MASCARET v8.1

API

SPECIFICATIONS TECHNIQUES

EDF - SA au capital de 924.433.331 euros - R.C.S. Paris B 552 081 317
CEREMA - Centre d'Etudes et d'Expertise sur les Risques, l'Environnement, la Mobilité et l'Aménagement

Table des matières

Synthèse

Ce document constitue les spécifications techniques détaillées de l'API [1] proposée par le code de calcul `MASCARET`. `MASCARET` est utilisé pour la simulation numérique des écoulements à surface libre unidimensionnels. Il est capable de simuler les écoulements torrentiels et le transport de polluants.

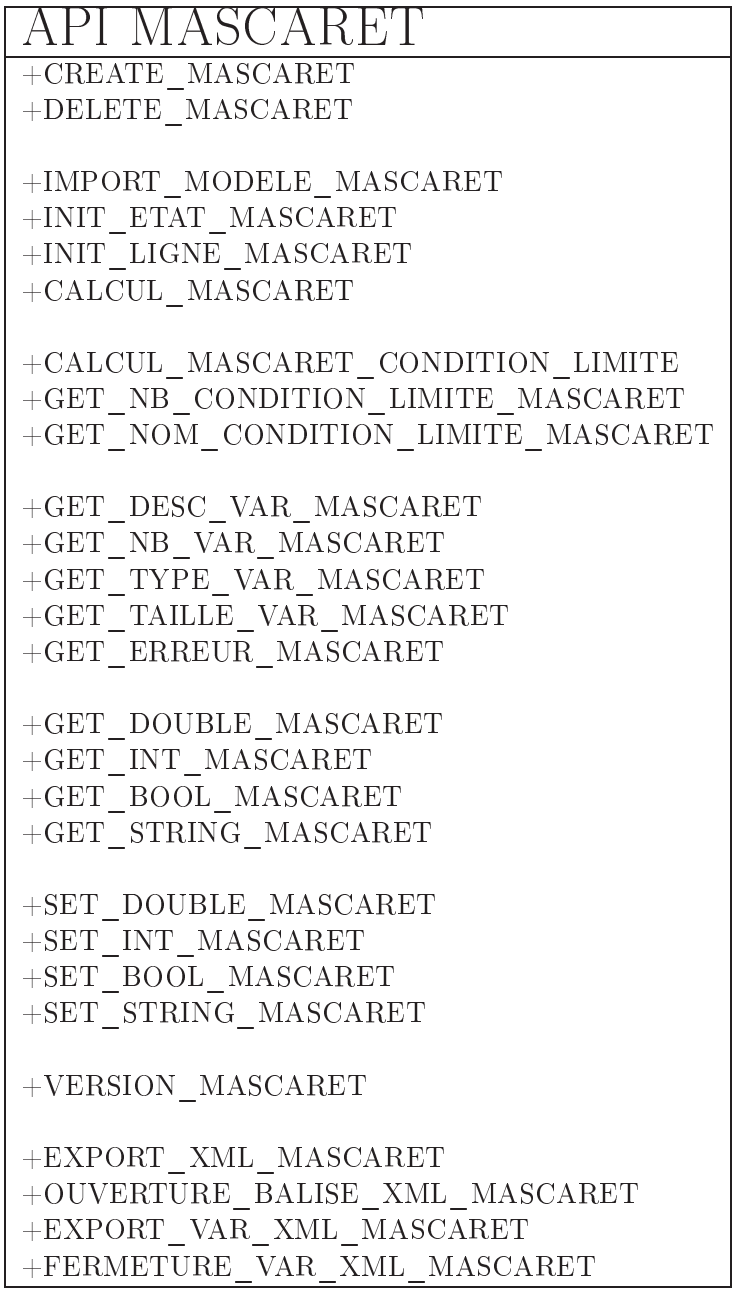
Le code de calcul propose une API pour faire face à des besoins déjà identifiés comme le couplage, le calcul d'incertitudes ou l'optimisation. Mais également, grâce à son approche générique, va permettre de répondre à des besoins non encore connus.

Cette nouvelle API a été conçue suite à un REX d'une précédente version (`MASCARET` v5.1) et tente de remédier aux différents problèmes qui sont apparus avec cette version. De plus, de nouveaux besoins ont été pris en compte.

Les principes qui ont servi de base pour la conception et le développement de l'API sont détaillés dans ce document.

L'ensemble des fonctions proposées par l'API sont :

1. API : Application Programming Interface – une interface fournie pour et par un programme informatique



Cette API a pour ambition de permettre un nouveau type d'usage de `MASCARET` en particulier permettre du calcul intensif dans des plateformes dédiées ou un pilotage fin de la simulation.

La version présentée ici risque d'évoluer pour inclure de nouvelles fonctions nécessaires à une utilisation dans des domaines très différents.

1 Introduction

Ce document constitue les spécifications techniques détaillées de l'API [2] proposée par le code de calcul `MASCARET`. `MASCARET` est utilisé pour la simulation numérique des écoulements à surface libre unidimensionnels. Il est capable de simuler les écoulements torrentiels et le transport de polluants.

Le code de calcul propose une API pour faire face à des besoins déjà identifiés comme le couplage, le calcul d'incertitudes ou l'optimisation. Mais également, grâce à son approche générique, va permettre de répondre à des besoins non encore connus.

Cette nouvelle API a été conçue suite à un REX d'une précédente version (`MASCARET` v5.1) et tente de remédier aux différents problèmes qui sont apparus avec cette version. De plus, de nouveaux besoins ont été pris en compte.

Les principes qui ont servi de base pour la conception et le développement de l'API sont détaillés dans ce document.

2 Versions antérieures

La première version de l'API (`MASCARET` v5.1) date de 2005 et fait apparaître les problèmes suivants :

- la maintenance corrective et évolutive a été difficile à assurer car il y avait un volume de code source spécifique à l'API et une compléxité du "*Wrapper*" C++/Fortran importants. De plus il y avait des redondances entre structures C++ et Fortran ;

- la solution technique pour mélanger C++ et Fortran était spécifique au compilateur, qui aujourd'hui ne fonctionne plus sous les systèmes Windows actuels (VISTA, SEVEN ou 8). De plus, ce compilateur ne fonctionnait que sous Windows et donc interdisait d'utiliser l'API sous GNU/Linux ou tout autre système ;

- les performances de l'API étaient moyennes car le "*Wrapper*" imposait de dupliquer et copier toutes les structures de données C++ vers le Fortran (et inversement) à chaque simulation élémentaire ;

- il était impossible d'interdire au code de calcul d'écrire certains fichiers qui peuvent être au final très volumineux et ralentir l'exécution de la simulation ;

- des problèmes sont apparus lors de l'utilisation de plusieurs modèles en simultané sur la même machine. Ces problèmes étaient liés à des conflits sur des fichiers qui ne sont pas contrôlés par l'API ;

- l'API était accessible uniquement en C++. Ce choix a posé quelques problèmes liés directement à la complexité du langage vis-à-vis des compétences des développeurs et du manque de standardisation des librairies dynamiques basées sur ce langage où les compilateurs *décorent* le nom des fonctions avec des caractères spéciaux spécifiques. De plus certaines plateformes, ne sont pas capables d'intégrer facilement des API C++ (par exemple `SCILAB`) ;

- cette API nécessitait parfois une compréhension du fonctionnement interne du code de calcul pour pouvoir être utilisée directement.

2. API : Application Programming Interface – une interface fournie pour et par un programme informatique

3 Objectifs actuels

Par conséquent, pour cette nouvelle version de l'API, de nouveaux objectifs techniques ont été introduits en plus des fonctions de bases nécessaires.

Ainsi, le pilotage fin du code de calcul est toujours présent, à savoir :

- arrêter la simulation suivant des critères spécifiques ;
- consulter et modifier les conditions limites ou les coefficients de frottement, ainsi que les autres variables de `MASCARET` ;
- contrôler les ressources système utilisées : fichiers et mémoire.

Mais de nouveaux objectifs ou contraintes sont introduits :

- ne pas être restreint à un système d'exploitation, un langage informatique, une plateforme de simulation, ni même à une norme/standard, ni à un modèle de données spécifiques ;
- proposer une solution performante pour du calcul intensif ;
- prendre en compte les évolutions du code facilement.

4 Principes de l'API

4.1 Indépendance

La solution retenue pour permettre l'indépendance vis à vis du langage et du système d'exploitation s'est faite à plusieurs niveaux.

L'API et les versions 7.1 et supérieures de `MASCARET` utilisent un compilateur multi-plateformes (GNU/Linux, Windows, Mac OS, architecture ARM, etc.) et multi-langages. Il s'agit de GCC (GNU Compiler Collection) et en particulier, sa déclinaison pour le Fortran : GFortran. Il existe depuis 1984, et il est considéré comme pérenne et fiable. Ce compilateur permet de génerer des librairies dynamiques aussi bien pour GNU/Linux que pour Windows.

De plus, l'API permet d'être utilisée directement en langage C ou en Fortran (même en Fortran 77). Le choix de ces deux langages permet d'être supporté par un maximum de plateformes, tout en conservant une performance excellente. De plus, la quasi-totalité des langages informatiques et des plateformes permet d'intégrer des librairies dynamiques écrites en langage C.

Enfin, pour faciliter l'intégration, l'API ne propose que des fonctions avec des arguments dont le type est primitif (tableau ou scalaire d'entiers, de réels ou de caractères). Cette approche permet d'éviter des "*Wrappers*" potentiellement complexes, lents et pas toujours portables. Le fait de proposer une API sans type complexe évite d'être dépendant d'un modèle de données particulier. Ainsi, les fonctions de l'API (décrites dans les fichiers "*apimascaret.h*" et "*m_ apimascaret_ i.f90*" (7)) sont autoportantes et ne dépendent d'aucun autre élément et en particulier des structures internes du code `MASCARET`.

4.2 Performance

Pour améliorer les performances, il fallait éviter de dupliquer les données entre les différents langages. Ainsi, en évitant d'écrire un "*Wrapper*" et en supprimant l'existence de données spécifiques au langage C, la duplication d'information a été supprimée. Cette simplification n'a été possible que grâce à la suppression de structures complexes en arguments des fonctions de l'API.

De plus, avec la précédente version de l'API, il était impossible d'empêcher `MASCARET` d'écrire certains fichiers qui pouvaient être très volumineux et ralentir considérablement l'exécution. Cette nouvelle API permet la suppression de toute écriture de fichier sur le disque.

