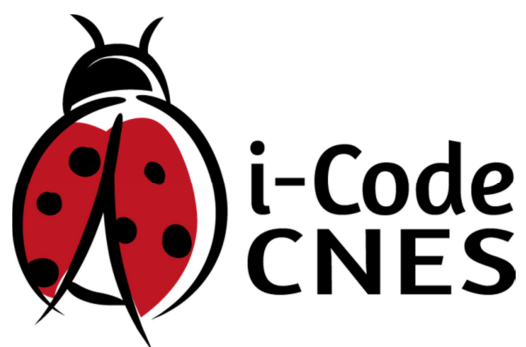


Manuel Utilisateur

i-Code

DNO/DA /AQ - 2017.0002478
Version 2.0.4



**CENTRE NATIONAL
D'ÉTUDES SPATIALES**

Siège
2, place Maurice Quentin
75039 Paris cedex 01
tél. : 33 (0)1 44 76 75 00

Direction des lanceurs
52, rue Jacques Hillairet
75612 Paris cedex
tél. : 33 (0)1 80 97 71 11

Centre spatial de Toulouse
18, avenue Edouard Belin
31401 Toulouse cedex 9
tél. : 33 (0)5 61 27 31 31

Centre spatial guyanais
BP 726
97387 Kourou cedex
tél. : 33 (0)5 94 33 51 11

TABLE DES MATIERES

1. INTRODUCTION	4
2. DOCUMENTS DE REFERENCE	4
3. CONFIGURATION DE L'OUTIL	4
3.1. ACTIVATION / DESACTIVATION D'UN LANGAGE	5
3.2. ACTIVATION / DESACTIVATION DE LA VERIFICATION DU RESPECT DES REGLES DE CODAGE ET DU CALCUL DE METRIQUE	6
3.3. CONFIGURATION DES REGLES DE CODAGE	6
3.4. CONFIGURATION DES METRIQUES	7
4. VERIFICATION DU RESPECT DES REGLES DE CODAGE	8
4.1. LANCEMENT DE L'ANALYSE	8
4.2. AFFICHAGE DES RESULTATS	8
4.3. PARCOURS DES VIOLATIONS DETECTEES DANS LE CODE	10
4.4. EXPORT DES RESULTATS	12
5. MESURE DE LA COMPLEXITE	13
5.1. LANCEMENT DU CALCUL DES METRIQUES	13
5.2. AFFICHAGE DES RESULTATS	13
5.3. PARCOURS DES VIOLATION DE METRIQUE DANS LE CODE	13
5.4. EXPORT DES RESULTATS	13
6. TABLES DES VIOLATIONS DETECTEES	14
6.1. REGLES COMMUNES	14
6.2. REGLES SPECIFIQUES	18
6.2.1. FORTRAN 77	18
6.2.2. FORTRAN 90	20
6.2.3. SHELL	26
7. TABLES DES METRIQUES CALCULEES	28
8. MESSAGES UTILISATEUR	31
8.1. MESSAGES DES VIOLATIONS	31
8.1.1. REGLES COMMUNES	31
8.1.2. FORTRAN 77	32
8.1.3. FORTRAN 90	33
8.1.4. SHELL	35

Table des modifications

Version	Date	Objet
1.0.0	05/12/2014	Initialisation du document
1.1.0	23/01/2015	Prise en compte des modifications des standards Fortran 77, Fortran 90 et des règles communes
2.0.0	03/06/2015	Intégration des règles Shell
2.0.1	03/09/2015	Ajout de l'export xml
2.0.2	21/12/2015	Ajout des marqueurs de violations de métriques et de la fonctionnalité double-clique
2.0.3	14/03/2016	Fusion des modifications des documentations 1.0, 2.0.1, 2.0.2
2.0.4	16/12/2016	Mise à jour suite à la validation

1. Introduction

Ce document constitue le manuel utilisateur de l'outil i-Code. Il vise à décrire le fonctionnement et l'utilisation de ce plug-in eclipse.



Avant d'utiliser i-Code, il est fortement conseillé de :

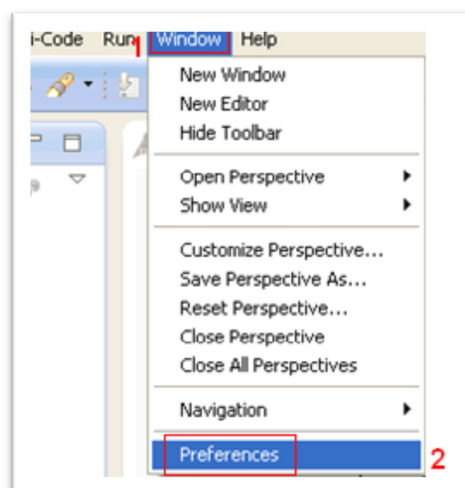
- prendre connaissance de l'environnement eclipse. La documentation d'eclipse est disponible sur le site en référence [R1].
- prendre connaissance des règles normatives du CNES sur les langages à analyser [R2] [R3] [R4]

2. Documents de référence

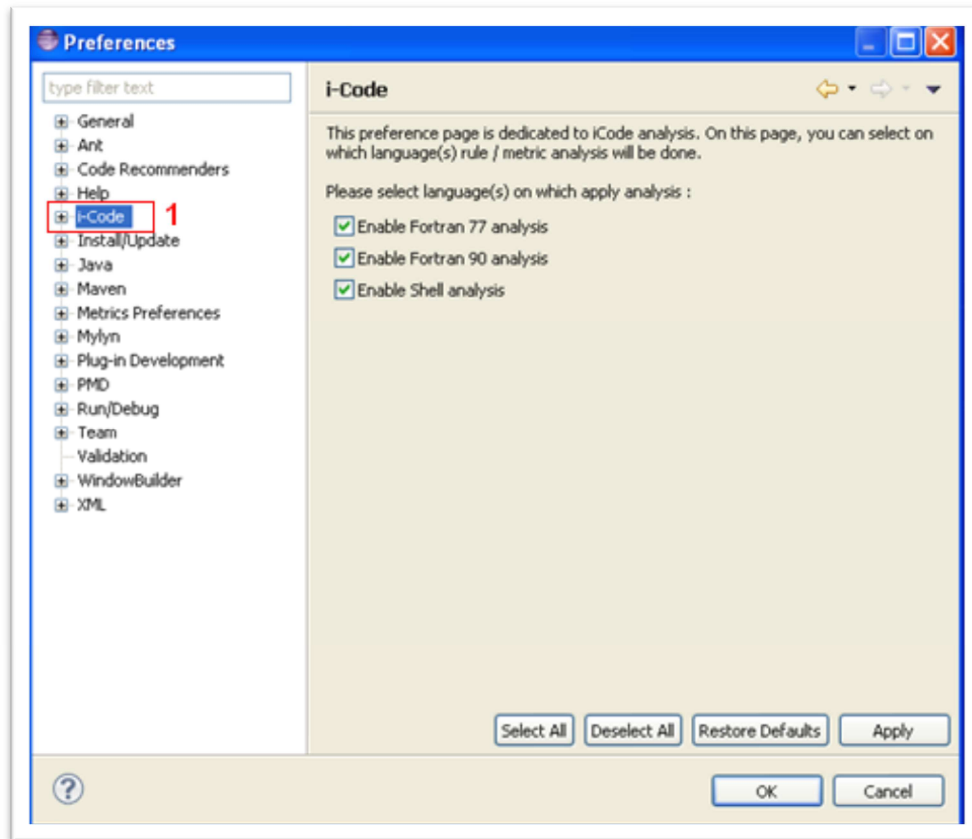
- [R1]. Documentation d'Eclipse :
 - <http://www.eclipse.org/documentation/>
- [R2]. Règles pour l'utilisation du langage Fortran 77
 - RNC-CNES-Q-HB-80-505 Version 7
- [R3]. Règles pour l'utilisation du langage Fortran 90
 - RNC-CNES-Q-HB-80-517 Version 5
- [R4]. Règles communes pour l'utilisation des langages de programmation
 - RNC-CNES-Q-HB-80-501 Version 5
- [R5]. Règles pour l'utilisation du SHELL
 - RNC-CNES-Q-HB-80-516 Version 6

3. Configuration de l'outil

L'ensemble de la configuration de l'outil est accessible dans les pages de préférences d'eclipse. Pour ouvrir les pages de préférence d'eclipse, cliquer sur **Windows > Préférences**



Dans la fenêtre des préférences, un item « i-Code » est visible dans la partie gauche, c'est dans cette partie que la configuration i-Code peut être modifiée.

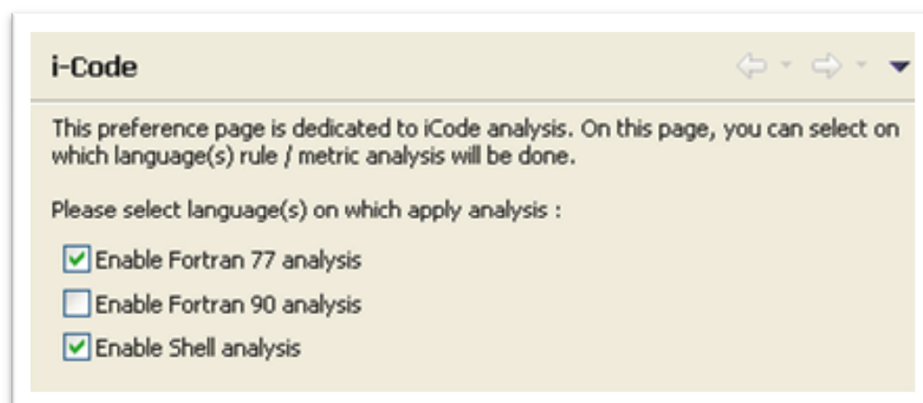


3.1. ACTIVATION / DESACTIVATION D'UN LANGAGE

i-Code permet de vérifier le respect des règles de codage et de calculer les métriques de différents langages.

L'activation/désactivation d'un langage se fait sur la page de préférence principale i-Code.

