Fortran 202Y / Fortran 2028

Aperçu des nouvelles fonctionnalités

Accent sur la généricité (templates)

Vincent LAFAGE

IJCLab, Laboratoire de Physique des 2 Infinis Irène Joliot-Curie
CNRS/IN2P3 & Université Paris-Saclay, Orsay, France

Journée Fortran 2026

CC BY SA



Plan

1. Contexte et calendrier
2. Généricité – le cœur de Fortran 202Y
3. Autres fonctionnalités majeures
4. État d'avancement et perspectives



Plan

1. Contexte et calendrier
2. Généricité – le cœur de Fortran 202Y
3. Autres fonctionnalités majeures
4. État d'avancement et perspectives



Fortran 202Y / Fortran 2028

- Fortran 2023 (ISO/IEC 1539-1 :2023) est le standard actuel.
- La prochaine révision est appelée Fortran **202Y** en interne (Y $\approx$ 8 $\Rightarrow$ Fortran **2028**).
- Compatibilité ascendante : tout programme conforme Fortran 2023 (sauf limitations mineures) reste conforme Fortran 2028.
- Liste initiale de fonctionnalités approuvée par les National Bodies en juin 2023 (WG5 N2222).
- Travail en cours au sein de J3 / WG5 ; détails encore susceptibles d'évoluer.

CC BY SA

https://j3-fortran.org/doc/year/26/26-010r1.pdf

F202Y Workplan

Live doc link

Label	Short Desc	Level (see 24-167)	Background / Use Cases	Subgroup	Rationale / Discussion Papers	Requirements	Specifications	Syntax	Edits	Applied
JP01	Include generic subprograms described in N2217	5	N2217 (WG5) 25-102 25-150	Data	24-140r2 (info, straw votes) 24-184r1 24-188r1 24-180 24-181 25-115	23-223r2 24-147r1 25-163r2	23-244r1 24-148r1 25-103 25-128 25-129 25-164r2	24-139r2 25-103 25-128 25-168r2	25-169r5	
US01	Obsolete default implicit typing	2	23-177	JOR					25-192r2	
US02	Obsolete D format edit descriptor	2	23-178	JOR					25-113	
US03	Add note that the real model is not IEEE 754	1	23-180	JOR					25-193r1	
US04	Asynchronous Collective Subroutines		N2245 (WG5)	HPC		25-162r2	25-165r1	25-166r2	25-202r3	
US05	Add extended floating-point types defined in ISO C23 to ISO_C_BINDING	3	23-176	HPC		25-189r1	25-189r1		25-203r1 26-112	
US06	Provide a mechanism to specify global binding name for non C-interoperable	3	23-201	HPC						
US07	Improve rank-independent functionality	4	23-184r1	Data / JOR						
US08	Polymorphic PURE function results	4	23-186	Data		25-116r1	25-116r1 25-178	25-116r1 25-178	26-107r1	
US09	Allow I/O of enumeration type names	4	23-151r1	JOR					25-201r1	
US10	Define a standard Fortran preprocessor	6	23-192r1, 24-108, 24-109	FPP		25-114r2	25-142r2 25-176r3			
US11	Provide intrinsics for source location	3	23-193r1	JOR						
US12	Add maximum rank/corank constants to ISO_FORTRAN_ENV	3	23-194	Data (JP01)						
US13	Remote access to module entities	3	23-196r1	Data	25-119r1					
US14	Provide scoped access to enumeration enumerators	4	23-197	Data		25-180	25-180	25-180	26-106	
US15	Readonly pointers: a new attribute be application to pointers that determined whether the target of the pointer cannot be changed trhough the pointer	4	23-198	Data		25-179r1	25-179r1	25-182r1	25-197r2	
US16	Provide a facility to specify the default kinds within the program unit	3	23-199	Data		25-191r2	25-191r2	26-108r1	26-109r2	
US17	Generic programming using templates	8	23-148, 23-103	Generic	23-211r1 23-212r2 23-213r2 23-214r3 23-215r1 25-158r1	24-105r3	24-105r3	24-125r5 24-126r4 24-127r4 24-159r4	25-109r2 25-135r2 25-136r2 25-172r1 25-173r1 25-174r2 25-175r6 25-204r3	
US18	Allow Polymorphism in Coarrays	4	23-217	HPC (?)						
US19	Add more math functions from IEEE-754	4	23-234r2	JOR		25-143	25-143			
US20	Add Intrinsic and collective subroutines for prefix operations (formerly "SCAN" and "CO_SCAN")	4	23-235r2, 24-157	HPC + JOR	25-195r1	25-144r1 (co_*) 25-145r1 (local)	25-177r1 (co_*) 25-186r2 (sum) 25-196r1 (reduce)	25-177r1 (co_*) 25-186r2 (sum) 25-196r1 (reduce)	25-199r2 (co_*) 26-103r3 (local)	
US23/DIN4	Add generic processing of assumed-rank object	5	24-136r1	Data	24-169	24-171 24-172	24-183r1	24-183r1	24-183r1	
US24	Add rank-independent looping	4	24-143	Data + JOR						
US25	Add ability to interpret complex as real and vice versa	5	24-129	Data		24-173	25-188r1	25-188r1	26-110	
DIN1	Add execution of collective procedures on a specified team	4	N2230 (WG5) 25-125r1	HPC		25-127r1	25-127r1	25-166r2	25-202r3	
DIN3a	Add support for atomic operations in local memory	5	N2230 (WG5)	HPC						

