

PROGRAMME D'INTERFACE

Spécifications Fonctionnelles Détailées

Réf. POM3-CS-2021-DS-PI-011

	Nom	Société	Fonction	Date	Visa
Rédigé par :	M. Renon	CS GROUP	Equipe CS		
Validé par :	C. Mertz	CS GROUP	Chef de projet		
Pour application :	J. Covès	CS GROUP	Directeur de projet		

CS GROUP
6 rue Brindejonc des Moulinais
Parc de la Grande Plaine
BP 15872
31506 Toulouse Cedex 5

ED.	RÉV.	DATE	MOTIF
01	00	21/08/15	Création du document
3.0	02	03/09/18	Prise en compte des remarques
3.0	03	05/09/18	Prise en compte des remarques
3.0	04	17/09/18	Ajout de la méthode d'initialisation « run permanent »
3.0	05	18/01/19	Prise en compte des remarques
3.0	06	18/12/19	Gestion des dates T0, T1, T2 ; Définition des nouveaux modes de fonctionnement ; Modification des dossiers dans les fichiers '.ini' ; Ajout des nouveaux Processeurs
3.0	07	10/02/20	Dépendance numpy ≥ 1.11 pour libhydro 0.8
3.0	08	09/03/2020	Règle d'archivage ; critère données prévision ; règles de reconstruction des données pour la POM
3.0	09	16/03/2020	[#189280] ajout d'un mode OneRun pour le PIG 3
3.0	10	26/03/2020	Prise en compte des remarques du SCHAPI
3.0	11	30/03/2020	[#192372] gestion erreur calcul dans progression
3.0	12	14/08/2020	Modifications pour PIPT3 :amélioration méthodes init Refonte de la gestion de la configuration Ajout de l'option « --version »
3.0	13	11/09/2020	Mise à jour de la détermination de T0 Erreur bloquante si des données d'entrée sont MBDLAMEDOSYMPO ou MDOUTPUT
3.1	00	20/10/2020	Développement du PI 3.1 et mise en œuvre dans les Plx (PIG, PIPT, PIT, PIM)
3.1	01	04/03/2021	Mise à jour du format de la table de conversion (§3.5.1) correction typo pour les fichiers .ini (§3.5.6)
3.2	00	09/06/2022	§1.5.4 Maj en PI3.2 avec libbdimage 1.5.8
3.2	01	02/02/2023	Mise à jour BC9 - migration Python3 - homogénéisation des clés dans .ini - parallélisation des calculs prévision
3.2	02	04/02/2023	§4.8.4.2 Prise en compte des retours du SCHAPI
3.2	03	30/06/2023	§2.4 Compléments pour le calcul de T0, T1, T2, lié au ft#301599 : §3.5 Support du format « .toml »
4.0	00	09/10/2023	§1.5.4 Maj en PI4.0 avec libbdimage 1.5.11 §3.2.4 argument « modeles » dans le mode de calcul POM §4.2.1 suppression argument « --verbose », ajout de

			l'argument « --logs »
4.0	01	27/03/2024	§ 3.5.3 Paramètres de calculs prévision parallèles Mise à jour des valeurs par défaut pour les clés 'min_free_mem_mo' et 'max_attente_sec'
4.1	00	28/10/2024	[issue #39] Sandre V2 §3.2 Protocole POM §3.3.1 XML Sandre §4.7.3.1 XML Sandre §4.9 Sorties(OutputsBaseProcessor)

Sommaire

Table des matières

1. Généralités.....	8
1.1 Objet du document.....	8
1.2 Glossaire.....	8
1.3 Présentation globale du système.....	8
1.3.1 But.....	8
1.3.2 Contexte du système.....	9
1.4 Architecture générale.....	9
1.4.1 Déploiement et architecture des composants.....	9
1.4.2 Architecture logicielle.....	9
1.5 Version des composants.....	10
1.5.1 Version POM.....	10
1.5.2 Version Python.....	10
1.5.3 Version libhydro.....	10
1.5.4 Version libimage.....	10
2. Fonctionnalités.....	11
2.1 Initialisation d'un calcul.....	11
2.1.1 Sans reprise.....	11
2.1.2 Reprise d'un calcul récent.....	11
2.1.3 Initialisation à partir fichiers par défaut.....	12
2.1.4 Initialisation à partir d'une cote constante.....	12
2.1.5 Initialisation à partir d'observations.....	13
2.1.6 Initialisation sans calcul d'analyse.....	13
2.2 Propagation des séries.....	13
2.3 Modes de fonctionnement.....	13
2.4 Dates des simulations analyse/prévision.....	14
2.4.1 Détermination de T0.....	14
2.4.2 Détermination de T1.....	15
2.4.3 Détermination de T2.....	15
2.4.4 Règles de cohérence.....	16
3. Interfaces.....	17
3.1 Ligne de commande.....	17
3.2 Protocole POM.....	17
3.2.1 Fichier de paramétrage POM (parameters.xml).....	17
3.2.2 Fichier de progression.....	18
3.2.3 Arborescence des fichiers d'échange avec la POM.....	19
3.2.4 Mode de calcul POM.....	19
3.3 Format des fichiers d'entrée.....	20
3.3.1 XML sandre.....	20
3.3.2 XML Image.....	20
3.3.3 JSON Lamedo.....	20
3.3.4 Autres.....	21
3.3.5 Format des fichiers de sortie.....	21
3.4 Données.....	21
3.5 Paramétrage du PI.....	22
3.5.1 Paramètres généraux.....	23

3.5.2 Paramètres des modèles.....	25
3.5.3 Paramètres de calculs prévision parallèles.....	25
3.5.4 Paramètres de reprise (ou d'initialisation).....	26
3.5.5 Paramètres de propagation des incertitudes.....	27
3.5.6 Paramètres de nettoyage.....	27
3.5.7 Paramètres d'environnement.....	29
3.5.8 Héritage de fichiers « .ini ».....	29
3.5.9 Lecteur de fichier « ini » (classe IniBase).....	30
3.6 Configuration du PI.....	30
3.7 Définition du mode de fonctionnement.....	31
3.8 Définition de la méthode d'initialisation pour le mode ANALYSE/PREVISION.....	32
3.9 Fichiers de reprises.....	32
3.10 Fichiers défaut.....	33
4. Processeurs.....	34
4.1 Généralités.....	34
4.1.1 Exceptions.....	34
4.1.2 Processeur de base.....	34
4.2 Démarrage (StartProcessor).....	35
4.2.1 Analyse de la ligne de commande.....	35
4.2.2 Création du fichier PID de verrou.....	36
4.2.3 Procédure principale « main() ».....	36
4.3 Classe ModelBase.....	36
4.3.1 ModelAnalysePrevision.....	38
4.3.2 ModelOneRun.....	39
4.3.3 Gestion du statut du calcul (OK/Erreur).....	40
4.3.4 Gestion de fin de traitement.....	41
4.4 Initialisation (InitializeBaseProcessor).....	41
4.4.1 Vérification des lancements simultanés.....	41
4.4.2 Création des méthodes de reprise.....	42
4.5 Nettoyage (CleanerBaseProcessor).....	42
4.5.1 Processeur.....	42
4.5.2 Cleaner.....	42
4.6 Scénario (ScenarioBaseProcessor).....	43
4.6.1 Création.....	43
4.6.2 Recherche scénario d'analyse (uniquement pour ScenarioAnalysePrevision).....	43
4.6.3 Exécution.....	44
4.6.3.1 ScenarioAnalysePrevision.....	44
4.6.3.2 ScenarioOneRun.....	44
4.6.3.3 Vérifications pour AnalysePrevision.....	44
4.6.4 Traitement des sorties.....	45
4.7 Entrées (InputsBaseProcessor).....	45
4.7.1 Création.....	45
4.7.2 Exécution.....	46
4.7.3 Lecture des fichiers.....	46
4.7.3.1 XML Sandre.....	46
4.7.3.2 XML Image.....	47
4.7.3.3 JSON Lamedo.....	47
4.7.3.4 Fichier / format inconnu.....	47

4.7.4 Détermination du pas de temps de l'entrée.....	47
4.7.5 Règles d'ajout des données.....	48
4.7.5.1 Détermination du type OBS ou PREV.....	48
4.7.5.2 Gestion du chevauchement des données.....	48
4.7.5.3 Métadonnée Sympo (MDBDLAMEDOSYMPO).....	48
4.7.5.4 Métadonnée de sortie (MDOUTPUT).....	49
4.7.6 Traitement sur l'entrée (conversion de codes).....	49
4.8 Gestion des Runs.....	49
4.8.1 LaunchProcessor.....	50
4.8.2 RunConfigProcessor.....	50
4.8.2.1 Mode Unique.....	50
4.8.2.2 Mode Multiple.....	50
4.8.3 RunProcessor.....	51
4.8.4 Gestion des calculs prévision en parallèle.....	52
4.8.4.1 Principe général.....	52
4.8.4.2 Principe de l'ordonnanceur.....	52
4.8.4.3 Contrôle du système.....	53
4.8.4.4 Gestion de la progression.....	53
4.9 Sorties (OutputsBaseProcessor).....	54
4.9.1 Création.....	54
4.9.2 Exécution.....	54
4.9.3 OutputsAnalysePrevision.....	55
4.9.3.1 Lecture des résultats.....	55
4.9.3.2 Constructions des prévisions.....	56
4.9.4 OutputsOneRun.....	56
4.9.4.1 Lecture des résultats.....	56
4.9.4.2 Constructions des prévisions.....	57
5. Exécution du PI.....	58
5.1 Point d'entrée (main).....	58
5.2 LogMessage.....	58

Liste des tableaux

Tableau 1.1 : Glossaire.....	8
Tableau 3.1 : Comparaison de syntaxe « .ini » et « .toml » pour les PIX.....	23
Tableau 3.2 : paramètres généraux du PI.....	24
Tableau 3.3 : paramètres des modèles du PI.....	25
Tableau 3.4 : paramètres des calculs prévisions du PI.....	26
Tableau 3.5 : paramètres de reprise du PI.....	27
Tableau 3.6 : paramètres de propagation des incertitudes du PI.....	27
Tableau 3.7 : paramètres de nettoyage du PI.....	28
Tableau 3.8 : paramètres d'environnement du PI.....	29
Tableau 4.1 : règles de chevauchement des données.....	48

Liste des figures

Figure 1 : architecture matérielle.....	9
Figure 2 : détermination des dates de reprise d'un calcul récent.....	12
Figure 3 : dates T0, T1 et T2.....	14
Figure 4 : diagramme de classes des paramètres.....	18
Figure 5 : diagramme de classes de progression.....	18
Figure 6 : diagramme de classes de données.....	21
Figure 7 : diagramme de classes des Exceptions.....	34
Figure 8 : diagramme de classes de processeurs.....	35
Figure 9 : attributs de la classe ModelBase.....	37
Figure 10 : classe ModelBase.....	38
Figure 11 : hiérarchie des classes héritant de ModelBase.....	38
Figure 12 : classe ModelAnalysePrevision.....	39
Figure 13 : classe ModelOneRun.....	40
Figure 14 : classes utilisées pour la gestion des runs.....	49

1. Généralités

1.1 Objet du document

Ce document présente la spécification logicielle du programme d'interface générique (PI) entre la plateforme opérationnelle pour la modélisation (POM) et les potentielles plateformes de modélisation. Ce programme est nommé PI (Programme d'Interface).

Il ne constitue pas un programme utilisable opérationnellement mais plutôt une librairie (également nommée PI pour PomInterface) facilitant la réalisation de programmes d'interfaces opérationnels.

Note : le code source du PI est disponible sur Gitlab : <https://gitlab.com/vigicrues/pom/pi>

Note : le présent document est créé pour la version 3 du PI. Les versions précédentes étaient décrites dans les Plx associés.

1.2 Glossaire

Acronyme	Signification
BDH	Base de Données Hydro.
BDTR	Également appelée BDTR (Base de Données Temps Réel).
PHYC/PHYL	Également appelée PHYC/PHYL (Plateforme Hydro Centrale/Locale) pour inclure ses services web.
CS	CS GROUP
IHM	Interface Homme Machine
Plx	Ensemble des Programmes d'Interface basés sur le PI : PIG, PIT, PIM, PIPT
POM	Plateforme Opérationnelle pour la Modélisation
SCHAPI	Service Central d'Hydrométéorologie et d'Appui à la Prévision des Inondations
SPC	Service de Prévision des Crues

Tableau 1.1 : Glossaire

1.3 Présentation globale du système

1.3.1 But

La POM est un outil du SCHAPI permettant de lancer des calculs de prévision des crues sur différentes plateformes de modèles. Parmi ces plateformes, certaines sont recommandées nationalement par le SCHAPI : Mascaret, Telemac, GRP, Plathynes.

La POM entre en interaction avec ces plateformes selon un protocole standardisé. Les modèles doivent donc s'interfacer avec ce protocole pour interagir avec la POM.

Le présent document définit les méthodes et procédures génériques implémentées pour faciliter la réalisation de programmes de pilotage de plateformes de modélisation à partir des informations fournies par la POM.

1.3.2 Contexte du système

Conceptuellement, le serveur POM et le serveur modèle (le plus souvent une machine virtuelle hébergeant le logiciel de calcul) sont des serveurs distants. La présente librairie est pensée pour un lancement à distance par la POM à l'aide du protocole SSH.

L'exécutable doit donc être lancé en ligne de commande avec pour paramètre

- ✓ le nom du fichier de paramétrage que la POM fournit (« parameters.xml »)
- ✓ éventuellement, le nom du fichier de paramétrage au format « .ini » du Plx

1.4 Architecture générale

1.4.1 Déploiement et architecture des composants

Le programme d'interface doit être déployé sur un serveur accessible en SSH depuis le serveur POM.

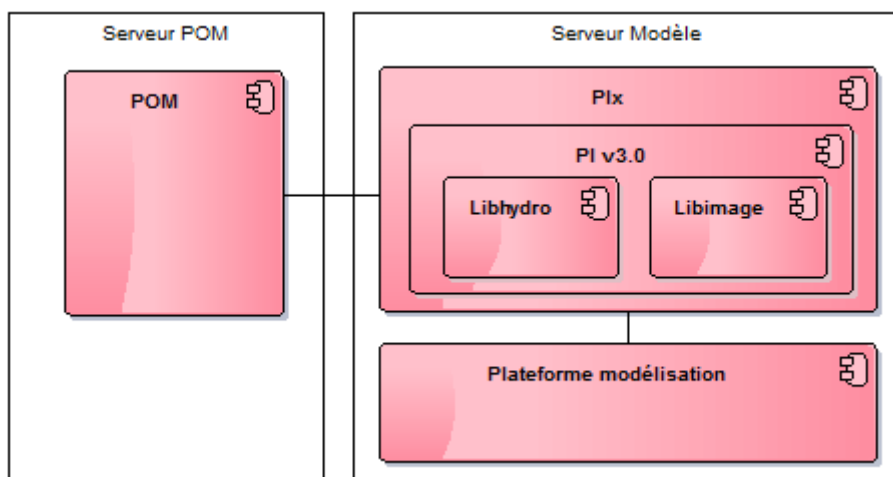


Figure 1 : architecture matérielle

1.4.2 Architecture logicielle

L'architecture logicielle retenue est une architecture à base de « **processeurs** ».

Chaque **processeur** est une classe Python qui prend en charge une partie du cycle de vie d'un calcul dans le protocole POM.

Tous les processeurs héritent d'une classe commune, et peuvent faire l'objet d'une surcharge par héritage dans les différents Plx. Ils sont donc conçus de manière très atomique, avec de nombreuses fonctions simples permettant d'en spécialiser les fonctionnalités (par héritage / surcharge).

