
JPIR	Jauges Pirani
JPEN	Jauges Penning

7 Fonctions spécifiques

Champ	Contenu
LT1init	Initialisation de l'optique de LT1
LT1setup	Panneau pour appeler les fonctions principales pour LT1
LT1optics	Outils pour gérer l'optique de LT1
Mesure d'émittance	emittance_v15

Le fichier LT1init est le fichier contenant tout la configuration spécifique à LT1 : il contient l'interface de communication avec le monde TANGO, avec le simulateur, et définis tous les chemins de configuration.

Quelques captures d'écran:

Version 2

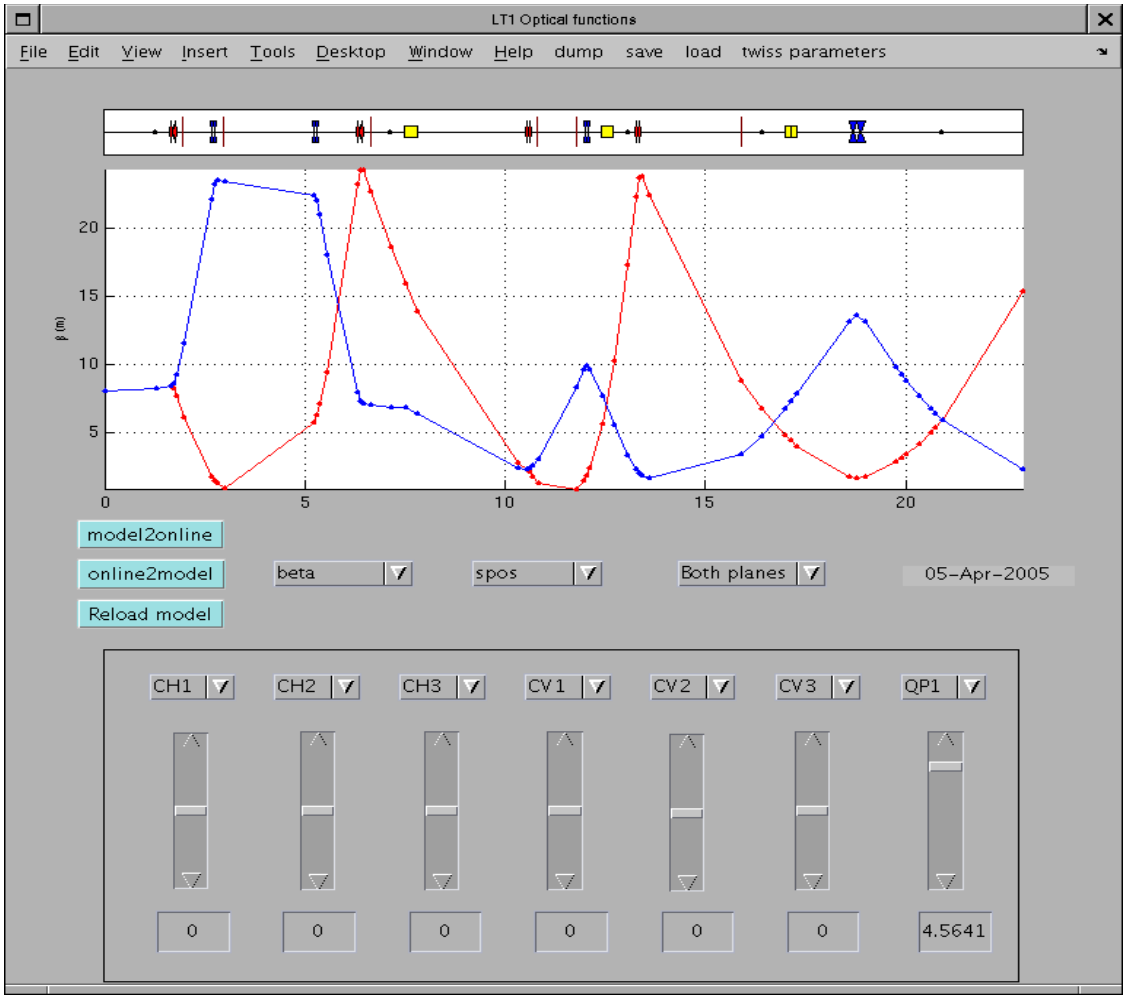


Illustration 3.1 Application de simulation et réglage de l'optique de LT1

Version 2

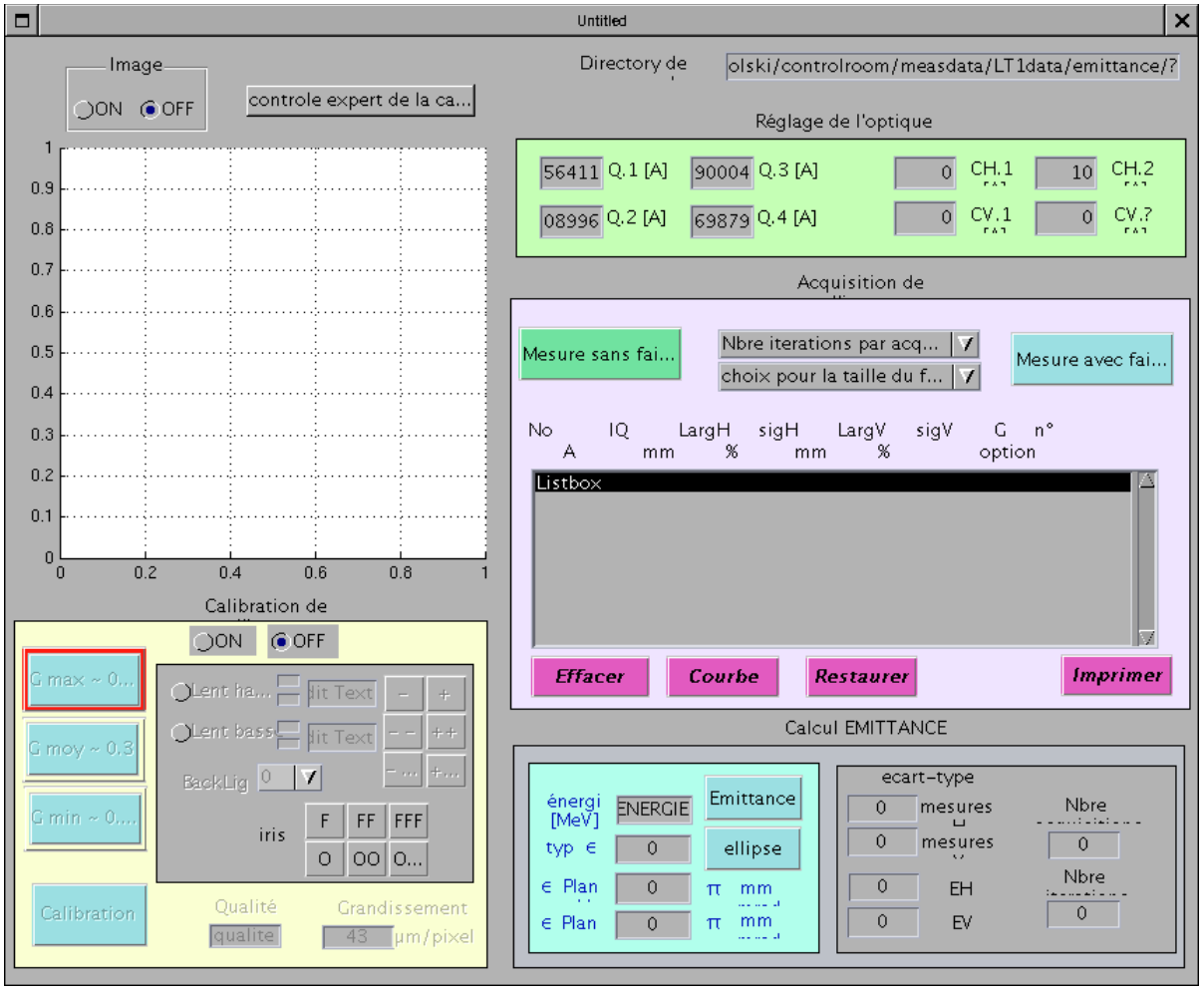


Illustration 3.2Application de la mesure d'émittance de LT1

8 Fonctions pour la gestion du cyclage des aimants

Fonction en cours de développements

Champ	**Contenu**
LT1cycling	IHM simplifiée gérant le cyclage
magnetcycle	Fonction principale gérant le cyclage
CycleLT1magnet	Fonction pour communiquer avec les Dserveurs cyclage
plotcyclecurve	Affiche une Courbe de cyclage
getcyclecurve	Relit une courbe de cyclage sur un Dserveur cyclage
setcyclecurve	Ecrite une nouvelle courbe de cyclage
configcyclecurve	
createcyclercurve	Génére les courbes de cyclage