4.3 Maintenance

Le code de calcul `MASCARET` est écrit en Fortran 90, il fallait donc, là encore, éviter d'écrire un "*Wrapper*" en C complexe qui doit être modifié quand le noyau de calcul `MASCARET` est modifié. La suppression de ce "*Wrapper*" facilite donc la maintenance mais décale l'essentiel du développement de l'API vers le Fortran.

Pour faciliter le développement initial et ses évolutions, un générateur de code Fortran a été écrit pour permettre la génération de fonctions qui permettent de consulter et modifier plusieurs centaines de variables réparties dans différentes structures de données du code.

4.4 Structure de données

Le manque de structuration est compensé par l'accès à des méta-données directement accessibles par l'API. Par exemple, on peut obtenir la liste des variables du modèle et de l'état, connaître leur type, leur dimension, et même obtenir une description (en anglais) de chaque variable.

5 Spécifications détaillées

Cette API est accessible depuis le langage C et le Fortran 77. Ces fonctions sont strictement identiques. En effet l'appel en C fait appel quasi directement à la subroutine Fortran correspondante, il n'y a plus d'utilisation de "*Wrappers*", seules les chaînes de caractères C sont converties en chaînes Fortran.

Les fonctions principales de l'API permettent de gérer les instances d'un couple (modèle,état) de `MASCARET` et en particulier la création et la destruction d'une instance ; ainsi que la possibilité d'effectuer des simulations fines ; enfin de consulter et modifier les variables de l'état ou du modèle.

L'ensemble de ces fonctions est décrit en détails dans les paragraphes suivants.

API MASCARET en Fortran 77	API MASCARET langage C
+CREATE_MASCARET	+C_CREATE_MASCARET
+DELETE_MASCARET	+C_DELETE_MASCARET
+IMPORT_MODELE_MASCARET	+C_IMPORT_MODELE_MASCARET
+INIT_ETAT_MASCARET	+C_INIT_ETAT_MASCARET
+INIT_LIGNE_MASCARET	+C_INIT_LIGNE_MASCARET
+CALCUL_MASCARET	+C_CALCUL_MASCARET
+CALCUL_MASCARET_CONDITION_LIMITE	+C_CALCUL_MASCARET_CONDITION_LIMITE
+GET_NB_CONDITION_LIMITE_MASCARET	+C_GET_NB_CONDITION_LIMITE_MASCARET
+GET_NOM_CONDITION_LIMITE_MASCARET	+C_GET_NOM_CONDITION_LIMITE_MASCARET
+GET_DESC_VAR_MASCARET	+C_GET_DESC_VAR_MASCARET
+GET_NB_VAR_MASCARET	+C_GET_NB_VAR_MASCARET
+GET_TYPE_VAR_MASCARET	+C_GET_TYPE_VAR_MASCARET
+GET_TAILLE_VAR_MASCARET	+C_GET_TAILLE_VAR_MASCARET
+GET_ERREUR_MASCARET	+C_GET_ERREUR_MASCARET
+GET_DOUBLE_MASCARET	+C_GET_DOUBLE_MASCARET
+GET_INT_MASCARET	+C_GET_INT_MASCARET
+GET_BOOL_MASCARET	+C_GET_BOOL_MASCARET
+GET_STRING_MASCARET	+C_GET_STRING_MASCARET
+SET_DOUBLE_MASCARET	+C_SET_DOUBLE_MASCARET
+SET_INT_MASCARET	+C_SET_INT_MASCARET
+SET_BOOL_MASCARET	+C_SET_BOOL_MASCARET
+SET_STRING_MASCARET	+C_SET_STRING_MASCARET
+VERSION_MASCARET	+C_VERSION_MASCARET
+EXPORT_XML_MASCARET	+C_EXPORT_XML_MASCARET
+OUVERTURE_BALISE_XML_MASCARET	+C_OUVERTURE_BALISE_XML_MASCARET
+EXPORT_VAR_XML_MASCARET	+C_EXPORT_VAR_XML_MASCARET
+FERMETURE_VAR_XML_MASCARET	+C_FERMETURE_VAR_XML_MASCARET

5.1 Fonctions pour gérer les instances

5.1.1 Création d'une instance

Cette fonction permet d'initialiser les ressources (allocation mémoire et initialisation de variables) associées à une instance `MASCARET`, c'est-à-dire les ressources concernant un modèle et son état.

C'est cette fonction qui doit être utilisée en premier avant de mettre œuvre un modèle et lancer des simulations. Il n'y a aucun pré-requis avant de pouvoir utiliser cette fonction.

La signature de la fonction en langage C est :

```
int C_CREATE_MASCARET(int *Identifiant) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine CREATE_MASCARET(Erreur, Identifiant)
    implicit none
```

```
        integer, intent(out) : : Erreur
        integer, intent(out) : : Identifiant
end subroutine CREATE_MASCARET
```

Il n'y a pas d'argument en entrée de la fonction.

L'argument *Identifiant* en sortie de la fonction identifie de manière unique l'instance de `MASCARET`. Cet identifiant est utilisé en entrée des autres fonctions de l'API.

Actuellement, la création d'au plus 1000 instances de `MASCARET` est possible.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.1.2 Destruction d'une instance

Cette fonction permet de libérer les ressources (désallocation mémoire) associées à une instance de `MASCARET`, c'est-à-dire les ressources concernant un modèle et son état.

C'est cette fonction qui doit être utilisée en dernier après la création d'un modèle et le lancement des simulations.

Avant d'utiliser cette fonction, il est nécessaire d'avoir initialisé les ressources associées à une instance de `MASCARET`, c'est-à-dire d'avoir fait appel à la fonction décrite dans le paragraphe précédent.

La signature de la fonction en langage C est :

```
int C_DELETE_MASCARET(int Identifiant) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine DELETE_MASCARET(Erreur, Identifiant)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
end subroutine DELETE_MASCARET
```

L'argument en entrée est :

– *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*".

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.1.3 Sauvegarde de l'état d'un calcul

Cette fonction permet de sauvegarder en mémoire l'état courant d'une instance de `MASCARET` dans une structure interne de l'API. Elle sera utilisée conjointement avec la fonction "*SET_ETAT_MASCARET*" qui permet de rendre courant un état préalablement sauvegardé.

Cette fonction est utilisée dans le cas, par exemple, d'un coupleur, qui calculera différents états pour le même temps de simulation pour converger.

Avant d'utiliser cette fonction, il est nécessaire d'avoir initialisé l'état du modèle sauvegardé.

La signature de la fonction en langage C est :

```
int C_SAVE_ETAT_MASCARET(int Identifiant,int *IdentifiantEtat );
```

La signature de la subroutine en langage Fortran est :

```
subroutine SAVE_ETAT_MASCARET(Erreur, Identifiant, IdentifiantEtat)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       integer, intent(out) : : IdentifiantEtat
end subroutine SAVE_ETAT_MASCARET
```

L'argument en entrée est :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" dont on souhaite sauvegarder l'état courant pour une utilisation ultérieure.

L'argument en sortie est :

- *IdentifiantEtat* : identifiant de l'état `MASCARET` sauvegardé.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.1.4 Récupération de l'état d'un calcul

Cette fonction permet de récupérer un état préalablement sauvegardé par la fonction "*SAVE_ETAT_MASCARET*" pour le rendre courant à l'instance. Elle doit donc être utilisée conjointement avec la fonction précédente.

Avant d'utiliser cette fonction, il est nécessaire d'avoir sauvegardé l'état que l'on souhaite utiliser par la suite.

La signature de la fonction en langage C est :

```
int C_SET_ETAT_MASCARET(int Identifiant,int IdentifiantEtat );
```

La signature de la subroutine en langage Fortran est :

```
subroutine SET_ETAT_MASCARET(Erreur, Identifiant, IdentifiantEtat)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       integer, intent(in) : : IdentifiantEtat
end subroutine SET_ETAT_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance MASCARET retourné par "*CREATE_MASCARET*" dont on souhaite remplacer l'état courant par l'état sauvegardé ;

- *IdentifiantEtat* : identifiant de l'état MASCARET sauvegardé qui va remplacer l'état courant.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.2 Fonctions principales de la simulation

5.2.1 Importation d'un modèle

Cette fonction permet l'importation d'un modèle MASCARET à partir des fichiers natifs du code de calcul. Ces fichiers sont disponibles dans FUDAA-MASCARET via le menu : "Fichier | Exporter | Mascaret".

La signature de la fonction en langage C est :

```
int C_IMPORT_MODELE_MASCARET(int Identifiant,char *TabNomFichier[],
char *TypeNomFichier[],int Taille,int Impression) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine IMPORT_MODELE_MASCARET(Erreur, Identifiant, TabNomFichier, TypeNomFi-
chier, Taille, Impression)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(LEN=255), dimension(*), intent(in) : : TabNomFichier
         character(LEN=40), dimension(*), intent(in) : : TypeNomFichier
         integer, intent(in) : : Taille
         integer, intent(in) : : Impression
end subroutine IMPORT_MODELE_MASCARET
```

Avant d'utiliser cette fonction, il est nécessaire d'avoir initialisé les ressources associées à une instance de MASCARET. Attention, cette fonction ne permet pas d'importer des modèles avec qualité d'eau (le module Tracer n'est pas géré par l'API).

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance MASCARET retourné par "*CREATE_MASCARET*" ;

- *TabNomFichier* : tableau des noms de fichiers natifs MASCARET à importer (un nom de fichier ne doit pas dépasser 255 caractères, les noms peuvent comporter le chemin des répertoires d'accès au fichier, le séparateur dépend du système hôte) ;

- *TypeNomFichier* : tableau des types de fichiers natifs MASCARET à importer. Il n'est pas possible d'importer une ligne d'eau initiale avec cette fonction. Les valeurs possibles sont :

  - "xcas" ;
  - "geo" ;

- "loi" ;
- "casier" ;
- "listing" ;
- "listing_casier" ;
- "listing_liaison" ;
- "res" ;
- "res_casier" ;
- "res_liaison".

- *Taille* : taille des deux tableaux *TabNomFichier* et *TypeNomFichier* ;

- *Impression* : impression sur les fichiers listing (1 –> Vrai, 0 –> Faux)

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.2.2 Initialisation de l'état initial

Deux fonctions permettent d'initialiser l'état de `MASCARET`. Elles peuvent être utilisées indifféremment. Avant d'utiliser ces fonctions, il est nécessaire d'avoir importé un modèle à une instance de `MASCARET`, c'est-à-dire d'avoir fait appel à la fonction décrite précédemment.

Dans le cas où le modèle importé est permanent, l'initialisation de l'état ne doit pas être faite.

Fonction "*INIT_ETAT_MASCARET*"

La première fonction permet d'initialiser l'état à partir d'un fichier natif de `MASCARET` de type ligne d'eau adapté au modèle importé préalablement. Ce fichier est disponible dans `FUDAA-MASCARET` via le menu : "Hydraulique | Conditions Initiales | Ligne d'eau initiale | Exporter".

La signature de la fonction en langage C est :

```
int C_INIT_ETAT_MASCARET(int Identifiant,char *NomFichier, int Impression) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine INIT_ETAT_MASCARET(Erreur, Identifiant, NomFichier, Impression)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       character(LEN=255), dimension(*), intent(in) : : NomFichier
       integer, intent(in) : : Impression
end subroutine INIT_ETAT_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;

- *NomFichier* : nom du fichier natif contenant la ligne d'eau initiale (maximum 255 caractères) ;

- *Impression* : impression sur les fichiers listing (1 –> Vrai, 0 –> Faux)

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

Fonction "*INIT_LIGNE_MASCARET*"

La deuxième fonction permet d'initialiser l'état à partir de variables (2 tableaux) spécifiant le débit et la cote initiale dans le modèle.