1.5 Version des composants

1.5.1 Version POM

La version de compatibilité avec la POM est définie par la version du protocole POM implémenté, comme indiqué au chapitre 3.2.1 Fichier de paramétrage POM (parameters.xml).

1.5.2 Version Python

Les développements sont réalisés en Python 3.7.

1.5.3 Version libhydro

Le PI exploite les fonctionnalités du composant libhydro (v0.940), maintenu par le SCHAPI, pour la manipulation des fichiers au format XML Sandre.

La libhydro est incluse dans le PI : elle est installée à côté du dossier 'pominterface'.

La libhydro 0.9.4 dépend de numpy ≥ 1.11 pour la gestion des dates (fin de prise en compte des timezones). Le PI définit cette dépendance dans le fichier 'setup.ini' :

```
install_requires=(  
    'numpy >= 1.11.0',
```

1.5.4 Version libimage

Le PI exploite les fonctionnalités du composant libimage (v1.5.11 light : libbdimage_light-1.5.11.zip), maintenu par le SCHAPI, pour la manipulation des fichiers au format XML Image.

2. Fonctionnalités

Ce chapitre vise à décrire certaines fonctionnalités, dont l'implémentation décrite ci-après ne permet pas de les identifier simplement.

2.1 Initialisation d'un calcul

Nouveauté PI v3

Le calcul d'une prévision par un modèle événementiel nécessite de définir un état initial de départ, que le calcul va faire évoluer pour produire l'anticipation attendue. La constitution de cet état est appelée « l'initialisation ».

Plusieurs méthodes d'initialisation existent. Les chapitres suivants précisent les méthodes prises en charge par le PI :

- ✓ REPRISE (reprise d'un calcul récent) : on tente de repartir d'un état généré lors d'un calcul précédent (idéalement assez récent)
- ✓ DEFAUT (reprise à partir des fichiers par défaut) : on tente de repartir d'un état par défaut, stocké dans un des fichiers de reprise « par défaut ».
- ✓ CONSTANTE : ce cas correspond à l'initialisation à partir de valeurs constantes. Certaines plateformes de modélisation permettent de définir une valeur constante comme état de départ.
- ✓ INIT_OBS : cette méthode permet d'effectuer un calcul d'analyse en utilisant uniquement les données fournies par la POM,
- ✓ SANS_ANALYSE : cette méthode permet de ne pas effectuer de run d'analyse.

Les méthodes d'initialisation sont paramétrées par le mode de calcul du scénario courant (cf. 3.2.4) ou à défaut par le fichier « .ini » (cf. 3.5). Il s'agit d'une liste de méthodes, testées les unes à la suite des autres, jusqu'à ce que l'une d'entre elle aboutisse à un résultat. Le PI tente donc des lancements de runs pour chaque méthode, dans l'ordre, tant que le calcul de la plateforme de modélisation n'a pas abouti sans erreur.

Note : il est du ressort du Plx d'implémenter les méthodes qui permettent de mettre à jour le paramétrage de la plateforme de modélisation pour les différents types d'initialisation. Ces fonctions sont déclarées dans le PI, non implémentées (abstraites) en vue d'une surcharge par les Plx. Le PI définit seulement l'algorithme permettant la détection et l'appel dynamique de ces méthodes (cf 3.8)

2.1.1 Sans reprise

Si le lancement POM est identifié comme « initialisation » (balise « initialize » du fichier parameters.xml, i.e. le lancement est « personnel avec initialisation »), le PI ne cherche pas à repartir de données archivées (c'est-à-dire qu'il ignore le mode REPRISE).

Dans ce cas, les autres méthodes d'initialisation sont utilisées (DEFAUT, CONSTANTE).

2.1.2 Reprise d'un calcul récent

L'objectif est de trouver le fichier de reprise le plus récent entre deux dates, pour initialiser le calcul.

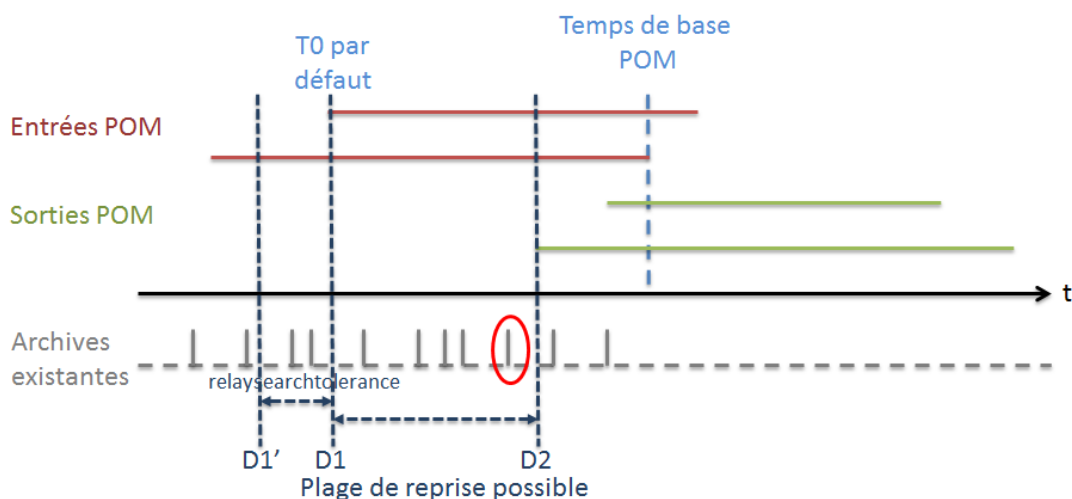


Figure 2 : détermination des dates de reprise d'un calcul récent

Pour déterminer les dates de recherche D1 et D2, un paramètre de tolérance est lu dans le fichier « .ini » (cf. relaysearchtolerance au chapitre 3.5.4).

- ✓ D1 : Temps de base POM – min(profondeur des entrées observées)
- ✓ D1' : D1 - relaysearchtolerance
- ✓ D2 : Temps de base POM – max(profondeur des sorties)

Note si D1 ou D'1 est postérieur à D2 une erreur est levée et le calcul s'arrête.

Si le fichier de reprise à utiliser se trouve avant D1 (donc entre D1' et D1), on considère que tout se passe comme s'il avait été trouvé à D1.

Le fichier de reprise sélectionné est le plus récent entre D1 et D2. Sa date constitue le « T0 » du calcul.

Si aucune archive n'existe entre D1' et D2, une erreur est levée et ce run s'arrête.

2.1.3 Initialisation à partir fichiers par défaut

Le répertoire des fichiers de reprise par défaut est paramétré dans le fichier « .ini » (cf. 3.5.4). Les fichiers à traiter sont, par ordre de priorité :

- ✓ ceux du mode de calcul du scénario en cours de traitement s'il est renseigné
- ✓ sinon, ceux du fichier « .ini » s'ils sont renseignés
- ✓ sinon, tous les fichiers du répertoire de reprise par défaut (par ordre alphabétique)

Les fichiers sont testés dans l'ordre, les uns après les autres, jusqu'à ce qu'un calcul se termine en succès.

2.1.4 Initialisation à partir d'une cote constante

Le mode de calcul du scénario courant (ou à défaut le paramétrage du fichier « .ini ») doit contenir au moins une valeur de constante pour paramétrer le calcul (génération d'un fichier ou mise à jour des fichiers de configuration de la plateforme de modélisation selon le Plx).

Il peut contenir plusieurs valeurs qui seront potentiellement testées dans l'ordre paramétré.

2.1.5 Initialisation à partir d'observations

Cette méthode « INIT_OBS » permet d'effectuer un calcul d'analyse en utilisant uniquement les données fournies par la POM

2.1.6 Initialisation sans calcul d'analyse

Cette méthode « SANS_ANALYSE » permet de ne pas effectuer de run d'analyse, tout en indiquant au Plx que cette méthode a fonctionné sans erreur.

2.2 Propagation des séries

La propagation des séries doit permettre d'effectuer plusieurs runs de la plateforme de modélisation en l'alimentant avec les différentes séries de données prévues en entrée.

L'idée est de produire en sortie autant de séries que de séries utilisées en entrées. Par exemple, produire une série 90 % à partir des séries 90 % (ou moyenne) en entrée. Cela signifie que seules les séries MIN, MOY, MAX, les quantiles et les probabilités sont « propageables » (les quantiles seront associés en sortie aux probabilités correspondantes).

Le PI prévoit donc autant de lancements que de séries à traiter (cf. paramétrage, au chapitre 3.5.5), plus un premier run basé uniquement sur les observations.

Le PI lance donc :

- ✓ Le cas échéant¹, un run d'analyse sur les données observées, sans les données prévues.
- ✓ Puis un run de prévision pour chaque série à traiter (cf. ci-après). Les données d'entrée de ces runs de prévision sont :
 - ↳ un fichier de reprise issu du run d'analyse (spécifique à chaque Plx : le PI crée les fonctions génériques, non implémentées, destinées à être surchargées dans le Plx)
 - ↳ les données d'entrée observées situées dans la plage temporelle du run de prévision
Note : généralement il n'y en aura pas car on n'a pas de vraies observations dans le futur.
 - ↳ les données d'entrée prévues dont les valeurs sont prises égales à la série à traiter

Si aucune donnée d'entrée n'a de valeur pour la série en cours de calcul, le run n'est pas lancé et un message non bloquant le signale dans le contexte d'exécution. C'est par exemple le cas d'un calcul sur la proba 90 % qui ne serait pas renseigné en entrée. Afin d'assurer une compatibilité avec les modèles pluie - débit, le PI peut adopter un fonctionnement plus simple : un seul run par scénario sans filtre sur les données en entrée (comme pour le PIG par exemple).

2.3 Modes de fonctionnement

Afin de synthétiser les points précédents, nous définissons 2 modes de fonctionnement : « ONERUN » et « ANALYSE/PREVISION »

¹ Si les données prévues démarrent dès le début, ce run n'existe pas et tous les runs partent donc de la même initialisation.

- ✓ mode « ONERUN » : quelque soit le nombre de scénarios, un unique run est lancé sur toutes les données de tous les scénarios (utilisé par à partir du PIG 2.x)
- ✓ mode « ANALYSE/PREVISION » : nouveau mode PI 3 :
 - ↳ pour chaque scénario ayant des données observées, un run d'analyse est lancé sur les données observées du scénario (ie avant le temps de base).
Ce run d'analyse va tester toutes les méthodes d'initialisation les unes à la suite des autres, jusqu'à ce que l'une d'entre elles fonctionne et produise un fichier résultat au temps de base
 - ↳ pour chaque scénario ayant des données prévues, des runs de prévision sont lancés sur les données prévues (ie après le temps de base)
Ces runs de prévision sont obligatoirement en mode « REPRISE » à partir d'un fichier défini au temps de base.
Il y a autant de runs que de séries de données à propager.

La description précise du choix du mode de fonctionnement est décrite au chapitre 3.7.

2.4 Dates des simulations analyse/prévision

Quand le PI fonctionne en ANALYSE / PREVISION :

- le calcul d'Analyse est effectué sur les données entre T0 et T1,
- le calcul de Prévision est effectué sur les données entre T1 et T2.

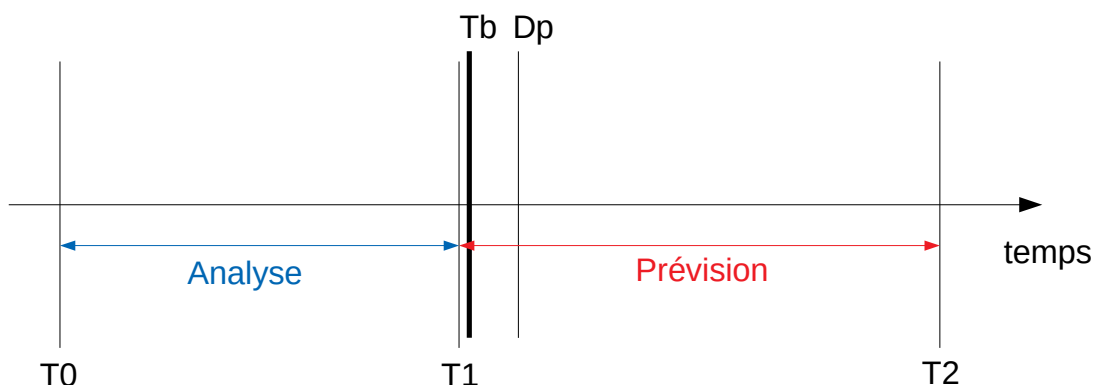


Figure 3 : dates T0, T1 et T2

2.4.1 Détermination de T0

La date T0 est fonction du type de démarrage :

- ✓ Démarrage sur un fichier de reprise (cf 2.1.2 Reprise d'un calcul récent) : la date du fichier constitue la date T0. Attention au cas particulier d'un fichier choisi entre D'1 et D1 : sa date est prise égale à D1. (NB : On doit toujours avoir $D1 \leq T1$).
- ✓ Démarrage sans date de reprise (fichier par défaut, démarrage à partir d'un run permanent ou cote constante) : dans ce cas, la date T0 est calculée comme suit

$$T0 = \{\{\text{TEMPS_BASE}\}\} - \min(\text{profondeur données observées})$$

où « spinup_length » ou « min(profondeur entrées) » est la durée minimale des profondeurs des données observées.

« spinup_length » ou « min(profondeur entrées) » est exprimée en minutes et est > 0 vers le passé.

2.4.2 Détermination de T1

T1 correspond au temps de base défini par la POM (dans le fichier « parameters.xml »), arrondi au pas de temps :

$$T1 = \text{arrondi}(\{\{\text{TEMPS_BASE}\}\}, \{\{\text{PAS_DE_TEMPS}\}\})$$

La fonction d'arrondi est définie ainsi :

```
T1 = TEMPS_DE_BASE
SI PAS_DE_TEMPS >= 24h
    T1.minute = 0
    T1.heure = 0
SI PAS_DE_TEMPS > 1h
    # calage sur l'heure inférieure la plus proche
    T1.minute = 0
    T1.heure = TEMPS_DE_BASE.heure -
(TEMPS_DE_BASE.heure % (PAS_DE_TEMPS/60))
SINON
    # calage sur la minute inférieure la plus proche
    T1.minute = TEMPS_DE_BASE.minute -
(TEMPS_DE_BASE.minute % PAS_DE_TEMPS)
```

Le PAS_DE_TEMPS est défini comme le minimum des pas de temps de chaque <file> de chaque <input>.

Le pas de temps de chaque <file> est l'écart en minutes entre les 2 premières valeurs.

2.4.3 Détermination de T2

La date T2 est calculée comme suit :

$$T2 = \{\{\text{TEMPS_BASE}\}\} + \text{forecast_length}$$

avec « forecast_length » le maximum des échéances des sorties.
« forecast_length » est exprimée en minutes et est >0 vers le futur.

2.4.4 Règles de cohérence

Si $T1 < T0$, une erreur bloquante est générée avec le message : « $T1 < T0$ run d'analyse impossible. Il convient de modifier les plages temporelles de certaines ressources (d'entrée et/ou de sortie). ».

Si $T2 < T1$, une erreur bloquante est générée avec le message : « $T2 < T1$ run de prévision impossible. Il convient de modifier les plages temporelles de certaines ressources de sortie.) ».

Si $T1 = T0$, une erreur bloquante est générée avec le message : « $T1 = T0$ run d'analyse impossible. Il convient de modifier les plages temporelles de certaines ressources (d'entrée et/ou de sortie). ».

Si $T1 = T2$, un warning est affiché dans la progression, avec le message : « $T1 = T2$ donc pas de run de prévision. Il convient de modifier les plages temporelles de certaines ressources de sortie. ».

