

# Whitepaper — Uniforme Interoperabiliteitsarchitectuur via C-ABI DLL's

## Whitepaper - Uniforme Interoperabiliteitsarchitectuur via C-ABI DLL's

### Executive Summary

Organisaties met een heterogene technologie-stack - COBOL voor bedrijfslogica, Fortran voor numerieke kernels, C voor infrastructuur, ML64 voor low-level optimalisaties en moderne talen zoals C# en Python - lopen structureel tegen interoperabiliteitsproblemen aan. Deze whitepaper introduceert een uniforme architectuur gebaseerd op C-ABI DLL's, met expliciete symbol-exports, consistente type-mapping en gestandaardiseerde memory- en error-modellen. Het resultaat is een duurzame, stabiele en toekomstbestendige multi-language omgeving.

---

### 1. Probleemstelling

Heterogene systemen kampen met:

- Incompatibele calling conventions
- Compiler-specifieke symbol-mangling
- Verschillende memory-modellen
- Onvoorspelbare struct-layouts
- Niet-uniforme error-afhandeling
- Moeilijke integratie tussen legacy- en moderne talen

Zonder een centrale interoperabiliteitslaag ontstaat een fragiel systeem met runtime-fouten en hoge onderhoudskosten.

---

### 2. Oplossing: C-ABI als uniforme taalgrens

De kern van de oplossing is één harde regel:

**Alle talen communiceren uitsluitend via een C-ABI-compatibele DLL-laag.**

Dit elimineert:

- Fortran-module-afhankelijkheden
- COBOL-runtime-exposure
- Compiler-specifieke naamgeving
- Taal-specifieke memory-modellen

De C-ABI fungeert als stabiel contract tussen alle talen.

---

### 3. Architectuurmodel

```
COBOL →  
\  
→ C-ABI DLL ← Fortran 95  
/  
ML64 →  
\  
→ C# / Python / Rust
```

#### Kernprincipes

- C is de enige contractlaag
  - Alle exports zijn expliciet via `.def` en `bind(C)`
  - Geen directe taal-naar-taal aanroepen
  - Volledige scheiding tussen implementatie en interface
- 

### 4. Calling-Convention Framework

Alle talen gebruiken de Win64 x64-ABI:

- RCX, RDX, R8, R9 voor parameters
- Shadow space verplicht
- Returnwaarde in RAX
- Structs via pointer
- Geen variadische functies

COBOL gebruikt **CALL-CONVENTION 0**, Fortran gebruikt **bind(C)** - beide exact compatibel.

---

### 5. Type-Mapping Framework

#### 5.1 Primitieve types

Concept	C	Fortran	COBOL	C#	Python
int32	int32_t	integer(c_int)	BINARY-LONG	int	ctypes.c_int
double	double	real(c_double)	BINARY-DOUBLE	double	ctypes.c_double
pointer	void*	type(c_ptr)	USAGE POINTER	IntPtr	ctypes.c_void_p

---

## 5.2 Struct-mapping

Structs worden identiek gedefinieerd in alle talen, met vaste offsets en padding.

---

## 6. Memory-Ownership Framework

### 6.1 Principes

- De aanroeper is eigenaar, tenzij anders gespecificeerd
- Fortran mag alloceren, maar moet een free-routine exporteren
- COBOL mag nooit impliciet alloceren
- C beheert ownership-overdracht

### 6.2 Patronen

- **Caller-allocates:** C/COBOL/C# alloceert → Fortran vult → caller free't
  - **Callee-allocates:** Fortran alloceert → caller free't via free\_xxx()
  - **Buffer-passing:** COBOL en ML64 gebruiken BY VALUE of BY REFERENCE buffers
- 

## 7. Error-Propagation Framework

Elke export gebruikt:

- Integer returncodes
- Uniforme foutcodes (0 = OK, 1 = INVALID\_ARGUMENT, ...)
- Optionele get\_last\_error()
- Optionele error-callback

Geen exceptions over de ABI.

---

## 8. Exportbeheer en ABI-stabiliteit

- .def-file verplicht
  - Symbolen nooit wijzigen
  - Nieuwe functionaliteit = nieuwe functie
  - ABI-versie via get\_abi\_version()
  - Struct-offset-tests bij elke release
-

## 9. Debugging Framework

- dumpbin /exports voor symbol-inspectie
  - Fortran: /traceback
  - COBOL: runtime tracing
  - ML64: register-inspectie
  - C#: P/Invoke marshaling logs
  - Python: ctypes argument tracing
- 

## 10. Teststrategie

Elke export wordt getest in:

- C
- COBOL
- Fortran
- ML64
- C#
- Python

Daarnaast:

- ABI-regressietests
  - Struct-offset-vergelijkingen
  - Memory-ownership tests
  - Error-propagation tests
- 

## 11. Conclusie

Deze interoperabiliteitsarchitectuur:

- Elimineert compiler-lock-in
- Maakt alle talen gelijkwaardig
- Creëert een stabiele, voorspelbare ABI
- Reduceert onderhoudskosten
- Maakt toekomstige taaluitbreidingen triviaal

Het C-ABI-model vormt een duurzame ruggengraat voor organisaties die legacy- en moderne talen willen combineren zonder technische schuld.