La signature de la fonction en langage C est :

```
int C_INIT_LIGNE_MASCARET(int Identifiant,double Q[],double Z[], int Taille) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine INIT_LIGNE_MASCARET(Erreur, Identifiant, Q, Z, Taille)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       real(8), dimension(*), intent(in) : : Q
       real(8), dimension(*), intent(in) : : Z
       integer, intent(in) : : Taille
end subroutine INIT_LIGNE_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;
- Q : tableau des débits de la ligne d'eau initiale ($m^3.s^{-1}$) ;
- Z : tableau des cotes de la ligne d'eau initiale (m) ;
- *Taille* : taille des deux tableaux Q et Z. Cette taille doit être égale au nombre de sections de calcul dans le modèle (par la taille de la variable "*Model.X*", accessible via la fonction *GET_TAILLE_VAR_MASCARET*). Si la taille n'est pas égale au nombre de sections de calcul, une erreur est retournée.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.2.3 Simulation d'un modèle

Deux fonctions permettent d'effectuer une simulation d'un ou plusieurs pas de temps. Ces fonctions vont calculer un nouvel état courant à partir du modèle courant et de l'état précédent. L'un possède en paramètre les conditions aux limites, l'autre non.

Habituellement ces fonctions sont appelées plusieurs fois en effectuant éventuellement des modifications au niveau du modèle. La fonction ayant les conditions aux limites en arguments évite l'utilisation de la fonction *SET_DOUBLE_MASCARET* sur l'une des variables "*Model.Graph.Level*", "*Model.Graph.Discharge*" ou "*Model.Graph.Time*".

Simulation d'un modèle sans modification des conditions aux limites

La signature de la fonction n'intégrant pas les conditions aux limites en langage C est :

```
int C_CALCUL_MASCARET(int Identifiant,double TpsInitial,double TpsFinal,double PasTps,
int Impression);
```

La signature de la subroutine en langage Fortran est :

```
subroutine CALCUL_MASCARET(Erreur, Identifiant, TpsInitial, TpsFinal, PasTps, Impression)
    implicit none
     integer, intent(out) : : Erreur
     integer, intent(in) : : Identifiant
     real(8), intent(in) : : TpsInitial
     real(8), intent(in) : : TpsFinal
     real(8), intent(in) : : PasTps
     integer, intent(in) : : Impression
end subroutine CALCUL_MASCARET
```

Avant d'utiliser ces fonctions, il est nécessaire d'avoir initialisé l'état (5.2.2) quand on utilise un modèle non permanent. Pour un modèle permanent, il est nécessaire d'avoir importé un modèle à une instance, c'est-à-dire d'avoir fait appel à la fonction décrite dans (5.2.1).

ATTENTION, CETTE FONCTION NE PERMET PAS D'IMPORTER DES MODÈLES AVEC QUALITÉ D'EAU (LE MODULE TRACER N'EST PAS GÉRÉ PAR L'API)

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;
- *TpsInitial* : temps initial du calcul (s) ;
- *TpsFinal* : temps final du calcul (s) ;
- *PasTps* : pas de temps interne du calcul (s) ;
- *Impression* : impression sur le fichier listing (1 –> Vrai, 0 –> Faux)

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

Simulation d'un modèle avec modification des conditions aux limites

Cette fonction est à utiliser conjointement avec les fonctions
"*GET_NB_CONDITION_LIMITE_MASCARET*" et
"*GET_NOM_CONDITION_LIMITE_MASCARET*".

La signature de la fonction intégrant les conditions aux limites en langage C est :

```
int C_CALCUL_MASCARET_CONDITION_LIMITE(int Identifiant,double TpsInitial,double
TpsFinal,double PasTps,double TpsCl[],int TailleTpsCL,double Cl1[][TailleTpsCL],
```

```
double Cl2[][TailleTpsCL],int Impression) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine CALCUL_MASCARET_CONDITION_LIMITE(Erreur, Identifiant, TpsInitial,
TpsFinal, PasTps, Impression)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       real(8), intent(in) : : TpsInitial
       real(8), intent(in) : : TpsFinal
       real(8), intent(in) : : PasTps
       real(8), dimension(*), intent(in) : : TpsCl
       integer, intent(in) : : TailleTpsCL
       real(8), dimension(TailleTpsCL,*), intent(in) : : Cl1
       real(8), dimension(TailleTpsCL,*), intent(in) : : Cl2
       integer, intent(in) : : Impression
end subroutine CALCUL_MASCARET_CONDITION_LIMITE
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;

- *TpsInitial* : temps initial du calcul (s) ;

- *TpsFinal* : temps final du calcul (s) ;

- *PasTps* : pas de temps interne du calcul (s) ;

- *TpsCl* : le vecteur temps commun à toutes les nouvelles conditions aux limites, dimensionné à la taille *TailleTpsCL*

- *TailleTpsCL* : nombre de pas pour les conditions aux limites

- *Cl1* : composante 1 de la condition limite, c'est une matrice décrivant l'évolution des nouvelles conditions aux limites :
  - si la condition limite est de type 1 ou 2 alors cette composante n'est pas utilisée ;
  - si la condition limite est de type 3 alors cette composante est une cote ;
  - si la condition limite est de type 7 alors cette composante est la cote inférieure ;
  - la dimension de cette matrice est différente en Fortran ou en C, à savoir :
    TailleTpsCL, NbCL en Fortran ;
    NbCL, TailleTpsCL en C.

- *Cl2* : composante 2 de la condition limite, c'est une matrice décrivant l'évolution des nouvelles conditions aux limites quand la condition limite nécessite une deuxième composante :
  - si la condition limite est de type 1 ou 3 alors cette composante est un débit ;
  - si la condition limite est de type 2 alors cette composante est une cote ;
  - si la condition limite est de type 7 alors cette composante est la cote supérieure ;
  - la dimension de cette matrice est différente en Fortran ou en C, à savoir :
    TailleTpsCL, NbCL en Fortran ;
    NbCL, TailleTpsCL en C.

- *Impression* : impression sur le fichier listing (1 –> Vrai, 0 –> Faux)

5.3 Fonctions pour consulter le modèle ou l'état

Ces fonctions permettent d'accéder ou de modifier les variables d'une instance courante de l'état ou du modèle. Elles sont décrites en respectant le même "pattern", c'est-à-dire :

- pour consulter : "GET_TYPEDEBASE_MASCARET(identifiant,nomVariable,index1,index2, index3,valeur)"
  - *valeur* : argument en sortie de la fonction dont le type dépend du type de base

- pour modifier : "SET_TYPEDEBASE_MASCARET(identifiant,nomVariable,index1,index2, index3,valeur)"
  - *valeur* : argument en entrée de la fonction dont le type dépend du type de base

5.3.1 Consulter le modèle ou l'état (accesseurs)

Il existe quatre fonctions, une par type de base (réel, entier, booléen, chaîne de caractères) qui permettent d'accéder aux valeurs du modèle ou de l'état courant d'une instance de `MASCARET`.

On peut connaître le type de base d'une variable grâce à l'argument *TypeVar* en sortie de la fonction "*GET_TYPE_VAR_MASCARET*". Le choix de la fonction adaptée au type de base peut ainsi être déterminé via l'API.

La signature de ces quatre fonctions en langage C est :

```
int C_GET_DOUBLE_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,double *valeur) ;
int C_GET_INT_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,int *valeur) ;
int C_GET_BOOL_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,int *valeur) ;
int C_GET_STRING_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,char **valeur) ;
```

La signature de ces quatre subroutines en langage Fortran est :

```
subroutine GET_DOUBLE_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       character(len=40), intent(in) : : NomVar
       integer, intent(in) : : index1
       integer, intent(in) : : index2
       integer, intent(in) : : index3
       real(8), intent(out) : : valeur
end subroutineGET_DOUBLE_MASCARET

subroutine GET_INT_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       character(len=40), intent(in) : : NomVar
       integer, intent(in) : : index1
```

```
        integer, intent(in) : : index2
        integer, intent(in) : : index3
        integer, intent(out) : : valeur
end subroutineGET_INT_MASCARET

subroutine GET_BOOL_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
        implicit none
        integer, intent(out) : : Erreur
        integer, intent(in) : : Identifiant
        character(len=40), intent(in) : : NomVar
        integer, intent(in) : : index1
        integer, intent(in) : : index2
        integer, intent(in) : : index3
        logical, intent(out) : : valeur
end subroutineGET_BOOL_MASCARET

subroutine GET_STRING_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
        implicit none
        integer, intent(out) : : Erreur
        integer, intent(in) : : Identifiant
        character(len=40), intent(in) : : NomVar
        integer, intent(in) : : index1
        integer, intent(in) : : index2
        integer, intent(in) : : index3
        character(len=256), intent(out) : : valeur
end subroutineGET_STRING_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;

- *NomVar* : nom de la variable (notation pointée), la liste des variables est retournée par "*GET_DESC_VAR_MASCARET*". La taille de cette chaîne ne doit pas dépasser 40 caractères (sans compter le caractère de fin de chaîne en langage C) ;

- *index1* : valeur du 1er indice. Cet argument peut être ignoré si la dimension de la variable est 0 (cf. l'argument en sortie *DimVar* de la fonction "*GET_TYPE_VAR_MASCARET*") ;

- *index2* : valeur du 2ème indice. Cet argument peut être ignoré si la dimension de la variable est inférieure ou égale à 1 (cf. l'argument en sortie *DimVar* de la fonction "*GET_TYPE_VAR_MASCARET*") ;

- *index3* : valeur du 3ème indice. Cet argument peut être ignoré si la dimension de la variable est inférieure ou égale à 2 (cf. l'argument en sortie *DimVar* de la fonction "*GET_TYPE_VAR_MASCARET*") ;

L'argument en sortie est :

- *valeur* : valeur de la variable pour les index spécifiés. Le type de cet argument est associé directement au type de base de la fonction.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.3.2 Modifier le modèle ou l'état (mutateurs)

Il existe quatre fonctions, une par type de base (réel, entier, booléen, chaîne de caractères) qui permettent de modifier aux valeurs du modèle ou de l'état courant d'une instance de `MASCARET`. Elles ne permettent pas de modifier la dimension des tableaux.

On peut connaître le type de base d'une variable grâce à l'argument *TypeVar* en sortie de la fonction "*GET_TYPE_VAR_MASCARET*". Le choix de la fonction adaptée au type de base peut ainsi être déterminé via l'API.

La signature de ces quatre fonctions en langage C est :