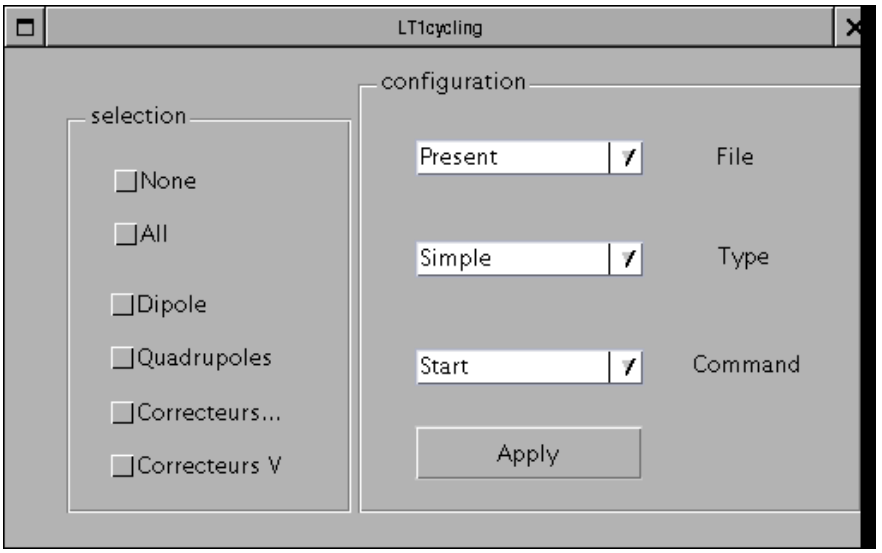


Illustration 3.3 IHM de gestion du cyclage de LT1

Fonctions propres au booster

Le MML a été entendu pour prendre en compte le booster

1 Familles du booster

2 Fonction spécifiques au alimentations 3 Hz

Champ	Contenu
writeramp3Hz	écrit une rampe dans le format binaire des alimentations 3 Hz PSI
readramp3Hz	Lit une rampe binaire pour les alimentations 3 Hz.

Annexes

Annexe 1 Résumé des commandes utiles

Les exemples peuvent être directement utilisés dans Matlab par copier/coller.

1.1 Mesurer/sauvegarder une orbite

Mesurer une orbite

x = *getx*; %ou x = *getam* ('BPMx');
z = *getz*; %ou z = *getam* ('BPMz');

Notes

1. L'option 'Struct' permet d'obtenir en retour de commande une structure autodescriptive.
2. Pour visualiser une orbite, utiliser l'interface *familyplot.*

Sauvegarder une orbite dans le répertoire par défaut

getx('archive'); % ou getx archive
getz('archive'); % ou getz archive

Notes:

1. getfamilydata('Directory','BPMData') est le répertoire de sauvegarde par défaut
2. getfamilydata('OpsData','PhysDataFile') est le fichier de référence 'GoldenPhysData'
3. getfamilydata('Directory','OpsData') est le répertoire du fichier de référence.

Obtenir l'orbite de référence (goldenorbit) des BPM

Xgolden = getgolden('BPMx'); %plan horizontal
Zgolden = getgolden('BPMz'); %plan vertical

Obtenir les offsets des BPM

Xoffset = getoffset('BPMx'); %plan horizontal
Zoffset = getoffset('BPMz'); %plan vertical

Sauvegarder l'orbite présente comme orbite de référence

```
BPMxData = getx('struct');
setphysdata('BPMx','Golden',BPMxData);

BPMzData = getz('struct');
setphysdata('BPMz','Golden',BPMzData);
```

Sauvegarder de l'orbite actuelle comme offset d'orbite de référence

```
BPMxOffset = getx('struct');
BPMxData = setphysdata('BPMz','Offset',BPMzData);
```

Notes

1. Les offsets sont normalement déterminés en mesurant le centre des quadrupôles, voir le programme de « beam based alignment »

Déterminer et sauvegarder les écarts- type de l'orbite

```
BPMSigma = measbpmsigma('struct');
FileName = getfamilydata('OpsData','BPMSigmaFile');
DirName = getfamilydata('Directory','OpsData');
save([DirName FileName], 'BPMSigma');
```

1.2 Les fonctions getpv et setpv

Les fonctions *getpv* et *setpv* appellent respectivement les fonctions *getpvonline* et *setpvonline* pour communiquer avec TANGO et *getpvmodel* et *setpvmodel* pour communiquer avec AT.

```
%>> help getpv
%
%GETPV - Gets a TANGO attribute (or AT simulated attribute)
%
% FamilyName/DeviceList Method
% [AM, tout, ErrorFlag] = getpv(Family, Field, DeviceList, t, FreshDataFlag, TimeOutPeriod)
% [AM, tout, ErrorFlag] = getpv(DataStructure, t, FreshDataFlag, TimeOutPeriod)
%
% TangoName Method
% [AM, tout, ErrorFlag] = getpv(TangoName, t, FreshDataFlag, TimeOutPeriod)
%
% CommonName Method
%    [AM, tout, ErrorFlag] = getpv(Family, Field, CommonName, t, FreshDataFlag, TimeOutPeriod)
%
% INPUTS
% 1. Family = Family Name
%          Data Structure
%          Tango Name
%          Accelerator Object
%          For CommonNames, Family=[] searches all families
%          (or Cell Array of inputs)
% 2. Field = Accelerator Object field name ('Monitor', 'Setpoint', etc) {'Monitor'}
%         Cell Array of fields
% 3. TangoName =  Access using Tango name (scalar or vector outputs),
%              Matrix of Tango names (scalar outputs only)
%              Cell array of Tango names
% 4. CommonName = Common name (scalar or vector outputs),
%             Matrix of common names (scalar outputs only)
%             Cell array of common names
% 5. DeviceList = [Sector Device #] or [element #] list {default or empty list: whole family}
%             Cell array of DeviceLists
%             Note: if input 1 is a cell array then DeviceList must be a cell array
```

```
% 6. t = Time vector (t can not be a cell) {default: 0}
% 7. FreshDataFlag =  0  -> Return after first measurement {default}
%                else -> Return after FreshDataFlag number of new measurements have been read
%               ie, getpv('BPMx',[1 1],0,2) measures the orbit then continues to read the orbit
%                 until 2 new orbits have been measured and returns the last measurement.
% 8. TimeOutPeriod = Time-out period when waiting for fresh data {10 seconds}
% 9. 'Struct' will return a data structure {default for data structure inputs}
%    'Numeric' will return numeric outputs {default for non-data structure inputs}
% 10. 'Physics'  - Use physics units (optional override of units)
%     'Hardware' - Use hardware units (optional override of units)
% 11. 'Online' - Get data online (optional override of the mode)
%     'Model'  - Get data from the model (optional override of the mode)
%     'Manual' - Get data manually (optional override of the mode)
%
% OUTPUTS
% 1. AM  = Monitor values (Column vector or matrix where each column is a data point in time)
% 2. tout = Time when measurement was completed (row vector)
% 3. ErrorFlag =  0  -> no errors
%            else -> error or warning occurred
%
% The output will be a data structure if the input is a data structure or the word 'struct'
% appears somewhere on the input line.
%
% NOTES
% 1. For data structure inputs:
%    Family    = DataStructure.FamilyName
%    Field     = DataStructure.Field
%    DeviceList = DataStructure.DeviceList
%    Units     = DataStructure.Units  (Units can be overridden on the input line)
%    (The Mode field is ignored!)
%
% 2. diff(t) should not be too small.  If the desired time to collect the data is too
%    short then the data collecting will not be able to keep up.  Always check tout.
%    (t - tout) is the time it took to collect the data on each iteration.
%
% 3. An easy way to measure N averaged monitors is:
%    PVmean = mean(getpv(Family,DeviceList,1:N)')';  %1 second between measurements
```