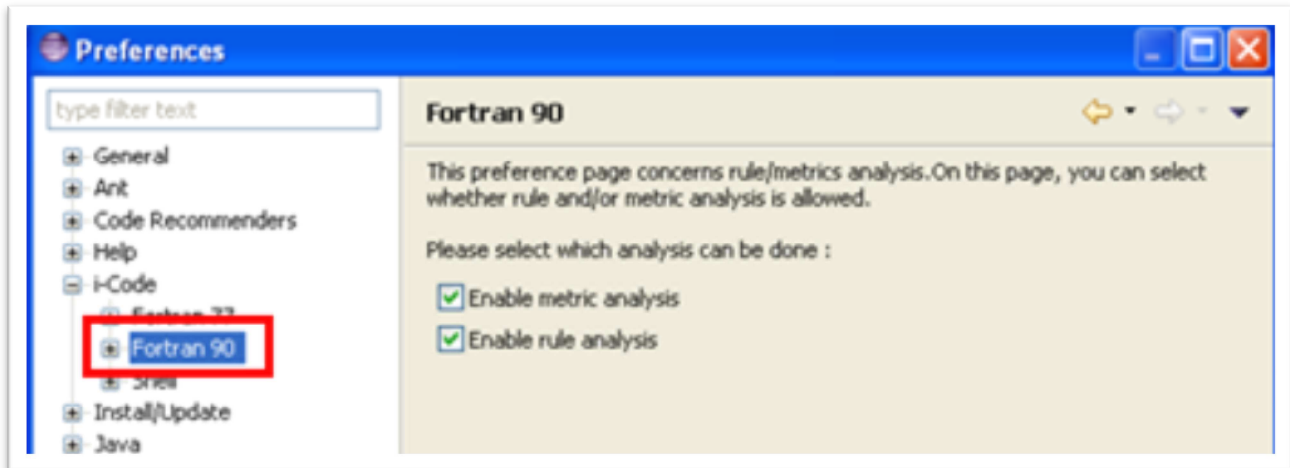
L'activation d'un langage rend accessible le menu correspondant à ce langage dans le menu i-Code, nécessaire pour lancer une analyse. Elle rend également accessible les pages de préférence permettant de configurer les analyses de ce langage.



3.2. ACTIVATION / DESACTIVATION DE LA VERIFICATION DU RESPECT DES REGLES DE CODAGE ET DU CALCUL DE METRIQUE

Pour chaque langage, i-Code permet de vérifier le respect des règles de codage et le calcul des métriques. L'activation/désactivation de ces deux fonctionnalités se fait sur la page de préférence spécifique à chaque langage.

L'activation d'une fonctionnalité rend accessible le sous-menu correspondant dans le menu du langage concerné.

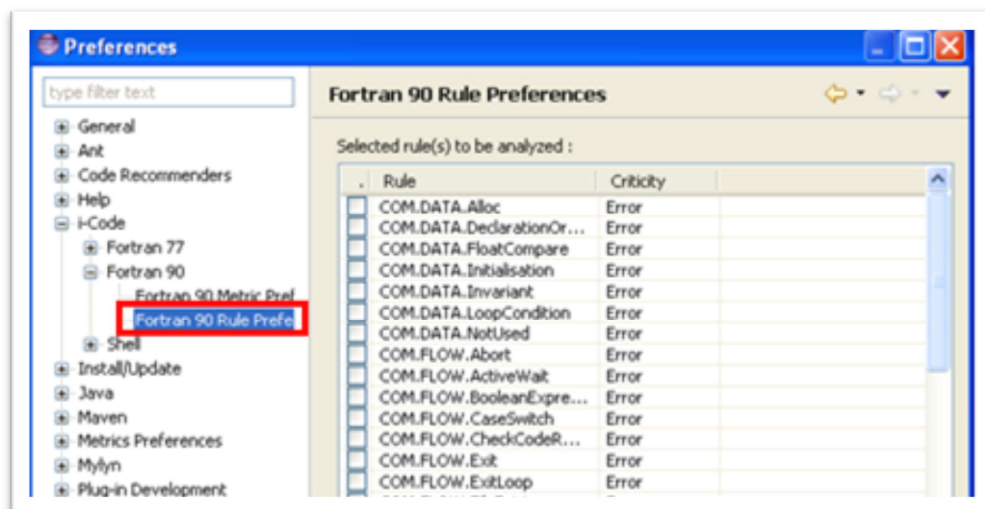


3.3. CONFIGURATION DES REGLES DE CODAGE

La configuration des règles de codage vérifiées par i-Code se fait sur la page de préférence relative aux règles de codage du langage concerné.

La page de préférence relative aux règles d'un langage présente l'ensemble des règles que peut vérifier l'outil. Cette table présente la liste des règles ainsi que leur criticité (par défaut toutes les règles sont en criticité error).

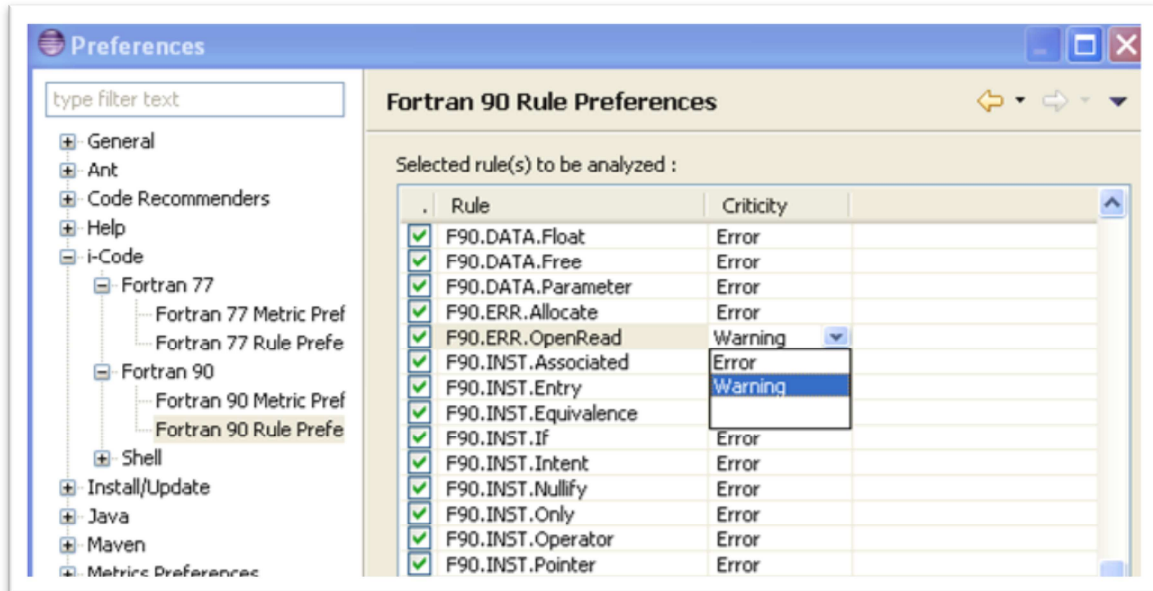
Pour activer/désactiver une règle, il suffit de cocher/décocher la case correspondante dans la table.



- Le check de la règle est *active* : dans la prochaine analyse, cette règle sera analysée.
- Le check de la règle est *désactivé* : cette règle ne sera pas analysée à la prochaine analyse.
- Select all : toutes les règles seront activées.
- Deselect all : toutes les règles seront désactivées.

- Restore defaults : valeurs par défaut mise à jour (toutes les règles activées)
- Apply : Valider et sauvegarder les modifications.

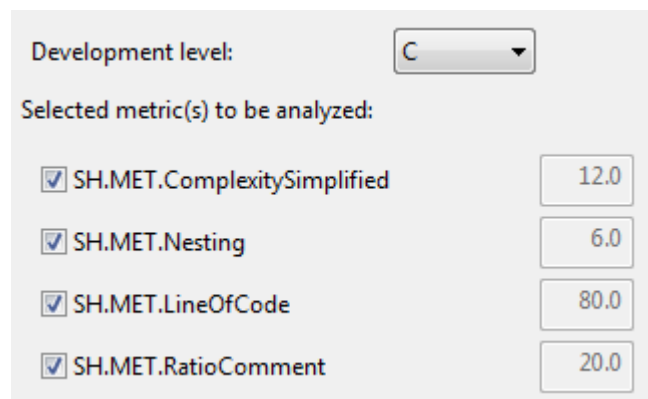
Pour modifier la criticité d'une règle de codage, il suffit de sélectionner la criticité choisie dans la table.



3.4. CONFIGURATION DES METRIQUES

La configuration des métriques se fait sur la page de préférence relative aux métriques du langage concernée.

Pour activer/désactiver une métrique, il suffit de cocher/décocher la case correspondante dans la table.



- Le niveau de développement permet de prendre les valeurs RNC définies pour chaque niveau A, B, C et D. Un niveau Custom permet de modifier manuellement les valeurs
- Le check de la métrique est *active* : dans la prochaine analyse, cette métrique sera analysée.
- Le check de la métrique est *désactivé* : cette métrique ne sera pas analysée au prochain analyse.
- *Restore defaults* : valeurs par défaut mise à jour (toutes les métriques activées)
- *Apply* : Valider et sauvegarder les modifications.

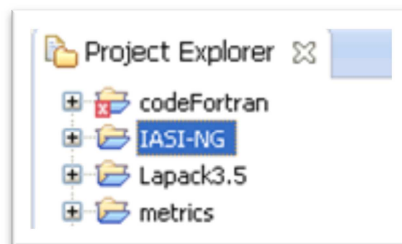
4. Vérification du respect des règles de codage et calcul des métriques

i-Code CNES permet de vérifier certaines règles de codage des standards de Fortran 77 [R2], Fortran 90 [R3], règles communes [R4] et Shell [R5], de visualiser les résultats dans un tableau et d'accéder rapidement à la ligne incriminée. Il permet également de calculer 4 métriques, et d'exporter l'ensemble des résultats aux formats xml et csv.

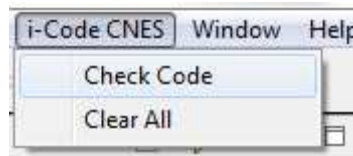
4.1. LANCEMENT DE L'ANALYSE

Prérequis : L'ensemble des fichiers à analyser est disponible dans projet eclipse.