```
int C_SET_DOUBLE_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,double *valeur) ;
int C_SET_INT_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,int *valeur) ;
int C_SET_BOOL_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,int *valeur) ;
int C_SET_STRING_MASCARET(int Identifiant,char *NomVar,int index1,int index2,
int index3,char *valeur) ;
```

La signature de ces quatre subroutines en langage Fortran est :

```
subroutine SET_DOUBLE_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=40), intent(in) : : NomVar
         integer, intent(in) : : index1
         integer, intent(in) : : index2
         integer, intent(in) : : index3
         real(8), intent(in) : : valeur
end subroutineSET_DOUBLE_MASCARET


subroutine SET_INT_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=40), intent(in) : : NomVar
         integer, intent(in) : : index1
         integer, intent(in) : : index2
         integer, intent(in) : : index3
         integer, intent(in) : : valeur
end subroutineSET_INT_MASCARET
```

```
subroutine SET_BOOL_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=40), intent(in) : : NomVar
         integer, intent(in) : : index1
         integer, intent(in) : : index2
         integer, intent(in) : : index3
         logical, intent(in) : : valeur
end subroutineSET_BOOL_MASCARET


subroutine SET_STRING_MASCARET(Erreur, Identifiant, NomVar, index1, index2,
index3, valeur)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=40), intent(in) : : NomVar
         integer, intent(in) : : index1
         integer, intent(in) : : index2
         integer, intent(in) : : index3
         character(len=256), intent(in) : : valeur
end subroutineSET_STRING_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;

- *NomVar* : nom de la variable (notation pointée), la liste des variables est retournée par "*GET_DESC_VAR_MASCARET*". La taille de cette chaîne ne doit pas dépasser 40 caractères (sans compter le caractère de fin de chaîne en langage C) ;

- *index1* : valeur du 1er indice. Cet argument peut être ignoré si la dimension de la variable est 0 (cf. l'argument en sortie *DimVar* de la fonction "*GET_TYPE_VAR_MASCARET*") ;

- *index2* : valeur du 2ème indice. Cet argument peut être ignoré si la dimension de la variable est inférieure ou égale à 1 (cf. l'argument en sortie *DimVar* de la fonction "*GET_TYPE_VAR_MASCARET*") ;

- *index3* : valeur du 3ème indice. Cet argument peut être ignoré si la dimension de la variable est inférieure ou égale à 2 (cf. l'argument en sortie *DimVar* de la fonction "*GET_TYPE_VAR_MASCARET*") ;

L'argument en sortie est :

- *valeur* : nouvelle valeur de la variable pour les index spécifiés. Le type de cet argument est associé directement au type de base de la fonction. En C, quand cette variable est une chaîne de caractères, elle doit être désallouée par l'utilisateur de l'API.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème

pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.4 Fonctions donnant des informations structurelles et métier

5.4.1 Lister les variables de MASCARET

Cette fonction permet de récupérer la liste des variables gérées par l'API. Elles sont toutes consultables via les fonctions "*GET_TYPEDEBASE_MASCARET*" et pour la plupart modifiables via les fonctions "*SET_TYPEDEBASE_MASCARET*". Cette liste est constituée de deux parties :

- liste des noms de chaque variable (identifiant unique) déterminés directement à partir des structures internes du code de calcul en utilisant une notation pointée (par exemple pour la cote d'eau, variable *Z*, de la structure *State*, le nom de la variable est "*State.Z*" ;
- liste des descriptions en anglais de chaque variable permettant d'obtenir une information compréhensible d'un point de vue métier sur le contenu de la variable. Ainsi la description de la variable "*State.Z*" est "Water level (m)".

C'est cette fonction qui permet de faire un *pont* entre une vision technique et une vision métier.

Avant d'utiliser cette fonction, il est utile d'avoir initialisé les ressources associées à une instance de MASCARET, c'est-à-dire d'avoir fait appel à la fonction décrite dans (5.2.2). Cette fonction ne dépend pas directement d'une instance de MASCARET mais l'initialisation des ressources permet de mieux gérer les messages d'erreurs éventuels.

La signature de la fonction en langage C est :

```
int C_GET_DESC_VAR_MASCARET(int Identifiant,char *TabNom[],char *TabDesc[],
int *Taille);
```

La signature des deux subroutines correspondantes en langage Fortran sont :

```
subroutine GET_DESC_VAR_MASCARET(Erreur, Identifiant, TabNom, TabDesc,
NbVarMascaret)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       character(len=40), dimension(*), intent(out) : : TabNom
       character(len=110), dimension(*), intent(out) : : TabDesc
       integer, intent(in) : : NbVarMascaret
end subroutineGET_DESC_VAR_MASCARET
subroutine GET_NB_VAR_MASCARET(Taille)
      implicit none
       integer, intent(out) : : Taille
end subroutine GET_NB_VAR_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance MASCARET retourné par "*CREATE_MASCARET*" ;

- *NbVarMascaret* : taille des tableaux *TabNom* et *TabVar* en argument : uniquement en Fortran. Cette valeur doit être égale à la valeur retournée par la subroutine "*GET_NB_VAR_MASCARET*".

Les arguments en sortie sont :

- *TabNom* : tableau des noms de variable du modèle ou de l'état ;
- *TabDesc* : tableau des descriptions de variable du modèle ou de l'état ;
- *Taille* : taille des tableaux des noms et des descriptions de variable.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.4.2 Identifier le type d'une variable

Cette fonction permet de récupérer des informations structurelles d'une variable particulière. C'est-à-dire son type (entier, réel, booléen ou chaîne de caractères et si c'est un tableau), sa catégorie (appartient au modèle ou à l'état), si elle est modifiable (via la fonction "*SET_TYPEDEBASE_MASCARET*" et sa dimension (c'est-à-dire le nombre d'index à utiliser pour accéder à la valeur de la variable.

Avant d'utiliser cette fonction, il est utile d'avoir initialisé les ressources associées à une instance de `MASCARET`, c'est-à-dire d'avoir fait appel à la fonction décrite dans le paragraphe (5.2.2). Cette fonction ne dépend pas directement d'une instance de `MASCARET` mais l'initialisation des ressources permet de mieux gérer les messages d'erreurs éventuels.

La signature de la fonction en langage C est :

```
int C_GET_TYPE_VAR_MASCARET(int Identifiant,char *NomVar,char *TypeVar,
char *Categorie,int *Modifiable,int *dimVar) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine GET_TYPE_VAR_MASCARET(Erreur, Identifiant, NomVar, TypeVar, Categorie,
Modifiable, dimVar)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=40), intent(in) : : NomVar
         character(len=10), intent(out) : : TypeVar
         character(len=10), intent(out) : : Categorie
         integer, intent(out) : : Modifiable
         integer, intent(out) : : dimVar
end subroutineGET_TYPE_VAR_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;

- *NomVar* : nom de la variable (notation pointée), la liste des variables est retournée par "*GET_DESC_VAR_MASCARET*". La taille de cette chaîne ne doit pas dépasser 40 caractères (sans compter le caractère de fin de chaîne en langage C).

Les arguments en sortie sont :

- *TypeVar* : précise le type de la variable. Les valeurs possibles sont :
  - "INT" : scalaire de type entier ;

  - "DOUBLE" : scalaire de type réel ;
  - "BOOL" : scalaire de type booléen ;
  - "STRING" : chaîne de caractères ;
  - "TABINT" : vecteur de type entier ;
  - "TABDOUBLE" : vecteur de type réel ;
  - "TABBOOL" : vecteur de type booléen.

- *Categorie* : précise si la variable appartient à l'état. Les valeurs possibles sont :
  - "MODEL" : la variable n'est pas modifiée par la fonction de simulation ;
  - "STATE" : la variable est modifiée par la fonction de simulation.

- *Modifiable* : indique si la variable est modifiable via les fonctions décrites dans (5.3.2). Les valeurs possibles sont :
  - 1 –> la variable est modifiable par une fonction mutateur (cf. 5.3.2) ;
  - 0 –> la variable n'est pas modifiable par une fonction mutateur (cf. 5.3.2).

- *dimVar* : la dimension de la variable, c'est-à-dire le nombre d'index pour accéder à la variable. Les valeurs possibles varient entre 0 et 3.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.4.3 Connaître la taille d'une variable

Cette fonction permet de récupérer des informations sur la taille d'une variable particulière. C'est-à-dire la valeur maximale des index qui permet d'accéder en consultation ou en modification à la variable.

Avant d'utiliser cette fonction, il est nécessaire d'avoir initialisé une instance d'un modèle `MASCARET`. Si la variable appartient à l'état, il faut avoir initialisé l'état. Cette fonction dépend directement d'une instance de `MASCARET`.