3. Interfaces

Le PI doit s'interfacer avec le protocole POM, qui impose différentes contraintes : arborescence des fichiers, formats des fichiers fournis et attendus par la POM, fichier de paramétrage, fichier de progression, ...

L'objectif est de simplifier cet interfaçage pour les PIx.

3.1 Ligne de commande

Le PI gère dans le « StartProcessor » l'analyse de la ligne de commande qui a lancé l'exécution.

Celle-ci

- ✓ Doit contenir un premier paramètre référençant le chemin complet du fichier parameters.xml généré par la POM
- ✓ Peut contenir un second paramètre référençant le chemin complet du fichier « .ini » de configuration du PI

3.2 Protocole POM

La présente spécification logicielle est prévue pour s'interfacer avec la version 2.3 ou 3.0 du protocole POM.

3.2.1 Fichier de paramétrage POM (parameters.xml)

Le fichier de paramétrage POM (parameters.xml) contient toutes les informations nécessaires à l'exécution du PI(x). Il est interfacé à l'aide du module « pomparameters.py » du package « pominterface.xmlpom ».

Ce module contient un ensemble de classes Python organisées de manière identique à la structure XML du fichier de paramétrage : à chaque balise complexe correspond une classe de même nom.

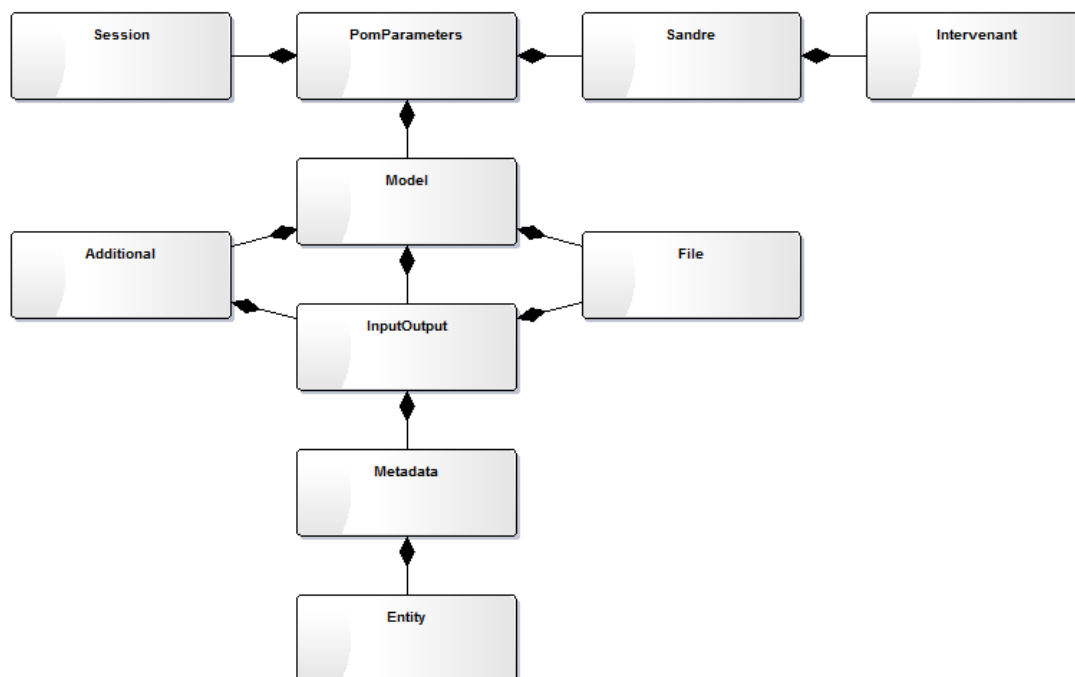


Figure 4 : diagramme de classes des paramètres

La classe PomParameters permet de gérer la lecture du fichier. Elle ne contient pas de fonctionnalités d'écriture du fichier parameters.xml, étant entendu que c'est à la POM de le générer (le PI ne fait que le lire).

3.2.2 Fichier de progression

Le fichier de progression permet au PI d'indiquer son avancement et son statut à la POM. Il doit être généré par le PI dès son lancement, étant attendu que la POM le scrute régulièrement pour déterminer l'état du calcul en cours.

Ce fichier est interfacé par le module « pomprogression.py » du package « pominterface.xmlpom ».

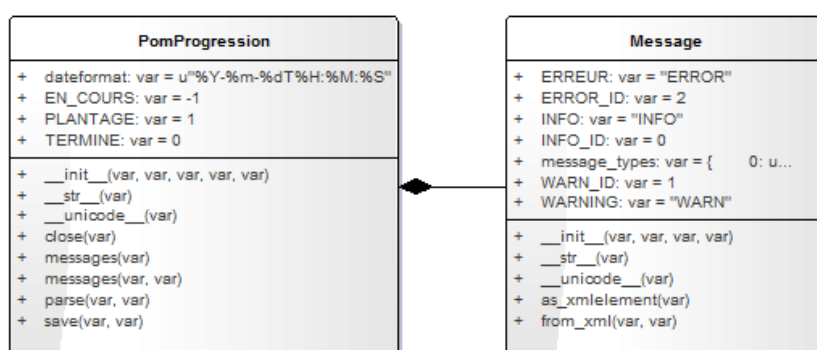


Figure 5 : diagramme de classes de progression

Ce module contient deux classes Python organisées de manière identique à la structure XML du fichier de progression : une classe pour gérer le fichier (et sa balise racine) et une classe pour gérer les balises de message.

Ce module contient des fonctionnalités d'écriture et de lecture (car l'écriture nécessite la lecture préalable du fichier).

3.2.3 Arborescence des fichiers d'échange avec la POM

L'arborescence POM est générée par la POM et autodéscriptive : le fichier parameters.xml référence pour chaque fichier sa localisation exacte. C'est notamment le cas :

- ✓ du fichier de progression
- ✓ des fichiers d'entrée fournis par la POM
- ✓ des fichiers de sortie attendus par la POM

Il faut donc utiliser les informations du fichier parameters.xml pour générer ou lire les fichiers dans cette arborescence. Les fonctions mises à disposition par le PI masquent généralement cette complexité : les fonctions à surcharger pour un Plx sont déjà encapsulées dans les mécanismes de lecture / écriture (sauf cas très particulier).

3.2.4 Mode de calcul POM

Le mode de calcul est paramétré par le mode de calcul du scénario en cours de traitement (renseigné dans le fichier « parameters.xml »). C'est une association de mots clefs (avec leurs valeurs séparées par un « = »), séparés par des espaces.

Pour le PI, il s'écrit comme suit :

```
[series={SERIES}] [inits={INITS}]
[init_files={INIT_FILES}] [init_cotes={INIT_COTES}]
[analyse={ANALYSE}] [modeles={MODELES}]
```

où :

- ✓ le mot clef « series » renseigne les séries à propager (cf. 2.2). Il est facultatif et sa valeur est une liste de noms ou de pourcentages (i.e. un entier suivi de%) séparés par des « | ». Les noms désignent des séries de données qui sont renseignées sur les entrées du fichier « parameters.xml ». Si ce mot clef est absent du mode de calcul, il est pris égal à la valeur paramétrée dans le fichier « ini » (cf. 3.5.5). Exemple :

```
series=MIN|MOY|MAX|75%
```

- ✓ le mot clef « inits » permet de définir les méthodes d'initialisation à lancer consécutivement et dans l'ordre indiqué (cf. 2.1). Il est facultatif. Si ce mot clef est absent du mode de calcul, il est pris égal à la valeur paramétrée dans le fichier « ini » (cf. 3.5.4). Sa valeur est une liste des codes de méthodes séparés par des « | ».

Chaque code ne peut apparaître qu'une fois.

Les valeurs possibles sont : REPRISE, DEFAULT, CONSTANTE, SANS_ANALYSE, INIT_OBS

- ✓ le mot clef « init_files » permet d'imposer les noms de fichiers pour le mode d'initialisation « par défaut », séparés par des « | ». Il n'est utilisé que si le mot clef INIT contient « DEFAULT ». S'il n'est pas renseigné, c'est la valeur du paramètre correspondant du fichier « .ini » qui est utilisée (cf. paramétrage « .ini » au chapitre 3.5.4). Les fichiers indiqués doivent être dans le répertoire de reprise par défaut (clé 'backupdirectory').
- ✓ Le mot clef « init_cotes » contient les valeurs cotes constantes. Il n'est utilisé que si le mot clef INIT contient « CONSTANTE ». Il s'agit de nombres réels séparés par des « | ». S'il n'est pas renseigné, sa valeur est prise égale à la valeur du paramètre correspondant dans le fichier « .ini » (cf. 3.5.4)
- ✓ {ANALYSE} (facultatif) : code d'un des scénarios POM (dit « scénario d'analyse ») présent dans le fichier « parameters.xml » et qui donne lieu à un run d'analyse. Si ce

paramètre n'est pas renseigné, il est pris égal au code du scénario en cours de traitement. Ce mot clef « analyse » n'est disponible que pour les Plx fonctionnant en mode Analyse/Prevision.

- ✓ {MODELES} (facultatif) : liste de codes des modèles à lancer avec les données de ce scénario. Les codes sont séparés par des « | ». Chaque modèle doit correspondre à un répertoire de modèle existant (dans le répertoire des modèles, cf. 3.5.2). Si ce mot clef n'est pas renseigné, il est pris égal à celui du fichier « .ini » (cf. 3.5). Ce mot clef « modeles » n'est disponible que pour les Plx fonctionnant en mode Analyse/Prevision.

3.3 Format des fichiers d'entrée

Le format des fichiers fournis par la POM dépendent de la source de données (i.e. le type de métadonnée) qui les a générés.

3.3.1 XML sandre

Les métadonnées de types suivants sont au format XML Sandre:

- ✓ Observation hydro : des séries d'observation hydro ou météo sont dans le fichier XML Sandre
- ✓ Prévision interne, prévision externe, de sortie : des simulations sont dans le fichier XML Sandre

A partir de la version 3.0 du protocole POM, ces fichiers XML Sandre sont au format Sandre V2.

3.3.2 XML Image

Les métadonnées de types suivants sont au format XML Image (format natif des webservices BdImage) :

- ✓ Image

Depuis la version 2016 de la BdImage, ces fichiers peuvent contenir des observations et des prévisions (alors qu'ils ne contenaient que des observations auparavant).

Les fichiers au format XML Image sont uniquement lus (pour les fichiers d'entrée, produits par la POM) à l'aide de la librairie Python « libimage » (cf. 1.5.4).

Note : les versions antérieures du PI utilisaient un module PI dédié, et non la librairie externe libimage.

3.3.3 JSON Lamedo

Les métadonnées de types suivants sont au format JSON Lamedo (format natif des webservices BdLamedo) :

- ✓ BP

Note : depuis le PI v2.2 et son adaptation à la BDImage 2016, les données Symposium 1 sont obsolètes. Le PI v3 n'implémente plus leur lecture et un message d'erreur le signale.

Ces fichiers ne contiennent que des données prévues.

Ils sont uniquement lus (non générés) à l'aide du module PI « lamedojsonfile.py » du package « pominterface.common »

3.3.4 Autres

Les autres formats de fichiers ne sont pas gérés par la POM, qui les exploite sous forme de métadonnée de type « Fichier », non introspectée.

Le PI ne les gère donc pas non plus nativement mais prévoit des fonctions facilitant leur manipulation.

3.3.5 Format des fichiers de sortie

Les fichiers de sorties produits par le PI sont :

- ✓ Soit des fichiers de données numériques au format XML Sandre (associés à des métadonnées « de sortie »)
- ✓ Soit des fichiers de format inconnu, associés à des métadonnées de type « Fichier »

Le second cas n'est pas pris en charge nativement par le PI, mais il offre des fonctions facilitant leur manipulation.

3.4 Données

Les données lues en entrées ou générées en sortie sont interfacées de manière commune à l'aide de classes du module « hydrodata.py » du package « pominterface.common » :

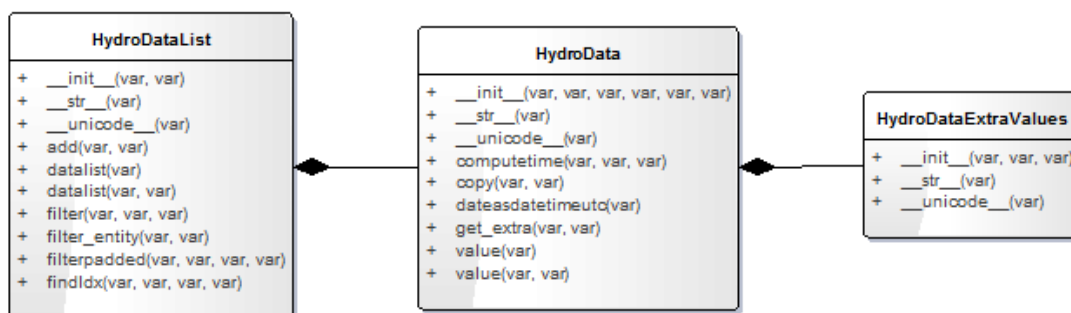


Figure 6 : diagramme de classes de données

Les données sont groupées dans une liste HydroDataList qui dispose de fonctions de recherche / filtre parmi ses données. Chaque donnée est une instance de la classe HydroData caractérisée par :

- ✓ un code entité
- ✓ un code grandeur
- ✓ une date (ainsi qu'éventuellement un temps relatif en secondes par rapport à une date de référence)
- ✓ une valeur (correspondant à la série moyenne le plus souvent)
- ✓ une liste de valeurs complémentaires éventuelles (dites « Extra Values », instances de la classe HydroDataExtraValues) pour stocker les séries de données autres que la moyenne. Chaque « Extra Value » est caractérisée par :
 - ↳ le code de la série associée
 - ↳ la valeur pour cette série

Note : les « Extra Values » sont en fait une association « clef / valeur » qui peut permettre de stocker d'autres informations que les valeurs des autres séries (comme par exemple une date de production, ...).

3.5 Paramétrage du PI

Le PI intègre un fichier de paramètres, au format « .ini ». Il contient le paramétrage général du PI et un paramétrage spécifique au nettoyage de fichiers.

A partir du PI 4.0.0, ce fichier de paramètres peut être au format « .toml ». Toutes les sections et paramètres définis dans la suite de ce document sont identiques dans le « .ini » et dans le « .toml ». Seule la syntaxe du « ;toml » change : elle permet de définir certains types de manière plus rigoureuse et lisible.