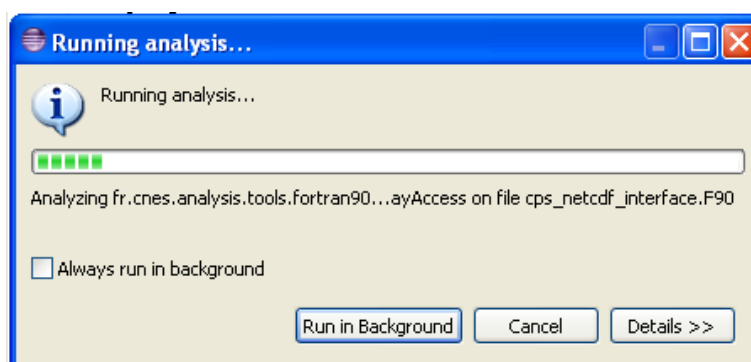
1. Sélectionner le projet ou le fichier à analyser.



2. Lancer l'analyse via le menu i-Code (menu principal ou menu contextuel)



3. Une barre de progression apparaît. A la fin de l'analyse, les résultats s'affichent dans la vue « I-Code CNES Violations » et « i-Code CNES Metrics »



4.2. AFFICHAGE DES RESULTATS

La vue *i-Code CNES Violations* présente les résultats de l'analyse pour chaque règle.

Tasks

i-Code CNES Metrics

i-Code CNES Violations

Filter : Enter part of file path, name, rule name or function's name... (not case sensitive)

!	Rule	Line	Num...	Message
	F90.TYPE.Real	--	3	
	fortran90.f90	--	3	
	PROGRAM TEST	14	--	It misses the declaration SELE...
	PROGRAM TEST	43	--	It misses the declaration SELE...
	PROGRAM TEST	44	--	It misses the declaration SELE...
	F90.TYPE.Integer	--	10	
	F90.REF.Open	--	1	
	F90.REF.Label	--	3	
	F90.NAME.KeyWords	--	1	
	F90.INST.Pointer	--	2	
	F90.INST.Operator	--	1	

La signification des colonnes de la vue est la suivante :

- *Rule* : nom de la règle qui provoque l'erreur
- *Line* : Numéro de ligne où est détectée l'erreur
- *Number of violations* : nombre de violations total.
- *Message* : détails sur l'erreur (variable qui lance l'erreur, petite description, etc.)

L'ensemble des violations d'une même règle est présentée sous forme arborescente. Le premier niveau correspond au fichier présentant la violation, le deuxième niveau la méthode et la ligne.

La vue i-Code CNES Metrics présente l'ensemble des métriques calculées

Tasks

i-Code CNES Metrics

i-Code CNES Violations

Metric	Total	Mean	Minimum	Maximum
▶ F77.MET.ComplexitySimplified	--	7.25	1.0	26.0
▶ F77.MET.LineOfCode	172.0	42.75	3.0	154.0
▶ F77.MET.Nesting	--	0.5	0.0	2.0
▶ F77.MET.RatioComment	--	--	28.638496	36.363636
▶ F90.MET.ComplexitySimplified	--	2.25	1.0	4.0
▶ F90.MET.LineOfCode	177.0	38.5	8.0	118.0
▶ F90.MET.Nesting	--	0.75	0.0	1.0
▶ F90.MET.RatioComment	--	--	25.0	39.772728

Les dépassements de seuil sont identifiés en rouge dans le tableau.

La signification des colonnes est la suivante :

- *Metric* : nom de la métrique qui est analysée
- *Total* : addition de toutes les valeurs de cette métrique pour tous les fichiers analysés
- *Mean* : moyenne des métriques pour les fichiers analysés
- *Minimum* : valeur minimum dans tous les fichiers
- *Maximum* : valeur maximum dans tous les fichiers

4.3. PARCOURS DES VIOLATIONS DETECTEES DANS LE CODE

La table des résultats présentant le fichier et la ligne d'une violation, il suffit d'ouvrir le fichier incriminé pour voir l'erreur dans le contexte. Pour cela, i-Code permet d'ouvrir directement le fichier à la ligne correspondant à la violation par un double-clic sur la violation.

Un marqueur dans la colonne de gauche de l'éditeur permet d'identifier plus facilement les violations dans le code. Le survol d'un marqueur avec la souris permet d'afficher le nom de la règle dans un tooltip.

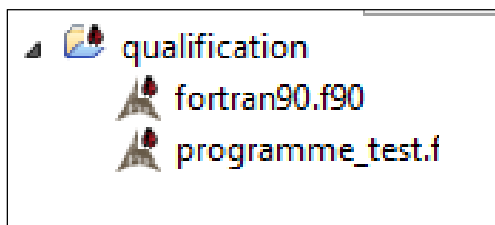
2. ouverture du fichier

3. positionner la souris

1. double clic

!	Rule	Line	Number of violations	Message
+	COM.DATA.NotUsed	--	4902	
+	COM.FLOW.Exit	--	1655	
+	GS_interface.F90	--	26	
+	MSP_ACCES.F90	--	46	
+	MSP_ACTSOL_DEF.F90	--	7	
+	subroutine MSP_creer_actsol	665	--	
+	subroutine MSP_creer_actsol	674	--	
+	subroutine MSP_creer_actsol	688	--	
+	subroutine MSP_creer_actsol	699	--	

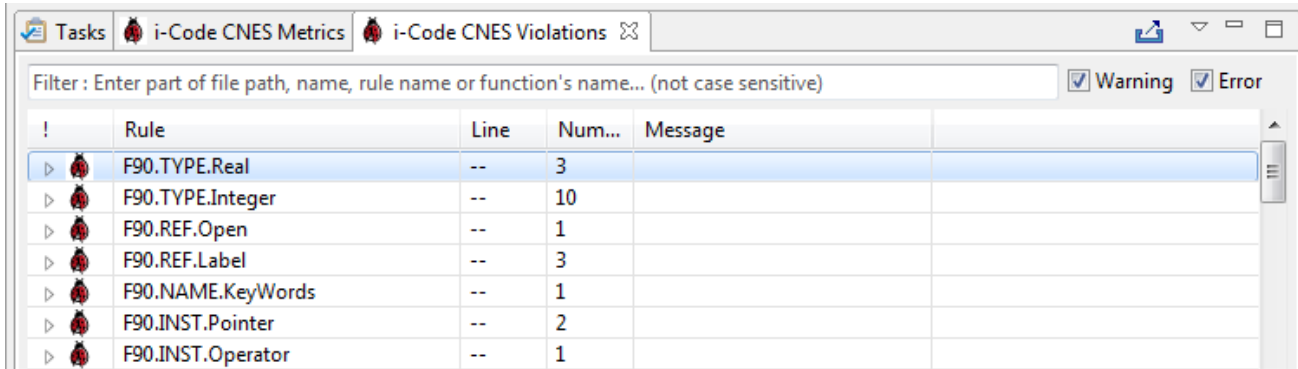
Dans La vue ProjectExplorer, le marqueur apparait sur les fichiers présentant des violations aux règles de codage.



4.4. EXPORT DES RESULTATS

Les résultats de l'analyse peuvent être exportés dans un fichier au format csv.

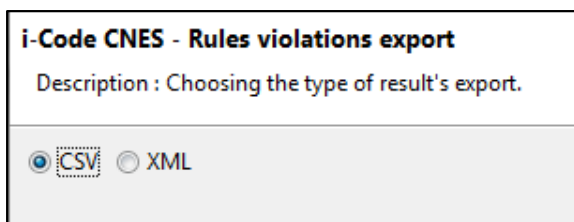
1. Cliquer sur le bouton d'export en haut à droite de la vue « i-Code CNES Violations » ou « i-Code CNES Metrics »



The screenshot shows a software window titled "i-Code CNES Metrics". It features a filter bar at the top with the text "Filter : Enter part of file path, name, rule name or function's name... (not case sensitive)". Below the filter is a table with the following columns: "!", "Rule", "Line", "Num...", and "Message". The table contains several rows of data, with the first row highlighted in blue.

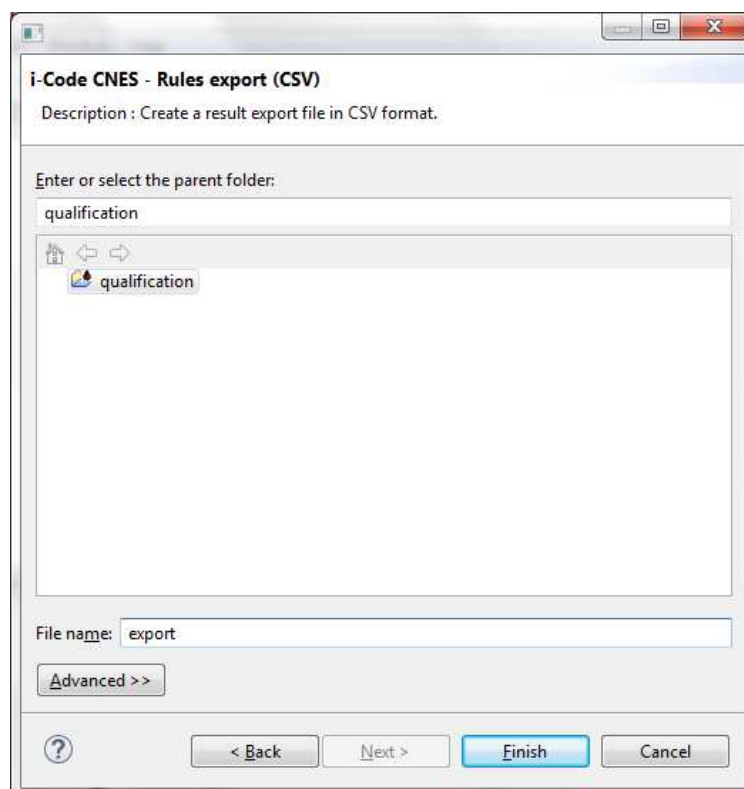
!	Rule	Line	Num...	Message
▶	F90.TYPE.Real	--	3	
▶	F90.TYPE.Integer	--	10	
▶	F90.REF.Open	--	1	
▶	F90.REF.Label	--	3	
▶	F90.NAME.KeyWords	--	1	
▶	F90.INST.Pointer	--	2	
▶	F90.INST.Operator	--	1	

2. Sélectionner le format d'export csv ou xml :

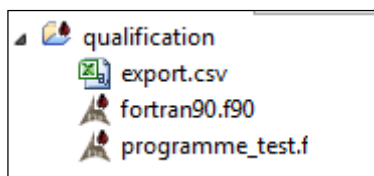


The screenshot shows a dialog box titled "i-Code CNES - Rules violations export". It contains a description: "Description : Choosing the type of result's export." Below the description are two radio buttons: "CSV" (which is selected) and "XML".