CC BY SA



Historique synoptique

	66	77	90	95	03	08	18/23	28

Procedural		ELSE
		ENDIF
		ENDDO
		DO WHILE
			SELECT CASE			BLOCK		

Modular	COMMON	INCLUDE	MODULE TYPE					
Inheritance					CLASS EXTENDS			
Dynamic			ALLOCATE POINTER		*			
Functional	passing subprogram argument		RECURSIVE INTENT()	PURE ELEMENTAL			SIMPLE	
Concurrent			array exp	FORALL		DO CONCURRENT CONTIGUOUS		
Distributed						coarray	TEAM	
Generic	preprocessing	intrinsic	INTERFACE					template GENERIC



Principaux axes de travail (N2222)

Généricité

- JP01 – Sous-programmes génériques (N2217)
- US17 – Templates + exigences fortes

Autres fonctionnalités majeures

- ~~US04 – Tâches asynchrones~~
- US10 – Préprocesseur `Fortran` standard
- US07 – Fonctionnalités indépendantes du rang

Modernisation & qualité

- US01 – Implicit typing par défaut obsolète
- US02 – Descripteur D obsolète
- US16 – KIND par défaut dans une unité
- US11 – Intrinsics de localisation source
- US12 – Constantes rank/corank max

Interopérabilité & autres

- US05 – Nouveaux types flottants C
- US09 – E/S sur énumérations
- US13/14 – Accès namespace / scoped



Plan

1. Contexte et calendrier
2. Généricité – le cœur de Fortran 202Y
3. Autres fonctionnalités majeures
4. État d'avancement et perspectives



Pourquoi la généricité ?

(... ou polymorphisme paramétrique)

Limitations actuelles

- Interfaces génériques + surcharge (polymorphisme faible) : il faut écrire *chaque* version spécifique.
- Parametrized Derived Types (PDT) : limités (kind/length uniquement).
- Polymorphisme (fort) illimité (`class(*)`) : perte de typage statique, performance.
- Pas de conteneurs génériques sûrs (vecteurs, maps, ...) sans duplication massive.

Objectif

Fournir un mécanisme de **templates** type-safe, vérifiable *avant* instanciation, inspiré de `C++ concepts` / `Rust traits`, adapté à `Fortran`.

CC BY SA



Deux approches complémentaires

1. **Generic Subprograms** (proposition japonaise N2217 – JP01)
   Extension du mécanisme d'identifiants génériques existant.
   Une seule écriture → plusieurs procédures spécifiques (kind, rank,...).
2. **Templates** (US17 – travail du groupe j3-fortran/generics)
   Paramètres de type, de constantes, d'opérateurs, d'exigences (*requires*).
   Instanciation explicite ou en ligne.
   Support de conteneurs, algorithmes génériques, monades, etc.

Les deux sont poursuivis ; les templates sont le sujet le plus ambitieux.