Voici un tableau décrivant les différences de syntaxe entre les « .ini » et les « .tom » appliqués aux Plx :

	Version « .ini »	Version «.toml »
booléen	lancementsimultanes = true	lancementsimultanes = true
entier	timeout = 1800	timeout = 1800
numérique	valeur_exemple = 3.14159	valeur_exemple = 3.14159
chaîne	rootdirectory = /home/telemac/pom/Modeles	rootdirectory = "/home/telemac/pom/Modeles"
Liste de chaînes	# 1 seule valeur inits = REPRISE # plusieurs valeurs inits = REPRISE CONSTANTE	# 1 seule valeur inits = ["REPRISE"] # plusieurs valeurs inits = ["REPRISE", "CONSTANTE"] # ou inits = ["REPRISE", "CONSTANTE"]
Liste de num.	# 1 seule valeur init_cotes = 0.4 # plusieurs valeurs init_cotes = 10.0 20.0	# 1 seule valeur init_cotes = [0.4] # plusieurs valeurs init_cotes = [10.0, 20.0] ou init_cotes = [10.0, 20.0

]
Liste multiples	liqfilecolumns = Q9340001;Q Q8390001;Q Q935002001;H	liqfilecolumns = [["Q9340001", "Q"], ["Q8390001", "Q"], ["Q935002001", "H"]]
	telemacnodes = Q935002001;29306 Q935251001;32939	telemacnodes = [["Q935002001", 29306], ["Q935251001", 32939]]

Tableau 3.1 : Comparaison de syntaxe « .ini » et «.toml » pour les PIX

3.5.1 Paramètres généraux

Les paramètres généraux sont définis dans la section [general] :

Section	Paramètre	Format	Obliga toire	Remarque
general	include	Chaîne de caractère	Non	Nom du fichier « .ini » à inclure avant la lecture de celui-ci (cf. 3.5.8)
general	timeout	Entier	Oui	Timout d'exécution du modèle, en secondes. Au delà de cette durée, la commande est interrompue et le modèle est en erreur.
general	rootdirectory	Chaîne de caractère	Oui	Chemin complet du répertoire racine de l'exécutable à lancer (exécutable de la

				plateforme de modélisation). Ce répertoire doit exister. <i>[voir note sous le tableau]</i>
general	commandlinewor kdirectory	Chaîne de caractère	Oui (mais peut être vide)	Chemin du répertoire dans lequel on doit se positionner pour exécuter la ligne de commande. Ce répertoire doit exister <i>[voir note sous le tableau]</i>
general	commandline	Chaîne de caractère	Oui	Commande à lancer dans le répertoire racine. <i>[voir note sous le tableau]</i>
general	lancementsimult anes	Booléen	Non	Active ou désactive la vérification des lancements simultanés.
general	versionprotocole pom	Chaîne de caractère	Oui	Version du protocole POM compatible avec cette version du PI.
general	conversiontable	Chaîne de caractère	Non	Table de conversion des codes d'entités (cf. ci-dessous).
general	workspacedirect ory	Chaîne de caractère	non	Chemin complet du dossier de travail. S'il n'existe pas, le Pix s'arrête. Il ne faut le définir que si ce chemin ne correspond pas au chemin défini par défaut dans les specs des Plx. Exemples : PIT : /home/telemac/workspace PIM : /home/admin/PIM/workspace PIPt : /home/plathynes_pom/PIPt/WORKDIR

Tableau 3.2 : paramètres généraux du PI

Pour les 3 clés 'rootdirectory', 'commandlineworkdirectory', 'commandline', chaque Plx peut les utiliser comme il veut . Par exemple :

- PIT et PIM n'utilisent pas 'rootdirectory',
- PIM suppose que 'commandlineworkdirectory' est relatif au dossier de travail mascaret de chaque run
- PIT a besoin que 'commandline' soit une commande shell (« python /home/telemac/.opentelemac/v7p2r3/scripts/python27/runcode.py telemac2d »
- PIM suppose que 'commandline' soit un exécutable du dossier 'commandlineworkdirectory'

La table de conversion est une série de couples de codes séparés par des « | ». Chaque couple de code est une chaîne de caractères dont les deux codes sont séparés par un « ; ».

3.5.2 Paramètres des modèles

Les paramètres suivants sont lus par le PI uniquement pour les Plx en mode Analyse/Prévision.

Section	Paramètre	Format	Obliga toire	Remarque
models	directory	Chaîne de caractère	Non	Chemin complet du répertoire des modèles. Ce répertoire doit exister. La valeur par défaut est définie par chaque Plx.
models	names	Chaîne de caractère	oui	Code des modèles à lancer, séparés par des « ». Chaque code doit correspondre à un répertoire existant du répertoire des modèles.

Tableau 3.3 : paramètres des modèles du PI

3.5.3 Paramètres de calculs prévision parallèles

Les paramètres suivants sont utilisés par le PI pour la gestion des calculs de prévision en parallèle (cf §4.8.4).

Section	Paramètre	Format	Obliga toire	Remarque
multi_previ	max_nb_calculs_previ	entier	non	Nombre maximum de calculs à lancer par le Plx. La valeur est comprise entre 1 et 1000. La valeur par défaut est « 2 ».
multi_previ	max_cpu_load_percent	entier	non	Maximum de la charge CPU pour pouvoir lancer un calcul. Valeur en % La valeur est comprise entre 10 et

				100. La valeur par défaut est « 80 ».
multi_previ	min_free_mem_mo	entier	non	Minimum de RAM disponible pour pouvoir lancer un calcul. Valeur en Mo. La valeur est comprise entre 1 et 1000. La valeur par défaut est « 150 ».
multi_previ	max_attente_sec	entier	non	Nombre maximum de secondes d'attentes sans lancement de calcul. La valeur est comprise entre 1 et 10000. La valeur par défaut est 180, soit 3 minutes

Tableau 3.4 : paramètres des calculs prévisions du PI

3.5.4 Paramètres de reprise (ou d'initialisation)

Nouveauté PI v3

Certains paramètres permettent de configurer le mode de reprise (ou d'initialisation d'un calcul). Ils sont regroupés dans la clef [relay] :

Section	Paramètre	Format	Obligatoire	Remarque
relay	relaysearchtolerance	Entier	Non	Tolérance de recherche des fichiers de reprise, en secondes. Si elle n'est pas renseignée, cette clef est prise égale à 0
relay	backupdirectory	Chaîne de caractère	Non	Chemin complet du répertoire des fichiers de reprise par défaut. Il ne faut le définir que si les fichiers par défaut sont utilisés, via la clé 'init_files' (paramètres de reprise ou 'mode de calcul' du fichier parameters.xml) La vérification de son existence est faite lors de la vérifications des 'init_files'.
relay	inits	Chaîne de caractère	Non	Codes des méthodes d'initialisation à tester, séparées par des « », parmi REPRISE, DEFAUT, CONSTANTE,

				<p>AUCUNE, INIT_OBS.</p> <p>Le code AUCUNE ne peut apparaître que seul. Un code ne peut pas apparaître deux fois.</p> <p>S'il n'est pas renseigné, ce paramètre est pris égal à AUCUNE.</p>
relay	init_files	Chaîne de caractère	Non	<p>Noms des fichiers du répertoire « backupdirectory » pour le mode d'initialisation « DEFAULT », séparés par des « ».</p> <p>Il n'est utilisé que si le mot clef « inits » contient « DEFAULT ».</p> <p>Cf 3.10</p>
relay	init_cotes	Liste de réels	Non	<p>Contient des nombres réels, séparés par des « ».</p> <p>Il n'est utilisé que si le mot clef « inits » contient « CONSTANTE ».</p> <p>S'il n'est pas renseigné, il est pris égal à « vide » (None).</p>

Tableau 3.5 : paramètres de reprise du PI

Si la clef « inits » n'est pas renseignée, la gestion des méthodes d'initialisation est désactivée. Pour cela, la méthode d'initialisation est prise égale à « AUCUNE ».

3.5.5 Paramètres de propagation des incertitudes

Nouveauté PI v3

Certains paramètres permettent de configurer la propagation des incertitudes. Ils sont regroupés dans la clef [incertitudes] :

Section	Paramètre	Format	Obligation	Remarque
incertitudes	series	Chaîne de caractère	Non	<p>Nom (ou pourcentage) des séries à propager, séparés par des « ».</p> <p>S'il n'est pas renseigné, ce paramètre est pris égal à « MOY ».</p>

Tableau 3.6 : paramètres de propagation des incertitudes du PI

3.5.6 Paramètres de nettoyage

Le PI implémente un processeur « cleanprocessor » (cf. 4.5) chargé du nettoyage des fichiers, exécuté pendant l'initialisation (cf. 4.4). Il se base sur le paramétrage saisi dans le fichier « ini ».

Une première clef « cleaners » de la section « cleaning » référence les noms des sections du fichier « ini » qui contiennent des paramètres de nettoyage. Chacune de ces sections doit être déclarée dans le fichier « ini » avec différents paramètres :

Section	Paramètre	Format	Obligatoire	Remarque
cleaning			Non	La section est facultative. Si elle est renseignée, elle doit contenir la clef de paramétrage ci-dessous.
cleaning	cleaners	Chaîne de caractère	Oui	Liste des noms de section de nettoyage, séparés par des « , »
XXXX			Oui	Nom d'une section référencée dans la liste des « cleaners ».
XXXX	directory	Chaîne de caractère	Oui	Chemin complet du répertoire à traiter.
XXXX	filter	Chaîne de caractère	Non	Filtre des noms de fichiers à traiter.
XXXX	age	Entier	Non	Age maximum (en jour) des fichiers à conserver. Par défaut : 1 jour
XXXX	mode	Chaîne de caractère	Non	Mode de suppression parmi prerun, postrun, pre-postrun Valeur par défaut si non renseigné : prerun

Tableau 3.7 : paramètres de nettoyage du PI

Note : l'exploitation de ce paramétrage est décrit au chapitre 4.5.

Exemple :

```
[CLEANING]
cleaners=cleaner1,nettoyage,purge

# supprime récursivement tous les fichiers et répertoires
# plus vieux que 7 jours dans « c:\grp\echanges\ »
[cleaner1]
directory=c:\grp\echanges\
filter=*
age=7

# supprime récursivement tous les fichiers XML dans « c:\
# grp\echanges\session_tr »
[nettoyage]
directory=c:\grp\echanges\session_tr
```

```
filter=*.xml

# supprime récursivement tous les fichiers et répertoires
dans « c:\grp\modele\RAs\base_TR\tmp »
[purge]
directory=c:\grp\modele\RAs\base_TR\tmp
filter=*
age=0
```

3.5.7 Paramètres d'environnement

Le PI permet de gérer les variables d'environnement du système lors du lancement des exécutables (cf RunProcessor, §4.8.3). Il se base sur le paramétrage saisi dans le fichier « .ini ».

Cela comprend 2 sections « setenv » et « appendenv » qui contiennent des clés suivantes au format texte :

Section	Paramètre	Format	Obliga toire	Remarque
setenv	XXXX	Chaîne de caractère	Non	La section est facultative. Si elle est renseignée, elle doit contenir une ou plusieurs clef de paramétrage
appendenv	XXXX	Chaîne de caractère	Non	La section est facultative. Si elle est renseignée, elle doit contenir une ou plusieurs clef de paramétrage

Tableau 3.8 : paramètres d'environnement du PI

Note : l'exploitation de ce paramétrage est décrit au chapitre 4.8.3.

Exemple :

```
[setenv]
MARINE_DIR = /home/plathynes_pom/modeles/plathynes/CODE/
PLATHYNES_DIR =
/home/plathynes_pom/modeles/plathynes/CODE/
LD_LIBRARY_PATH =
/home/plathynes_pom/modeles/plathynes/CODE/lib/:
/home/plathynes_pom/modeles/plathynes/CODE/bin

[appendenv]
PYTHONPATH = /home/plathynes_pom/modeles/plathynes/CODE/:
/home/plathynes_pom/modeles/plathynes/CODE/src:
/home/plathynes_pom/modeles/plathynes/CODE/lib
```

3.5.8 Héritage de fichiers « .ini »

Le paramétrage du PI permet de faire bénéficier dans un fichier « ini » du contenu d'un autre un fichier « ini ».

Il suffit de référencer la clef « include » de la section « general ». Si cette clef référence un chemin complet vers un fichier existant, ce chemin est utilisé. Sinon, on tente de chercher un fichier « ini » de même nom situé à côté du fichier « ini » en cours de lecture.

Si ce fichier « .ini » n'existe pas, il est ignoré. Sinon, son paramétrage est lu puis celui du fichier « ini » courant. Toutes les clefs de paramétrage déclarées dans le fichier courant viennent écraser les clefs du fichier « include ».

Cela offre l'avantage de pouvoir constituer un fichier « ini » complet dont seulement quelques clefs de paramétrage sont « surchargées » (écrasées) dans le fichier à utiliser.

Exemple de fichier « base » complet (pi-base.ini) :

```
[general]
timeout = 600
lancementsimultanes = false
rootdirectory = /home/modele/bin
...
```

Exemple de fichier spécifique (pi-modele2.ini) :

```
[general]
include = pi-base.ini
lancementsimultanes = true
```

Si on lance le PI avec le second fichier « ini », le paramétrage sera identique à celui déclaré dans le fichier « pi-base.ini » à l'exception du paramètre « lancementsimultanes » qui sera « true ». C'est également très utile pour différencier uniquement certains chemins de répertoires par exemple.

3.5.9 Lecteur de fichier « ini » (classe IniBase)

Cette fonctionnalité est implémentée dans la classe « IniBase » du module « pominterface.common.ini.py ». Cette classe facilite la manipulation des fichiers « .ini » de configuration du PI.

La fonction de lecture du fichier « read() » permet de lire :

- ✓ les options de paramétrage du PI (cf. 3.5)
- ✓ les éventuelles options de paramétrages personnalisées dans les Plx (la fonction est créée non implémentée pour être surchargée dans les Plx).

Mise à part leur caractère obligatoire ou leur format, aucune vérification sémantique n'est faite sur les paramètres, à l'exception de :

- ✓ la table de conversion de codes d'entité : ce paramètre est converti en une instance de la classe « Codestable » du module « pominterface.common.codestable.py » qui facilite l'accès aux couples de codes.
- ✓ les paramètres de nettoyage (cf. 3.5.6) qui sont convertis en instances de la classe « Cleaner » du module « pominterface.common.cleaner.py ». Cette classe dispose en particulier d'une fonction « clean() » qui réalise le nettoyage tel que paramétré (cf. 4.5.2).

3.6 Configuration du PI

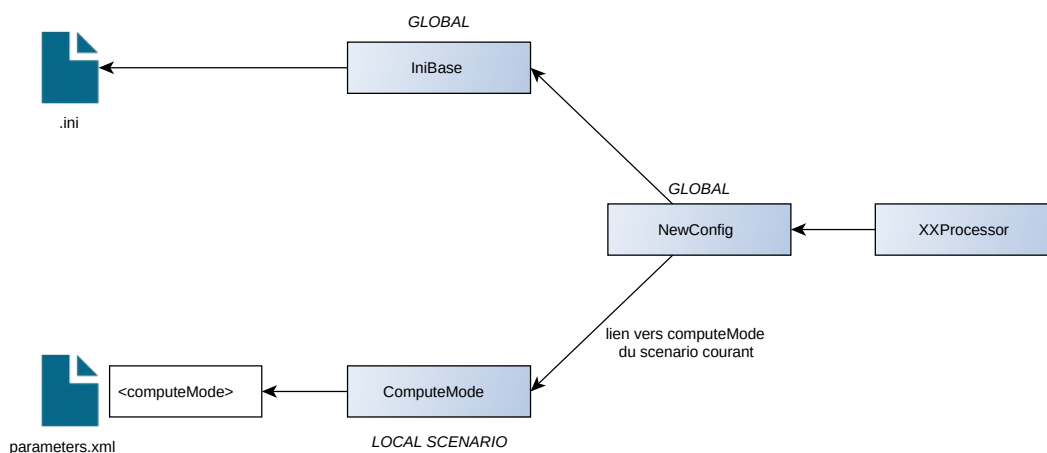
La gestion du paramétrage des calculs pour les Plx est faite par un système de clé/valeur qui peuvent être définies de 2 façons différentes :

- ✓ via un fichier ini
 - ↳ ces valeurs sont globales (ie tous les scénarios, tous les modèles...)
- ✓ via une balise « computemode » dans le fichier « parameters.xml »
 - ↳ ces valeurs sont uniquement utilisées pour le scénario courant

Les valeurs peuvent être définies dans le fichier ini et/ou dans le « computemode ».

Lorsque qu'une valeur est définie dans le fichier ini et le « computemode », c'est la valeur définie dans le « computemode » qui est utilisée.

La gestion du fichier ini et du « computemode » est implémentée par un objet **NewConfig**.