Version 2

```
%
% 4. Tango name method is always Online!
%
% 5. For cell array inputs:
%    a. Input 1 defines the size of all cells
%    b. All of the cell array inputs must be the same size or size=[1 1]
%    c. t (if used) can not be a cell!
%    d. FreshDataFlag and TimeOutPeriod can be a cell but they don't have to be
%
% See also getam, getsp, getx, getz, setpv
```

```
%>> help setpv
%
%SETPV - Absolute setpoint change via MATLAB Tango access or AT simulator
%
% FamilyName/DeviceList Method
% ErrorFlag = setpv(FamilyName, Field, NewSP, DeviceList, WaitFlag)
% ErrorFlag = setpv(DataStructure, WaitFlag)
%
% TangoName Method
% ErrorFlag = setpv(TangoName, NewSP)
%
% CommonName Method
% ErrorFlag = setpv(FamilyName, Field, NewSP, CommonNames, WaitFlag)
%
% INPUTS
% 1. Family = FamilyName
%          Data Structure
%          Tango Name
%          AcceleratorObject
%          Use Family=[] in CommonName method to search all Families
%         (or Cell Array of inputs)
%
% 2. TangoName = Tango access Tango name (scalar or vector inputs)
%              Matrix of Tango names (scalar inputs only)
%              Cell array of Tango names
%
% 3. CommonName = CommonNames (scalar or vector inputs)
```

```
%              Matrix of CommonNames (scalar inputs only)
%              Cell array of CommonNames
%              Must use Family=[] in to search all Families
%
% 4. Field = AcceleratorObject Field name ('Monitor', 'Setpoint', etc) {'Monitor'}
%          If Family is a cell array then Field must be a cell array
%
% 5. NewSP = New Setpoints or cell array of Setpoints
%
% 6. DeviceList = ([Sector Device #] or [element #]) {default or empty list: whole family}
%              Note: all numerical inputs must be column vectors
%
% 7. WaitFlag = 0   -> return immediately
%            > 0  -> wait until ramping is done then adds an extra delay equal to WaitFlag
%            = -1 -> wait until ramping is done {SLAC default}
%            = -2 -> wait until ramping is done then adds an extra delay for fresh data
%                    from the BPMs {ALS default}
%            = -3 -> wait until ramping is done then adds an extra delay for fresh data
%                    from the tune measurement system
%            = -4 -> wait until ramping is done then wait for a carriage return
%            else -> wait until ramping is done
%              Note: change WaitFlag default in setpv.m and BPM delay in the Accelerator Data
structure
%
% 8. ErrorFlag = 0 -> OK
%            -1 -> MATLAB Tango Access error
%              -2 -> SP-AM warning, i.e. setpoint minus analogmonitor not within tolerance
(only if WaitFlag=1)
%
% 9. 'Physics'  - Use physics  units (optional override of units)
%    'Hardware' - Use hardware units (optional override of units)
%
% 10. 'Online' - Set data online (optional override of the mode)
%     'Model'  - Set data on the model (optional override of the mode)
%     'Manual' - Set data manually (optional override of the mode)
%
% NOTES
% 1. For data structure inputs:
%    Family     = DataStructure.FamilyName
```

```
%    Field      = DataStructure.Field
%    NewSP      = DataStructure.Data
%    DeviceList = DataStructure.DeviceList
%    Units      = DataStructure.Units  (Units can be overridden on the input line)
%    (The Mode field is ignored!)
%
% 2. The number of colomns of NewSP and DeviceList must be equal,
%    or NewSP must be a scalar.  If NewSP is a scalar, then all
%    devices in DeviceList will be set to the same value.
%
% 3. When using cell array all inputs must be the same size cell array
%    and the output will also be a cell array.  Field and WaitFlag can be
%    cells but they don't have to be.
%
%  4. For Familys and AcceleratorObject structures unknown devices or elements are ignored.
%
% 5. TangoName method is always Online!
%
% 6. For cell array inputs:
%    a. Input 1 defines the maximum size of all cells
%    b. The cell array size must be 1 or equal to the number of cell in input #1
%    c. WaitFlag can be a cell but it doesn't have to be
%
% 7. WaitFlag
%    a. If no change is seen on the AM then an error will occur.  The tolerance for this
%       may need to be changed depending on the accelerator (edit the end of this function to do so)
%    b. The delay for WaitFlag = -2 is in the AD.  It is often better to use the FreshDataFlag when
%       getting BPM data but the data must to noisy for this to work.
%
% EXAMPLES
% 1. setpv('HCOR','Setpoint',1.23);              Sets the entire HCOR family to 1.23
%  2. setpv({'HCOR','VCOR'},'Setpoint',{10.4,5.3}); Sets the entire HCOR family to 10.4 and VCOR family to 5.3
% 3. setpv('HCOR','Setpoint',1.23,[1 3]);    Simple DeviceList method
% 4. setpv('HCOR','Setpoint',1.23, 3);       Simple ElementList method
% 5. setpv(AO('HCOR'),'Setpoint',1.5,[1 2]);     If AO is a properly formatted AcceleratorObject Structure
```

```
%                                   then this sets the 1st sector, 2nd element to 1.5
% 6. setpv('HCOR','Setpoint',1.23,'1CX3');   CommonName method with Family specified (spear3 naming convection)
% 7. setpv([],'Setpoint',1.23,'1CX3');       CommonName method without Family (spear3 naming convection)
%
% See also getam, getsp, getpv, setsp, steppv, stepsp
```

1.3 Lire la valeur de consigne et de relecture / spécifier une nouvelle valeur de consigne sur une alimentation

a) Lire toutes les valeurs de consigne sur les correcteurs lents horizontaux

sp = getsp('HCOR')

b) Lire toutes les valeurs de relecture sur les correcteurs lents horizontaux

am = getam('HCOR')

c) Assigner 0.5 A sur HCOR(2,1) et 0.33 A sur HCOR(4,2)

setsp('HCOR', [0.5; 0.33], [2 1; 2 4])

d) Assigner 0 A sur tous les correcteurs horizontaux

setsp('HCOR', 0);

Notes

1. *getfamilylist* revoie le nom de toutes les familles d'éléments
2. *plotfamily* permet de tracer les valeurs de relecture et de consigne d'une famille d'équipements.

1.4 Mesure d'une matrice réponse

La commande *measbpmrep* permet de mesurer la matrice réponse pour les correcteurs par défaut, i.e. les correcteurs lents. Pour utiliser un autre jeu de correcteurs, il suffit spécifier les correcteurs à utiliser en paramètres d'entrée de la fonction. Voir l'aide en ligne pour plus de détails (*help measbpmresp*).

Commande pour une mesure en ligne de la matrice réponse

% this command

R = measbpmresp;

% is the same as,

R = measbpmresp('BPMx', 'BPMz', 'HCOR', 'VCOR', 'Online', 'Numeric', 'Archive');

Commande pour une mesure d'après le modèle

% Matrices pour le modèle en ligne

R = measbpmresp('Model');

% is the same as,

R = measbpmresp('BPMx', 'BPMz', 'HCOR', 'VCOR', 'Model', 'Numeric', 'NoArchive', 'FixedPathLength', 'Full');

Cette commande appelle la fonction *locoresponsematrix* (voir l'aide en ligne pour plus de détails). On rappelle les principaux arguments :

- *'FixedMomentum'* : l'orbite fermée n'est pas l'orbite synchrone (RF constante)
- *'FixedPathLength'* : l'orbite fermée est l'orbite synchrone
- *'Full'* : modèle nonlinéaire
  - *'FixedMomentum'* : findorbit4 --> sans facteur d'Amman
  - *'FixedPathLength'* : findsyncorbit --> Il doit y avoir une cavité dans la maille
- *'Linear'* : approximation linéaire
  - *'FixedMomentum'* : sans facteur d'Amman
  - *'FixedPathLength'* : avec facteur d'Amman
- *'Oneway' / 'bidirectional'* : option pour le calcul de l'orbite et/ou pour le calcul de la dispersion.
  - Option *'bidirectional'*
    - Pour l'orbite : les correcteurs sont allumés avec +théta/2 et -théta/2
    - Pour la dispersion: la RF est changée de manière symétrique + DeltaRF/2 et – DeltaRF/2
  - Option *'Oneway'*
    - Pour l'orbite : les correcteurs sont allumés avec +théta
    - Pour la dispersion: la RF est changée de manière symétrique + DeltaRF

Version 2

Notes :

- Si une cavité est présente dans la maille, alors le d'Amman est calculé
- L'orbite synchrone ne peut être trouvée que s'il y a une cavité dans la maille.

Matrice réponse en ligne pour 3 BPM et 2 correcteurs sans sauvegarde

% Get a response matrix for 3 BPMs and 2 Correctors (w/o saving data)

% The data is usually saved to disk. The 'NoArchive' flag stops this.