3. Indiquer le nom du fichier d'export et sa localisation.



4. Le fichier est créé dans le projet sélectionné



5. Tables des violations détectées

5.1. REGLES COMMUNES

Règle	Vérification	Couverture du standard
COM.DATA.Array	Obligation d'utiliser les tableaux à deux dimensions de manière à avoir "ligne x colonne"	Non
COM.DATA.DeclarationOrder	Fortran 77 : Non applicable	Non
	Fortran 90 : Les paramètres des fonctions doivent être déclarés, en premier les données d'entrée, après entrée/sortie puis les données de sortie. La déclaration se fait au travers du mot clé 'INTENT' puis 'IN' pour les données d'entrée, INOUT pour les entrées/sorties et OUT pour les sorties. S'il n'y a pas de paramètre INTENT, ce n'est pas possible de faire la vérification.	Oui
	Shell : non applicable	Non
COM.DATA.FloatCompare	Comparaison d'égalité/inégalité (.EQ., ==, .NE., /=) interdite entre des nombres réels (REAL, DOUBLE PRECISION ou COMPLEX).	Oui
	Shell : non applicable	Non
COM.DATA.Initialisation	Les variables doivent être initialisées avant d'être utilisées. Quand une variable est utilisée dans le code, le programme vérifie qu'elle est initialisée (nom de la variable puis signe d'égalité), sinon, il renvoie une erreur.	Oui
	Shell : Non implémenté	Non
COM.DATA.Invariant	Les données déclarées dans une sous-routine, fonction, etc et jamais modifiées (pas d'occurrence de la variable puis signe d'égalité) doivent être définies comme constantes	Oui
	Shell : Non implémenté	Non
COM.DATA.LoopCondition	Interdiction de modifier les données de condition de sortie des boucles à l'intérieur de celle-ci	Oui
COM.DATA.NotUsed	Fortran : Toute variable déclarée doit être utilisée, sinon une erreur est remontée.	Oui
	Shell : Toute variable déclarée doit être utilisée. Limitation : Les assignations dans les options de la commande awk peuvent engendrer des faux-positifs. Une variable est considérée comme utilisée si elle est utilisée avec \${variable}.	
COM.DATA.Using	Interdiction de réutiliser un objet local dans des traitements de type différent.	Non
COM.DESIGN.Alloc	Fortran : L'allocation et la désallocation des ressources doit être dans le même niveau (fonction, sous-routine, ...). Chaque fois que le mot 'DEALLOCATE' est trouvé, le programme vérifie qu'il y a le mot 'ALLOCATE' et que les deux utilisent la même ressource.	PC ¹
	Shell : non applicable	Non
COM.DESIGN.ActiveWait	Fortran : dans une boucle, les instructions SLEEP, WAIT et PAUSE sont interdites.	Oui
	Shell : La boucle WHILE [1] et le mot READ sont interdits	Oui
COM.FLOW.Abort	Fortran : Le mot STOP est interdit.	Oui

¹ Les ressources vérifiées sont les blocs de mémoire, pas les fichiers. Si le développeur encapsule les allocations et désallocations dans les sous-routines, l'application remonte une erreur.

Règle	Vérification	Couverture du standard
	Shell : Les mot KILL, PKILL et KILLALL sont interdit Limitation : Ne prend pas en compte les reutrn,break et exit, même s'ils peuvent interrompre l'exécution de commandes.	
COM.FLOW.BooleanExpression	Dans une instruction conditionnelle (IF, DO) il n'est pas possible de définir plus de cinq expressions conditionnelles (AND, OR, NEQV, XOR, EQV, NOT, LT, <, LE,<=, GT, >, GE, >=, EQ, ==, NE, /=).	Oui
COM.FLOW.CaseSwitch	Fortran77 : Non Applicable	Non
	Fortran90 : Obligation de finir l'instruction SWITCH avec DEFAULT, afin de traiter tous les cas possibles.	Oui
	Shell : Obligation de finir l'instruction CASE avec *), afin de traiter tous les cas possibles	Oui
COM.FLOW.CheckArguments	Fortran : Obligation de contrôler les paramètres passés à un programme	Non
	Shell : voir SH.FLOW.CheckArguments	
COM.FLOW.CheckCodeReturn	Fortran : Obligation de tester tous les retours de fonction	Oui
	Shell : renommée en SH.FLOW.CheckCodeReturn	Non
COM.FLOW.CheckUser	Fortran : Obligation de vérifier l'identité de l'utilisateur qui exécute un programme	Oui
	Shell : voir SH.FLOW.CheckUser	Non
COM.FLOW.Exit	Fortran : Interdiction d'implémenter plusieurs points de sortie dans les fonctions, procédures ou méthodes.	Oui
	Shell : Non implémenté	Non
COM.FLOW.ExitLoop	Interdiction d'implémenter plus d'une sortie dans les boucles.	Oui
COM.FLOW.FileExistence	Fortran : Avant d'ouvrir ou créer (mot clé OPEN, READ, WRITE) un fichier, doit apparaître l'instruction INQUIRE avec le flag EXIST sur le même fichier.	Oui
	Shell : Avant d'accéder à un fichier (> nom_fichier), il faut apparaître la vérification : if [! -f \$nom_fichier] Limitations : <ul style="list-style-type: none">Pas de détection dans les commandes \$(...) ou `...`Une variable (or redirections standards) sur laquelle est réalisée une redirection peut être interprétée comme un fichier	Oui
COM.FLOW.FilePath	Dans l'instruction OPEN, il est interdit d'utiliser directement le nom du fichier (fichier.txt). Le chemin d'accès doit être défini au travers d'une variable qui contient le chemin vers le fichier.	Oui
	Shell : Non implémenté	Non
COM.FLOW.Recursion	Fortran77 : Non Applicable	Non
	Fortran90 : Interdiction d'utiliser la récursivité. En Fortran, une fonction récursive est définie comme suit : RECURSIVE FUNCTION (params)	Oui
	Shell : Non implémenté	Non
COM.INST.BoolNegation	La double négation est interdite sur les expressions booléennes. Les négations sont définies avec le mot .NOT. donc les expressions suivantes ne sont pas permis : .NOT. (.NOT. a) -> (a) .NOT. (a .AND. .NOT. b) -> .NOT. a .OR. b	Oui
	Shell : Non implémenté	Non

Règle	Vérification	Couverture du standard
COM.INST.Brace	Fortran : Toute expression doit être parenthésée, ainsi le nombre de parenthèses ouvertes doit être supérieur ou égal au nombre d'opérateurs utilisés (+, -, *, /, **) $a + b * c$ $\rightarrow a + (b * c)$	Oui
	Shell : Non implémenté	Non
COM.INST.CodeComment	Fortran : Interdiction de commenter le code. Tout mot clé (ASSIGN, BACKSPACE, BLOCK DATA, CALL, ...) dans une ligne de commentaire est une erreur.. Le header (commentaires juste avant ou juste après de la déclaration de la fonction ou sous-routine) peut contenir ces mots.	Oui
	Shell : Interdiction de commenter le code. Le premier mot après le symbole de commentaire (#), ne doit pas être un mot clé (cd, date,...), ni une affectation de variable (var=). Limitation : un texte courant commençant par un mot clé du langage sera incorrectement détecté comme du code commenté. Ex : # date de mise à jour # set the starting value	Oui
COM.INST.GOTO	Fortran: L'instruction GO TO est interdite.	Oui
	Shell: non applicable	Non
COM.INST.Line	Fortran 77 : non applicable	Non
	Fortran 90 et Shell : Chaque ligne doit contenir maximum une expression. En Fortran il est possible d'implémenter plusieurs instructions dans une ligne grâce au point-virgule. $a = b + c ; d = e * f$ $\rightarrow a = b + c$ $d = e * f$	Oui
	Shell: Interdiction de réaliser plusieurs instructions sur une même ligne. Limitation : L'utilisation de mots clés dans les messages de commandes non encadrés de guillemets peut engendrer des erreurs sur l'analyse du fichier.	
COM.INST.LoopCondition	Dans une instruction de boucle, les comparaisons d'égalité (.EQ., ==) ou de différence (!=, .NE.) sont interdites.	Oui
	Shell : Non implémenté	Non
COM.NAME.Homonymy	Fortran : Une variable doit avoir un nom unique. Chaque fois qu'une variable est trouvée, le programme vérifie que le nom de cette variable n'est pas déjà utilisé dans le programme.	Oui
	Shell : Non implémenté	Non
COM.PRES.Data	Obligation de commenter par une description détaillée les objets importants.	Non
COM.PRES.Indent	Le code doit être indenté par espaces. Une ligne doit commencer à la même colonne que la ligne précédente. Après l'instruction DO, IF, WHILE, WHERE, SELECT et TYPE la ligne doit commencer dans une colonne supérieure, à l'exception de la fin de l'expression (dénoté par END). Exemple : $\begin{array}{l} \text{DO } i = 2, nb \\ \quad \text{somme} = \text{somme} + x(i) \\ \quad \text{IF (isnan(somme)) THEN} \\ \quad \quad \text{print *, 'somme is a NaN'} \\ \quad \quad \text{moy} = -1.0 \\ \quad \text{END IF} \\ \text{END DO} \end{array}$	Oui

Règle	Vérification	Couverture du standard
	Shell : Non implémenté	Non
COM.PRES.LengthLine	Une ligne de code doit contenir un maximum de 100 caractères. Le caractère 101 doit être écrit à la ligne suivante.	Oui
	Shell : Non implémenté	Non
COM.PROJECT.Analyser	Obligation de passer un outil d'analyse statique sur tous les codes sources d'un projet.	Non
COM.PROJECT.CodeCloning	Interdiction de dupliquer / cloner du code.	Non
COM.PROJECT.Header	Obligation de définir et d'appliquer les entêtes/cartouche de chaque module et fonctions en début de projet. <i>Shell : nommer COM.PRES.Header</i>	Oui
COM.PROJECT.Warnings	Obligation d'afficher tous les warnings et de les corriger	Non
COM.TYPE.Expression	Fortran : Dans une expression (une expression est définie par un opérateur comme +, -, *, /, **), Les variables doivent être de même type : soit REAL, soit INTEGER, etc.	Oui
	Shell : Non implémenté <i>Shell : Les variables d'une même expression doit être du même type : integer ou char/string définis selon première initialisation.</i>	Non