Son rôle est :

- ✓ d'être le seul point d'entrée pour les autres Processeurs
- ✓ de fournir une API simple et unique aux Processeurs pour qu'ils puissent accéder à toutes les configurations (du fichier .ini ou du computemode)
- ✓ de gérer les règles d'écrasement des configurations entre le fichier .ini et le computemode
- ✓ de gérer les computemode (création, changement en fonction du scénario courant)

3.7 Définition du mode de fonctionnement

Le mode de fonctionnement « ONERUN » ou « ANALYSE/PREVISION » est défini au niveau du code, par le choix de hiérarchie des classes :

Le PI va définir des classes spécifiques au mode OneRun et Analyse/Prevision. Chaque Plix devra choisir une de ces classes pour définir son comportement.

Exemple de hiérarchie :

- ModelBase
 - ModelAnalysePrevision
 - ModelPIT
 - ModelPIM
 - ModelPIPt
 - ModelOneRun
 - ModelPIG

3.8 Définition de la méthode d'initialisation pour le mode ANALYSE/PREVISION

Voici l'algorithme implémenté dans NewConfig afin de permettre au PI de déterminer la méthode d'initialisation à utiliser :

1. lors de l'initialisation du Plx, lecture de la clé « inits » dans le fichier ini (cf 3.5.4)
 - ♦ les valeurs autorisées par le PI sont : "AUCUNE|REPRISE|DEFAULT|CONSTANTE|INIT_OBS|SANS_ANALYSE"
 - ♦ la vérification de chaque valeur est effectuée par le Plx
 - ♦ si la clé « inits » est non définie dans fichier ini, c'est la valeur "AUCUNE" qui est utilisée par défaut
2. lors du traitement de chaque scénario, lecture du « computemode » dans le fichier « parameters.xml » (cf 3.2.4)
 - ♦ les valeurs autorisées par le PI sont : "REPRISE|DEFAULT|CONSTANTE|INIT_OBS|SANS_ANALYSE"
 - ♦ la vérification de chaque valeur est effectuée par le Plx
3. fusion des valeurs
 - ♦ il prend la valeur définie dans le « computemode »
 - ♦ si elle n'est pas définie, il prend celle du fichier ini
 - ♦ à cet instant, la méthode d'initialisation peut valoir
 - "REPRISE/DEFAULT/CONSTANTE|INIT_OBS|SANS_ANALYSE"
 - ou autre (définie par Plx)

3.9 Fichiers de reprises

Pour rappel, le dossier de reprise est {WORKSPACEDIRECTORY}/Archives.

Comme les fichiers de reprise sont liés à un seul modèle, le PI ajoute un niveau de rangement : ces fichiers sont donc rangés dans le dossier de leur modèle :

{WORKSPACEDIRECTORY}/Archives/{CODE_MODELE}

Lors de l'archivage d'un calcul, si un fichier de même nom existe déjà, il est simplement remplacé par le nouveau.

3.10 Fichiers défaut

La clé 'init_files' liste les fichiers défaut : il faut que cela corresponde exactement au chemin relatif de chaque fichier dans le dossier {BACKUPDIRECTORY}.

Les fichiers peuvent être directement dans le dossier {BACKUPDIRECTORY} ou dans des sous-dossiers.

Exemple :

```
init_files = fichier1.slf
|{CODE_MODEL1}/fichier2.slf|{CODE_MODEL2}/fichier10.slf
|mon_sous_dossier/sous_dossier2/sous_dossier3/fichier2.slf
```

Pour windows, le caractère « / » est possible ; il n'est pas nécessaire de le remplacer par « \ ».

Pour informations, il est possible de mettre cette clé sous plusieurs lignes pour une meilleure lisibilité :

```
; clé init_files sur plusieurs lignes :
; il suffit d'ajouter au minimum 1 espace en début des lignes suivantes
init_files = fichier1.slf|
  {CODE_MODEL1}/fichier2.slf| ; 4 espaces en début de ligne
  {CODE_MODEL2}/fichier10.slf| ; 4 espaces en début de ligne
  {CODE_MODEL2}/fichier20.slf ; 4 espaces en début de ligne
```

Si la clé 'init_files' n'est pas définie, le PI va lister automatiquement les fichiers du dossier {BACKUPDIRECTORY}, sans prendre en compte les sous-dossiers. Les fichiers sont triés alphabétiquement.

4. Processeurs

Ce chapitre décrit les classes de « processeurs » du PI, chargés chacun d'une étape du calcul.

4.1 Généralités

4.1.1 Exceptions

Des classes d'exception sont déclarées pour cloisonner les causes d'exception du PI. Elles héritent toutes d'une classe commune pour le PI.

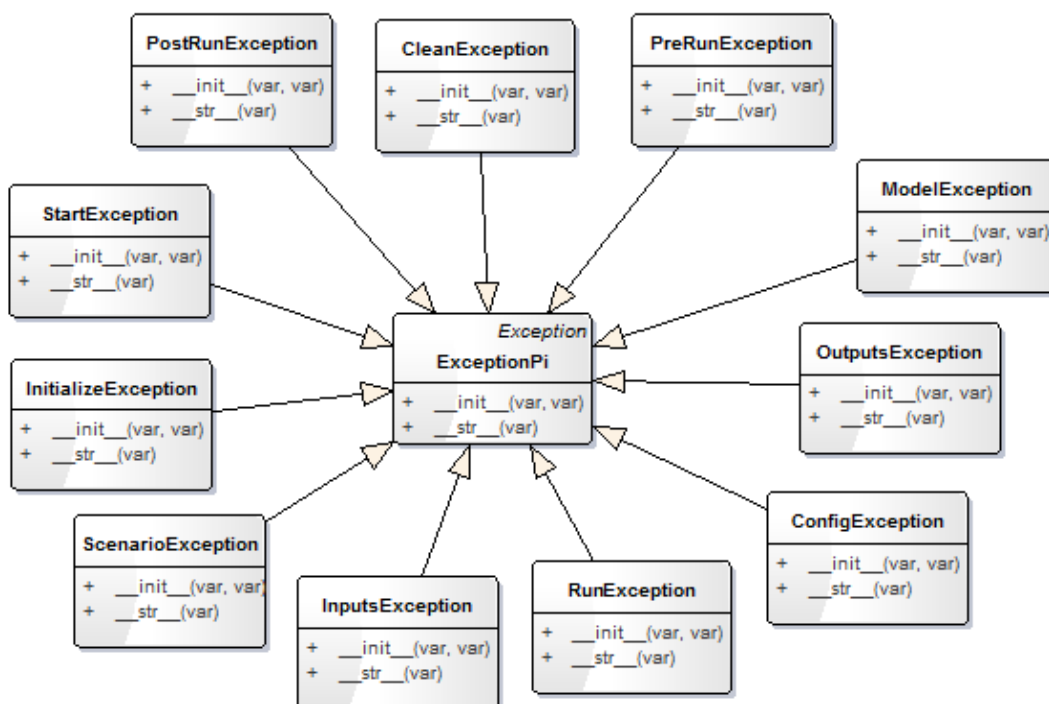


Figure 7 : diagramme de classes des Exceptions

4.1.2 Processeur de base

Toutes les classes de processeurs

- ✓ sont déclarées dans le package « pominterface.processors »
- ✓ héritent d'une même classe « BaseProcessor », et disposent de ce fait
 - ↳ d'un accès au fichier de paramétrage « .ini » (propriété « config »)
 - ↳ de la progression initiale du calcul (au début de l'exécution du processeur)
 - ↳ d'un pas et d'une plage d'avancement de la progression, pour faciliter le calcul de la progression
 - ↳ de fonctions de gestion du verrou (cf. PID, ci-dessous)
 - ↳ d'une méthode « process() » pour lancer l'exécution du processeur (à surcharger dans les classes filles)

↳ d'une méthode « finalize() »

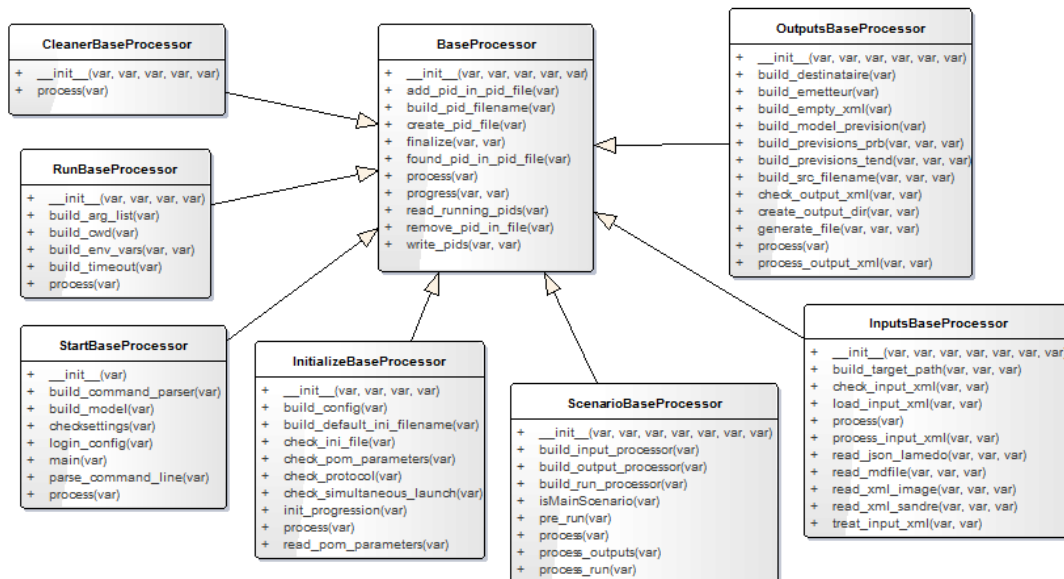


Figure 8 : diagramme de classes de processeurs

4.2 Démarrage (StartProcessor)

Le StartProcessor est la classe de démarrage du PI. Son exécution passe par les étapes suivantes :

- ✓ Analyse de la ligne de commande (cf. ci-dessous)
- ✓ Création du fichier PID de verrou (cf. ci-dessous)
- ✓ Lancement de la procédure principale (fonction « main() »)
- ✓ Suppression dans le fichier de verrou (cf. ci-dessous), même en cas d'erreur
- ✓ Finalisation (mise à jour de la progression et du message de fin dans le fichier de progression), même en cas d'erreur.

4.2.1 Analyse de la ligne de commande

L'analyse de la ligne de commande est réalisée à l'aide du module générique Python « ArgumentParser ».

La ligne de commande :

- ✓ Doit contenir un premier paramètre référençant le chemin complet du fichier parameters.xml
- ✓ Peut contenir un second paramètre référençant le fichier « ini » ou « toml » à utiliser (cf. 3.5)
- ✓ Peut contenir un paramètre optionnel « --force » pour ne pas tenir compte des éventuels verrous

- ✓ Peut contenir un paramètre optionnel « --version » ou « -V » qui permet d'afficher la version du PI et du Plx
- ✓ Peut contenir un paramètre optionnel « --logs » pour générer les fichiers de debug « pix_xxx.log »

Une fois lue, la ligne de commande est analysée :

- ✓ Paramètre parameters.xml
 - ↳ les éventuelles quotes « ' » encadrant le nom du fichier sont supprimées
 - ↳ les éventuels « \ » sont remplacés par des « / »
 - ↳ le chemin ainsi modifié doit pointer vers un fichier existant.
- ✓ Fichier « .ini » ou « toml »
 - ↳ Si ce paramètre est renseigné, il est modifié de la même manière que le paramètre « parameters.xml » et il doit pointer vers un fichier existant

4.2.2 Création du fichier PID de verrou

Le PI est lancé en tant que processus (Windows ou Linux) et dispose donc à ce titre d'un identifiant unique.

Par défaut :

- ✓ le fichier de verrou est positionné dans le répertoire temporaire du serveur de calcul : sous Linux « /tmp »
- ✓ le fichier est nommé « pi.pid ».

Dans chaque Plx, il est possible de définir l'utilisation d'un fichier de verrou autre que le fichier de verrou par défaut.

Au lancement d'un Plx, si l'identifiant du processus courant n'est pas déjà dans le fichier de verrou, il y est ajouté (avec si besoin la création du fichier).

L'implémentation de la gestion du verrou dans le PI est réalisée dans la méthode `has_global_pid()` de la classe `InitializeBaseProcessor`.

4.2.3 Procédure principale « main() »

La procédure principale vise à créer une instance de la classe **ModelBase** (cf. 4.3) et à appeler sa méthode de démarrage « `start()` ».

La classe **ModelBase** a vocation à être surchargée pour s'adapter aux besoins spécifiques des modèles à interfacier dans le Plx.

4.3 Classe ModelBase

La classe **ModelBase** permet de définir les comportements communs à tous les modèles.

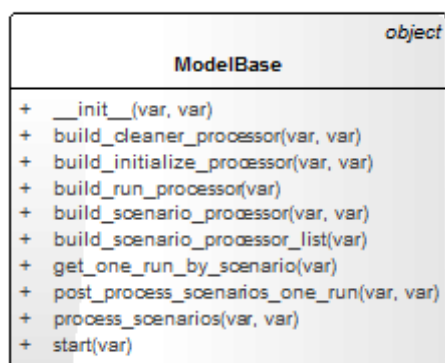


Figure 9 : attributs de la classe ModelBase

La fonction « get_one_run_by_scenario » renvoie « vrai » ou « faux » selon que :

- ✓ le PI doit exécuter un run de la plateforme de modélisation pour chaque scénario (comme Mascaret par exemple)
- ✓ ou au contraire, le PI ne doit exécuter un unique run à la fin du traitement de tous les scénarios (comme GRP par exemple).

Cette information transite vers les processeurs de scénarios afin d'adapter leur comportement.

Dans le PI3, la classe ModelBase est transformée pour être plus modulaire : elle ne gère que les traitements les plus génériques possibles :

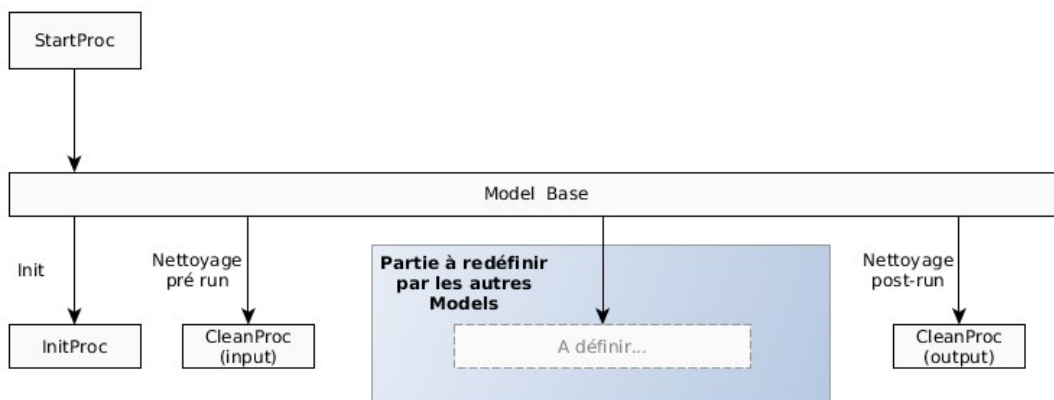


Figure 10 : classe ModelBase

Le PI3 définit de nouvelles classes qui précisent les fonctionnements des « Models » plus spécifiques :

- ✓ **ModelOneRun** (pour GRP)
- ✓ **ModelAnalysePrevision** (pour Telemac, Mascaret et Plathynes)

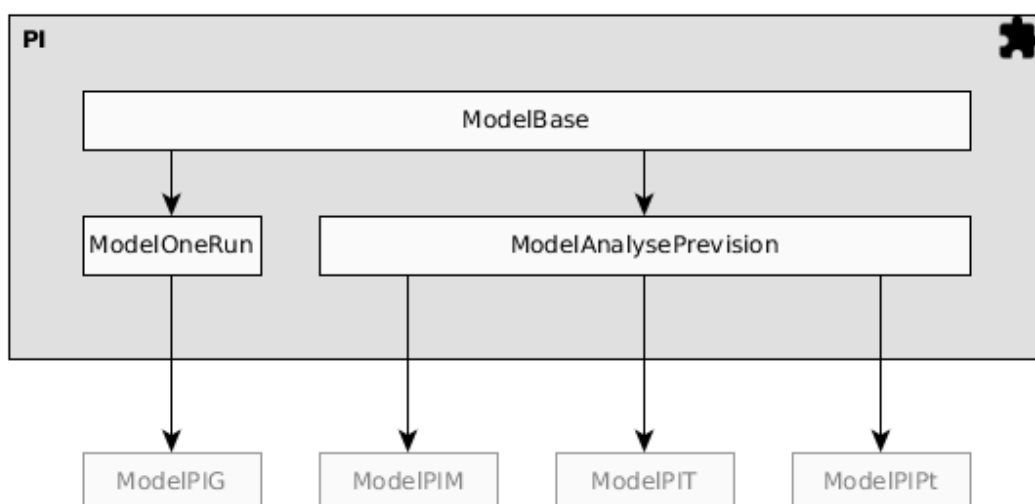


Figure 11 : hiérarchie des classes héritant de ModelBase

4.3.1 ModelAnalysePrevision

Le point d'entrée est la méthode « start » lancée par le StartProcessor.