La signature de la fonction en langage C est :

```
int C_GET_TAILLE_VAR_MASCARET(int Identifiant,char *NomVar,int index1,int *taille1,
int *taille2,int *taille3) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine GET_TAILLE_VAR_MASCARET(Erreur, Identifiant, NomVar, index1, taille1,
taille2, taille3)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=40), intent(in) : : NomVar
         integer, intent(in) : : index1
         integer, intent(out) : : taille1
         integer, intent(out) : : taille2
         integer, intent(out) : : taille3
end subroutine GET_TAILLE_VAR_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;

- *NomVar* : nom de la variable (notation pointée), la liste des variables est retournée par "*GET_DESC_VAR_MASCARET*". La taille de cette chaîne ne doit pas dépasser 40 caractères (sans compter le caractère de fin de chaîne en langage C) ;

- *index1* : valeur du 1er indice utilisé de l'instance de la variable dont on souhaite connaître la taille. Cet argument est utilisé pour les variables commençant par "*Model.CrossSection*", "*Model.Graph*","*Model.Weir*","*Model.LateralWeir*","*Model.Boundary*","*Model.StoArea*" et "*Model.Junction*".
  Pour les autres variables, cet indice est ignoré car il n'a pas d'utilité.

Les arguments en sortie sont :

- *taille1* : valeur maximale du 1er indice pour accéder à la variable ;

- *taille2* : valeur maximale du 2ème indice pour accéder à la variable ;

- *taille3* : valeur maximale du 3ème indice pour accéder à la variable ;

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.4.4 Obtenir une description d'une erreur

Cette fonction permet de récupérer le message d'erreur correspondant à une erreur qui s'est produite lors de l'appel précédent d'une fonction de l'API.

La signature de la fonction en langage C est :

```
int C_GET_ERREUR_MASCARET(int Identifiant,char **Message) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine GET_ERREUR_MASCARET(Erreur, Identifiant, Message)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       character(len=256), intent(out) : : Message
end subroutine GET_ERREUR_MASCARET
```

L'argument en entrée est :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;

L'argument en sortie est :

– *Message* : le message décrivant l'erreur apparue lors de l'exécution de la fonction de l'API appelée précédemment. La longueur maximale du message est de 256 caractères. En C, cette variable doit être désallouée par l'utilisateur de l'API.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.4.5 Récupérer la version du code de calcul

Cette fonction utilitaire permet de récupérer simplement la version du code de calcul `MASCARET`.

La signature de la fonction en langage C est :

```
int C_GET_VERSION_MASCARET(int *Majeur,int *Mineur,int *Micro) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine GET_VERSION_MASCARET(Majeur, Mineur, Micro)
      implicit none
       integer, intent(out) : : Majeur
       integer, intent(out) : : Mineur
       integer, intent(out) : : Micro
end subroutine GET_VERSION_MASCARET
```

La fonction n'a pas d'argument en entrée.

Les arguments en sortie sont :

– *Majeur* : numéro de la version majeure de `MASCARET`. Exemple, pour la version 7.1.2 de `MASCARET`, la valeur retournée est 7 ;

– *Mineur* : numéro de la version mineure de `MASCARET`. Exemple, pour la version 7.1.2 de `MASCARET`, la valeur retournée est 1 ;

– *Micro* : numéro de la version micro de `MASCARET`. Exemple, pour la version 7.1.2 de `MASCARET`, la valeur retournée est 2 ;

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.4.6 Récupérer des informations sur les conditions aux limites

Ces fonctions permettent de récupérer le nombre de conditions aux limites, ainsi que leur nom.

Les signatures de ces fonctions en langage C sont :

```
int C_GET_NB_CONDITION_LIMITE_MASCARET(int Identifiant,int *NbCL) ;
int C_GET_NOM_CONDITION_LIMITE_MASCARET(int Identifiant,int NumCL,
char **NomCL,int *NumLoi) ;
```

Les signaturse des subroutine en langage Fortran sont :

```
subroutine GET_NB_CONDITION_LIMITE_MASCARET(Erreur, Identifiant, NbCL)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       integer, intent(out) : : NbCL
end subroutine GET_NB_CONDITION_LIMITE_MASCARET
subroutine GET_NOM_CONDITION_LIMITE_MASCARET(Erreur, Identifiant, NumCL,
NomCL, NumLoi)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       integer, intent(in) : : NumCL
       character(LEN=30), intent(out) : : NomCL
       integer, intent(out) : : NumLoi
end subroutine GET_NOM_CONDITION_LIMITE_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;
- *NumCl* : numéro de la condition limite dont on souhaite connaître le nom, ce numéro doit être compris entre 1 et NbCL

Les arguments en sortie sont :

- *NbCL* : nombre de conditions aux limites dans le modèle ;
- *NomCL* : nom de la condition à la limite. En C, cette variable doit être désallouée par l'utilisateur de l'API ;
- *NumLoi* : numéro de la loi dans le modèle correspondant à la condition limite demandée.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.5 Fonctions utilitaires XML

5.5.1 Exporter un modèle ou un état dans un fichier XML

Cette fonction permet d'exporter dans un fichier XML, le modèle ou l'état d'une instance de `MASCARET`.

Avant d'utiliser cette fonction, il est nécessaire d'avoir initialisé une instance d'un modèle `MASCARET`, c'est-à-dire d'avoir fait appel à la fonction décrite dans le paragraphe (5.2.2). Si l'on souhaite exporter l'état, il faut l'avoir initialisé.

La signature de la fonction en langage C est :

```
int C_EXPORT_XML_MASCARET(int Identifiant,char *NomFichier,int AvecDesc,
```

```
int exportModele);
```

La signature de la subroutine en langage Fortran est :

```
subroutine EXPORT_XML_MASCARET(Erreur, Identifiant, NomFichier, AvecDesc,
exportModele)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=255), intent(in) : : NomFichier
         logical, intent(in) : : AvecDesc
         logical, intent(in) : : exportModele
end subroutine EXPORT_XML_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;
- *NomFichier* : nom du fichier XML qui sera créé. La taille de cette chaîne ne doit pas dépasser 255 caractères (sans compter le caractère de fin de chaîne en langage C) ;
- *AvecDesc* : si vrai (valeur 1), ajoute la description de la variable ;
- *exportModele* : si vrai (valeur 1), exportation du modèle, sinon de l'état.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.5.2 Ecrire l'en-tête d'un fichier XML

Cette fonction doit être utilisée quand on souhaite créer un fichier XML contenant une liste spécifique de variables. Cette fonction permet d'écrire l'en-tête d'un fichier XML et une balise ouvrante au choix de l'utilisateur. On doit utiliser cette fonction avant d'utiliser la fonction qui permet d'écrire une variable spécifique ("*EXPORT_VAR_XML_MASCARET*").

La signature de la fonction en langage C est :

```
int C_OUVERTURE_BALISE_XML_MASCARET(int Identifiant,char *NomFichier,
int uniteLogique,char *balise);
```

La signature de la subroutine en langage Fortran est :

```
subroutine OUVERTURE_BALISE_XML_MASCARET(Erreur, Identifiant, NomFichier,
uniteLogique, balise)
        implicit none
         integer, intent(out) : : Erreur
         integer, intent(in) : : Identifiant
         character(len=255), intent(in) : : NomFichier
         integer, intent(in) : : uniteLogique
         character(len=255), intent(in) : : balise
end subroutine OUVERTURE_BALISE_XML_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;
- *NomFichier* : nom du fichier XML qui sera créé. La taille de cette chaîne ne doit pas dépasser 255 caractères (sans compter le caractère de fin de chaîne en langage C) ;
- *uniteLogique* : unité logique utilisé par le fichier XML, elle doit être strictement supérieure à 36 ;
- *balise* : balise racine du fichier XML.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.5.3 Ecrire une variable dans un fichier XML

Cette fonction doit être utilisée à la suite de la fonction d'écriture de l'en-tête de fichier XML et permet d'écrire une variable spécifique de `MASCARET` dans un fichier XML. Cette fonction peut être appelée plusieurs fois successivement.

La signature de la fonction en langage C est :

```
int C_EXPORT_VAR_XML_MASCARET(int Identifiant,int uniteLogique,char *nomVar,
int avecDesc) ;
```

La signature de la subroutine en langage Fortran est :

```
subroutine EXPORT_VAR_XML_MASCARET(Erreur, Identifiant, uniteLogique, nomVar,
avecDesc)
      implicit none
       integer, intent(out) : : Erreur
       integer, intent(in) : : Identifiant
       integer, intent(in) : : uniteLogique
       character(len=40), intent(in) : : nomVar
       logical, intent(in) : : avecDesc
end subroutine EXPORT_VAR_XML_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;
- *uniteLogique* : unité logique utilisé par le fichier XML, elle doit être égale à la valeur utilisée lors de l'appel à la fonction "*OUVERTURE_BALISE_XML_MASCARET*" ;
- *nomVar* : nom de la variable (notation pointée) à exporter dans le fichier XML. La liste des variables est retournée par "*GET_DESC_VAR_MASCARET*". La taille de cette chaîne ne doit pas dépasser 40 caractères (sans compter le caractère de fin de chaîne en langage C) ;
- *avecDesc* : si vrai (valeur 1), ajoute la description de la variable dans le fichier XML.

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la

fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

5.5.4 Fermeture du fichier XML

Cette fonction doit être utilisée à la suite de la fonction d'écriture de l'en-tête de fichier XML et des appels à la fonction d'écriture d'une variable en XML. Elle ferme la balise et le fichier XML.

La signature de la fonction en langage C est :

```
int C_FERMETURE_BALISE_XML_MASCARET(int Identifiant,int uniteLogique,
char *balise);
```

La signature de la subroutine en langage Fortran est :