5.2. REGLES SPECIFIQUES

5.2.1. FORTRAN 77

Règle	Vérification	Couverture au standard
F77.BLOC.Common	Interdiction d'utiliser des COMMON blanc.	Oui
F77.BLOC.Else	Dans une instruction IF, le dernier ELSE IF doit toujours être suivi d'un ELSE.	Oui
F77.BLOC.File	Obligation d'utiliser les instructions OPEN et CLOSE pour accéder aux fichiers.	Non
F77.BLOC.Function	Obligation d'utiliser les parenthèses d'argument pour l'instruction FUNCTION, même s'il n'y a pas d'argument	Oui
F77.BLOC.Loop	Les boucles DO imbriquées doivent avoir des indicateurs de fermeture différents. Ce n'est pas possible de partager le label.	Oui
F77.DATA.Array	Obligation de déclarer explicitement les dimensions des tableaux. Par contre, est possible d'utiliser la notation * pour la dernière dimension mais toujours avec la justification d'un commentaire avant. <code>A(*), A(4, *), A(4, *, *), A(4, 4, *),</code> mais <code>A(*, 4)</code>	Oui
F77.DATA.Common	Obligation d'utiliser l'instruction INCLUDE pour déclarer les COMMON dans les unités de programmes qui les référencent.	Oui
F77.DATA.Double	Dans une initialisation de constante ou dans l'évaluation d'une expression arithmétique, l'utilisateur souhaite que cette constante soit évaluée en double précision, la présence d'un exposant double précision (lettre D) est obligatoire.	Oui
F77.DATA.Initialisation	Obligation d'initialiser toutes les variables avant leurs utilisations avec l'instruction DATA ou BLOCKDATA.	Oui
F77.DATA.IO	les unités logiques implicites définies par * sont interdites. <code>READ (*,f) [iolist], READ f [,iolist], WRITE (*,f) [iolist], PRINT f [,iolist]</code>	Oui
F77.DATA.LoopDo	Obligation d'utiliser un type ENTIER comme paramètre de contrôle des boucles DO.	Oui
F77.DATA.Parameter	Interdiction d'utiliser des constantes, expression calculé ou appel de fonction comme paramètres de fonction. <code>CALL function (3, x*y, f(z), var)</code>	Oui
F77.ERR.OpenRead	Obligation des tester le statut de retour des instructions OPEN et READ, de préférence à l'aide du paramètre " IOSTAT = ", et vérifier le value de cette variable.	Oui
F77.INST.Assign	Interdiction d'utiliser l'instruction ASSIGN.	Oui
F77.INST.Dimension	Interdiction d'utiliser l'instruction DIMENSION.	Oui
F77.INST.Equivalence	Interdiction d'utiliser l'instruction EQUIVALENCE.	Oui
F77.INST.Function	Il faut utiliser l'instruction FUNCTION avec une déclaration explicite de type, à la définition de la fonction.	Oui
F77.INST.If	Interdiction d'utiliser le IF arithmétique : IF (Expression arithmétique) <code>e1,e2,e3</code> Où les « eN » sont des étiquettes.	Oui
F77.INST.Include	Avec une instruction INCLUDE, le fichier inclus, ne peut pas inclure instructions exécutables (ASSIGN, GOTO, IF, ELSE, CONTINUE, STOP, PAUSE ; DO, READ, WRITE, PRINT,	Oui

Règle	Vérification	Couverture au standard
	REWIND ;BACKSPACE, ENDFILE, OPEN, CLOSE, INQUIER, CALL, RETURN, END)	
F77.INST.Pause	Interdiction d'utiliser l'instruction PAUSE.	Oui
F77.INST.Return	L'instruction RETURN(i) est interdite dans les sous-programmes.	Oui
F77.INST.Save	Interdiction d'utiliser l'instruction SAVE hormis pour des variables locales avec une justification par commentaire.	Oui
F77.MET.Line	Interdiction de dépasser 72 caractères par ligne.	Oui
F77.NAME.GenericIntrinsic	Obligation d'utiliser les noms génériques des fonctions intrinsèques.	Oui
F77.NAME.Intrinsic	Interdiction de réutiliser les noms des fonctions intrinsèques. Quand une fonction définie par le développeur a le même nom que une fonction intrinsèque (définie aux standards), l'application lance une erreur	Oui
F77.NAME.KeyWords	Interdiction de réutiliser les mots-clés u Fortran77 pour les variables	Oui
F77.NAME.Label	Restriction des étiquettes aux instructions FORMAT et CONTINUE.	Oui
F77.PROTO.Declaration	Obligation de déclarer les fonctions externes (lesquelles quisont pas dans le même fichier) par le mot EXTERNAL avant de leur utilisation.	Oui
F77.REF.IO	Obligation d'identifier les unités logiques par un nom symbolique. READ (5, *) NOMBRE -> READ (STDIN, *) NOMBRE	Oui
F77.REF.Open	Obligation de définir les paramètres FILE, STATUS et POSITION de l'instruction OPEN	Oui
F77.REF.Parameter	Interdiction de transmettre comme paramètre d'une subroutine els variables que son déjà dans un bloc COMMON accessible par la subroutine et le programme qui l'appel. COMMON /CONTROL/ A, B, C, D ... PROGRAM ESSAI CALL MY_SUB1 (A, B, C, D) ... END PROGRAM ESSAI SUBROUTINE MY_SUB1 (C _A, B, _C, C_D)	Oui
F77.TYPE.Basic	Obligation d'utiliser les types standards (INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, CHARACTER) uniquement. Autres typs non standards seront considérées erreurs INTEGER*4, LOGICAL*n	Oui
F77.TYPE.Hollerith	Les données et les constantes de type HOLLERITH sont interdites. Une donnée Hollerith est de la forme : numeroH par exemple 0.8H	Oui

5.2.2. FORTRAN 90

Règle	Vérification	Couverture au standard
F90.DESIGN.Free	Obligation de libérer la mémoire allouée dans le même niveau conceptuel.	Non
F90.DESIGN.Include	L'inclure d'un fichier est interdit. Si l'INCLUDE contient un fichier écrit en F90, le programme retourne une erreur. INCLUDE 'file_to_include.F90'	Oui
F90.DESIGN.Interface	Le contenu des modules doit être limité aux clauses USE, PRIVATE et PUBLIC MODULE interface_syslog Implicit none PRIVATE Interface Subroutine f_syslog(cdata) USE message_syslog Type(opendata_type) End subroutine End interface PUBLIC f_syslog End module interface_syslog	Oui
F90.DESIGN.IO	Le nombre d'unité dans une fonction OPEN doit dépendre d'une autre fonction ou un tableau. Integer :: f_unit = 15 OPEN (UNIT = f_unit, ...) -> integer :: f_unit f_unit = getNumber() OPEN(UNIT = f_unit, ...)	PC ²
F90.DESIGN.Obsolete	Cette règle vérifie les clauses suivantes : - Ne pas utiliser le GOTO calculé. - Ne pas utiliser la syntaxe : CHARACTER*N - Ne pas utiliser le IF arithmétique - Dans une boucle DO, ne pas utiliser de variables réelles ni comme indice, ni pour les bornes de l'intervalle de contrôle, ni pour le pas d'incrément. - Ne pas utiliser de terminaison de boucle DO autre que END DO ou CONTINUE. - Ne pas faire de branchements sur ENDIF. - Ne pas utiliser l'instruction PAUSE. - Ne pas utiliser l'instruction GOTO assigné. - Ne pas utiliser l'affectation d'étiquette de FORMAT - Ne pas utiliser le descripteur H (Hollerith) dans les formats.	Oui
F90.BLOC.File	Tout fichier ouvert doit être fermé. Le programme cherche l'instruction CLOSE pour chaque fichier ouvert en amont. OPEN (unit = f_unit, ...) ... CLOSE (unit = f_unit, ...)	Oui
F90.DATA.Array	La dimension du tableau doit être respectée. Il faut obligatoirement utiliser les paramètres qui ont été déclarés à la création du tableau lors de l'appel dans une fonction ou une procédure. Exemple :	Oui

² La règle demande de vérifier ces trois cas : 1) Des primitives d'allocation et de libération de numéros d'unité. 2) Une primitive de réservation d'un numéro d'unité donné, pour permettre l'utilisation de sous-programmes Fortran 77 utilisant des numéros fixés. 3) Des constantes nommées pour l'entrée et la sortie standard. Notre vérification inclue les options 2 et 3.

