

# Fortran DLL Interoperabiliteit — Gecombineerd ADR + Richtlijnen

## Fortran DLL Interoperabiliteit - Gecombineerd ADR + Richtlijnen

### ADR-07 - Fortran 95 DLL Interoperabiliteitsstrategie

#### Status

Goedgekeurd - Dit besluit is vastgesteld als standaard voor alle numerieke kernels en interoperabiliteitscomponenten binnen de multi-language architectuur (Fortran 95, Rocket COBOL, Intel C, ML64).

---

#### 1. Context

Het project vereist een stabiele, taal-agnostische ABI-laag die door meerdere talen gebruikt kan worden, waaronder Rocket COBOL 11.0, Intel C (CL), ML64-assembly en toekomstige bindings voor Python, C#, Rust. Fortran wordt ingezet voor numerieke kernels vanwege performance en legacy-compatibiliteit.

Moderne Fortran-modules genereren compiler-specifieke .mod-bestanden, veroorzaken symbol-mangling en zijn niet bruikbaar buiten Fortran-compilers. Daarom is onderzocht of Fortran 95 zonder modules, met uitsluitend C-ABI-compatibele DLL-exports, een geschikte strategie is.

---

#### 2. Beslissing

Het project kiest voor een Fortran 95-architectuur zonder modules, waarbij:

1. Alle procedures worden gedefinieerd als bind(C) voor een stabiele C-ABI.
2. Alle functionaliteit wordt aangeboden via een DLL-exportlaag.
3. Geen gebruik wordt gemaakt van Fortran-modules.
4. Interfaces worden beheerd via C-headers, COBOL-copybooks en ML64-prototypes.

Deze strategie wordt toegepast op alle numerieke kernels en interoperabiliteitscomponenten waar ABI-stabiliteit en multi-language integratie belangrijker zijn dan moderne Fortran-features.

---

## 3. Motivatie

### 3.1 Maximale interoperabiliteit

Een pure C-ABI-laag is de enige interface die door alle betrokken talen wordt ondersteund.

### 3.2 Compiler-onafhankelijkheid

.mod-bestanden zijn niet gestandaardiseerd en verschillen per compiler en versie. DLL-exports met C-ABI zijn langdurig stabiel.

### 3.3 Volledige controle over symbolen

Elke export is expliciet en voorspelbaar, zonder verborgen module-symbolen of naam-mangling.

### 3.4 Geschikt voor numerieke kernels

De Fortran-code is klein, functioneel en numeriek; modules bieden hier weinig extra waarde.

---

## 4. Gevolgen

### 4.1 Positieve gevolgen

- Stabiele ABI voor alle talen
- Geen compiler-lock-in
- Eenvoudige integratie met COBOL, C, ML64, Python, C#, Rust
- Volledige controle over exports en symbolen
- Eenvoudige deployment (DLL + header)

### 4.2 Negatieve gevolgen

- Geen interface-controle door Fortran zelf
  - Geen moderne Fortran-features zoals modules en submodules
  - Meer boilerplate in interface-definities
  - Hogere foutkans bij verkeerde argumenten (runtime-detectie)
- 

## 5. Alternatieven

### A. Fortran 95 met modules

Voordelen: type-veiligheid, encapsulatie, moderne features. Nadelen: .mod-bestanden zijn niet bruikbaar buiten Fortran.

## B. Mixed approach (modules intern, C-exports extern)

Voordelen: interne structuur + externe interoperabiliteit. Nadelen: complexere build-chain, dubbele interface-definitie.

## C. Alles in C implementeren

Voordelen: maximale interoperabiliteit. Nadelen: verlies van Fortran-performance en numerieke stabiliteit.

---

## 6. Besluitcriteria

Deze strategie is gekozen omdat:

- Interoperabiliteit belangrijker is dan taalfeatures.
  - ABI-stabiliteit belangrijker is dan module-functionaliteit.
  - Multi-language integratie een harde eis is.
  - De Fortran-code klein en numeriek is.
  - De C-laag de centrale interface vormt.
- 

## Richtlijnen voor Fortran DLL-Interoperabiliteit

### 1. Scope en uitgangspunten

- Doel: een stabiele, taal-agnostische DLL-laag op basis van C-ABI.
  - Talen: Fortran 95 (kernels), Rocket COBOL, C, ML64, Python/C#/Rust.
  - Principe: Fortran is de engine, C-ABI is de interface.
- 

### 2. Algemene ontwerpregels

- Enkel C-ABI: alle extern zichtbare routines krijgen bind(C).
  - Geen modules: de C-header is de bron van waarheid.
  - Flat interfaces: geen optional args, geen Fortran-specifieke structuren.
  - Geen globale state over de ABI: state expliciet via handles.
-

## 3. Richtlijnen aan Fortran-zijde

### 3.1 Declaratie van exports

#### **Functie:**

```
function f_example(a, b) result(res) bind(C, name="f_example")
  use iso_c_binding
  implicit none
  real(c_double), value :: a, b
  real(c_double) :: res
end function f_example
```

#### **Subroutine:**

```
subroutine s_example(n, arr) bind(C, name="s_example")
  use iso_c_binding
  implicit none
  integer(c_int), value :: n
  real(c_double) :: arr(n)
end subroutine s_example
```

### 3.2 Types en parameters

- Altijd `iso_c_binding` gebruiken.
- Strings: `character(kind=c_char)`, `dimension(*)` + nulterminatie.
- Structs: type, `bind(C)`.

### 3.3 Memory-beheer

- Eén kant is eigenaar.
- Geen verborgen allocaties.
- Free-API verplicht bij Fortran-allocaties.

### 3.4 Foutenafhandeling

- Geen exceptions over de ABI.
- Gebruik integer-returncodes of status-out-parameters.

---

## 4. Richtlijnen voor C-headers en ABI-contract

- C-header is leidend.
- Exportnamen zijn stabiel en expliciet.
- Calling convention: standaard x64-ABI.
- Structs: identieke layout in C en Fortran.

---

## 5. Richtlijnen voor COBOL en ML64

### COBOL

- Copybooks genereren vanuit de C-header.
- Let op mapping van integers, floats en pointers.

### ML64

- Prototypes direct uit de C-header.
- Geen variadische functies.

---

## 6. Versiebeheer en compatibiliteit

- Exportlijst bevroren.
- Nieuwe functionaliteit → nieuwe functie.
- Versie-API verplicht.

---

## 7. Testen en validatie

- Cross-language tests voor elke export.
- ABI-checks bij compiler-upgrade.

---

## 8. Documentatie-afspraken

- Elke functie krijgt beschrijving, pre/postcondities, ownership-regels en foutcodes.
- Eén centraal document beschrijft calling convention, endianness en thread-safety.

---

## Conclusie

Deze gecombineerde ADR + richtlijnen vormen een volledig en consistent kader voor het ontwerpen, bouwen en onderhouden van een stabiele Fortran-DLL-interoperabiliteitslaag binnen een multi-language architectuur.