R1 = measbpmresp('BPMx', [1 1;2 2;6 3], 'BPMz', [1 1;2 2;6 3], 'HCOR', [1 1; 1 3], 'VCOR', [1 1; 1 2], 'NoArchive');

Matrice réponse modèle pour 3 BPM et 2 correcteurs sans sauvegarde

% Get a the same data from the model

% The default for the model is not to save data to disk.

R2 = measbpmresp('BPMx', [1 1;2 2;6 3], 'BPMy', [1 1;2 2;6 3], 'HCM', [1 1; 1 3], 'VCM', [1 1; 1 2], 'Model');

Comparaison

```
%Compare a column (ie, a corrector magnet response)
plot(R2(:,1) – R1(:,1));
```

```
%Get a the same data from the model w/ a FixedMomentum, Linear model
R2 = measbpmresp('BPMx', 'BPMz', 'HCOR', [1 1; 1 3], 'VCOR', [1 1; 1 2],
    'FixedMomentum', 'Linear', 'Model');
```

Le répertoire et fichier de sauvegarde par défaut est :

<DataRoot>\Response\BPM\BPMRespMat<Date><Time>.mat

Pour que cette matrice réponse devienne la matrice utilisée par les programme, il faut la copier à :

<OpsData>\GoldenBPMResp.mat

1.5 Préparer un fichier pour une analyse avec LOCO

Version 2

LOCO requires an orbit response matrix and the standard deviations of the BPM difference orbits. Measure a new orbit response matrix with *measbpmres* and new BPM standard deviations *measbpmsigma*. Combine the output from these functions with *makelocoinput*. ***(this function is still under development)***

1.6 Get/Set/Step RF frequency

(to be written)

1.7 Mesurer, sauvegarder, dessiner la fonction dispersion

La fonction *measdisp* permet de mesurer la fonction dispersion. En fin de mesure, le déplacement d'orbite en fonction de la fréquence RF est affiché. A termes, le déplacement d'orbite en fonction de l'énergie sera également affiché. Les unités par défaut pour la dispersion sont actuellement le changement d'orbite par unité de fréquence [mm/MHz] (unité « Hardware »). Cependant, il est possible de sélectionner d'autres unités [meters/(dp/p)] (unités « Physics ») en ajoutant le paramètre 'Physics' or 'Zeta' à l'appel de la fonction. La fonction renvoie une structure si le paramètre 'Struct' est ajouté. Les champs de la structure sont semblables à ceux de la structure la matrice réponse (variation d'orbite pour une variation de la fréquence RF) pour pouvoir être utilisés par le programme LOCO. La fonction *plotdisp* permet de tracer une mesure ancienne de la dispersion. Voir *help measdisp* pour plus de détails.

```
%Measure the dispersion (vector output)
[Dx, Dz] = measdisp;

%Measure the dispersion (structure output)
[Dx, Dz] = measdisp('Struct');

%Measure the dispersion in physics units [1/(dp/p)]
[Dx, Dz] = measdisp ('Struct', 'Physics');

%Measure the dispersion and archive (output is optional)
measdisp ('Archive');

%Measure and plot the dispersion
%No outputs or 'Display' on the input line will automatically plot
```

[Dx, Dz] = measdisp('Display');

measdisp;

%Plot using plotdisp function

[Dx, Dz] = measdisp('Struct');

plotdisp(Dx, Dz);

%Model dispersion (override the Mode to Simulate)

[Dx, Dz] = measdisp('Simulate');

%Model dispersion (calls AT directly)

[Dx, Dy] = measdisp('Model'); % calls modeldisp

1.8 Nombres d'ondes

1.8.1 Lire et changer les nombres d'ondes

%Measure the tune and return a 2x1 vector

Tune = gettune;

%Measure the tune and return a structure

Tune = gettune('Struct');

1.8.2 Mesure de la matrice réponse des nombres d'onde

La fonction *measTuneResp* permet de mesurer la matrice réponse des nombres d'ondes. L'unité de la matrice est [nombre d'onde fractionnaire/Amplitude] (hardware units). Voir *help measTuneResp* pour plus de détails.

Le répertoire et fichier de sauvegarde par défaut est :

<DataRoot>\Response\Tune\TuneRespMat<Date><Time>.mat

Pour faire de cette matrice la matrice de référence des nombre d'onde, il suffit de la copier dans :

<OpsData>\GoldenTuneResp.mat

1.8.3 Changements relatifs des nombres d'ondes

Pour modifier les nombres d'ondes de [-.05; .05], il suffit d'utiliser la commande *steptune*.

```
steptune([-.05; .05]);
```

On peut vérifier à la main ce que la commande précédente fait :

```
%On mesure les nombres d'ondes
Tune1 = gettune;

%on mesure la matrice des nombres d'ondes
m = gettuneresp;

% On calcule les variations de courant pour les deux familles de quadripôles et on applique la correction :
DeltaAmps = inv(m) * [-.05; .05];
setsp({'Q9', 'Q7'}, {getsp('Q7')+DeltaAmps(1), getsp('Q9')+DeltaAmps(2)});

% On mesure de nouveau les nombres d'ondes et vérifie la variation effectuée :
Tune2 = gettune;
DeltaTune = Tune2 – Tune1
```

1.9 Chromaticités

1.9.1 Mesure de la chromaticité

La commande *measchro* permet de mesurer la chromaticité. Les variations des nombres d'ondes avec la fréquence RF ou l'énergie sont tracées. Les chromaticités sont les parties linéaires des ajustements des courbes. A chaque mesure, bien vérifier que la courbe a une allure correcte. Note : Lorsque la chromaticité est proche de zéro de petites erreurs sur les nombre

d'ondes peuvent engendrer une mesure de chromaticité incorrecte. L'unité par défaut de la chromaticité est le variation du nombre d'ondes par unité de fréquence RF [1/MHz] (« Hardware units »). Cependant, il es possible de sélectionner comme unité, la variation de nombre d'onde en fonction de l'énergie [1/(dp/p)] (« Physics units ») en spécifiant 'Physics' ou 'Zeta' en paramètre de la commande. Le format de sortie par défaut est un vecteur de chromaticités 2x1. Pour obtenir une structure, il suffit de préciser 'Struct' en argument de la fonction. Les champs de la structure sont similaires à ceux de la matrice réponse. La fonction *measchro* dessine automatiquement les courbes de variation des nombres d'ondes avec l'énergie et par ajustement détermine les chromaticités linéaire et d'ordre 2. Voir *help measchro* pour plus de détails.

```
%Mesure de la chromaticité, vecteur 2x1 (units [1/MHz])
Chro = measchro;

% Mesure de la chromaticité avec le résultat sous forme de structure
ChroStruct = measchro('Struct');

%Mesure de la chromaticité en unités physiques [1/(dp/p)]
ChroStruct = measchro('Struct', 'Physics');

%Mesure de la chromaticité et archivage
measchro('Archive');          % Output is optional
```

1.9.2 Mesure de la matrice réponse des chromaticités

La commande *measchroresp* permet de mesurer a matrice réponse des chromaticité en utilisant les familles de sextupôles par défaut. Les unités sont exprimées en [1/MHz] (hardware units). Taper help *measchroresp* pour plus de détails.

Répertoire et fichier de sauvegarde par défaut :

<DataRoot>\Response\Chromaticity\ChroRespMat<Date><Time>.mat

Pour faire ce cette matrice, la matrice de référence pour modifier la chromaticité :

<OpsData>\GoldenChroResp.mat

1.9.3 Variations relatives des chromaticités et mesure de la matrice réponse

Pour changer les valeurs de chromaticité de [-.25; .25] [1/MHz], il suffit d'utiliser la commande :

stepchro([-.25; .25]);

Si les chromaticités ont été mesurées en unités physiques et si la matrice réponse a été prise en unité hardware, il suffit de la convertir avant d'utiliser la commande *stepchro*. Pour, RF=352.2 et MCF=0.001016,

stepchro([.1346; -.1346] / -RF / MCF);

Cependant il est recommander d'avoir les mêmes unités durant l'ensemble des mesures.

La fonction *stepchro* peut être facilement réalisée à la main :

```
%Mesure la chromaticité pour vérifier à la fin le résultat
figure(1);
Chro1 = measchro;

%Lecture de la matrice réponse des chromaticité en utilisant les familles par défaut
m = getchroresp;

%Calcul des variations en courant pour les deux familles de sectupôles et application des nouvelles valeurs de consigne.

DeltaAmps = inv(m) * [-.25; .25];
setsp({'S9', 'S10'}, {getsp('S9')+DeltaAmps(1), getsp('S10')+DeltaAmps(2)});

%Mesure finale des chromaticités et vérification des résultats:
figure(2);
Chro2 = measchro;
DeltaCrho = Chro2 – Chro1
```

1.10 Sauvegarde/restauration

- Archiver une maille
  Faire de la maille actuelle, la maille pour l'opération courante.