Règle	Vérification	Couverture au standard
	Subroutine s1(tab) -> Subroutine s1(tab, x) Integer :: tab(2) Integer :: tab(x)	
F90.DATA.ArrayAccess	Dans un tableau d'indirection n'est pas possible de spécifier plusieurs fois le même élément. Integer,dimension(3) :: a Integer,dimension(3) :: b a = (/ 1,1,3 /) b(a) = (/ 1,2,3 /)	Oui
F90.DATA.Constant	Les constantes qui apparaissent dans plusieurs sous-programmes doivent être définies dans un module. Subroutine s1() Real PI = 3,141519 End subroutine s1 module precision 3.141519 Function f1() Real PI = 3,141519 End function -> real PI =	Oui
F90.DATA.ConstantFloat	Les constantes littérales numériques doivent être suivies par le paramètre de sous-type Integer,parameter :: DOUBLE=selected_real_kind(15) Real (DOUBLE) :: x = 0.1_DOUBLE	Oui
F90.DATA.Declaration	Il est obligatoire de mettre l'instruction IMPLICITE NONE après chaque en-tête de méthode. De plus chaque variable devra au préalable être déclaré avant leur utilisation. Cette règle n'est pas prise en compte pour les fonctions et les tableaux.	Oui
F90.DATA.Float	Est interdit d'utiliser le format * en sortie (instruction WRITE) pour les nombres flottants Real :: X Write (std_out, *), x	Oui
F90.DATA.Parameter	Les fonctions intrinsèques SELECTED_REAL_KIND et SELECTED_INT_KIND doivent être regroupées dans un module. MODULE precision Integer, parameter :: DOUBLE = SELECTED_REAL_KIND(15) END MODULE	Oui
F90.ERR.Allocate	L'allocation (ALLOCATE) et libération (DEALLOCATE) doit contenir le paramètre STAT. Ensuite, le valeur du STAT doit être testé après l'instruction. ALLOCATE(x, STAT = iom) IF (iom > 0) THEN ...	Oui
F90.ERR.OpenRead	Les instructions OPEN et READ qui travaille avec des fichiers doivent contenir le paramètre IOSTAT, et vérifier le valeur de cette variable. Pour vérifier ça, l'instruction OPEN doit contenir l'attribut FILE. Une exemple qui ne respecte pas la règle est : OPEN (UNIT = FILE_UNIT, FILE = C_ARG) Parce que il n'y a pas d'attribut IOSTAT, et le valeur de IOSTAT n'est pas testé. Le même exemple correct sera : OPEN (UNIT = FILE_UNIT, FILE = C_ARG, IOSTAT=IOS) IF (IOS .NE. 0) ... Pour l'instruction READ, comme l'attribut FILE n'existe pas, la vérification sera sur ces reads que le UNIT désigne une unité logique qui a été ouverte par un OPEN avant a le code. En continuation avec	Oui

Règle	Vérification	Couverture au standard
	l'antérieur exemple : <code>READ (*, *) //pas vérifiable, lecture du clavier</code> <code>READ (UNIT = FILE_UNIT, FMT = 9011, IOSTAT = IOS) // verifiable</code> <code>IF (IOS .LT. 0) ...</code>	
F90.INST.Associated	<p>Entre le mot ASSOCIATED et la déclaration il faut avoir l'instruction NULLIFY.</p> <p>Exemple :</p> <pre> Real, pointer :: ptr ... NULLIFY(ptr) ASSOCIATED(ptr) </pre>	Oui
F90.INST.Entry	L'instruction ENTRY est interdite Subroutine s1 <code>ENTRY</code> rien	Oui
F90.INST.Equivalence	Interdiction d'utiliser l'instruction EQUIVALENCE. INTEGER total (3,2) INTEGER sum (6) <code>EQUIVALENCE</code> (sum, total)	Oui
F90.INST.If	L'instruction IF logique est interdite quand est suivi par un mt autre que EXIT, CYCLE, GOTO et RETURN. <code>IF</code> (x == 0) THEN <code>GO TO</code> 1000 End IF	Oui
F90.INST.Intent	Chaque paramètre des sous-programmes doit avoir le mot clé INTENT à sa déclaration. Function f1(x, y, z) Integer, <code>INTENT</code> (IN) :: x Integer, <code>INTENT</code> (IN) :: y Integer, <code>INTENT</code> (OUT) :: z	Oui
F90.INST.Nullify	Après une desallocation il y a obligation d'utiliser l'instruction NULLIFY sur la même unité logique. DEALLOCATE (C, stat = iom) <code>NULLIFY</code> (C)	
F90.INST.Only	Interdiction d'utiliser le mot clé ONLY sans commentaire avant qui explique son utilisation. Use mes_fonctions_intrinseques , <code>ONLY</code> :: getuid -> my_getuid	Oui
F90.INST.Operator	Ne pas utiliser la notation ancienne pour les opérateurs relationnels. Substituer .EQ., .NE., .LT., .LE., .GT., .GE. pour ==, /=, <, <=, >, >=	Oui
F90.INST.Pointer	Le POINTER est interdit à exception des cas suivantes : - pour créer des structures de données complexes (i.e. liste de	Oui

Règle	Vérification	Couverture au standard
	<p>chaînes, arbre, etc.) ;</p> <ul style="list-style-type: none"> - pour manipuler des références à des tableaux (référence à un tableau alloué dans un sous-programme, recopie par échange de pointeurs,...) et à des parties de tableaux; - pour utiliser de l'allocation dynamique dans les composants de types dérivés. <p>Le programme lance, donc, une erreur quand il trouve l'attribut POINTER et il référence une variable simple.</p> <p>real, pointer :: ppi</p>	
F90.NAME.GenericIntrinsic	<p>Ne pas se servir des fonctions intrinsèques spécifiques (INT, IFIX, IDINT, REAL, FLOAT, SNGL, ICHAR, CHAR, AINT, DINT, ANINT, DNINT, NINT, IDNINT, IABS, ABS, DABS, CABS, MOD, AMOD, DMOD, ISIGN, SIGN, DSIGN, IDIM, DIM, DDIM, DPROD, MAX0, AMAX1, DMAX1, AMAX0, MAX1, MIN0, AMIN1, DMIN1, AMIN0, MIN1, AIMAG, CONJG, SQRT, DSQRT, CSQRT, EXP, DEXP, CEXP, ALOG, DLOG, CLOG, ALOG10, DLOG10, SIN, DSIN, CSIN, COS, DCOS, CCOS, TAN, DTAN, ASIN, DASIN, ACOS, DACOS, ATAN, DATAN, ATAN2, DATAN2, SINH, DSINH, COSH, DCOSH, TANH, DTANH), utiliser les génériques (INT, REAL, AINT, ANINT, NINT, ABS, MOD, SIGN, DIM, MAX, MIN, SQRT, EXP, LOG, LOG10, SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2, SINH, COSH, TANH).</p> <p>resultat = AMOD (argument, diviseur) resultat = MOD (argument, diviseur)</p>	Oui
F90.NAME.KeyWords	<p>Les variables du code ne peuvent être nommées comme les mots clés en Fortran (ALLOCATABLE, ALLOCATE ; ASSIGN, BACKSPACE, ...). De plus les noms de fonctions doivent être différents des fonctions intrinsèques (ABS, ACHAR, ACOS, ...)</p> <p>integer, parameter :: DATA integer, parameter :: MY_DATA</p>	Oui
F90.PROTO.Overload	<p>La surcharge des opérateurs est interdite. Celle-ci est définie par l'instruction INTERFACE OPERATOR (symbole) et ensuite par leur utilisation.</p>	Oui
F90.REF.ARRAY	<p>Dans une expression, quand on veut représenter l'utilisation total d'un tableau, il est obligatoire de se servir de la notation (:)</p> <p>Y = A*X + B Y(:) = A(:)*X + B où Y et A sont des tableaux</p>	Oui
F90.REF.Interface	<p>Le sous-programme appelé doit être visible</p> <p>subroutine Pas_1 subroutine Pas_1 call Mouv(3.0,resul) call Mouv(3.0,resul) end subroutine Pas_1 contains</p> <p>subroutine Mouv(oper0, resul) -> subroutine Mouv(oper0, resul) end subroutine Mouv end subroutine Mouv</p> <p>end subroutine Pas_1</p>	Oui
F90.REF.Label	<p>L'END doit être suivi par le type (FUNCTION, SUBROUTINE, ...) et le nom</p> <p>function f function f end function end function f</p>	Oui
F90.REF.Open	<p>Toute instruction OPEN doit avoir les paramètres FILE,</p>	Oui

Règle	Vérification	Couverture au standard
	STATUS, IOSTAT et POSITION OPEN (UNIT=f_unit, FILE=c_args, STATUS='old', POSITION='rewind', IOSTAT=ios)	
F90.REF.Variable	Une même variable doit être référencée sous le même nom dans un sous-programme. Call incr(i) call incr(i) subroutine incr(j) -> subroutine incr(j) i = i + 1 j = j + 1 end subroutine incr end subroutine incr	Oui
F90.TYPE.Derivate	Toute déclaration de type doit être dans un module.	Oui
F90.TYPE.Integer	Les paramètres INTEGER doivent être suivis par l'expression SELECTED_INT_KIND. Integer, parameter :: LONG integer, parameter :: LONG = SELECTED_INT_KIND(5)	Oui
F90.TYPE.Real	Les paramètres REAL doivent être suivis par l'expression SELECTED_REAL_KIND. Real, parameter :: LONG real, parameter :: LONG = SELECTED_REAL_KIND(5)	Oui

5.2.3. SHELL

Règle	Vérification	Couverture au standard
SH.DATA.Constant	Obligation de définir les constantes en utilisant <code>'typeset -r'</code> .	Non
SH.DATA.IFS	Interdiction de modifier la variable IFS	Oui
SH.DATA.Integer	À la déclaration d'un integer doit apparaître <code>'typeset -i'</code>	Oui
SH.DESIGN.Bash	La première ligne d'un script doit être <code># !/bin/bash</code> , <code># !/bin/ksh</code> ou <code># !/bin/false</code> .	Oui
SH.DESIGN.Exit	Interdiction pour un programme de prendre fin avant les programmes qu'il a lancés en tâches de fond.	Non
SH.DESIGN.Options	Le cas <code>getopts</code> doit être suivi par un case ou ses options sont évalués.	Oui
SH.ERR.Args	Obligation d'afficher un message particulier lorsqu'un script ne reconnaît pas une option. Ce message doit inclure le synopsis d'utilisation du script.	Non
SH.ERR.Help	Les options doivent être gérées par un <code>getopts</code> ou <code>getopt</code> suivi par un case. Une de ces options doit être <code>-h</code> ou <code>-help</code> . Limitation : si un "h" apparaît dans une chaîne de caractères dans le traitement du case, alors que l'option <code>-h</code> n'est pas gérée, il n'y aura pas de violation.	Oui
SH.ERR.NoPipe	Avant le premier pipe d'un script il faut avoir <code>'set -o pipefail'</code> <u>Limitations</u> : <ul style="list-style-type: none"> Un symbole " " apparaissant dans une chaîne de caractères ou dans les options d'une commande provoquera une violation, sauf suivant un <code>printf</code> ou un <code>sed</code>. Les utilisations du pipe qui suivent un caractère "#" ne seront pas détectées (ex : <code>grep -v ^# grep "\$POINT_DE_MONTAGE"</code>) 	Oui
SH.ERR.String	Dans les cas de <code>if</code> ou <code>while</code> avec les traitements de chaînes de caractères, il faut avoir le traitement de chaînes vides, désignée par <code>[[]]</code> <u>Limitation</u> : L'appel de <code>ls</code> à l'intérieur d'une commande <code>\$(...)</code> ou <code>`...`</code> est susceptible d'être ignoré.	Oui
SH.FLOW.CheckArguments	Dans une fonction, la première instruction, sera la vérification des paramètres. L'instruction sera de la forme suivante : <code>if [\$# -ne 0]</code> Limitation : Un argument d'une même fonction sera signalé autant de fois qu'il est présent dans la fonction	Oui
SH.FLOW.CheckCodeReturn	Shell : Non implémenté <i>Pour toutes les fonctions existantes et appelées dans le script, faudra de vérifier son return avec l'aide de <code>\$#</code> La fonction <code>CD</code> est aussi considérée.</i>	Non
SH.FLOW.CheckUser	Si le script demande des droits d'administrateur, après la vérification de l'utilisateur il faut demander l'action directe de l'admin.	Oui
SH.INST.Basename	Ne pas se servir de <code>\$0</code> , utiliser <code>'basename \$0'</code>	Oui
SH.INST.Continue	Shell : Non implémenté	Non
SH.INST.Copy	Obligation d'utiliser des arguments de même nature pour les commandes <code>'cp'</code> . La nature des arguments est soit de type fichier ou soit de type répertoire.	Non
SH.INST.Find	L'instruction <code>LS</code> est interdite	Oui