```
subroutine FERMETURE_BALISE_XML_MASCARET(Erreur, Identifiant, uniteLogique,
balise)
    implicit none
     integer, intent(out) : : Erreur
     integer, intent(in) : : Identifiant
     integer, intent(in) : : uniteLogique
     character(len=255), intent(in) : : balise
end subroutine FERMETURE_BALISE_XML_MASCARET
```

Les arguments en entrée sont :

- *Identifiant* : identifiant de l'instance `MASCARET` retourné par "*CREATE_MASCARET*" ;
- *uniteLogique* : unité logique utilisé par le fichier XML, elle doit être égale à la valeur utilisée lors de l'appel à la fonction "*OUVERTURE_BALISE_XML_MASCARET*" ;
- *balise* : balise racine du fichier XML à fermer. Elle doit être identique à celle utilisée pour l'ouverture ("*OUVERTURE_BALISE_XML_MASCARET*") ;

La valeur de retour de la fonction C ou de l'argument Fortran *Erreur* est identique et indique si la fonction s'est déroulée correctement (valeur 0). Pour toute valeur différente de 0, il y a eu un problème pendant l'exécution. Pour avoir plus de détails concernant l'erreur, il faut faire appel à la fonction "*GET_ERREUR_MASCARET*".

6 Liste des variables

6.1 Variables du modèle

NOM DE LA VARIABLE	DESCRIPTION	TYPE	DIMENSION
Model.Connect.FirstNdNum	Number of the first node for a reach	TABINT	1
Model.Connect.LastNdNum	Number of the last node for a reach	TABINT	1
Model.Connect.NumReachJunction	The number of reaches for a junction	TABINT	1
Model.Connect.ReachNum	The numbers of the reaches for a junction	TABINT	2
Model.Connect.NodeNum	Node number for a junction	TABINT	2
Model.Connect.ReachNumFreeOutflow	Reach number for the free outflow	TABINT	1
Model.Connect.NodeNumFreeOutflow	Node number for the free outflow	TABINT	1
Model.CrossSection.Name	Name of the cross section	STRING	1
Model.CrossSection.RelAbs	Relative abscissa of the cross section	DOUBLE	1
Model.CrossSection.AbsAbs	Absolute abscissa of the cross section	DOUBLE	1
Model.CrossSection.ReachNum	Reach number of the cross section	INT	1
Model.CrossSection.ReachName	Name of the reach	STRING	1
Model.CrossSection.NumStep	Number of steps of the vertical discretisation	INT	1
Model.CrossSection.Step	Step value for the vertical discretisation	DOUBLE	1
Model.CrossSection.Zbot	Bottom level of the cross section	DOUBLE	1
Model.CrossSection.Zbank	Bank level of the cross section	TABDOUBLE	2
Model.CrossSection.X	X points	TABDOUBLE	2
Model.CrossSection.Y	Y points	TABDOUBLE	2
Model.CrossSection.Bound1	Boundary of the main channel	TABINT	2
Model.CrossSection.Bound2	Boundary of the floodplain	TABINT	2
Model.CrossSection.FricCoef1	Friction coefficient of the main channel	DOUBLE	1
Model.CrossSection.FricCoef2	Friction coefficient of the floodplain	DOUBLE	1
Model.VDSection.S	Total wetted area	TABDOUBLE	2
Model.VDSection.S1	Main channel wetted area	TABDOUBLE	2
Model.VDSection.S2	Floodplain wetted area	TABDOUBLE	2
Model.VDSection.SS	Ineffective flow area	TABDOUBLE	2
Model.VDSection.S1GEO	Non-hydrostatic termsTermes (Scube)	TABDOUBLE	2
Model.VDSection.CELER	Celerity	TABDOUBLE	2
Model.VDSection.B	Total width	TABDOUBLE	2
Model.VDSection.INV	Riemann invariants	TABDOUBLE	2
Model.VDSection.INTE	INTE AIGEO	TABDOUBLE	2
Model.VDSection.DYDX	dY/dx|s	TABDOUBLE	2

Model.VDSection.PRESS	Pressure	TABDOUBLE	2
Model.VDSection.DEB	Conveyance	TABDOUBLE	2
Model.VDSection.DEB1	Main channel conveyance	TABDOUBLE	2
Model.VDSection.DEB2	Floodplain conveyance	TABDOUBLE	2
Model.VDSection.SD	Shifted - Total wetted area	TABDOUBLE	2
Model.VDSection.SD1	Shifted - Main channel area	TABDOUBLE	2
Model.VDSection.SD2	Shifted - Floodplain	TABDOUBLE	2
Model.VDSection.PRESSD	Shifted - Pressure	TABDOUBLE	2
Model.VDSection.BD	Shifted - Total width	TABDOUBLE	2
Model.VDSection.DEBD	Shifted - Total conveyance	TABDOUBLE	2
Model.DryArea.FirstNode	Number of the first node	INT	1
Model.DryArea.LastNode	Number of the last node	INT	1
Model.Graph.Name	Name of the graph	STRING	1
Model.Graph.Type	Graph type : 1->Q=f(T) 2->Z=f(T) 3->Q,Z=f(T) 4->Z=f(Q) 5->Q=f(Z) 6->Zus=f(Zds,Q) 7->Zinf,Zsup=f(T)	INT	1
Model.Graph.Time	Time value	TABDOUBLE	2
Model.Graph.Level	Level value	TABDOUBLE	2
Model.Graph.Discharge	Discharge value	TABDOUBLE	2
Model.Graph.SupLevel	Superior level	TABDOUBLE	2
Model.Graph.InfLevel	Inferior level	TABDOUBLE	2
Model.Graph.DownLevel	Downstream level	TABDOUBLE	2
Model.Graph.UpLevel	Upstream level	TABDOUBLE	3
Model.Weir.Name	Name of the weir or dam	STRING	1
Model.Weir.Number	Weir number	INT	1
Model.Weir.ReachNum	Numero de la branche	INT	1
Model.Weir.RelAbscissa	Relative abscissa	DOUBLE	1
Model.Weir.Node	Node number	INT	1
Model.Weir.Type	Type	INT	1
Model.Weir.State	Enabled or disabled	BOOL	1
Model.Weir.Thickness	Thick or thin	INT	1
Model.Weir.CrestLevel	Crest level (m)	DOUBLE	1
Model.Weir.DischCoef	Discharge coefficient	DOUBLE	1

Model.Weir.BrkLevel	Breaking water level	DOUBLE	1
Model.Weir.Slope	Gradient of crest lowering (m/s)	DOUBLE	1
Model.Weir.Discharge	Constant discharge (m^3/s)	DOUBLE	1
Model.Weir.GateLength	Length of the gate	DOUBLE	1
Model.Weir.GraphNum	Graph number	INT	1
Model.Weir.PtZ	Z points (levels) for the graph	TABDOUBLE	2
Model.Weir.PtQ	Q points (discharges) for the graph	TABDOUBLE	2
Model.Weir.PtZus	Z upstream points (levels) for the graph	TABDOUBLE	3
Model.Weir.PtZds	Z downstream points (levels) for the graph	TABDOUBLE	2
Model.Weir.PtZup	Upper levels of the gate	DOUBLE	1
Model.Weir.PtZlo	Lower levels of the gate	DOUBLE	1
Model.Weir.PtX	Crest abscissa	TABDOUBLE	2
Model.Weir.PtY	Crest ordinate	TABDOUBLE	2
Model.LateralWeir.Name	Name of the lateral weir	STRING	1
Model.LateralWeir.Type	Lateral weir definition	INT	1
Model.LateralWeir.ReachNum	Reach number of the lateral weir	INT	1
Model.LateralWeir.RelAbscissa	Relative abscissa of the lateral weir	DOUBLE	1
Model.LateralWeir.Length	Length of the lateral weir	DOUBLE	1
Model.LateralWeir.UpNode	Upstream node of the lateral weir	INT	1
Model.LateralWeir.DownNode	Downstream node of the lateral weir	INT	1
Model.LateralWeir.CrestLevel	Crest level of the lateral weir	DOUBLE	1
Model.LateralWeir.DischCoef	Discharge coefficient of the lateral weir	DOUBLE	1
Model.LateralWeir.GraphNum	Graph number associated with the lateral weir	INT	1
Model.LateralWeir.PtZ	Z points of the graph (level (m))	TABDOUBLE	2
Model.LateralWeir.PtQ	Q points of the graph (discharge (m^3/s))	TABDOUBLE	2
Model.File.Listing.Unit	Logical unit number Listing	INT	0
Model.File.Listing.Name	Name of the file Listing	STRING	0
Model.Link.Type	Type of the link (Weir, channel, syphon or culvert)	INT	1
Model.Link.Kind	River-TO-Storage Area or Storage Area-TO-Storage Area	INT	1
Model.Link.CulverFlowDir	Flow direction for the culvert	INT	1
Model.Link.StoR.NumS	Corresponding storage area number (link Storage Area — River)	INT	1
Model.Link.StoR.Abscissa	Abscissa of the link (link Storage Area — River)	DOUBLE	1
Model.Link.StoR.Node	Corresponding node of the link (link Storage Area — River)	INT	1

Model.Link.StoR.ReachNum	Number of the corresponding reach (link Storage Area — River)	INT	1
Model.Link.StoR.DQDZsto	Derivative of the discharge with respect to the storage area level (link Storage Area — River)	DOUBLE	1
Model.Link.StoR.DQDZriv	Derivative of the discharge with respect to the river level (link Storage Area — River)	DOUBLE	1
Model.Link.StoS.S_i	Number of the storage area 'i' (discharge flowing from 'i' towards 'j')	INT	1
Model.Link.StoS.S_j	Number of the storage area 'j' (discharge flowing from 'i' towards 'j')	INT	1
Model.Link.StoS.DQDZup	Derivative of the flow discharge with respect to the water level in the upstream storage area	DOUBLE	1
Model.Link.StoS.DQDZdown	Derivative of the flow discharge with respect to the water level in the downstream storage area	DOUBLE	1
Model.Link.Width	Width of the link (m)	DOUBLE	1
Model.Link.Length	Length of the link (m)	DOUBLE	1
Model.Link.Level	Level of the link (m)	DOUBLE	1
Model.Link.MeanLevel	Mean level of the link (m)	DOUBLE	1
Model.Link.CoefCulvertDischarge	Coefficient of the culvert discharge	DOUBLE	1
Model.Link.CSection	Cross section of the culvert or the syphon (m^2)	DOUBLE	1
Model.Link.Roughness	Roughness (Strickler) of the link	DOUBLE	1
Model.Link.CoefWeirDischarge	Weir type discharge coefficient (weir or culvert)	DOUBLE	1
Model.Link.HeadLossCoef	Head loss coefficient (syphon or culvert)	DOUBLE	1
Model.Link.ActivationCoef	Activation coefficient for a weir	DOUBLE	1
Model.Link.FlowExchange	Flow exchanged (m^3/s)	DOUBLE	1
Model.Link.PrevDischarge	Discharge value at the previous time step (m^3/s)	DOUBLE	1
Model.Link.MaxDischarge	Maximal discharge value (m^3/s)	DOUBLE	1
Model.Link.TimeMaxDischarge	Time corresponding to the maximal discharge value (s)	DOUBLE	1