1.11 « Beam based alignment »

(to be written)

1.12 Fonctions pour manipuler les insertions

(à venir)

1.13 Fonctions propres au modèle AT

Exemple d'utilisation :

```
%Model dispersion function
modeldisp;                  % Plots with units mm/MHz
modeldisp('BPMx', 'BPMz');  % Plots at 'BPMx', 'BPMz' families [mm/MHz]
modeldisp('Physics');  % Plots with units meters/(dp/p)
[Dx, Dy] = modeldisp;  % Returns Dx, Dy with units mm/MHz

%Model beta function
modeltwiss('beta');              % Plot beta
modeltwiss('beta', 'BPMx');  % Plot beta at the BPMx family
[Betax, Betaz] = modeltwiss('beta', 'BPMx');  % Returns beta at BPMx

%Model closed orbit
[x, y] = modeltwiss('ClosedOrbit');    % Closed orbit at all AT elements
[x, z] = modeltwiss('x');                  % Closed orbit at all AT elements
z = modeltwiss('z', 'BPMz');             % Vertical orbit at BPMy family
```

1.14 Exemple de script Matlab pour corriger l'orbite fermée

```
%Introduce an orbit error
setsp('HCOR',rand(72,1));
```

Version 2

```
%Get the response matrix
%Sx = getrespmat('BPMx','HCOR');
Sx = measbpmresp('BPMx', 'BPMz','HCOR', 'VCOR', 'Model');
Sx = Sx(1,1).Data;

%Gets all horizontal BPMs (vector)
X = getx;

%Computes the SVD of the response matrix, Sx(120x56)
%Use singular vectors 1 thru 24
Ivec = 1:24;
[U, S, V] = svd(Sx);

%Find the corrector changes (vector)
DeltaAmps = -V(:,Ivec)*((U(:,Ivec)*S(Ivec,Ivec))\X) ;

%Changes the current in all horizontal corrector magnets
stepsp('HCOR', DeltaAmps);

%Plot new orbit
plot(getspos('BPMx'), X, 'b', getspos('BPMx'), getx, 'r');
xlabel('BPM Position [meters]');
```

Annexe 2 Installation du MML

A FAIRE

Annexe 3 Aides de programmation

3.1 Règles de programmation

La langue de Matlab est l'anglais

Casse des fonctions : Tous les noms de fonctions sont en minuscules. Unix est sensible à la casse contrairement à Windows.

Nom des fonctions : Ne pas utiliser un nom commun lorsqu'une nouvelle fonction est définie. Vérifier que le nom est valide (>> *which* nom_fonction ou >> *genvarname* pour Matlab R14).

Nom des familles : Les applications doivent être écrites en utilisant des noms de fonctions génériques. Ainsi changer de nom de famille ne demande pas de récrire les applications. Ceci est vital pour la portabilité et la robustesse des applications.

Arborescence : L'arborescence des répertoires ne doit pas être écrite en dur dans une application. La racine de l'arborescence peut être obtenue grâce à la fonction *getfamilydata('Directory', 'DataRoot')*. Les données doivent être sauvegardées dans un sous répertoire par type, date et temps.

3.2 Aide en ligne

Toute fonction doit être accompagnée d'une aide en ligne. Le minimum minimorum est une entête (voir ci-après).

La langue de Matlab est l'anglais : la documentation est écrite dans cette langue

Pour rester consistant avec l'aide en ligne de Matlab, l'entête souhaitée pour fonctions est la suivante :

```
%FUNCTIONNAME – Description
% [Out1, Out2, ...] = functionname(Input1, Input2, ...)
%
% INPUTS
% 1.
```

```
% 2.
%
% OUTPUTS
% 1.
% 2.
%
% NOTES
% 1.
% 2.
%
% EXAMPLES
% 1.
% 2.
%
%See also

% Written by ____
```

Note : le saut d'une ligne après « See Also » est requis pour ne pas avoir apparaître la ligne « Written by » à chaque fois que l'aide en ligne est appelée.

3.3 Gestion des erreurs

Il est recommandé d'utiliser à chaque fois que possible les fonctions *error* or *mexerror*, et d'éviter l'emploi de drapeau (Flag). Cela évite ainsi d'avoir à vérifier le statut des erreurs après chaque exécution d'une fonction. Pour les cas complexes, l'utilisation du mécanisme try/catch est fortement recommandé (Attention cependant car ce dernier n'est pas supporter par le Matlab Compiler !).

3.4 Génération de documentation

3.4.1 Contenu d'un répertoire

mkcontnt : crée un fichier contents.m en reprenant la H1 ligne de chaque fichier '.m' du répertoire
Le fichier contents.m est afficher lorsque la commande *help repertoire* ou *doc repertoire* est tapé dans Matlab. *Repertoire* est le repertoire contenant les fichiers Matlab.

3.4.2 Génération de documentation html

La commande docgen permet de générer une documentation html associé à une arborescence Matlab. Voir *help docgen* pour plus de détails.

3.4.3 Note importante

Les commandes *mkcontent* et *docgen* ne sont pas des commandes natives de Matlab.

Annexe 4 Création de familles

4.1.1 Introduction

Les quatre fonctions principales pour lire ou assigner une valeur de consigne ou relecture sont *getam*, *getsp*, *setsp*, *stepsp* ; ces fonctions sont habituellement utilisées sur des familles d'éléments. Leur rôle principale est soit de lire, soit d'écrire un attribut, que ce dernier soit un « attribut » TANGO correspondant à la machine en ligne ou au simulateur. Ces deux fonctions principales sont les fonctions *getpv* and *setpv* dont dérivent entre autres les quatres fonctions principales précitées :

1. La fonction *getpv* permet d'aller lire une valeur de contrôle (Process Variable).
2. La fonction *setpv* permet d'écrire une valeur de contrôle.

Toutes les informations nécessaires pour le bon fonctionnement de ces fonctions sont regroupées dans une structure Matlab appelée Accelerator Object (AO), stockée dans l'objet « application data » de la fenêtre racine d'une session Matlab. L'objet AO possède un grand nombre de champs. Le premier d'entre eux est le champ famille (FamilyName). Le champ AO.(FamilyName) est lui même une structure contenant les informations nécessaires au bon fonctionnement des deux commandes *getpv* et *setpv*. Ce format est maintenant détaillé.

4.2 Structure principale pour l'agencement des données

AcceleratorObject.(FamilyName)

Champ	Description
FamilyName	- Nom de la famille (’BPMx’, ‘HCOR’, etc.) (unicité requise)
FamilyType	- Nom de la catégorie d’éléments, par exemple ‘QUAD’
MemberOf	- Tableau de cellules, par exemple {’QUAD’, ‘Magnet’}
Status	- 1 pour statut valide, 0 pour invalide
DeviceList	- Vecteur colonne [1 1; 1 2; 2 1, ...]
ElementList	- Vecteur colonne [1;2;3; ..; n]
Desired	- Structure (cf. infra)
Monitor	- Structure (cf. infra)
Setpoint	- Structure (cf. infra)
DeviceNames	- Matrice de cellules pour le nom du device Tango
CommonNames	- Matrice de chaîne de caractères pour le nom commun
Position	- Vecteur colonne avec la position longitudinal le long de l’anneau (mètres)
AT	- Structure pour le simulateur AT (facultatif)
Golden	- Structure avec les valeurs de référence (facultatif)

Tableau 4.1 Champs d'une famille du MML.

4.3 Structure pour le champ 'Monitor'

AcceleratorObject.(FamilyName).Monitor

Version 2

Champ	Description
DataType	– 'Scalar' ou 'Vector' ou 'Group' (groupe TANGO)
DataTypeIndex	– Sub-indexing of the EPICS record for
DataType	– *'Vector'* (facultatif)
Mode	– *'Online'*, *'Simulator'*, *'Manual'* ou *'Special'*
Units	– Unité utilisé: 'Hardware' ou 'Physics'
HW2PhysicsFcn	– Fonction pour passer des unités 'Hardware' à 'Physics' (voir Appendix VI)
HW2PhysicsParam	– Paramètres pour passer des unités 'Harware' à 'Physics'.
HWUnits	– Nom de l'unité 'Hardware' (String)
PhysicsUnits	– Nom de l'unité 'Physics' (String)
TangoNames	– Nomenclature complète de l'attribut
TANGO Handles	– Vecteur de référence (NaN si non ouvert) Inutile pour TANGO ? Ou sont rangées les références ????
SpecialFunctionGet	– Nom de la fonction si Mode='*Special*' (facultatif)
Golden	– Valeur de référence (généralement pour la famille BPM) (facultatif)
Offset	– Vecteur d'offset [BPM, aimants] (facultatif)
Gain	– Vecteur de gain [aimants] (facultatif)

Tableau 4.2 Champs "Monitor" d'une famille du MML.