Elle réalise les actions suivantes :

- ✓ Création et exécution du processeur d'initialisation (cf. 4.4)
- ✓ Création et exécution du processeur de nettoyage « prerun » (cf. 4.5)
- ✓ Création des processeurs de scénario pour l'Analyse (cf. 4.6)
- ✓ Exécution des processeurs de scénario pour le mode Analyse, dans l'ordre (le principal puis les complémentaires dans l'ordre)
- ✓ Création des processeurs de scénario pour la Prévision (cf. 4.6)
- ✓ Exécution des processeurs de scénario pour le mode Prévision, dans l'ordre (le principal puis les complémentaires dans l'ordre)
- ✓ Création et exécution du processeur de nettoyage « postrun » (cf. 4.5)

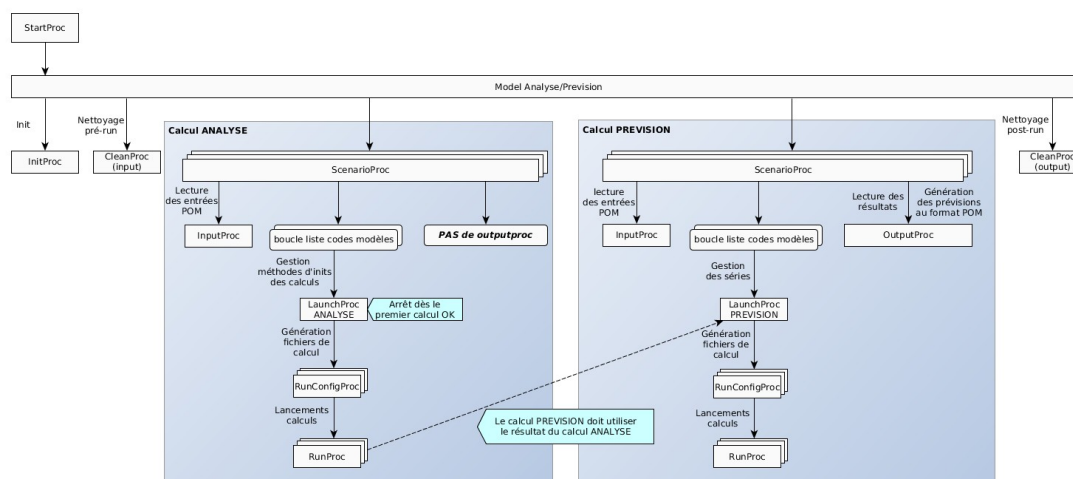


Figure 12 : classe ModelAnalysePrevision

4.3.2 ModelOneRun

Le point d'entrée est la méthode « start » lancée par le StartProcessor.

Elle réalise les actions suivantes :

- ✓ Création et exécution du processeur d'initialisation (cf. 4.4)
- ✓ Création et exécution du processeur de nettoyage « prerun » (cf. 4.5)
- ✓ Création des processeurs de scénario
- ✓ Exécution des processeurs de scénario (lecture des inputs et la préparation du calcul), dans l'ordre (le principal puis les complémentaires dans l'ordre)
- ✓ Création et exécution d'un seul processeur de lancement de runs LaunchProcessor (cf. 4.8)
- ✓ Exécution des processeurs de scénario (lecture des résultats et génération des fichiers pour la POM), dans l'ordre (le principal puis les complémentaires dans l'ordre)
- ✓ Création et exécution du processeur de nettoyage « postrun » (cf. 4.5)

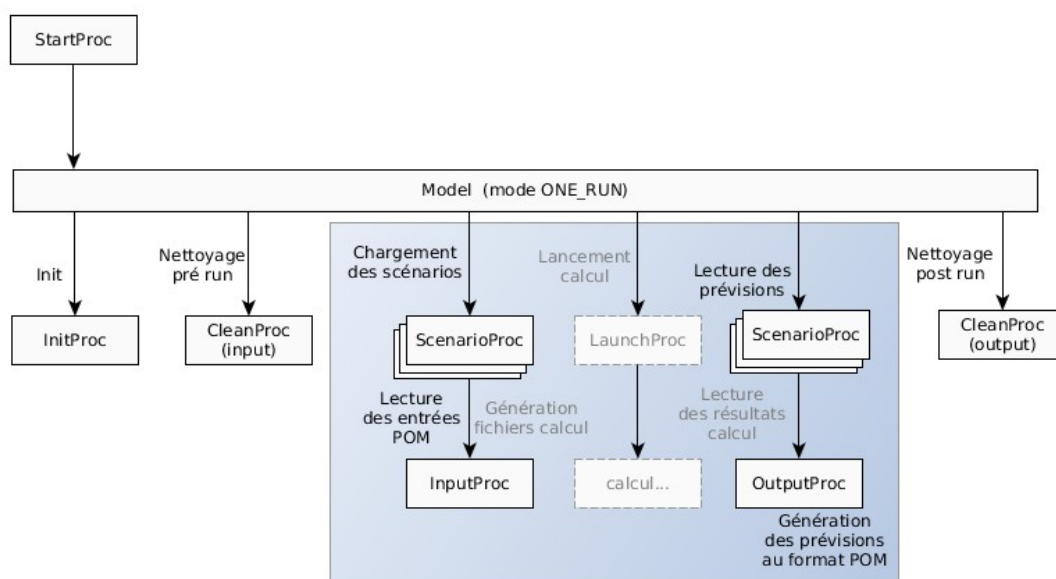


Figure 13 : classe ModelOneRun

4.3.3 Gestion du statut du calcul (OK/Erreur)

La classe ModelBase gère le calcul du status (OK/erreur) à renvoyer à la POM via le fichier 'progression.xml'.

Elle effectue ce traitement en 2 parties :

- ✓ elle récupère les statistiques de génération des outputs :
 - ↳ pour le ModelAnalysePrevision : après le process de chaque scénario, cela ne concerne que les ScénarioPrevision
 - ↳ pour le ModelOneRun : après le process des outputs de chaque scénario
 - ↳ le nombre total d'outputs attendus comprend tous les outputs de tous les scénarios du fichier 'parameters.xml'
 - ↳ le nombre total d'outputs générés comprend tous les fichiers de sortie effectivement générés de tous les scénarios du fichier 'parameters.xml'
- ✓ après l'exécution du processeur de nettoyage « postrun » : en fonction de l'écart entre le nombre total d'outputs générés et le nombre total d'outputs attendus, le contenu du fichier 'progression.xml' est mis à jour :
 - ↳ aucun fichier généré : le statut du calcul est en erreur ; le fichier 'progression.xml' contient le message d'erreur "Aucun fichier output généré"
 - ↳ tous les fichiers générés : le statut du calcul est ok ; le fichier 'progression.xml' ne contient aucun message sur ce sujet
 - ↳ certains fichiers générés : le statut du calcul est ok ; le fichier 'progression.xml' contient un message de warning : "{} fichiers de sortie n'ont pas été générés", suivi de la liste des fichiers manquants, regroupés par code scénario

4.3.4 Gestion de fin de traitement

Si le fichier de verrou est géré par le PI de manière globale, l'identifiant du processus en cours est supprimé du fichier de verrou.

Si le fichier de verrou n'est pas global, c'est chaque Plx qui a la responsabilité de gérer la mise à jour du fichier de verrou.

4.4 Initialisation (InitializeBaseProcessor)

Le processeur d'initialisation vise à vérifier le paramétrage de l'exécution et indiquer à la POM que l'exécution est en cours. Pour cela, les étapes suivantes sont réalisées :

- ✓ Lecture du fichier parameters.xml
- ✓ Création et initialisation du fichier de progression
- ✓ Vérification du contenu du fichier parameters.xml
 - ↳ Cette fonction existe mais ne fait rien dans le PI. Elle a pour objectif une éventuelle surcharge dans les Plx pour des vérifications spécifiques (sur le nombre de métadonnées « Fichier » par exemple ...)
- ✓ Vérification du contenu du fichier « .ini »
 - ↳ Création de l'objet de lecture du fichier « .ini » (cf. 3.5.9) et lecture du fichier :
 - Si le chemin complet de ce fichier n'est pas renseigné en paramètre de la ligne de commande de lancement du Plx, le Plx définit un nom de fichier par défaut (via la méthode `pix_build_default_ini_filename()`) qui doit exister dans le répertoire racine du Plx. S'il n'en existe pas, une erreur est levée et le calcul s'arrête.
 - Si ce chemin du fichier « ini » ne correspond pas à un fichier, une erreur est levée et le calcul s'arrête
 - Toute erreur de lecture du fichier « ini » interrompt le calcul.
 - ↳ Vérification du protocole POM : le numéro de version du protocole POM indiqué dans le fichier « parameters.xml » doit correspondre au numéro renseigné dans le fichier « .ini », sinon une erreur bloquante est levée.
 - ↳ Vérification des lancements simultanés (cf. 4.4.1)
 - ↳ Création des différentes méthodes d'initialisation en fonction du paramétrage du fichier « .ini » (cf. 4.4.2)

4.4.1 Vérification des lancements simultanés

La vérification des lancements simultanés est réalisée par le PI, si les conditions sont suivantes sont vraies :

- ✓ le fichier de paramétrage « ini » (cf. 3.5.1) indique qu'il faut effectuer la vérification,
- ✓ le fichier de verrou est géré par le PI.

Dans ce cas, on cherche si un autre PI est en cours d'exécution. Pour cela, on inspecte le fichier de verrou (cf. 4.2.2). S'il existe :

- ✓ on liste les identifiants de processus (PID) qu'il contient
- ✓ si la liste contient d'autres PID que le PID courant, une erreur bloquante est levée.

L'erreur bloquante peut ne pas être activée, si l'option « force » est à « vrai » (cf. 4.2.1). Dans ce cas, un message indique que le lancement aurait du être interrompu mais qu'il a été forcé. Sinon, si l'option « force » est à « faux », une erreur bloquante stoppe l'exécution.

Si le fichier de verrou n'est pas global, donc géré par le Plx, la vérification des lancements simultanés doit être effectuée par le Plx.

4.4.2 Création des méthodes de reprise

Cette étape vise à préparer la liste des méthodes de reprise / initialisation du calcul (cf. PI).

Pour chacune des valeurs de la clef « inits », selon son type, une ou plusieurs méthodes sont ajoutées à la liste des méthodes à tester :

- ✓ REPRISE : dans le cas d'un lancement personnalisé POM « avec initialisation », ce mot clef est « sauté » (non pris en compte). Sinon, on ajoute cet élément à la liste des méthodes à tester
- ✓ DEFAUT : pour chaque valeur de la clef « init_files » du fichier « .ini » (cf. 3.5.4 et cf. 3.10), une méthode est ajoutée à la liste des méthodes à tester.
- ✓ CONSTANTE : pour chaque valeur de la clef « init_cotes » du fichier « .ini » (cf. 3.5.4), une méthode est ajoutée à la liste des méthodes à tester.
- ✓ INIT_OBS
- ✓ SANS_ANALYSE

4.5 Nettoyage (CleanerBaseProcessor)

4.5.1 Processeur

Le processeur de nettoyage est paramétré lors de sa création pour traiter le nettoyage « pré run » ou « poste run ». Chaque processeur de nettoyage est donc instancié (et lancé) deux fois par le ModelBase : une fois avant le run de la plateforme de modélisation, une fois après.

Il lui revient de déterminer s'il doit, ou non, procéder au nettoyage, en fonction du paramétrage des Cleaners du fichier « ini ».

Note : pour rappel, les « Cleaners » sont créés à la lecture du fichier « ini » à partir du paramétrage de celui-ci (cf. 3.5.9).

Pour cela, il boucle sur les différents Cleaners et lance leur méthode « clean » en précisant la phase de nettoyage (pré ou post run).

4.5.2 Cleaner

Un Cleaner est une instance de la classe Cleaner, du module « cleaner.py » du package « pominterface.common ». Il est caractérisé par :

- ✓ un nom
- ✓ un répertoire
- ✓ un filtre (optionnel)
- ✓ un age (optionnel)
- ✓ un mode :pré ou post run (optionnel)
- ✓ un mode forcé (optionnel)

Il est créé lors de la lecture du fichier « .ini » à partir du paramétrage des différents cleaners (cf. 3.5.6). Il s'exécute grâce à la méthode « clean() » qui est paramétrée avec le mode de

nettoyage à effectuer (pré ou post run). Si le mode du Cleaner est différent du mode courant (de la fonction clean), rien n'est fait . Sinon, le nettoyage est réalisé :

- ✓ Vérification de l'existence du répertoire. S'il n'existe pas, une erreur bloquante stoppe le nettoyage. Sinon, on parcourt tous les sous répertoires et fichiers contenus, et pour chacun :
 - ↳ on vérifie si le nom du fichier correspond au filtre du cleaner s'il est renseigné. Si oui, il est considéré comme « à supprimer ». Si le filtre n'est pas renseigné, le fichier est également considéré comme « à supprimer ».
 - ↳ si le fichier est « à supprimer », on vérifie ensuite son âge : il doit être inférieur à l'âge paramétré pour le cleaner. Si l'âge n'est pas renseigné sur le cleaner ou que le fichier est trop âgé, il est considéré comme « à supprimer ».
 - ↳ si le fichier est « à supprimer » il est supprimé.
 - ↳ Si au moins un fichier a été supprimé, on vérifie s'il y a des répertoires désormais vides à supprimer également. On boucle donc sur tous les sous répertoires et on supprimer ceux qui ne contiennent plus de fichiers.

4.6 Scénario (ScenarioBaseProcessor)

4.6.1 Création

Le processeur de scénario traite un seul scénario : le principal (sans numéro) ou un scénario complémentaire, référencé par son numéro. Les informations de ce scénario sont lues dans le fichier parameters.xml.

Lors de sa création, le processeur de scénario est paramétré avec :

- ✓ son numéro (cf. ci-avant)
- ✓ son type de run

Plusieurs variables sont également calculées à la création :

- ✓ profondeur minimale des entrées du scénario
- ✓ profondeur minimale des entrées 'observation' du scénario
- ✓ profondeur maximale des sorties du scénario
- ✓ échéance maximale des sorties du scénario
- ✓ le temps de base
- ✓ les temps T0, T1 (non recalé), T2

Enfin, la création donne également lieu à la création du processeur d'entrées (cf. 4.7).