Règle	Vérification	Couverture au standard
SH.INST.GetOpts	Cette règle vérifie les options passées comme paramètre dans le script. L'application lance un warning à la dernière ligne si il ne trouve pas le mot GETOPS dans le script. Ce celui qui permet contrôler les paramètres passés. <u>Limitation :</u> Après l'appel de la commande getopt, les violation de \$1 est ignoré jusqu'à la déclaration d'une nouvelle fonction dans le fichier.	Partiel
SH.INST.Interpreter	La première ligne d'un script doit être l'accès à l'interpréter	Oui
SH.INST.Keywords	Une variable ne peut pas être le nom d'un mot clé (if, while, then...) <u>Limitation :</u> L'appel d'option de commandes possédant un mot clef lèvent de faux-positifs (ex : l'option if de la commande dd).	Oui
SH.INST.Logical	Après le symbole logique && ou seulement est permes des instructions 'echo' et 'exit'	Oui
SH.INST.Move	Interdiction d'écraser un fichier en utilisant la commande 'mv'.	Non
SH.INST.POSIX	Utiliser les commandes POSIX aux scripts. Liste dans le RNC-CNES-Q-HB-80-516 <u>Limitation :</u> L'appel de la commande écho suivi d'une redirection (ex : echo &>2) peut engendrer des erreurs sur l'analyse.	Oui
SH.INST.SetShift	Les instructions SET et SHIFT sont interdites.	Oui
SH.INST.Variables	Toute variable doit être écrits comme \${nom_variable}	Oui
SH.IO.Redirect	Les redirection non standards doit etre commentées, et liu doit contenir le nom explicite de la redirection	Oui
SH.MET.LimitAWK	Pour toute instruction AWK, ne pas dépasser les 5 actions (denotées par {...})	Oui
SH.MET.LimitSed	Pour toute instruction SED, ne pas dépasser les 5 actions (denotées par -e, --expresion, -f, ...) et dans chaque action ne pas dépasser les 5 lignes.	Oui
SH.MET.PipeLine	Toute pipeline de ligne de commandes doit être commenté à priori.	Oui
SH.REF.Export	Shell : Non implémenté	Non
SH.REF.Inheritance	Interdiction pour les scripts SHELL d'hériter des alias ou des fonctions définis par l'utilisateur et d'utiliser des commandes interactives dans un script.	Non
SH.SYNC.Signals	Shell : Non implémenté	Non

6. Tables des métriques calculées

Dans le cas des métriques, on identifie les vérifications faites pour un fichier et celles faites pour une fonction – dans le cas où elles sont effectuées

Métrique	Vérification sur fichier
MET.Nesting	<p>Nombre maximum de niveaux d'imbrications de if, switch,while, for, ... dans une fonction/méthode. Ce nombre est 0 s'il y a aucune imbrication..</p> <pre> SUBROUTINE changer_coordonnees(Deplacement, NbPoints, Points) ! ! --- Cette routine effectue une modification de coordonnees sur un ta ! --- en appliquant un déplacement sur les 3 axes x, y et z ! IMPLICIT NONE INTEGER :: c ! colonne INTEGER :: l ! ligne INTEGER, parameter :: NbDim = 3 INTEGER, intent(in) :: NbPoints DOUBLE PRECISION, intent(inout), dimension(NbDim,NbPoints) :: Po DOUBLE PRECISION, intent(in), dimension(NbDim) :: Deplacement ! On applique a chaque point une valeur de déplacement selon les 3 axe 1 do c=1, NbDim, 1 2 do l=1, NbPoints, 1 Points(c, l) = Points(c, l) + Deplacement(c) end do end do END SUBROUTINE </pre> <p>NB.Imbric = 2</p>
MET.Cyclomatic	C'est le nombre de décisions du code (nb if, case, while, catch...). .

```
      SUBROUTINE changer_coordonnees(Deplacement, NbPoints, Points)
      !
      ! --- Cette routine effectue une modification de coordonnees sur un ta
      ! --- en appliquant un déplacement sur les 3 axes x, y et z
      !
      !
      IMPLICIT NONE

      INTEGER :: c          ! colonne
      INTEGER :: l          ! ligne

      INTEGER, parameter :: NbDim = 3
      INTEGER, intent(in) :: NbPoints

      DOUBLE PRECISION, intent(inout), dimension(NbDim,NbPoints) :: Po
      DOUBLE PRECISION, intent(in), dimension(NbDim) :: Deplacement

      ! On applique a chaque point une valeur de déplacement selon les 3 axe
1      do c=1, NbDim, 1
2          do l=1, NbPoints, 1
              Points(c, l) = Points(c, l) + Deplacement(c)
          end do
      end do

      END SUBROUTINE

      NB.Cycomatique = 2
```

MET.LineOfCode

C'est le nombre total les lignes dans du code source du composant logiciel sauf les lignes vides et les lignes de commentaires

```

1  SUBROUTINE changer_coordonnees(Deplacement, NbPoints, Points)
!
! --- Cette routine effectue une modification de coordonnees sur un ta
! --- en appliquant un déplacement sur les 3 axes x, y et z
2  !
3      IMPLICIT NONE
4
5      INTEGER :: c          ! colonne
6      INTEGER :: l          ! ligne
7
8      INTEGER, parameter :: NbDim = 3
9      INTEGER, intent(in) :: NbPoints
10
11     DOUBLE PRECISION, intent(inout), dimension(NbDim,NbPoints) :: Points
12     DOUBLE PRECISION, intent(in), dimension(NbDim) :: Deplacement
13
14     ! On applique a chaque point une valeur de déplacement selon les 3 axes
15     do c=1, NbDim, 1
16         do l=1, NbPoints, 1
17             Points(c, l) = Points(c, l) + Deplacement(c)
18         end do
19     end do
20
21     END SUBROUTINE

```

NB.Line = 15

MET.RatioComment

C'est la proportion de commentaires dans le code source du composant logiciel.
L'entête se prend en compte dans le calcul de le taux de commentaire (tant si elle est avant ou après la définition).

```

1  SUBROUTINE changer_coordonnees(Deplacement, NbPoints, Points)
1  !
2  ! --- Cette routine effectue une modification de coordonnees sur un ta
3  ! --- en appliquant un déplacement sur les 3 axes x, y et z
4  !
2  !
      IMPLICIT NONE

3      INTEGER :: c          ! colonne
4      INTEGER :: l          ! ligne

5      INTEGER, parameter :: NbDim = 3
6      INTEGER, intent(in) :: NbPoints

7      DOUBLE PRECISION, intent(inout), dimension(NbDim,NbPoints) :: Points
8      DOUBLE PRECISION, intent(in), dimension(NbDim) :: Deplacement

5  ! On applique a chaque point une valeur de déplacement selon les 3 axes
9  do c=1, NbDim, 1
10     do l=1, NbPoints, 1
11         Points(c, l) = Points(c, l) + Deplacement(c)
12     end do
13 end do
14

15] END SUBROUTINE

RATE.Comment = 5 / 15 = 0.33

```

7. Messages utilisateur

Il y a deux types de messages différents dans l'outil : les messages qui correspondent à des violations et les messages d'informations dans une fenêtre émergente.

7.1. MESSAGES DES VIOLATIONS

Les messages qui concernent les violations sont affichés dans la vue *i-Code violations* une fois que l'analyse est terminée. Ces messages donnent un peu d'information sur l'erreur remontée : variable qui provoque l'erreur, petite explication de pourquoi cette erreur.