4.4 Structure pour le champ 'Setpoint'

AcceleratorObject.(Family).Setpoint

Champ	Description
DataType	– *'Scalar'* ou *'Vector'*
DataTypeIndex	– Sub-indexing of the EPICS record for
DataType	– 'Vector' (optional)
Mode	– *'Online'*, *'Simulator'*, *'Manual'* ou *'Special'*
Units	– What units to work in: 'Hardware' or 'Physics'
Physics2HWFcn	– Fonction pour passer des unités 'Physics' a 'Hardware' (voir Appendix VI)
Physics2HWParams	– Paramètres de conversion utilisés par Physics2hw
HWUnits	– Nom de l'unité matérielle
PhysicsUnits	– Nom de l'unité.physique
TangoNames	– Matrice de noms d'attribut TANGO
Handles	– Vecteur de références (NaN sinon)
Range	– [Min Max] bornes de variation d'une valeur de consigne
DeltaRespMat	– Incréments pour mesurer une matrice réponse (facultatif)
Tolerance	– Tolérance pour comparaison entre valeur de consigne et de relecture SP-AM.
SpecialFunctionSet	– Nom de la fonction si Mode='*Special*' (facultatif)
Golden	– Valeur de référence (généralement pour la famille BPM) (facultatif)
Offset	– Vecteur d'offset [BPM, aimants] (facultatif)
Gain	– Vecteur de gain [aimants] (facultatif)

Tableau 4.3 Champs "Setpoint" d'une famille du MML.

4.5 Règles générales

Le nombre de lignes des champs DeviceList, ElementList, CommonNames, Positions, Range et Tolerance doit être identique.

Bien qu'il soit aisé de créer une nouvelle famille, il est sage de se tenir à un jeu donné de familles pour un accélérateur sinon le partage des développements entre laboratoires s'en trouve compliqué.

Le nombre de champs de AO (Monitor, SetPoint, etc) dépend du type de famille. N'importe quel nom de champ peut être choisi. Cependant les

champs Monitor et SetPoint sont réservés aux fonctions *getam*, *getsp*, *setsp* et *stessp*. Il est hautement recommandé d'utiliser ces fonctions. Les commandes getpv et setpv sont très similaires à *getam* et *setsp* à l'exception que leurs arguments d'entrée nécessitent une structure de données AO. En fait, les fonctions *getam, getsp, setsp* et *stepsp* peuvent être vues comme des fonctions raccourcis des fonctions de base *getpv* et *setpv* avec pour argument d'entrée 'Monitor' ou 'Setpoint'. Leur utilisation est plus aisée et cache la complexité du système de contrôle.

4.6 Champs supplémentaires pour utiliser l'Accelerator Toolbox.

4.6.1 AcceleratorObject.(Family).AT (simulateur uniquement)

Champ	Description
ATType	'X', 'Z', 'BPMX', 'BPMZ', 'HCOR', 'VCOR', autrement ATParameterGroup est utilisé
ATParameterGroup	Paramètre pour faire un groupe sous AT
ATIndex	Vecteur colonne d'indice dans la maille AT

Tableau 4.4 Sous champs d'une familles dans AT

Le simulateur (AT) et la machine en ligne (pilotage des accélérateurs) peuvent coexister en paramétrant correctement l'objet AO. Ainsi, si le Mode est en ligne, et que l'on désire calculer des paramètres à partir du modèle, le jeu de commande *model*(*twiss*, *beta* ...) est disponible. On peut également spécifier 'Model' comme argument de toutes les fonctions pour forcer à utiliser le modèle (*getmcf('Model'), getam('HCOR','Model')*). Ce raisonnement est encore valide si le mode actuel de la machine est 'Online', 'Simulator', 'Manual' ou 'Special'. En mode 'Simulator', les paramètres ATType peuvent être 'X', 'Z', 'BPMx', 'BPMz', 'HCOR', 'VCOR', ou un paramètre de groupe ATParameterGroup (voir l'aide en ligne *help setparamgroup* pour plus de détails).

4.6.2 Notes concernant le simulateur

1. L'élément Cavity doit être allumé THERING{ }.PassMethod = 'ThinCavityPass' afin que les fonctions touchant la RF fonctionnement correctement.
2. La fréquence RF ne modifie pas les nombres d'ondes.

3. Les unités physiques du simulateur doivent correspondre à celles de l'Accelerator Toolbox afin de fonctionner correctement.
4. Une attention particulière doit être portée au cas où les éléments de la maille AT sont découpés en plusieurs morceaux. Par exemples, pour connaître la valeur des fonctions bêtatrons au centre des quadrupôles. Voir à ce sujet le paramètre ATParameterGroup.
5. Le mode simulateur (Simulate') ne fonctionnent que pour les champs 'Monitor' et 'Setpoint'. Cette limitation pourra être levée par une meilleure utilisation du paramètre de groupe de l'AT.
6. Les noms TANGO et les noms communs ne fonctionnent pas correctement en mode simulateur.

Annexe 5 Stockage des données

5.1 Introduction

Actuellement, les données du système de contrôle, les données pour la Physique Machine et l'opération des installations sont stockées dans différents endroits et formats.

5.2 Accelerator Object (AO)

But	Information permettant la communication entre les familles et le système de contrôle/commmande
Lieu de stockage	Espace de travail de Matlab
Get/Set	*getfamilydata* / *setfamilydata*

Tableau 5.1Accelerator Object

5.3 Accelerator Data (AD)

But	Variables liées au MML
Lieu de stockage	Espace de travail de Matlab
Get/Set	*getfamilydata* / *setfamilydata*
AD.Machine	nom de la machine, eg. 'ALS' ou 'SOLEIL'
AD.Directory.DataRoot	Racine de l'arborescence des fichiers de sauvegardes
AD.OpsData.RespFiles	Tableau de cellules des fichiers de matrices réponses, eg. {'respmatbpm_08-06-2002', 'respmattune'}
AD.ATModel	Nom de la maille AT
AD.BPMDelay	Temps d'attente entre deux relectures des BPM (attendre que les données soient renouvelées)
AD.TUNEDelay	Temps de delai pour les nombres d'ondes (cf. BPM)

Tableau 5.2Accelerator Data

5.4 Physics Data

But	Données liées à la Physique Machine
Lieu de stockage	Fichier (pour l'instant)
Fichier	getfamilydata('OpsData','PhysDataFile');
Répertoire	getfamilydata('Directory','OpsData');
Get/Set	*getphysdata* / *setphysdata*

Tableau 5.3La structure Physics Data pour les données Physique Machine

Le fichier relatif aux données Physique Machine contient une structure dont le nom est *PhysData*. Chacun des champs de cette structure correspond à une famille d'éléments (voir Tableau 5.4).

PhysData.BPMx.Golden

PhysData.BPMx.Gain

PhysData.BPMx.Coupling

PhysData.BPMx.Offset

PhysData.BPMx.Sigma

PhysData.BPMx.PinCushion

PhysData.BPMx.Dispersion

PhysData.BPMx.DesignDispersion

PhysData.BPMx.DesignBeta

PhysData.HCOR.Gain

PhysData.HCOR.Offset

PhysData.HCOR.Coupling

PhysData.Tune.Golden

PhysData.Chro.Golden

Tableau 5.4Exemples de champs de la structure PhysData

La fonction *makephysdata* permet de créer une structure par défaut contenant tous les BPM et aimants de la machine. Attention, si une telle structure existe avant l'appel de fonction, alors elle est écrasée (un confirmation est néanmoins demandée à l'utilisateur).

Annexe 6 Unités hardware et physique

6.1 Introduction

Les variables de contrôle (*process variables*) issues du monde EPICS (*channel* access) ou TANGO (*attributs*) sont des signaux exprimées en unité matérielle (« Hardware unit »). Ces unités ne sont souvent pas celles utilisées par les physiciens machines. Le MML a été écrit de manière à pouvoir passer simplement d'un type d'unité à l'autre, ie. des unités « Hardward » aux unités « Physics » et vice versa. Cette section décrit comment configurer l'Accelerator Object pour arriver à cela.