CC BY SA



Generic Subprograms – idée (N2217)

```
GENERIC FUNCTION has_nan(x) RESULT(ans)
  TYPE(REAL(REAL32,REAL64,REAL128)), RANK(0), INTENT(IN) :: x
  LOGICAL :: ans
  ans = ieee_is_nan(x)
END FUNCTION

GENERIC FUNCTION has_nan(x) RESULT(ans)
  TYPE(REAL(REAL32,REAL64,REAL128)), RANK(1:15), INTENT(IN) :: x
  LOGICAL :: ans
  ans = any(ieee_is_nan(x))
END FUNCTION
```

- Une déclaration → plusieurs procédures spécifiques générées.
- Compatible avec le mécanisme générique existant.
- Particulièrement utile pour les bibliothèques numériques multi-précision.
- Pas seulement des `kind` différents, mais des types distincts.

CC BY SA



Generic Subprograms – idée (N2217)

```
GENERIC SUBROUTINE lift(x,y)
   TYPE (INTEGER, REAL), ALLOCATABLE, RANK(1:2) :: x
   TYPEOF(x), RANKOF(x) :: y
   ...
END SUBROUTINE
```



Templates – concepts clés (j3-fortran/generics)

- **template** : unité de programmation paramétrée.
- Paramètres possibles :
  - types (`type :: T`)
  - constantes entières (`integer, constant :: n`)
  - procédures / opérateurs
- **requires** : contraintes sur les paramètres (concepts / traits).
- **instantiate** : instanciation explicite (ou en ligne).
- Vérification de cohérence *avant* toute instanciation.



Exemple simple – template de swap

```
template swap_tmpl(T)
  private
  public :: swap
  type :: T ; end type
contains
  elemental subroutine swap(a, b)
    type(T), intent(inout) :: a, b
    type(T) :: tmp
    tmp = a ; a = b ; b = tmp
  end subroutine
end template

! Utilisation
instantiate swap_tmpl(real)
instantiate swap_tmpl(integer)
```


CC BY SA



Exemple avec exigences (requires)

```
template fallible_tmpl(E, T, combine_errs)
  requires bin_op(E, combine_errs)   ! E doit supporter combine_errs
  private
  public :: fallible_t
  type :: T ; end type
  type :: fallible_t
    private
    logical :: is_err_
    type(E) :: error_
    type(T) :: answer_
  contains
    procedure :: failed, error, answer
  end type
  ...
end template
```

Gestion d'erreurs monadic-style, typée et réutilisable.



Exemple plus riche – matrices génériques

```
template matrix_tmpl(T, plus_t, zero_t, times_t, one_t, n)
  requires semiring(T, plus_t, zero_t, times_t, one_t)
  integer, constant :: n
  type :: matrix
    type(T) :: elements(n, n)
  end type
  generic :: operator(+) => plus_matrix
  generic :: operator(*) => times_matrix
  ...
end template
```

- Paramètres de type + opérateurs + taille constante.
- Exigences algébriques (semiring, field,...).
- Permet d'écrire une fois un solveur de Gauss, un produit matriciel, etc.



Cas d'usage motivants (dépôt generics)

- Conteneurs génériques (vector, map, set) sans perte de typage.
- Algèbre linéaire multi-précision / multi-type.
- Gestion d'erreurs (fallible / Result types).
- Itération fonctionnelle (fold, scan, map) sur tableaux de type arbitraire.
- Bibliothèques scientifiques réutilisables sans duplication de code.

Références

GitHub : `https://github.com/j3-fortran/generics`
Papers J3 : 23-155r1, 23-182r1, 23-187, 23-189, 23-190,...

CC BY SA



Plan

1. Contexte et calendrier
2. Généricité – le cœur de Fortran 202Y
3. Autres fonctionnalités majeures
4. État d'avancement et perspectives



~~Tâches asynchrones (US04)~~

- Proposition 23-174. Deferred to 203X
- Introduction d'un construct `TASK` (inspiré d'Ada, 1983 / `OpenMP` tasks).
- Région d'exécution pouvant s'exécuter de façon asynchrone par rapport au parent.
- Spécificateurs de localité (local, shared,...).
- Complète les coarrays et `DO CONCURRENT`.
- Utile pour graphes de tâches (DAG), calculs hétérogènes, pipelines.

CC BY SA



Préprocesseur `Fortran` standard (US10)

- Actuellement : `cpp`, `fpp`, `coco`,...chacun avec ses dialectes.
- Objectif : un préprocesseur *Fortran-friendly* normalisé.
- Travail en cours dans le dépôt `j3-fortran/fpp-202y`.
- Facilite la génération de code, la configuration conditionnelle, les macros sûres.