7.1.1. REGLES COMMUNES

Règle	Message
COM.DATA.DeclarationOrder	The parameters are not defined in the right order. The order shall be: in, in/out, out.
COM.DATA.FloatCompare	It's not allowed to compare float variables(" <i>variable</i> ") with equality.
COM.DATA.Initialisation	The variable " <i>variable</i> " is used before being initialized.
COM.DATA.Invariant	The variable " <i>variable</i> " must be defined as constant.
COM.DATA.LoopCondition	The variable " <i>variable</i> " is modified inside the loop.
COM.DATA.NotUsed	The variable " <i>variable</i> " is declared and not used

Règle	Message
COM.DESIGN.ActiveWait	This process contains an active wait. SH : There is an active wait in this point.
COM.DESIGN.Alloc	The resource named " <i>variable</i> " has not been allocated and deallocated in the same algorithmic level.
COM.FLOW.Abort	The keyword STOP is not allowed.
COM.FLOW.BooleanExpression	Using more than five conditions in an expression is not allowed. SH : It is not allowed use five or more conditional expressions in the same instruction.
COM.FLOW.CaseSwitch	A DEFAULT case is needed in a switch case instruction. SH : The default case of the case switch condition is missing.
COM.FLOW.CheckCodeReturn	The return code of the function " <i>function</i> " is not checked.
COM.FLOW.CheckUser	The user identity is not verified in the main program.
COM.FLOW.Exit	There is more than one exit in the function.
COM.FLOW.ExitLoop	There is more than one exit in the loop.
COM.FLOW.FileExistence	The existences of the file " <i>file</i> " must be checked with the instruction INQUIRE before being opened or created. SH : The existence of the file " + name + " has not been checked.
COM.FLOW.FilePath	It is not allowed to use directly the file name. Store the file path in a variable. Use the variable instead.
COM.FLOW.Recursion	The use of recursivity is not allowed.
COM.INST.BoolNegation	Double negation is not allowed.
COM.INST.Brace	Parentheses are needed for readability.
COM.INST.CodeComment	Commented code is not allowed. It shall be suppressed. SH : The keyword " + yytext() + " is used in a comment. A variable is assigned in a comment.
COM.INST.GOTO	The keyword GOTO is not allowed.
COM.INST.Line	More than one instruction per line is not allowed.
COM.INST.LoopCondition	A loop condition shall be written with inequality (.LE.,<=, or .GT.,>=)
COM.NAME.Homonymy	Names must be unique. The name " <i>variable</i> " is already defined in this file.
COM.PRES.Indent	The code is not indented.
COM.PRES.LengthLine	There are more than 100 characters in this line.
COM.PROJECT.Header	- No file header existing. This module/function should have a header with a brief description. - No file header (file name not found). This module/function should have a header with a brief description. - The module/function should have a header with a brief description. SH : The function should have a header with a brief description.
COM.TYPE.Expression	Mixed types " <i>type_variable_1</i> " with " <i>type_variable_2</i> " in the same expression

7.1.2. FORTRAN 77

Règle	Message
F77.BLOC.Common	Unnamed COMMON is not allowed.
F77.BLOC.Else	The IF instruction shall finish with an ELSE after the last ELSE IF.

Règle	Message
F77.BLOC.Function	When calling a function, the brackets following the function name are mandatory.
F77.BLOC.Loop	Loops shall have distinct ends.
F77.DATA.Array	The dimension of the array "variable" is not well declared. The * shall be used for the last dimension.
F77.DATA.Common	The INCLUDE instruction shall be used to reference the needed common bloc.
F77.DATA.Double	The double precision variable is not correctly initialized. It misses the character D in its declaration.
F77.DATA.Initialization	The variable "variable" shall be initialized with DATA or BLOCK DATA before its use.
F77.DATA.IO	The use of * with logical units is not allowed.
F77.DATA.LoopDo	The control variable in a loop shall be an integer.
F77.DATA.Parameter	"variable" belongs to parameter types forbidden when calling a function: a constant, an expression to be evaluated, a call to another function
F77.ERR.OpenRead	The status of OPEN/READ shall be tested with the parameter IOSTAT.
F77.INST.Assign	The instruction ASSIGN Is not allowed.
F77.INST.Dimension	The instruction DIMENSION Is not allowed.
F77.INST.Equivalence	The instruction EQUIVALENCE is not allowed.
F77.INST.Function	It misses the type declaration in FUNCTION header.
F77.INST.If	The arithmetic if is not allowed.
F77.INST.Include	The executable instruction "variable" is not allowed in the include file.
F77.INST.Pause	The instruction PAUSE is not allowed.
F77.INST.Return	The instruction RETURN(i) is not allowed.
F77.INST.Save	The instruction SAVE is only permitted for local variables
F77.MET.Line	There are more than 72 characters in this line.
F77.NAME.GenericIntrinsic	It should be used the generic name of the intrinsic function instead of "variable"
F77.NAME.Intrinsic	It is not allowed to use the name of an intrinsic function.
F77.NAME.KeyWords	The variable "variable" is a keyword in Fortran77 language.
F77.NAME.Label	The use of labels is not allowed except with the instructions FORMAT and CONTINUE.
F77.PROTO.Declaration	The function "variable" shall be declared.
F77.REF.IO	The logical entities shall be declared using a symbolic name.
F77.REF.Open	The instruction OPEN shall be called with the parameters FILE, STATUS and POSITION.
F77.REF.Parameter	It is not allowed to provide as a parameter the variables of an accessible bloc COMMON. The variable "variable" is used in a wrong way.
F77.TYPE.Basic	"variable" is not a basic type. Basic types are INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL and CHARACTER.
F77.TYPE.Hollerith	Type Hollerith is not allowed. "variable" shall be a CHARACTER.

7.1.3. FORTRAN 90

Règle	Message
F90.BLOC.File	The file "variable" is not correctly closed.
F90.DATA.Array	The dimension's array must be declared as parameters' function.
F90.DATA.ArrayAccess	Array " variable1" initialized using other array named "variable2 " with repeated values.

F90.DATA.Constant	The constants shall be declared and initialized in a module.
F90.DATA.ConstantFloat	Float constant "variable" shall be declared using the subtype_parameter: <name>_<subtype_parameter>
F90.DATA.Declaration	The variable must be declared. The sequence IMPLICITE NONE must be declared after the method.
F90.DATA.Float	It is not allowed to use the format * for reals like "variable".
F90.DATA.Parameter	It misses the use of intrinsic function SELECTED_REAL_KIND or SELECTED_INT_KIND for the subtype specification.
F90.DESIGN.Include	Is it possible to use a module instead of this inclusion?
F90.DESIGN.Interface	Interface Module shall only contain: INTERFACE, USE, IMPLICIT instructions as well as PRIVATE or PUBLIC declaration.
F90.DESIGN.IO	The value of the logic unity should be a integer or a variable initialised directly.
F90.DESIGN.Obsolete	The instruction calculated GOTO is not allowed. The instruction PAUSE is not allowed. The alternate return statement is not allowed. There is a branch on an END IF statement. It is not allowed. The use of CHARACTER* is not allowed. The instruction HOLLERITH is not allowed inside FORMAT. Error in "variable" used. The instruction ASSIGN contains the label for the FORMAT instruction. Arithmetical IF is not allowed. A DO loop shall end with END DO. The variable "variable" is a real used in a do loop. Use only INTEGER. Each loop shall have its own END DO. Shared END DO is forbidden.
F90.ERR.Allocate	The status of the ALLOCATE or DEALLOCATE instruction is not checked
F90.ERR.OpenRead	- There is no parameter IOSTAT in the OPEN/READ instruction. - The return of IOSTAT is no checked in the OPEN/READ instruction.
F90.INST.Associated	The pointer « variable » is not set to null before the use of the instruction ASSOCIATED.
F90.INST.Entry	The instruction ENTRY is not allowed.
F90.INST.Equivalence	The instruction EQUIVALENCE is not allowed.
F90.INST.If	Logical IF (without THEN and ENDIF) is only allowed with EXIT, CYCLE, GOTO, RETURN statements.
F90.INST.Intent	It misses the attribute INTENT for the parameter "variable"
F90.INST.Nullify	It misses the instruction NULLIFY after the DEALLOCATION of "variable".
F90.INST.Only	The instruction ONLY must be preceded by a comment.
F90.INST.Operator	The symbolic notation (==, /=, <=, <, >=, >) must be used instead of (.EQ., .NE., .LT., .LE., .GT., .GE.). Error in "variable".
F90.INST.Pointer	This use of POINTER is not allowed.
F90.NAME.GenericIntrinsic	Use the generic name of the intrinsic functions instead of "variable".
F90.NAME.KeyWords	The variable "variable" is a keyword in Fortran90 language.
F90.PROTO.Overload	Overloading operator is not allowed. Overload of "variable"
F90.REF.ARRAY	It should be used the notation(:) to specify the entire use of the arrays: "list_variables".

F90.REF.Interface	The function "function" is not visible in this point.
F90.REF.Label	It misses the name of the subprogram. It must finish with END TYPE_PROGRAM NAME.
F90.REF.Open	It misses one or more parameters In OPEN instruction. Mandatory parameters are FILE, STATUS, IOSTAT, POSITION.
F90.REF.Variable	The variable "variable" is used with different names inside the subprogram.
F90.TYPE.Derivate	The variable " must be defined inside the module structure.
F90.TYPE.Integer	It misses the declaration SELECTED_INT_KIND in the initialisation of "variables"
F90.TYPE.Real	It misses the declaration SELECTED_REAL_KIND in the initialisation of "variables"

7.1.4. SHELL

Règle	Message
SH.DATA.IFS	The environment variable IFS can't be modified.
SH.DATA.Integer	The integer variables must be defined using the typeset -i declaration.
SH.DESIGN.Bash	The first line must declare the interpreter (/bin/bash, /bin/ksh or /bin/false)
SH.DESIGN.Options	It is mandatory to use getopts and getopt and to provide the -h, -help, -v and -version options at least.
SH.ERR.Help	The help option (-h or --help) must be implemented.
SH.ERR.NoPipe	When the pipe is used in the script the option set -o pipefail is mandatory.
SH.ERR.String	The empty strings must be taken into account
SH.FLOW.CheckArguments	The number of parameters received has not been checked.
<i>SH.FLOW.CheckCodeReturn</i>	<i>The function's return function_name has not been checked.</i>
SH.FLOW.CheckUser	The user has not been checked.
SH.INST.Basename	The use of the keyword basename before \$0 is mandatory.
<i>SH.INST.Continue</i>	<i>The keyword CONTINUE is not allowed.</i>
SH.INST.Find	The use of LS is not allowed. Use FIND instead.
SH.INST.GetOpts	It is mandatory to use getopts & getopt in the script.
SH.INST.Interpreter	The first line must declare the interpreter
SH.INST.Keywords	The keywords <i>variable</i> cannot be used as a variable.
SH.INST.Logical	The abbreviation and && must be followed only by ECHO or EXIT.
SH.INST.POSIX	The keyword <i>POSIX_word</i> is not allowed.
SH.INST.SetShift	The keyword SET/SHIFT is not allowed.
SH.INST.Variables	The variable <i>variable_name</i> is not correctly declared (must be declared using \${ } or " " notation)
SH.IO.Redirect	Thenon-standard redirection must be preceded by a comment.
SH.MET.LimitAWK	The AWK expression has more than 5 actions
SH.MET.LimitSed	The SED expression has more than 5 actions/lines
SH.MET.PipeLine	Every pipeline must be preceded by a comment.
<i>SH.REF.Export</i>	<i>The keyword EXPORT is no allowed.</i>

Règle	Message
<i>SH.SYNC.Signals</i>	<i>The keyword TRAP must be followed by a variable, not an integer.</i>