Le champ « Units » doit être configuré pour chaque famille, les deux valeurs possibles sont :

AO.(Family).Monitor.Units = 'Hardware' or 'Physics'
AO.(Family).Setpoint.Units = 'Hardware' or 'Physics'

Bien qu'il soit possible de choisir suivant les cas l'un ou l'autre mode, il est fortement recommandé d'utiliser toujours le même mode pour toutes les applications. Notons que la plupart des fonctions et applications permettent de modifier localement le mode et de passer par exemple du mode unité « Hardware » au mode unité « Physics ».

6.2 Unités matérielles (« Hardware Units »)

Toute application communiquant avec TANGO utilise les unités Hardware (des ampères pour les alimentations, ...). Les applications qui lisent des attribut TANGO ou qui assignent des nouvelles valeurs en unité Hardware, n'ont pas besoin de conversion d'unité si appelées au travers des fonctions *getpv* / *setpv*. L'unité matériel est habituellement utilisée pour toutes les applications en ligne, comme par exemple les mesures de matrices réponse, la correction d'orbite fermée. Les deux fonctions prenant en compte les unités et conversions éventuelles sont *getpv* et *setpv :*

1. Si la fonction *getpv* est appelée avec AO.(Family).Monitor.Units = 'Hardware', la valeur de relecture retournée par *getpv* est en unité Hardware après exécution de la commande *readattribute;*

2. Si la fonction *setpv* est appelée avec AO.(Family).Setpoint.Units = 'Hardware', la valeur de consigne est unitilsée telle quelle (sans conversion) lors de l'exécution de la commande *writeattribute*.

6.3 Unités physiques (« Physics Units »)

Les unités « Physics » sont utilisées par toute application calculant des paramètres et grandeurs accélérateur (valeur de gradient, radians pour des correcteurs, ...). Ainsi bien que les applications puissent lire ou écrire des valeurs de contrôle en unités physiques, le système de contrôle communique en unités matérielles. Ce qui signifie que les fonction de bas niveau *getpv* et *setpv* doivent faire la conversion.

1. Lorsque la fonction *getpv* est appelée avec AO.(Family).Monitor.Units = 'Physics', l'attribut à lire est converti d'unité « Hardware » (eg. Ampère) en unité physiques (eg. Radians) dans la fonction *getpv* juste après exécution de la commande *readattribute*.

2. Lorsque la fonction *setpv* est appelée AO.(Family).Setpoint.Units = 'Physics',, la valeur de consigne est convertie d'unité « Physics » (eg.

radians) en unité Hardware (eg. ampères) dans la fonction *setpv* juste avant l'appelle de la commande *writeattribute*.

6.4 Fonctions de conversion

Les deux fonctions de conversion sont les fonctions *hw2physics* et *physics2hw* dans le MML. La première (*hw2physics*) transforme des unités Hardware en unités Physics. La seconde (*physics2hw*) des unités « Physics » en unités « Hardware ».

La conversion pour un champ donné en faite en appelant la fonction spécifiée dans le champ *HW2PhysicsFcn* ou *Physics2HWFcn* en utilisant les paramètres définis dans les champs *HW2PhysicsParams* ou *Physics2HWParams*. Si la fonction définie dans le champ *HW2PhysicsFcn* ou *Physics2HWFcn* n'existe pas, alors la conversion revient juste à appliquer le facteur de gain spécifié par *HW2PhysicsParams* ou *Physics2HWParams*.

Note important

L'unité « Physics » doit correspondre à l'unité AT, lorsque que le simulatuer AT est utilisé.

Exemples

Si l'objet AO est en unités « Hardware », la fonction *getsp* renvoie une valeur dans cette unité. La fonction *hw2physics* permet de convertir le courant du quadrupôle Q1, en valeur de gradient (K).

```
>> val  = getsp('Q1');
>> Kval = hw2physics('Q1', 'Setpoint', val);
```

Pour sélectionner les quadrupôles 1, 2 et 4 de la famille Q1, il suffit de spécifier la liste de devices :

```
>> val  = getsp('Q1',[1; 2; 4]);
>> Kval = hw2physics('Q1', 'Setpoint', val, [1; 2; 4]);
```

6.5 Configuration de l' « Accelerator Object » pour les conversions

Le Tableau 6.1 contient les différents champs à spécifier afin que le MML soit configuré pour les conversion d'unités. Ces champs doivent être définis pour chaque sous champ d'une famille (« Monitor », « Setpoint », etc).

Champ	Fonctionnalité
HW2PhysicsFcn	– Nom ou référence de fonction (« mapping function ») entre unités « Hardware » et « Physics ».
HW2PhysicsParams	– Matrice ou tableau de cellules de paramètres nécessaires à la fonction *HW2PhysicsFcn*
Physics2HWFcn	– Nom ou référence de fonction (« mapping function ») entre unités « Physics » et « Hardware ».
Physics2HWParams	– Matrice ou tableau de cellules de paramètres nécessaires à la fonction *Physics2HWFcn*
PhysicsUnits	– Champ facultatif pour le nom d'unité « Physics » (string)
HWUnits	– Champ facultatif pour le nom d'unité « Hardware » (string)

Tableau 6.1 Champs de l' « Accelerator Object » à configurer pour la conversion entre unités.

6.6 Fonctions de conversion et paramètres

Les fonctions de conversion ou leur référence (*handles* au sens Matlab) sont stockées dans les champs *HW2PhysicsFcn* et *Physics2HWFcn*. Les fonctions *physics2hw* et *hw2phyics* appellent alors la fonction Matlab *feval* avec en argument la liste de paramètres pour faire la conversion. Si le champ fonction n'est pas défini, alors la conversion appliquée est juste un changement d'échelles en utilisant la liste de paramètres.

Les paramètres pour les fonctions de conversion sont stockés dans les champs *HW2PhysicsParams* et *Physics2HWParams*. Ils doivent absolument être consistants avec la syntaxe d'appel des fonctions *HW2PhysicsFcn* et *Physics2HWFcn*. Pour une famille de M devices et N paramètres à passer à la fonction de conversion alors les champs *HW2PhysicsParams* et *Physics2HWParams* sont exprimés dans un des cinq formats suivants :

1. vecteur 1xN
2. matrice MxN
3. matrice de string avec M colonnes
4. tableau de cellules de N éléments où chaque élément est un vecteur de dimension M
5. vide

Pour des matrices, le nombre de colonnes soit être égal au nombre d'équipements de la famille. Il vaut 1 si tous les équipements sont traités de la même manière. Chaque colonne de la matrice, correspond à un argument différent pour les fonctions *HW2PhysicsFcn* et *Physics2HWFcn*. Si la matrice est de type string, alors les colonnes correspondant au équipements sont

considérées comme argument unique. Dans le cas de paramètre multiples de type non scalaire, un tableau de cellule doit être utilisé. Le contenu de chaque cellule est passé aux fonctions *HW2PhysicsFcn* et *Physics2HWFcn* comme argument distinct (Ce format n'est utilisé en pratique que très rarement, pour des cas compliqués). Si les champs restent vide, aucun paramètre n'est passé à la fonction de conversion.

6.7 Exemples

Quelques exemples de configuration de l'Accelerator Object sont maintenant donnés.