Model.Link.ExchangeVelocity	Exchange velocity	DOUBLE	1
Model.Link.MaxVelocity	Maximal velocity value	DOUBLE	1
Model.Link.TimeMaxVelocity	Time corresponding to the maximal velocity value	DOUBLE	1
Model.File.Result.Unit	Logical unit number Result	INT	0
Model.File.Result.Name	Name of the file Result	STRING	0
Model.File.Result2.Unit	Logical unit number Result2	INT	0
Model.File.Result2.Name	Name of the file Result2	STRING	0
Model.File.GeoStoArea.Unit	Logical unit number GeoStoArea	INT	0
Model.File.GeoStoArea.Name	Name of the file GeoStoArea	STRING	0
Model.File.ResultStoArea.Unit	Logical unit number ResultStoArea	INT	0
Model.File.ResultStoArea.Name	Name of the file ResultStoArea	STRING	0

Model.File.ResultLink.Unit	Logical unit number ResultLink	INT	0
Model.File.ResultLink.Name	Name of the file ResultLink	STRING	0
Model.File.ListingStoArea.Unit	Logical unit number ListingStoArea	INT	0
Model.File.ListingStoArea.Name	Name of the file ListingStoArea	STRING	0
Model.File.ListingLink.Unit	Logical unit number ListingLink	INT	0
Model.File.ListingLink.Name	Name of the file ListingLink	STRING	0
Model.Boundary.Name	Name of a boundary	STRING	1
Model.Boundary.Slope	Slope of the bottom	DOUBLE	1
Model.Boundary.GraphNum	Graph number	INT	1
Model.Boundary.Type	Type of the boundary (flow or stage hydrograph, rating curve, open ,etc.)	INT	1
Model.Boundary.PtZ	Z points of the graph (level (m))	TABDOUBLE	2
Model.Boundary.PtQ	Q points of the graph (discharge (m^3/s))	TABDOUBLE	2
Model.StorageArea.BottomLevel	Bottom level of a storage area (m)	DOUBLE	1
Model.StorageArea.Level	Water level of a storage area (m)	DOUBLE	1
Model.StorageArea.Surface	Surface of a storage area (m^2)	DOUBLE	1
Model.StorageArea.Volume	Volume of a storage area (m^3)	DOUBLE	1
Model.StorageArea.InitVolume	Initial volume of a storage area (m^3)	DOUBLE	1
Model.StorageArea.Statement	Statement of a storage area	DOUBLE	1
Model.StorageArea.ErrorStatement	Error statement of a storage area	DOUBLE	1
Model.StorageArea.DZ	Water level elevation of a storage area for one time step	DOUBLE	1
Model.StorageArea.MaxLevel	Maximal water level of a storage area	DOUBLE	1
Model.StorageArea.TimeMaxLevel	Time for the maximal water level	DOUBLE	1
Model.StorageArea.SSLink	SSLink(:,1) : number of the link, SSLink(:,2) : number of the corresponding storage area	TABINT	3
Model.StorageArea.RSLink	RSLink(:,1) : number of the link, RSLink(:,2) : number of the corresponding node	TABINT	3
Model.StorageArea.Graph_Z_S	Surface of the storage area as a funtion of the elevation	TABDOUBLE	3
Model.StorageArea.Graph_Z_V	Volume of the storage area as a funtion of the elevation	TABDOUBLE	3
Model.StorageArea.VertDisc	Vertical discretisation step for a storage area	DOUBLE	1
Model.StorageArea.NbVertDisc	Number of data for the vertical discretisation	INT	1
Model.StorageArea.BoundPts	Coordinates of the boundary points	TABDOUBLE	3
Model.StorageArea.InternalPts	Coordinates of the internal points	TABDOUBLE	3
Model.Junction.Name	Name of the junction	STRING	1
Model.Connect.ReachNum	The number of reaches	TABINT	2

Model.Junction.Abscissa	Abscissa of the reaches	TABDOUBLE	2
Model.Junction.Ordinate	Ordinate of the reaches	TABDOUBLE	2
Model.Junction.Angle	Angle of the reaches	TABDOUBLE	2
Model.Junction.Isec	Node number for the reaches	TABINT	2
Model.Junction.Isecvo	Node number for the reaches	TABINT	2
Model.Junction.EndReach	End of the reach	TABINT	2
Model.Inflow.Name	Name of the inflow	STRING	1
Model.Inflow.ReachNum	Number of the corresponding reach	INT	1
Model.Inflow.RelAbscissa	Relative abscissa of the inflow	DOUBLE	1
Model.Inflow.Length	Length of the inflow	DOUBLE	1
Model.Inflow.UpNode	Number of the upstream node	INT	1
Model.Inflow.DownNode	Number of the downstream node	INT	1
Model.Inflow.GraphNum	Number of the graph	INT	1
Model.Inflow.Discharge	Value of the inflow (m^3/s or m^3/s/m)	DOUBLE	1
Model.Dam.ReachNum	Reach number of the dam	INT	0
Model.Dam.RelAbscissa	Relative abscissa of the dam	DOUBLE	0
Model.Dam.Node	Node number of the dam	INT	0
Model.Dam.BreakType	Type of dam breaking	INT	0
Model.Dam.CrestLevel	Crest level of the dam	DOUBLE	0
Model.ExternalInflow.Number	Number of the storage area with an external inflow	INT	1
Model.ExternalInflow.GraphNumber	Number of the graph for an external inflow	INT	1
Model.ExternalInflow.Discharge	Value of the external inflow (m3/s)	DOUBLE	1
Model.VDCrossSection.B1	Main channel width	TABDOUBLE	2
Model.VDCrossSection.B2	Floodplain width	TABDOUBLE	2
Model.VDCrossSection.BS	Width of the ineffective flow area	TABDOUBLE	2
Model.VDCrossSection.P1	Main channel wetted perimeter	TABDOUBLE	2
Model.VDCrossSection.P2	Floodplain wetted perimeter	TABDOUBLE	2
Model.VDCrossSection.S1	Main channel wetted area	TABDOUBLE	2
Model.VDCrossSection.S2	Floodplain wetted area	TABDOUBLE	2
Model.VDCrossSection.S2G	Left floodplain wetted area	TABDOUBLE	2
Model.VDCrossSection.SS	Wetted ineffective flow area	TABDOUBLE	2
Model.VDCrossSection.C	Celerity	TABDOUBLE	2

Model.VDCrossSection.Conv1	Conveyance of the main channel	TABDOUBLE	2
Model.VDCrossSection.Conv2	Conveyance of the floodplain	TABDOUBLE	2
Model.VDCrossSection.Pr	Pressure	TABDOUBLE	2
Model.VDCrossSection.Inv	Riemann invariant	TABDOUBLE	2
Model.VDCrossSection.S1D	Shifted mesh - Main channel wetted area	TABDOUBLE	2
Model.VDCrossSection.S2D	Shifted mesh - Floodplain wetted area	TABDOUBLE	2
Model.VDCrossSection.SSD	Shifted mesh - Ineffective flow area	TABDOUBLE	2
Model.VDCrossSection.PrD	Shifted mesh - Pressure	TABDOUBLE	2
Model.VDCrossSection.BD	Shifted mesh - Width	TABDOUBLE	2
Model.VDCrossSection.DebD	Shifted mesh - Conveyance	TABDOUBLE	2
Model.CSectionAbsX	Cross section in absolute abscissa ?	BOOL	0
Model.HeadLossEnlarg	Automatic head loss in case of enlargement ?	BOOL	0
Model.ImpSupCritKern	Implicitation of the super-critical kernel ?	BOOL	0
Model.HeadLossJunc	Automatic head losses at junctions ?	BOOL	0
Model.StoArea	Storage areas activation ?	BOOL	0
Model.RecVar	Recorded variables	TABBOOL	1
Model.CompVar	Computed variables	TABBOOL	1
Model.PrintComp	Write the listing along with the computation progress	BOOL	0
Model.PrintVertCSection	Write the vertical discretisation of the cross sections	BOOL	0
Model.HotStart	Hot start ?	BOOL	0
Model.InefFlowArea	Ineffective flow area ?	BOOL	0
Model.InterpFriction	Linear interpolation of the friction coefficients ?	BOOL	0
Model.ProgOverFlowIFA	Type of progressive overflow for the ineffective flow areas	BOOL	0
Model.ProgOverFlowFP	Type of progressive overflow for the floodplains	BOOL	0
Model.FricVertWall	Conservation of friction on the vertical walls ?	BOOL	0
Model.VarTimeStep	Variable time step ?	BOOL	0
Model.ImpFric	Implicitation of friction ?	BOOL	0
Model.ValidComp	Validate the computation ?	BOOL	0
Model.DamBrkFldWave	Computation of a dam break flood wave ?	BOOL	0
Model.Zbot	Bottom level (m)	TABDOUBLE	1

Model.LevRightBk	Level of the right bank (m)	TABDOUBLE	1
Model.LevLeftBk	Level of the left bank (m)	TABDOUBLE	1
Model.Abac	Abacus for the head losses at junctions	TABDOUBLE	3
Model.Heps	Miniam water depth (m)	DOUBLE	0
Model.FricCoefFP	Friction coefficients for the floodplains	TABDOUBLE	1
Model.FricCoefMainCh	Friction coefficients for the main channel	TABDOUBLE	1
Model.LocalHeadLoss	Local head losses	TABDOUBLE	1
Model.XDT	Relative position of nodes / cross sections	TABDOUBLE	1
Model.X	Abscissa of nodes (m)	TABDOUBLE	1
Model.CourantNum	Maximal Courant number	DOUBLE	0
Model.MaxCompTime	Maximal computation time (s)	DOUBLE	0
Model.InitTime	Initial time of the computation (s)	DOUBLE	0
Model.DT	Time step (s)	DOUBLE	0
Model.LimFroude	Limit Froude number for the boundary conditions	DOUBLE	0
Model.DZwave	Level threshold identifying the wave (m)	DOUBLE	0
Model.ResultFmt2	Result format #2	INT	0
Model.NodeRes	Nodes where the results are recorded	TABINT	1
Model.RecOption	Option for the records	INT	0
Model.ResultFmt	Result format	INT	0
Model.RecNbFstTimeStep	Recording : number of the first time step to write	INT	0
Model.RecNTimeStep	Recording : write data all N time steps	INT	0
Model.RecListNTimeStep	Recording : write to the listing file all N time steps	INT	0
Model.AlgoNet	Algorithm for the network solution	TABINT	1
Model.1DMesh	Computational method for the 1D mesh generator	INT	0
Model.IDT	Upstream cross section of a node	TABINT	1
Model.GeoFileFmt	Format of the geometry file	INT	0
Model.FricLaw	Type of friction law	INT	0
Model.MaxNbTimeStep	Maximal number of time steps	INT	0
Model.StopCriteria	Criteria for stopping calculations	INT	0
Model.Regime	Regime : steady or unsteady	INT	0
Model.CSectionLayout	Cross section layout	INT	0
Model.ValidType	Type of validation (number)	INT	0