4.6.2 Recherche scénario d'analyse (uniquement pour ScenarioAnalysePrevision)

Pour les runs de prévision, ce processeur implémente aussi la recherche standard de scénario d'analyse, via la méthode `pix_get_scenario_analyse()`.

Le résultat du run d'analyse est cherché par ordre de priorité :

- ✓ sur le scénario éventuellement référencé dans le mode de calcul (cf. 3.2.4 - Mode de calcul POM)

- ✓ sur le scénario courant s'il a donné lieu à un run d'analyse
- ✓ sinon sur le scénario principal

Si aucun run d'analyse ne correspond, une erreur interrompt le calcul.

Cette méthode permet aux Plx de compléter le test sur chaque scénario (pour l'instant, cela est utilisé uniquement par le PIM)

4.6.3 Exécution

4.6.3.1 ScenarioAnalysePrevision

L'exécution du processeur de scénario passe par les étapes suivantes :

- ✓ Exécution du processeur des entrées
- ✓ Appel à la fonction « prerun() » : cette fonction peut être surchargée dans les Plx pour les traitements qui précèdent le lancement d'un run. Elle effectue les traitements suivants :
 - ↳ calcul des temps recalés
 - ↳ Vérification des modèles par rapport au scénario courant (cf 4.6.3.3)
- ✓ Pour chaque modèle à gérer, création et lancement du processeur de lancement de runs LaunchProcessor (cf. 4.8)
- ✓ Traitement des sorties (cf. 4.6.4)

4.6.3.2 ScenarioOneRun

L'exécution du processeur de scénario passe par les étapes suivantes :

- ✓ Exécution du processeur des entrées
- ✓ Appel à la fonction « pix_pre_run() » : cette fonction ne fait rien dans le PI mais a vocation à être surchargée dans les Plx pour les traitements qui précèdent le lancement d'un run

Le traitement des sorties (cf. 4.6.4) n'est pas inclus dans l'exécution du processeur, mais est appelé directement par le ModelOneRun (cf 4.3.2).

4.6.3.3 Vérifications pour AnalysePrevision

Cette partie est utilisée uniquement par le PIPT 3 pour la gestion multi-modèles. Mais il est pertinent de gérer ces vérifications de manière générique.

Dans les paragraphes suivants, les termes spécifiques au PIPT ont été remplacés par des termes génériques :

- ✓ le terme « modèle Plathynes » a été remplacé par « modèle de bassin »
- ✓ le terme « PIPT3 » a été remplacé par « Plx »

Pour le Plx, il s'agit de rajouter des contrôles afin de se prémunir de conflits entre les sorties de chaque modèle de bassin.

Pour les différents cas possibles, les contrôles à effectuer sont les suivants :

- ✓ activation du Plx, avec un scénario POM, et un modèle de bassin, le Plx doit contrôler que :
 - ↳ les codes de sortie du modèle de bassin sont bien des codes de station attendues en sortie par la POM

- ✓ activation du Plx, avec un scénario POM, et plusieurs modèles de bassin, le Plx doit contrôler que :
 - ↳ les codes de sortie de ces modèles de bassin n'entrent pas en conflit (chaque code de sortie est dans un seul modèle de bassin)
 - ↳ les codes de sortie de ces modèles de bassin sont bien des codes de stations attendues en sortie par la POM

- ✓ activation du Plx avec plusieurs scénarios POM, et pour chaque scénario, un ou des modèles de bassin, le Plx doit contrôler que :
 - ↳ pour chaque scénario, si plusieurs modèles de bassin, les codes de sortie de ces modèles de bassin n'entrent pas en conflit (chaque code de sortie est dans un seul modèle de bassin),
 - ↳ les codes de sortie de ces modèles de bassin sont bien des codes de stations attendues par la POM

Si une des conditions ci-dessus n'est pas vérifiée, les traitements sont interrompus et un message d'erreur trace l'erreur.

Ces vérifications sont effectuées dans `ScenarionAnalysePrevision`.

4.6.4 Traitement des sorties

Le traitement de sorties consiste en la création du processeur des sorties (cf. 4.9) et son lancement.

Puis le Scenario va stocker les statistiques du processeur des sorties (variables "nb_outputs_expected", "nb_outputs_generated", "missing_outputs").

4.7 Entrées (InputsBaseProcessor)

4.7.1 Création

Le processeur d'entrée est chargé de lire les données d'entrées d'un scénario. Il est donc paramétré avec le scénario auquel il est associé (paramètres du fichier parameters.xml et numéro du scénario).

Lors de sa création, le processeur initialise une liste de données vide (cf. 3.4) qu'il va enrichir lors de la lecture des différents fichiers d'entrée. Cette liste contient à la fin toutes les données lues du scénario.

Le processeur lit toutes les données (toutes les séries), indépendamment de leur utilisation ultérieure (propagation des incertitudes ou pas).

Contrairement aux précédentes versions, le processeur ne lit les données que du scénario courant, il ne complète pas avec les données du scénario principal.

4.7.2 Exécution

Le processeur boucle sur toutes les entrées du fichier parameters.xml et les traite une à une :

- ✓ Vérification de la cohérence du paramétrage de cette entrée
 - ↳ Existence de la métadonnée
 - ↳ Existence du fichier associé
- ✓ Lecture et chargement des données du fichier associé, selon le type de métadonnée (cf. 4.7.3)
- ✓ Détermination du pas de temps de cette entrée
- ✓ Traitements sur cette entrée (cf. 4.7.6)

4.7.3 Lecture des fichiers

Selon le type de métadonnée, la méthode de chargement est spécifique :

- ✓ MDOBSBDH, MDPREVINT, MDPREVEXT, MDOUTPUT : format XML Sandre
- ✓ MDBDIMAGE : format XML Image
- ✓ MBDLAMEDOBP, MBDLAMEDOSYMPO : format JSON Lamedo
- ✓ MDFICHER : format inconnu
- ✓ Autre : erreur bloquante.

4.7.3.1 XML Sandre

Le format XML Sandre est lu à l'aide de la librairie Libhydro (cf. 1.5.3). Les données sont chargées dans une structure commune de liste de données (cf. 3.4).

Les formats Sandre V1.1 et Sandre V2 sont autorisés.

Pour le format Sandre V1.1 :

- ✓ les balises «Series» (données hydro observées) et «ObssMeteo» (données météo observées) sont lues. Les données lues sont chargées dans la série « MOY ».
- ✓ les balises « Simuls » sont lues. Elles contiennent séries « MOY », « MIN », « MAX » et « proba » sont chargées.

En format Sandre V2 :

- ✓ les balises «SeriesObsHydro» (données hydro observées) et «SeriesObsMeteo» (données météo observées) sont lues. Les données lues sont chargées dans la série « MOY ».
- ✓ les balises « Simuls » sont lues. Elles contiennent des prévisions déterministes, des prévisions de tendance, des probas.
 - ↳ Pour les prévisions déterministe, la seule série est « MOY »,

- ↳ Pour les prévisions d'ensemble, les séries sont « MOY », « MIN », « MAX »
- ↳ Pour les probas et les prévisions d'ensemble les valeurs sont chargées.

4.7.3.2 XML Image

En v2.1, le format XML Image est lu à l'aide de la librairie Libimage (cf. 1.5.4).

Note : auparavant, un module spécifique au PI permettait de lire les fichiers XML Image.

Les données sont chargées dans une structure commune de liste de données (cf. 3.4).

Les informations suivantes sont également ajoutées à chaque valeur :

- ✓ « prévision » ou « observation » (vrai ou faux pour chacun)
- ✓ durée de l'image
- ✓ valeurs des pixels
- ✓ autres statistiques que « MOY »

4.7.3.3 JSON Lamedo

Le format JSON Lamedo est lu à l'aide de la classe LamedoJsonFile du module « pominterface.common.lamedojsonfile.py ». Les données sont chargées dans une structure commune de liste de données (cf. 3.4).

Les séries « MOY », « INCMOY », « LOC », « INCLOC » sont chargées dans chaque valeur.

4.7.3.4 Fichier / format inconnu

La lecture des formats spécifique est du ressort de spécificités des Plx.

Le PI ne fait que créer une fonction, non implémentée, chargée de construire le nom du fichier cible où copier le fichier source fourni par la POM.

4.7.4 Détermination du pas de temps de l'entrée

Le PI détermine le pas de temps de chaque donnée d'entrée, ainsi le pas de temps de chaque données d'entrée est disponible dans les Plx.

Pour cela :

- ✓ la classe **HydroData** a un attribut dédié au pas de temps
- ✓ dans la classe **InputsBaseProcessor**, les fonctions de lecture (read_xml_sandre, read_xml_image, read_json_lamedo) déterminent le pas de temps pour chaque type de données d'entrée
- ✓ le pas de temps est calculé en faisant la différence entre les dates des 2 premières valeurs de la donnée. Si il y a moins de 2 valeurs dans la donnée, le pas de temps est positionné à 0.
- ✓ la classe **HydroDataList** dispose des méthodes suivantes permettant :
 - ↳ d'avoir la liste des pas de temps d'une liste de **Hydrodata**
 - ↳ de filtrer une liste de **Hydrodata** par un pas de temps

4.7.5 Règles d'ajout des données

4.7.5.1 Détermination du type OBS ou PREV

La distinction données observées/données prévues est effectuée à partir des informations de l'input correspondant dans le fichier « parameters.xml », via la balise « metadataTypeData ».

Les données de type MDPREVINT, MDPREEXT, MDOUTPUT, MDBDLAMEDOBP, MDBDLAMEDOSYMPO sont des données prévues.

L'implémentation de ces propriétés est effectuée dans les fonctions isObs() et isPrev() de la classe InputOutput du module pomparameters.

4.7.5.2 Gestion du chevauchement des données

Lorsqu'il existe déjà un hydrodata pour la même (entité, grandeur, date), en fonction du type de la donnée déjà présente et de la donnée à ajouter, voici les règles permettant de définir quelle donnée doit être utilisée :

Type de donnée	Type de la nouvelle donnée à ajouter	Action
OBS	OBS	
OBS	PREV	Utilisation de la donnée existante
PREV	OBS	Utilisation de la nouvelle donnée
PREV	PREV	

Tableau 4.1 : règles de chevauchement des données

La problématique est la même suivant la nature des données « spatialisée » ou « non spatialisée ». Si pour une même (entité, grandeur, date), il existe des données de même type (« observées » ou « prévisions ») mais de nature différente, une erreur bloquante sera générée, car nous estimons que c'est une erreur de modélisation dans la POM.

Note : La gestion du chevauchement peut être réalisée dans HydrodataList.add().

4.7.5.3 Métadonnée Sympo (MDBDLAMEDOSYMPO)

Ce type de métadonnée ne doit plus être utilisé. Le PIPT lève un message d'erreur (bloquant) pour chaque entrée de ce type :

La ressource {NOM} est associée à la métadonnée {NOM} de type 'Symposium'. Ce type de métadonnée ne doit plus être utilisé, pensez à modifier votre paramétrage POM pour utiliser les métadonnées Image correspondantes.

4.7.5.4 Métadonnée de sortie (MDOUTPUT)

Ce type de métadonnée ne doit pas être fourni une donnée d'entrée.

Si une donnée d'entrée est de type MDOUTPUT, une erreur bloquante est générée.

4.7.6 Traitement sur l'entrée (conversion de codes)

Si le fichier « .ini » (cf. 3.5.1) contient une table de conversion de codes, le PI traite les entrées de manière à modifier ou dupliquer les codes des entités de la table de conversion.

Pour cela, on va créer une nouvelle liste de données, vide initialement, que l'on va progressivement remplir avec les données véritablement lues dans les fichiers. Pour chaque donnée lue dans les fichiers d'entrée :

- ✓ Si elle concerne une entité dont le code n'est pas dans la table de conversion, la donnée est ajoutée telle qu'elle dans la liste de données résultat
- ✓ Sinon, on clone cette donnée et on change son code entité par le code entité correspondant dans la table de conversion puis on l'ajoute à la liste de données résultat.

La liste résultat constitue la nouvelle liste de données d'entrées.

4.8 Gestion des Runs

Le PI v3 réorganise le lancement du calcul : 2 nouvelles classes s'ajoutent au RunBaseProcessor existant : LaunchProcessor, RunConfigProcessor.

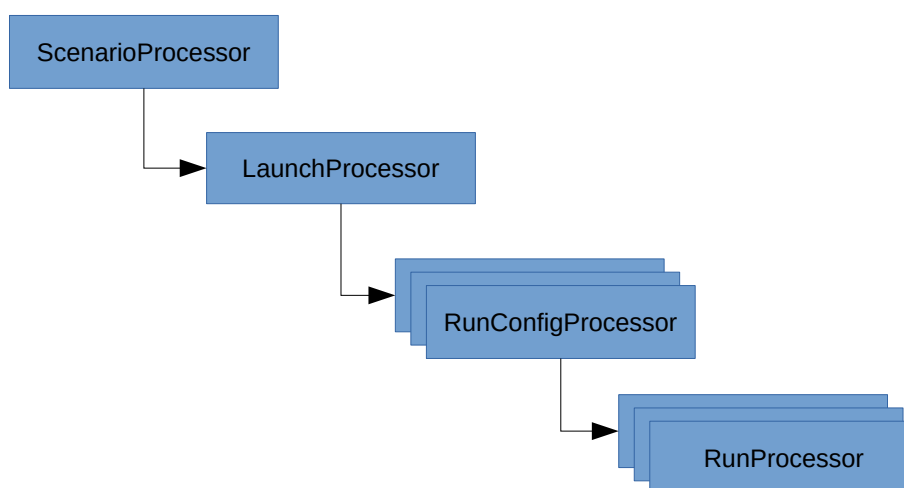


Figure 14 : classes utilisées pour la gestion des runs

Chaque scénario créé 1 **LaunchProcessor**.

Chaque **LaunchProcessor** créé 1 ou plusieurs **RunConfigProcessor**

Chaque **RunConfigProcessor** créé 1 ou plusieurs **RunProcessor**.

4.8.1 LaunchProcessor

Le LaunchProcessor va avoir un comportement spécial en fonction du mode : Analyse, Prévission :

- ✓ Analyse : lancement d'un RunConfig pour chaque méthode d'initialisation, en testant toutes les méthodes d'initialisation les unes à la suite des autres, jusqu'à ce que l'une d'entre elles fonctionne
- ✓ Prévission : lancement d'un RunConfig par série de données à propager

4.8.2 RunConfigProcessor

Le RunConfigProcessor va préparer tout l'environnement nécessaire au calcul puis lancer le calcul.

4.8.2.1 Mode Unique

Il s'agit du mode par défaut : le RunConfig prépare et lance un seul calcul.

Le RunConfig effectue les étapes suivantes :

- ✓ Constitution du jeu de données nécessaire au run (filtre sur les données d'entrées), comme indiqué au chapitre 2.2
 - ↳ analyse : données observées uniquement
 - ↳ prévission : cf. 2.2
- ✓ Copie des fichiers de référence nécessaires à la plateforme de modélisation. Cette fonction est créée, non implémentée en vue d'être surchargée dans les Plx.
- ✓ Paramétrage de la plateforme de modélisation, i.e. mise à jour des fichiers de configuration de la plateforme. Cette fonction est créée, non implémentée en vue d'être surchargée dans les Plx.
- ✓ Création et exécution du RunProcessor

4.8.2.2 Mode Multiple

Il s'agit d'un mode spécial (pour le PIM actuellement) qui permet d'enchaîner plusieurs calculs, chaque résultat d'un calcul servant d'entrée au calcul suivant. A la fin, seul le résultat du dernier calcul est pris en compte.