1. Si les champs *HW2PhysicsFcn* ou *Physics2HWFcn* ne sont pas définis, les paramètres *HW2PhysicsParams* et *Physics2HWParams* sont de changements d'échelle. Si les unités « Physics » pour les BPM sont des mètres, et des millimètres pour les unités « Hardware », alors la configuration est la suivante :

```
AO.(BPMx).FamilyName                = 'BPMx';
AO.(BPMx).Monitor.Units             = 'Hardware';
AO.(BPMx).Monitor.HW2PhysicsParams =  1e-3;
AO.(BPMx).Monitor.Physics2HWParams =  1000;
AO.(BPMx).Monitor.HWUnits           = 'mm';
AO.(BPMx).Monitor.PhysicsUnits      = 'm';
```

2. Le champ *HW2PhysicsFcn* peut être une fonction en ligne (*inline function* au sens Matlab). Pour le même exemple en ajoutant un offset à la conversion, la configuration devient :

```
AO.(BPMx).FamilyName                = 'BPMx';
AO.(BPMx).Monitor.Units             = 'Hardware';
AO.(BPMx).Monitor.HW2PhysicsFcn     = inline('P1.*x+P2',  2);
AO.(BPMx).Monitor.Physics2HWFcn     = inline('P1.*x+P2',  2);
AO.(BPMx).Monitor.HW2PhysicsParams =  [1e-3 0];
AO.(BPMx).Monitor.Physics2HWParams =  [1000 0];
AO.(BPMx).Monitor.HWUnits           = 'mm';
AO.(BPMx).Monitor.PhysicsUnits      = 'm';
```

3. Le champ *HW2PhysicsFcn* peut aussi être une fonction (voir plus loin la manière d'écrire cette fonction). Si les fonctions *amp2k* et *k2amp* permettent de convertir des ampères et gradients (K) en utilisant une expression polynomiale et un offset, la configuration devient :

```
AO.(QF).FamilyName                = 'Q1';
AO.(QF).Monitor.Units             = 'Hardware';
AO.(QF).Monitor.HW2PhysicsFcn     = @amp2k
AO.(QF).Monitor.Physics2HWFcn     = @k2amp;
AO.(QF).Monitor.HW2PhysicsParams = {[-0.06  .3  0],  0};
AO.(QF).Monitor.Physics2HWParams = {[-0.06  .3  0],  0};
AO.(QF).Monitor.HWUnits           = 'amperes';
AO.(QF).Monitor.PhysicsUnits      = 'K';
```

Dans le cas où les coefficients polynomiaux diffèrent d'un aimant à l'autre, le champ doit contenant les coefficients doit être une matrice dont le nombre de colonnes correspond au nombre d'aimants.

6.8 Ecriture d'une fonction de conversion

Les fonctions de conversion (eg. *k2amp* et *amp2k de l'exemple 4*) doivent avoir les propriétés suivantes :

1. La fonction de conversion est indépendante du MML
2. Son expression est la même pour tous les équipements d'une même famille. Seuls les paramètres peuvent varier d'un équipement à un autre.
3. Tous les paramètres nécessaires à la conversion sont passés comme arguments d'entrée de la fonction de conversion.
4. La fonction de conversion doit accepter des entrées vectorielles pour traiter un ensemble d'équipements.

La syntaxe générale pour une fonction de conversion est :

```
myhw2physicsfcn(Val, Param1, Param2, ..., ParamN)
```

avec Val, la valeur de consigne (issue de *hw2physics)* et les N paramètres provenant du champ *HW2PhysicsParams*.

6.9 Exemple de fonction de conversion

Soit la conversion suivante où x est la valeur à convertir et y le résultat de conversion :

$$y = s\left(c_o + c_1 x + c_2 x^2\right)$$

avec s comme facteur d'échelle, c0, c1 comme coefficients polynomiaux.

```
function Y = myhw2physicsfcn(X, s, c0, c1,c2)
Y = s * (c0 + c1*X + c2*X^2);
```

Appel de la fonction :

```
>> myhw2physicsfcn(1,2,3,4,5)
ans = 25
```

La même fonction peut s'écrire de manière vectorielle :

```
function Y = myhw2physicsfcn(X,s,c1,c2);
Y = s(:) .* (c0(:) + c1(:).*X(:) + c2(:).*X(:).^2);
```

Exemple d'appel de la fonction :

```
>> S = [1;  0.99;  1.01];
>> C0 = [1;  2;  3];
>> C1 = [4;  5;  6];
>> C2 = [7;  8;  9];

>> myhw2physicsfcn( [pi;  exp(1);  sqrt(2)],  S, C0; C1, C2)

ans = [82.6536
       73.9568
       29.7801]
```

Il est prudent de vérifier la consistance entre les deux fonctions *myhw2physicsfcn* et *HW2PhysicsParams* en procédant ainsi :

Si *HW2PhysicsParams* est une matrice :

```
>> feval(HW2PhysicsFcn,X,HW2PhysicsParams(:,1),  ... HW2PhysicsParams(:,N))
```

Si *HW2PhysicsParams* est un tableau de cellules :

```
>> feval(HW2PhysicsFcn,X,HW2PhysicsParams{:})
```

La fonction précédente peut être généralisée pour un nombre arbitraire de coefficients polynomiaux. L'Illustration 1.1 présente le schéma générale de fonctions telles *amp2k* et *k2amp*.

$$y = s\left(c_0 + c_1 x + c_2 x^2 + \ldots c_N x^N\right)$$

```
function  k = amp2k(Amps,  C,  ScaleFactor)
% C = [Cn  ... C2  C1  C0]
Amps = Amps ./  ScaleFactor;
brho = (10/2.998);
for  i  = 1:length(Amps)
    if  size(C,1)  == 1
        k(i,1)  = polyval(C,  Amps(i))  /  brho;
    else
        k(i,1)  = polyval(C(i,:),  Amps(i))  /  brho;
    end
end
```

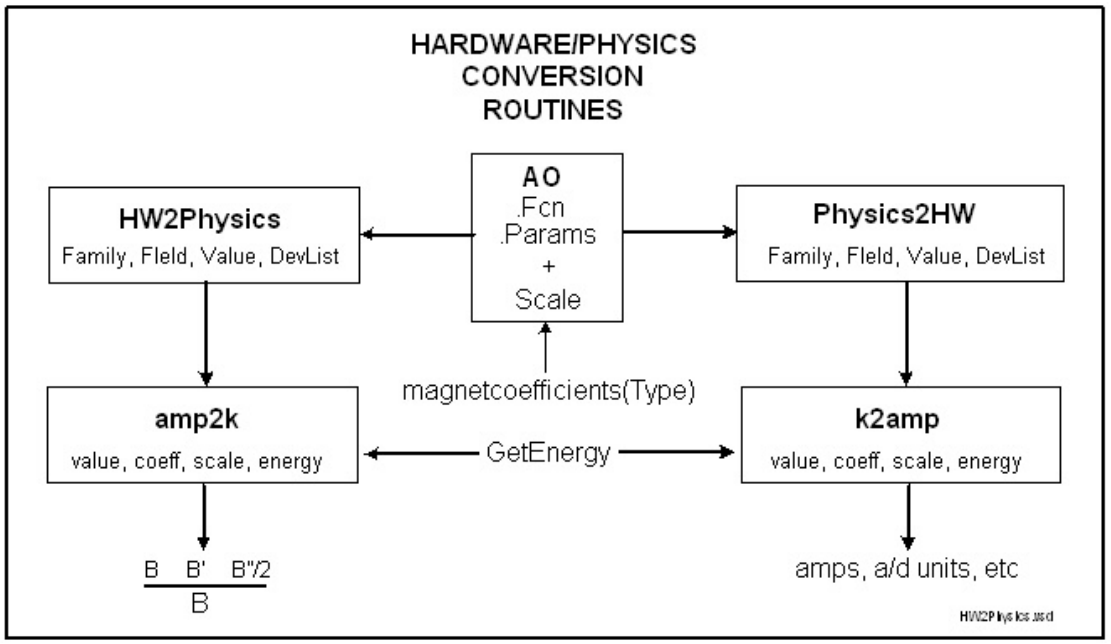


Illustration 6.1 Schéma de fonctionnement des commandes amp2k *et* k2amp *(SPEAR3)*

Annexe 7 Binding TANGO/Matlab

A FAIRE

Commande	Fonctionnalité
tango_get_attribute_property	Lit une propriété d'un attribut
tango_set_attribute_property	Ecrit une propriété d'un attribut
tango_get_db_property	Lit une propriété de la base de données statique
	tango_set_db_property Ecrit une propriété de la base de données statique
make_QT_db	
writeattribute	Fonction générique utilisée par MML pour communiquer avec TANGO (écriture)
readattribute	Fonction générique utilisée par MML pour communiquer avec TANGO (lecture)
getattribute	Renvoie l'attribute et le nom du device serveur dans une chaîne de caractères nom_device/nom_attribut

Tableau 6.2 Commandes ajoutées

Divers

Commande	Fonctionnalité
cell2field	Conversion d'une cellule en structure

Index des tables

Index des illustrations

Index lexical

Bibliographie

1: , Matlab TM, , http://www.mathworks.com/
2: Andrei Terebilo, Matlab Channel Access (MCA),
3: Andrei Terebilo, AT Users Manual,
4: , EPICS, ,
5: , Spear3, , http://www- ssrl.slac.stanford.edu/spear3/
6: , , , http://www- als.lbl.gov/als/
7: , Canadian Light Source, , http://www.cls.usask.ca/
8: , , , http://www.esrf.fr/computing/cs/tango/index.html
9: N. Leclercq, ??? Bindings TANGO/MATLAB,
10: L. Nadolski, A. Loulergue, Nomenclature Tango,