Model.Kernel	Computational kernel : Steady, Unsteady subcritical or Super-critical	INT	0
Model.Version	MASCARET version	INT	0
Model.Title	Title of the computation	STRING	0
Model.DZ	Features of the nodes	TABDOUBLE	1
Model.XD	Abscissa of interfaces	TABDOUBLE	1
Model.DZD	Vertical discretisation of interfaces	TABDOUBLE	1
Model.FirstCSReach	First cross section of a reach	TABINT	1
Model.LastCSReach	Last cross section of a reach	TABINT	1
Model.RelXFirstNdReach	Relative abscissa of the first node of a reach	TABDOUBLE	1
Model.RelXLastNdReach	Relative abscissa of the last node of a reach	TABDOUBLE	1
Model.Opt	Optimisation of the super-critical kernel	BOOL	0
Model.F1	Pulse function	TABDOUBLE	2
Model.Boussinesq	Non-hydrostatic terms for the super-critical kernel ?	BOOL	0
Model.NoConvection	Reduced momentum equation for sub-critical kernel ?	BOOL	0
Model.CQMV	Lateral inflow in the momentum equation ?	INT	0

6.2 Variables de l'état

NOM DE LA VARIABLE	DESCRIPTION	TYPE	DIMENSION
State.DPDZ2	Derivative of the floodplain wetted perimeter with respect to the level	TABDOUBLE	1
State.DPDZ1	Derivative of main channel wetted perimeter with respect to the level	TABDOUBLE	1
State.Qspilled	Spilled discharge (lateral weir)	TABDOUBLE	1
State.Qinflow	Inflow discharge	TABDOUBLE	1
State.Airs	State of the 2D junction	TABDOUBLE	2
State.W	State of the 2D junction	TABDOUBLE	3
State.YNode	Water depth (super-critical kernel)	TABDOUBLE	1
State.CNode	Celerity (super-critical kernel)	TABDOUBLE	1
State.UNode	Speed (super-critical kernel)	TABDOUBLE	1
State.XFron	Front wave position	TABDOUBLE	1
State.RH2	Hydraulic radius of the floodplain	TABDOUBLE	1
State.RH1	Hydraulic radius of the main channel	TABDOUBLE	1
State.BS	Width of the ineffective flow area	TABDOUBLE	1
State.B2	Width of the floodplain	TABDOUBLE	1
State.B1	Width of the main channelr	TABDOUBLE	1
State.P2	Floodplain wetted perimeter	TABDOUBLE	1
State.P1	Main channel wetted perimeter	TABDOUBLE	1
State.Froude	Froude number	TABDOUBLE	1
State.Beta	beta coefficient of the Debord's formula	TABDOUBLE	1
State.S2	Floodplain wetted area	TABDOUBLE	1
State.S1	Main channel wetted area	TABDOUBLE	1
State.SS	Wetted ineffective flow area	TABDOUBLE	1
State.Q2	Floodplain discharge	TABDOUBLE	1
State.Q1	Main channel discharge	TABDOUBLE	1
State.V1	Velocity in main channel	TABDOUBLE	1
State.V2	Velocity in floodplain	TABDOUBLE	1
State.Y	Water depth	TABDOUBLE	1
State.VOL	Volume of the active channel	TABDOUBLE	1

State.VOLS	Volume of the ineffective flow areas	TABDOUBLE	1
State.PreviousTime	Previous Time	DOUBLE	0
State.TimeStepNum	Time step number	INT	0
State.SimulPhase	Indicator of the simulation phase	INT	0
State.DT	Time step (s)	DOUBLE	0
State.Q	Total discharge (m^3/s)	TABDOUBLE	1
State.Z	Water level (m)	TABDOUBLE	1
State.Link.Discharge	Discharge (m^3/s)	DOUBLE	1
State.Link.PrevDischarge	Previous discharge (m^3/s)	DOUBLE	1
State.Link.MaxDischarge	Maximal discharge value (m^3/s)	DOUBLE	1
State.Link.MaxDischTime	Time (s) corresponding to the maximal discharge value	DOUBLE	1
State.Link.ExchangeV	Exchange velocity	DOUBLE	1
State.Link.MaxV	Maximal velocity value	DOUBLE	1
State.Link.MaxVTime	Time (s) corresponding to the maximal velocity value	DOUBLE	1
State.Link.DQDZus	Derivative of the discharge with respect to the level in the upstream storage area	DOUBLE	1
State.Link.DQDZds	Derivative of the discharge with respect to the level in the downstream storage area	DOUBLE	1
State.Link.DQDZsto	Derivative of the discharge with respect to the storage area level	DOUBLE	1
State.Link.DQDZriv	Derivative of the discharge with respect to the river level	DOUBLE	1
State.StoArea.Level	Water level (m)	DOUBLE	1
State.StoArea.Surface	Surface (m^2)	DOUBLE	1
State.StoArea.Volume	Volume (m^3)	DOUBLE	1
State.StoArea.InitVolume	Initial volume (m^3)	DOUBLE	1
State.StoArea.VolStatement	Volume statement	DOUBLE	1
State.StoArea.ErrVolStatement	Error volume statement	DOUBLE	1
State.StoArea.DzSto	Level variation	DOUBLE	1
State.StoArea.MaxLevel	Maximal water level (m)	DOUBLE	1
State.StoArea.MaxTime	Time corresponding to the maximal level	DOUBLE	1
State.JGNODE	index of the vertical discretisation GNode	TABINT	1
State.JDNODE	index of the vertical discretisation DNODE	TABINT	1
State.IFIGE	index of the vertical discretisation Fige	TABINT	1

State.FLUX	Flux of the Roe's solver (FV)	TABDOUBLE	2
State.Flow	Mass flow	TABDOUBLE	1
State.DTRezo	Time step (REZO - subcritical kernel)	DOUBLE	0
State.Rezomat.SOLV	Linear solver (REZO - subcritical kernel)	INT	0
State.Rezomat.N	Matrix rank (REZO - subcritical kernel)	INT	0
State.Rezomat.NNZ	Number of non-zero elements (REZO - subcritical kernel)	INT	0
State.Rezomat.NN	Work aray dimension SNR (Y12M solver)	INT	0
State.Rezomat.NN1	Work aray dimension RNR (Y12M solver)	INT	0
State.Rezomat.IHA	DWork aray dimension HA (Y12M solver)	INT	0
State.Rezomat.KL	Lower bandwidth (LAPACK-DGBSV solver)	INT	0
State.Rezomat.KU	Upper bandwidth (LAPACK-DGBSV solver)	INT	0
State.Rezomat.LDAB	Workspace (LAPACK-DGBSV solver)	INT	0
State.Rezomat.IFAIL	Error indicator	INT	0
State.Rezomat.ipiv	Permutation matrix (LAPACK-DGBSV solver)	TABINT	1
State.Rezomat.IFLAG	Numerical options and statistics (Y12M solver)	TABINT	1
State.Rezomat.RowA	Row number of a non-zero element	TABINT	1
State.Rezomat.ColA	Column number of a non-zero element	TABINT	1
State.Rezomat.snr	Work array SNR	TABINT	1
State.Rezomat.rnr	Work array RNR	TABINT	1
State.Rezomat.ha	Work array HA	TABINT	2
State.Rezomat.noVarDQ	Pointers to DQ solutions (discharge)	TABINT	1
State.Rezomat.noVarDZ	Pointers to DZ solutions (level)	TABINT	1
State.Rezomat.noVarDQl	Pointers to DQl solutions (link)	TABINT	1
State.Rezomat.noVarDZc	Pointers to DZc solutions (Storage area)	TABINT	1
State.Rezomat.KdNode	Kind of nodes : 0->undefined, -1 ->discharge, -2 ->level, >0 ->junction	TABINT	1
State.Rezomat.headJunction	For each junction, first element (node number)	TABINT	1
State.Rezomat.nxtNdJunction	Next element in the list of nodes of a junction	TABINT	1
State.Rezomat.WeirNd	Positions of weirs — nodes	TABINT	1
State.Rezomat.LinkNd	Positions of links — nodes	TABINT	1
State.Rezomat.NdLink	For each link, gives the upstream node or 0 if the link is not connected to the river	TABINT	1

State.Rezomat.AFLAG	Numerical parameters (Y12M solver)	TABDOUBLE	1
State.Rezomat.valA	Values of the non-zero elements	TABDOUBLE	1
State.Rezomat.b	RHS vector of the linear system : A.x = b	TABDOUBLE	1
State.Rezomat.pivot	Pivot value of the Gauss elimination	TABDOUBLE	1
State.Rezomat.AB	Band matrix (LAPACK-DGBSV solver)	TABDOUBLE	2
State.NBARAD	Number of downstream dams (super-critical kernel)	INT	0
State.IDEB	Beginning of the computation area per reach (super-critical kernel)	TABINT	1
State.IFIN	End of the computation area per reach (super-critical kernel)	TABINT	1
State.ITEM0	ITEM0 (super-critical kernel)	TABINT	1
State.Save.H2OIB	Initial water mass - REACH	TABDOUBLE	1
State.Save.H2OTB	Total water mass - REACH	TABDOUBLE	1
State.Save.H2OEB	Incoming water mass - REACH	TABDOUBLE	1
State.Save.H2OSB	Outgoing water mass - REACH	TABDOUBLE	1
State.Save.H2OIC	Initial water mass - JUNCTION	TABDOUBLE	1
State.Save.H2OTC	Total water mass - JUNCTION	TABDOUBLE	1
State.Save.H2OEC	Incoming water mass - JUNCTION	TABDOUBLE	1
State.Save.H2OSC	Outgoing water mass - JUNCTION	TABDOUBLE	1
State.Save.H2OTBS	Total outgoing water mass - REACH	TABDOUBLE	1
State.Save.H2OIBS	Initial outgoing water mass - REACH	TABDOUBLE	1
State.Save.SPREC	Previous wetted area	TABDOUBLE	1
State.Save.QPREC	Previous discharge	TABDOUBLE	1
State.Save.H2OIG	Global initial water mass	DOUBLE	0
State.Save.H2OIGS	Outgoing global initial water mass	DOUBLE	0
State.Save.H2OTG	Global total water mass	DOUBLE	0
State.Save.H2OTGS	Outgoing global total water mass	DOUBLE	0
State.Save.H2OEG	Global incoming water mass	DOUBLE	0
State.Save.H2OSG	Global outgoing water mass	DOUBLE	0
State.ZINIT	Initial water level (super-critical kernel)	TABDOUBLE	1

7 Fichiers d'inclusions

7.1 Langage C : 'apimascaret.h'

voir ce fichier : ∼/Mascaret/src/API/cpp/apimascaret.h

7.2 Langage Fortran : 'm_apimascaret_i.f90'

voir ce fichier : ∼/Mascaret/src/ModulesAPI/m_apimascaret_i.f90

8 Conclusion

L'API permet un nouveau type d'usage de `MASCARET` tourné vers le calcul intensif et la simulation fine. A moyen terme, l'API devrait devenir la seule façon d'utiliser le code de calcul.

En attendant un certain nombre d'évolutions sont possibles et souhaitées :

- ajout de nouvelles fonctions comme la possibilité d'importation de flux XML ;
- rendre modifiable la taille de certaines variables ;
- faciliter la consultation d'un tableau complet ;
- ajouter le module de qualité d'eau ;
- etc.