Règles concernant l'écriture des catalogues

Résumé :

Nous donnons dans ce document, quelques règles (ou conseils) que doit respecter le développeur lorsqu'il ajoute ou modifie un catalogue de commande ou un catalogue d'élément fini.

Table des matières

1 Catalogues de commandes

- Utiliser le plus possible les possibilités du superviseur concernant les exclusions, les valeurs par défaut…
- Lorsqu'un argument est de type "texte" :
  s'il peut prendre un nombre fini de valeurs, donner la liste intégrale des possibilités par le mot clé `into`.
- Les commentaires sont les bienvenus.
- Faire valider le vocabulaire en RTA.
- Consulter le document [D5.01.01].

2 Catalogues d'éléments

2.1 Catalogue `phenomenons_modelisations.py`

- Les noms des phénomènes et des modélisations doivent être validés par l'EDA / RTA car ils apparaissent à l'utilisateur.

2.2 Catalogue `physical_quantities.py`

- Donner un nom aux grandeurs de la forme `XXXX_S` où `S` peut valoir :
  - `R` : réel
  - `C` : complexe
  - `F` : fonction (K8)
- Lorsqu'on ne veut pas créer une nouvelle grandeur trop particulière, utiliser les grandeurs "neutres" : `NEUT_R` ou `NEUT_K24`.
- Quand on modifie le catalogue des GRANDEURS, penser à mettre à jour le document "description des grandeurs" [D4.04.02] et classer les noms de grandeurs par ordre alphabétique.
- Ne pas définir de grandeurs de type : L, K32, K80
- Ne pas détruire de composantes `CMPS` dans une grandeur existante sans avoir vérifié qu'aucun `type_element` ne l'utilise.
- Ne pas changer l'ordre des CMPS d'une grandeur existante sans modifier les `type_element` qui l'utilisent.
- Quand on introduit une nouvelle composante dans une grandeur, la mettre à la suite des CMPS existantes. Ceci évite de "casser" de la programmation trop "en dur" ; par exemple, un programmeur peut avoir fait :
  - vérification que '`DX`' et '`DY`' sont les deux premiers CMPS de la grandeur '`DEPL_R`',
  - puis utilisation de `DEPL_R(I), I = 1,2.`

2.3 Catalogues des OPTIONS (Options/*.py)

- Les noms d'options doivent être validés par l'EDA / RTA s'ils apparaissent à l'utilisateur.
- Avant d'ajouter un nouveau paramètre dans une option, il faut regarder s'il n'existe pas déjà dans le catalogue des paramètres partagés (Commons/parameters.py)
- Ne pas réinventer les noms des paramètres pour chaque option ; s'inspirer de ceux déjà choisis. La forme usuelle est la suivante : nom_par = 'P'//nom_gd.
  Exemples : PMATERF, PGEOMER
- Commenter champ paramètre : exemple :
  PCAGEPO = InputParameter(... comment="RAYON ET EPAISSEUR POUR LES TUYAUX"
- Si le champ qui est associé au paramètre a toujours la même « origine ». On peut indiquer son « container », c'est à dire la structure de donnée où se trouve ce champ.
  Par exemple :
  PCAGNBA = InputParameter(..., container='CARA!.CARGENBA',
- Quand on ajoute une nouvelle option, il faut remplir le bloc « CondCalcul ». Cette tâche n'est pas facile, car elle nécessite d'avoir une connaissance de tous les éléments. Dans le doute, il faut toujours préférer déclarer une liste d'éléments plus vaste que nécessaire. Lorsqu'un utilisateur sera confronté à un blocage dû à cet excès de prudence, il sera toujours temps de lever la restriction en ajoutant une ligne « - » dans le bloc.

2.4 Catalogues des type_element (Elements/*.py)

- Avant d'ajouter un nouveau mode local dans un catalogue d'élément, il faut regarder s'il n'existe pas déjà dans le catalogue des modes locaux partagés (Commons/located_components.py)
- Pour les noms des modes locaux s'inspirer des noms choisis par les type_element voisins.
  Respecter l'usage :
  - Cxxxx : mode de type carte
  - Nxxxx : mode de type cham_no
  - Exxxx : mode de type cham_elem
- Se poser des questions sur la cohérence du type_element que l'on modifie avec les autres type_element :
  - pourquoi le nouveau type_element aurait-il un mode local "aux noeuds" alors que tous les autres l'ont "aux points de GAUSS" ?
  - pourquoi le nouveau type_element n'utilise-t-il pas ce champ paramètre, cette composante ?