CC BY SA



Fonctionnalités indépendantes du rang (US07)

- Améliorer l'écriture de code qui fonctionne pour n'importe quel rang.
- Extensions autour de `RANK`, `SELECT RANK`, assumed-rank,...
- Objectif : réduire la prolifération de versions 0D/1D/2D/...

Autres points notables

- US01 : implicit typing par défaut → obsolète
- US16 : déclaration de KIND par défaut dans une unité de programme
- US11 : intrinsics de localisation source (`__FILE__`, `__LINE__`,...)
- US12 : constantes `MAX_RANK`, `MAX_CORANK` dans `ISO_FORTRAN_ENV`
- US13/14 : accès de type namespace / scoped aux modules et énumérations
- US15 : pointeurs en lecture seule (immutable target)



Plus de fonctions IEEE-754 (US19)

IEEE-754 Operation	Fortran Operation	Expression
expm1(x)	EXPM1(X)	exp(x)-1
exp2m1(x)	EXP2M1(X)	2**x-1
exp10m1(x)	EXP10M1(X)	10**x-1
log2(x)	LOG2(X)	log(x)/log(2.)
logp1(x)	LOGP1(X)	log(1+x)
log2p1(x)	LOG2P1(X)	log2(1+x)
log10p1(x)	LOG10P1(X)	log10(1+x)
rsqrt(x)	RSQRT(X)	1/sqrt(x)
compound(x,n)	COMPOUND(X,N)	(1+x)**n
rootn(x,n)	ROOTN(X,N)	x**(1./n)
pow(x,y)	POW(x,y)	x**y

CC BY SA



Plan

1. Contexte et calendrier
2. Généricité – le cœur de Fortran 202Y
3. Autres fonctionnalités majeures
4. État d'avancement et perspectives



État actuel (fin 2025 / 2026)

- Les grandes lignes sont stabilisées (N2222 + mises à jour N2249).

  `https://wg5-fortran.org/N2201-N2250/N2222.txt`

  `https://wg5-fortran.org/N2201-N2250/N2249.txt`

- Le design des templates continue d'évoluer (syntaxe des paramètres, `requires`, instanciation en ligne vs avancée).
- Des prototypes et de nombreux exemples existent dans le dépôt `j3-fortran/generics`.
- Les compilateurs (`gfortran`, `flang`, `ifx`,...) n'implémentent pas encore ces fonctionnalités.
- Publication visée autour de 2028.

CC BY SA



Impact attendu

- Réduction drastique de la duplication de code dans les bibliothèques numériques.
- Conteneurs et algorithmes génériques de qualité production.
- Meilleure interopérabilité avec les pratiques modernes (`C++`, `Rust`, `Julia`,...).
- `Fortran` reste compétitif pour le calcul scientifique haute performance.

La généricité est le changement le plus transformateur depuis les coarrays / OOP.

CC BY SA



Ressources

- WG5 documents : `https://wg5-fortran.org/`
- Dépôt generics : `https://github.com/j3-fortran/generics`
- Dépôt préprocesseur : `https://github.com/j3-fortran/fpp-202y`
- Discourse `Fortran` : `https://fortran-lang.discourse.group/`
- Papers clés :
  - N2217 – Generic Subprograms `https://wg5-fortran.org/N2201-N2250/N2217.pdf`
  - 23-155r1, 23-182r1 – Templates / Requirements

    `https://j3-fortran.org/doc/year/23/23-155r1.txt`

    `https://j3-fortran.org/doc/year/23/23-182r1.txt`
  - 23-174 – Asynchronous Tasks `https://j3-fortran.org/doc/year/23/23-174.pdf`
  - N2222 – Work items Manchester 2023 `https://wg5-fortran.org/N2201-N2250/N2222.txt`

CC BY SA



Et après ?

- Asynchronie ?
  `https://j3-fortran.org/doc/year/23/23-174.pdf`
- Convergence des bibliothèques mathématiques ?
- Types `UNSIGNED` ?
  `https://j3-fortran.org/doc/year/24/24-116.txt`
- Contraintes de domaines pour les types ?
  - Ada, 1983
- Programmation par contrats ?
  - Hoare, 1969
  - Eiffel, 1986
  - D, 2001
  - Ada, 2012
  - C++, 2026
- Validation Formelle ?
  - (Ada) SPARK, 2014
- Exceptions ?
- Constantes (≠ parameter) ?

CC BY SA

Questions ?

Vincent Lafage – IJCLab
vincent.lafage@ijclab.in2p3.fr