Le RunConfig effectue les étapes suivantes :

- ✓ boucle sur les runs à traiter
 - ✓ si premier run, constitution du jeu de données nécessaire au run (filtre sur les données d'entrées), comme indiqué au chapitre 2.2
 - ↳ analyse : données observées uniquement
 - ↳ prévision : cf. 2.2
 - ✓ sinon, reprise du résultat du run précédent
- ✓ Copie des fichiers de référence nécessaires à la plateforme de modélisation. Cette fonction est créée, non implémentée en vue d'être surchargée dans les Plx.
- ✓ Paramétrage de la plateforme de modélisation, i.e. mise à jour des fichiers de configuration de la plateforme. Cette fonction est créée, non implémentée en vue d'être surchargée dans les Plx.
- ✓ Création et exécution du RunProcessor
- ✓ si dernier run, gestion standard du résultat/reprise
- ✓ sinon, stockage du résultat/reprise pour run suivant

4.8.3 RunProcessor

Le RunBaseProcessor est chargé de lancer la commande du calcul (sur la plateforme de modélisation) proprement dit et d'attendre son exécution.

- ✓ Préparation du lancement à l'aide du fichier « .ini »
 - ↳ mise en place de l'environnement système avec la section « [setenv] » :
 - chaque clé de cette section est ajoutée dans l'environnement sous la forme
 - CLE=VALEUR
 - ↳ mise en place de l'environnement système avec la section « [appendenv] » :
 - chaque clé de cette section est ajoutée dans l'environnement sous la forme
 - CLE=VALEUR:\$CLE
 - le PI ne vérifie pas si la variable CLE existe déjà dans l'environnement
- ✓ Lancement du processus, à l'aide du paramétrage du fichier « .ini » (comme en PI v2), et attente de son exécution (ou du timeout). Les sorties standard et d'erreur de cette commande sont redirigées vers des fichiers temporaires.
- ✓ Vérification de la bonne exécution du run
 - ↳ Cette étape vise à s'assurer que le run est allé au bout sans erreur, lorsque le retour de la ligne de commande est 0 (retour sans erreur) ou après analyse des fichiers temporaires des sorties standards. La fonction est déclarée, non implémentée, en vue d'être surchargée dans les Plx (exemple : pour analyser le fichier de log du calcul)
- ✓ Gestion des éventuelles erreurs d'exécution du run

- ↳ Cette étape vise à remonter à l'utilisateur des informations pertinentes sur la cause de l'erreur. La fonction est déclarée, non implémentée, en vue d'être surchargée dans les Plx (exemple : pour lire le fichier de log du calcul)

4.8.4 Gestion des calculs prévision en parallèle

Le PI permet de lancer les calculs prévision en parallèle.

Le PI utilise les options suivantes , définies dans le fichier « .ini » :

- ✓ section « multi_previ »
 - ↳ option « max_nb_calculs_previ »
 - ↳ option « max_cpu_load_percent »
 - ↳ option « min_free_mem_mo »
 - ↳ option « max_attente_sec »

4.8.4.1 Principe général

Pour les calculs prévision, le PI exécute les 3 étapes distinctes suivantes :

- ✓ Préparation de tous les calculs

Le PI prépare tous les dossiers de calculs de tous les calculs prévision.
- ✓ Lancement et gestion des calculs

Le PI lance et gère les calculs en parallèle, suivant le principe « ordonnanceur », cf §4.8.4.2.

Le PI attend que tous les calculs soient terminés (sans erreur ou en erreur ou en « timeout »).
- ✓ Lecture de tous les résultats

Le PI lit tous les résultats, dans tous les dossiers de calculs.

Si la plateforme d'exécution du PI est Windows, il n'y a pas de parallélisme des calculs de prévision.

4.8.4.2 Principe de l'ordonnanceur

Le principe que nous retenons consiste à implémenter dans le PI un mécanisme « ordonnanceur » (« scheduler » en anglais), dont les principes généraux sont :

- ➔ pour s'initialiser l'ordonnanceur doit avoir la liste des calculs de prévision à lancer. Cette liste doit être complète dès le début de l'exécution.
- ➔ il doit avoir une liste des calculs en cours : cette liste ne doit pas avoir plus d'éléments que le nombre maximal de calculs définis dans le .ini .

➔ il va effectuer de manière répétitive différentes tâches, jusqu'à ce tous les calculs à lancer soient terminés (avec ou sans erreur)

➔ première tâche : exécuter les vérifications

- vérifier l'état des calculs déjà lancés (à la première itération, il n'y en a pas)
- vérifier si certains calculs déjà lancés et non terminés sont en « timeout »

➔ deuxième tâche : récupérer les informations sur l'état du système

- le nombre de calculs déjà lancés dans la plateforme ; ce nombre de calculs doit être indépendant des environnements virtuels Python installés
- l'état de la plateforme (charge CPU et RAM disponible)

Comme les Plx sont prévus pour fonctionner par défaut sur Debian, l'ordonnanceur récupère les informations sur l'état du système avec des appels à des fonctions de base Debian.

➔ troisième tâche :

- si l'état du système le permet (cf 4.8.4.3), lancer un des calculs en attente de lancement ;

➔ quatrième tâche : si aucun calcul n'a pu être lancé au bout de 'max_attente_sec' secondes, alors une erreur bloquante est générée.

➔ cinquième tâche : attendre avant de reprendre l'itération

4.8.4.3 Contrôle du système

Avant chaque lancement de calcul prévision, le PI applique les contrôles suivants pour déterminer si un nouveau calcul prévision peut être lancé :

- ✓ le nombre total de calculs prévision de la plateforme ne doit pas être supérieur à 'max_nb_calculs_previ'
- ✓ la charge CPU de la plateforme ne doit pas être supérieure à 'max_cpu_load_percent'
- ✓ la mémoire disponible de la plateforme doit être inférieure à 'min_free_mem_mo'

Si l'une des conditions n'est pas vérifiée, elle est tracée dans les logs avec la valeur de dépassement de la limite.

4.8.4.4 Gestion de la progression

Le PI met à jour le fichier « progression.xml » avec l'avancement des calculs prévision, avec le texte suivant :

```
calculs prevision : <nb1> | <nb2> | <nb3> / <nb4>
```

avec les valeurs pour l'instance du Plx en cours d'exécution :

nb1 : nombre de calculs en attente

nb2 : nombre de calculs en cours

nb3 : nombre de calculs terminés

nb4 : nombre total de calculs

Ce texte est ajouté dans le fichier « progression.xml » à chaque changement d'état d'un calcul.

4.9 Sorties (OutputsBaseProcessor)

Le processeur de sortie est chargé de lire les résultats produits par le(s) run(s) de la plateforme de modélisation et de produire les fichiers de sortie attendus par la POM.

4.9.1 Création

Lors de sa création le processeur est paramétré avec le scénario qu'il concerne.

4.9.2 Exécution

Lors de son exécution, le processeur va exécuter la méthode de lecture des résultats (cette méthode est vide dans **OutputsBase** et doit être implémentée dans les sous-classes), puis boucle sur les sorties attendues par la POM (issues du fichier parameters.xml). Pour chacune :

- ✓ il vérifie la cohérence du paramétrage du fichier « parameters.xml ». A cette occasion sont créés les répertoires de sortie si nécessaire.
- ✓ Génère le fichier attendu par la POM
 - ↳ Si la métadonnée est de type « fichier », le PI appelle la fonction qui crée le nom du fichier source à copier dans l'arborescence de la POM. Cette fonction ne fait rien dans le PI et a vocation à être surchargée dans les Plx.
 - ↳ Sinon, le PI initialise un fichier XML Sandre destiné à être rempli des données lus dans les fichiers produits par la plateforme de modélisation. Cette fonctionnalité spécifique de décodage des prévisions produites par le « modèle » est implémentée par les fonctions suivantes :
 - **build_previsions_det** : chargée de produire les prévisions déterministes
 - **build_previsions_prb** : chargée de produire les prévisions probabilistes
 - **build_previsions_tend** : chargée de produire les prévisions de tendance
 - **build_previsions_ens** : chargée de produire les prévisions d'ensemble

Ces fonctions sont destinées à être surchargées dans les Plx. Cependant, afin de constituer un modèle « bouchon », le PI les implémente pour produire des séries de données prévues aléatoires.

Ensuite, le processeur met à jour ses variables de statistiques des fichiers de sortie :

- ✓ **nb_outputs_expected** : nombre de fichier de sortie attendus par la POM
- ✓ **nb_outputs_generated** : nombre de fichiers de sortie générés par le Plx
- ✓ **missing_outputs** : la liste des noms de fichiers de sortie qui n'ont pas été générés par le Plx

4.9.3 OutputsAnalysePrevision

Ce processeur redéfinit 2 actions de **OutputsBase** qui permettent de :

- ✓ lire les résultats produits par la plateforme (Mascaret, Telemac, GRP, Plathynes., ...)
- ✓ construire les prévisions pour les fournir à la POM

4.9.3.1 Lecture des résultats

Le **OutputsAnalysePrevisionProcessor** redéfinit la lecture des résultats analyse/prévision.

Les données sont lues dans le fichier et intégrées à une liste commune (cf. ci-dessous).

Ce processeur définit 2 méthodes pour lire les fichiers résultats spécifiques :

- ✓ **pix_get_hydrodata_analyse**
- ✓ **pix_get_hydrodata_prevision**

Ces méthodes doivent être implémentées par les sous-classes (PIT, PIM...).

La liste commune des données regroupe toutes les productions de tous les runs lancés pour ce scénario POM.

Lors de l'ajout d'une ressource plusieurs cas particuliers se présentent :

- ✓ si le run en cours de lecture est un run d'analyse, la donnée ne devrait pas déjà s'y trouver (même entité, même date, même grandeur). Dans le cas contraire une erreur est levée. La valeur lue dans le fichier est transmis à la POM dans la valeur moyenne du fichier XML.
- ✓ Si le run en cours de lecture est un run de prévision, il doit être associé à une série à propager. Les valeurs lues sont donc ajoutées à la liste commune des données uniquement pour cette série :
 - ↳ seules les données dont la date est > T1 sont acceptées (pour garder la valeur du résultat d'analyse à T1)
 - ↳ s'il n'existe pas de données de même date / grandeur / entité :
 - une nouvelle donnée est créée sans valeur
 - affectation à la donnée de la valeur lue dans la série courante (sous forme d'« extra value »)
 - ajout de cette donnée dans la liste résultat
 - **Attention, pour la série « MOY », la valeur de la série est la valeur par défaut de la donnée (et non pas une « extra value »)**
 - ↳ Sinon, (il existe déjà une donnée identique)
 - écraser la valeur de la série courante dans cette donnée (l'« extra value » correspondante).
 - si cette valeur existe déjà, une erreur est levée et stoppe le calcul (cela ne devrait jamais arriver).

4.9.3.2 Constructions des prévisions

Le **OutputsAnalysePrevisionProcessor** redéfinit la construction des prévisions via les fonctions

- ✓ **build_previsions_det**
- ✓ **build_previsions_tend**
- ✓ **build_previsions_prob**
- ✓ **build_previsions_ens**

Seules les données comprises dans la plage de dates de la ressource de sortie POM sont prises en compte

Les données sont réparties d'après leur série, pour une date de sortie :

- ✓ si la série « MOY » et pas de série « MIN » ou « MAX » : la série « MOY » génère une prévision déterministe
- ✓ Si « MIN » ou « MAX » : les séries « MIN », « MOY », « MAX » génèrent une prévision de tendance
- ✓ pour les autres : les prévisions probabilistes ou les prévisions d'ensemble

4.9.4 OutputsOneRun

Ce processeur redéfinit 2 actions de **OutputsBase** qui permettent de :

- ✓ lire les résultats produits par la plateforme (Mascaret, Telemac, GRP, Plathynes., ...)
- ✓ construire les prévisions pour les fournir à la POM

4.9.4.1 Lecture des résultats

Le **OutputsOneRunProcessor** redéfinit la lecture des résultats en mode OneRun.

Les données sont lues dans le fichier et intégrées à une liste commune (cf. ci-dessous).

Ce processeur définit 1 méthode pour lire les fichiers résultats spécifiques :

- ✓ **pix_get_hydrodata**

Cette méthode doit être implémentée par les sous-classes (PIG...).

La liste commune des données regroupe toutes les productions pour ce scénario POM.

Pour les données déterministes et probabilistes, les valeurs lues sont donc ajoutées à la liste commune des données avec les règles :

- ✓ s'il n'existe pas de données de même date / grandeur / entité :
 - ↳ une nouvelle donnée est créée sans valeur
 - ↳ affectation à la donnée de la valeur lue dans la série courante
 - ↳ ajout de cette donnée dans la liste résultat
 - ↳ Attention, pour la série « MOY », la valeur de la série est la valeur par défaut de la donnée (et non pas une « extra value »)
- ✓ Sinon, (il existe déjà une donnée identique) :

↳ écraser la valeur de la série courante dans cette donnée..

4.9.4.2 Constructions des prévisions

Le **OutputsOneRunProcessor** redéfinit la construction des prévisions via les 2 méthodes :

- ✓ **build_previsions_det**
- ✓ **build_previsions_tend**
- ✓ **build_previsions_prob**
- ✓ **build_previsions_ens**

Seules les données comprises dans la plage de dates de la ressource de sortie POM sont prises en compte.

Les données sont réparties d'après leur série :

- ✓ si « MOY » et pas de « MIN » ou « MAX » : prévision déterministe pour la seule série « MOY », balise « **PrevsTendance** »
- ✓ Si « MIN » ou « MAX » : prévision tendance pour les séries « MIN », « MOY », « MAX », balise « **PrevsDeterministe** »
- ✓ pour les autres séries : prévision probabiliste « **PrevsProb** » ou prévision d'ensemble « **PrevsEnsemble** »

5. Exécution du PI

Même s'il n'est pas opérationnel, le PI peut être lancé comme un modèle « bouchon ». Il n'a pas vocation à l'être, mais son point d'entrée est un bon exemple de ce qu'il faut implémenter pour un véritable modèle.

5.1 Point d'entrée (main)

Il s'agit du module « bouchon.py » qui peut servir de base à l'implémentation du point d'entrée d'un Plx (par copier coller des lignes de code source).

Le point d'entrée vise à :

- ✓ construire et exécuter le StartProcessor (cf. 4.2)
- ✓ gérer les erreurs (exceptions) au plus haut niveau : ces erreurs sont tracées à l'aide du LogMessage (cf. 5.2) ainsi que le statut d'avancement (positionné à 1 en cas d'erreur).

5.2 LogMessage

La classe LogMessage du module « errormanager.py » constitue l'interface avec le fichier « progression.xml ».

Il dispose notamment d'une méthode d'ajout d'informations dans le fichier de progression : un type de message, un message et une progression (entre 0 et 100%). Chaque message est ajouté au fichier comme suit :

- ✓ Trans-typage du message en Unicode s'il ne l'est pas
- ✓ Si le répertoire du fichier progression n'existe pas, le message est affiché dans la sortie standard (print)
- ✓ Sinon
 - ↳ le fichier est créé s'il n'existe pas déjà
 - ↳ le fichier est lu
 - ↳ si le message à écrire n'a pas de progression associée, sa progression est prise égale à la progression générale du fichier
 - ↳ le message, son type et sa progression sont ajoutés au fichier
 - ↳ le fichier est sauvegardé
 - ↳ Le message, son type et sa progression sont également affichés dans la sortie standard

En cas d'erreur lors de la modification du fichier, l'erreur en question est affichée dans la sortie standard (print).

Note : le protocole POM redirige la sortie standard vers le fichier pom.log. Tous les messages redirigés vers la sortie standard s'y trouvent donc.