InGrid

Innovative Grid-Entwicklungen für ingenieurswissenschaftliche Anwendungen

1. Bericht AP 2.5: Anforderungsanalyse und Spezifikation der erforderlichen FEFLOW-Anpassungen

Projektpartner:

WASY Gesellschaft für wasserwirtschaftliche Planung
und Systemforschung mbH
Waltersdorfer Straße 105
12526 Berlin

Bearbeiter:

Dipl.-Ing. Olaf Arndt

Inhaltsverzeichnis

1 Inhalt

Im ersten Halbjahr des Forschungsprojektes Innovative Grid-Entwicklungen für ingenieurswissenschaftliche Anwendungen (INGRID) wurden für das Arbeitspaket 2.5: Integration und Optimierungen komplexer Grundwassermanagementsysteme die Anpassungen der Finite-Elemente-Simulation FEFLOW® spezifiziert, die erforderlich sind, um FEFLOW in eine GRID-Umgebung zu integrieren.

Bei der derzeitigen FEFLOW Version handelt es sich um eine monolithische Anwendung zur Simulation von 3D-Grundwasserströmungs- und Stofftransportprozessen. Das erste Ziel des Arbeitspaketes 2.5 (AP2.5) ist die Integration der FEM-Simulation FEFLOW in eine Grid-Umgebung. Der Schwerpunkt hierbei liegt auf der Bereitstellung des Rechenkerns als Grid-Service, um vorhandene Modelle unter Nutzung der Grid-Ressourcen berechnen zu können.

2 Zerlegung der FEM-Software FEFLOW

Die bisherige monolithische FEFLOW Architektur erschwert die Nutzung der Software in einer Grid-Architektur erheblich. Zwar bietet FEFLOW mit dem Interface-Manager (IFM) bereits eine Programmierschnittstelle, die Zugriff auf alle wesentlichen Modellparameter und Simulatorfunktionen ermöglicht, dennoch besitzt der IFM einige Einschränkungen, die die Bereitstellung eines „Grid-Services FEFLOW" erschweren. Hierzu zählen vor allem:

- Der IFM ist Dokument- bzw. Modell-abhängig, d.h. alle FEFLOW Funktionen, die über den IFM angesteuert werden, können erst genutzt werden, wenn das zu bearbeitende Modell bereits geladen ist.
- Alle Zugriffe auf die Modellparameter via IFM erfolgen über einen Callback-Mechanismus, wodurch ein Zugriff von außerhalb nur zu definierten Modell- oder Simulationszeitpunkten erfolgen kann.
- Die über den IFM realisierten Funktionalitäten müssen als externe Bibliotheken zur Verfügung gestellt werden und sind Bestandteil des Modells. Die Modelldatei enthält einen Verweis auf die zu aktivierenden IFM Erweiterungen. Modelldateien, die externe Bibliotheken nutzen sollen, müssen daher modifiziert werden.

Darüber hinaus besteht bei der aktuellen FEFLOW Version eine enge Verflechtung zwischen dem Rechenkern und der Benutzeroberfläche (OSF/Motif). Es ist daher zurzeit nicht möglich, eine fensterlose FEFLOW Version zur Berechnung zu nutzen, ohne die grafische Ausgabe in beispielsweise virtuelle XServer umzuleiten.

Für die Bereitstellung einer Grid-fähigen FEFLOW Version wird daher die monolithische Anwendung in Komponenten zerteilt werden. Die Komponenten umfassen hierbei drei wesentliche Bereiche:

- den Rechenkern, der u.a. den Finite-Element-Kode und die Gleichungslöser beinhaltet,
- die Datei Ein- und Ausgabe der Modell- und Ergebnisdateien, den so genannten Serializer und
- die grafische Benutzerschnittstelle, die derzeit noch auf OSF/Motif basiert.

Diese Architektur ermöglicht es, die grafische Benutzeroberfläche (GUI) durch Multiplattfrom-Toolkits (z.B. TrollTech Qt) zu ersetzen, oder sogar projektspezifische GUIs zu entwickeln, die nur eine spezielle oder angepasste Funktionalität bereitstellen. Letzteres ist insbesondere in Bezug auf das InGrid-Projekt von Bedeutung, da hierdurch WEB-basierte Oberflächen bereitgestellt werden können, die auf die entsprechenden Anwendungszenarien zugeschnitten sind. Des Weiteren bietet die Separation der Dateioperationen die Möglichkeit, spezielle Protokolle (z.B. Netzwerkprotokolle) zu realisieren, die das Starten des Rechenkerns auf entfernten Ressourcen erheblich vereinfacht.

Problematisch bei der Zerlegung von FEFLOW in Komponenten ist jedoch der Umstand, dass Aufgrund der Komplexität des Simulationskerns eine Revision von über 80% des Quelltextes erforderlich ist. Daher wurden Schnittstellen spezifiziert, die den Zugriff auf die erforderlichen Komponenten ermöglichen und die durch eine Kapselung der derzeitigen Anwendung schon frühzeitig zu einem lauffähigen, komponentenbasierten FEFLOW-Prototyp führen.

2.1 Der Rechenkern

Für den Rechenkern wurden die erforderlichen Schnittstellen definiert, um auf alle relevanten Funktionen des Simulators zugreifen zu können. Der Aufbau des Simulators ist in Abbildung 1: Struktur des FEFLOW Rechenkerns dargestellt.

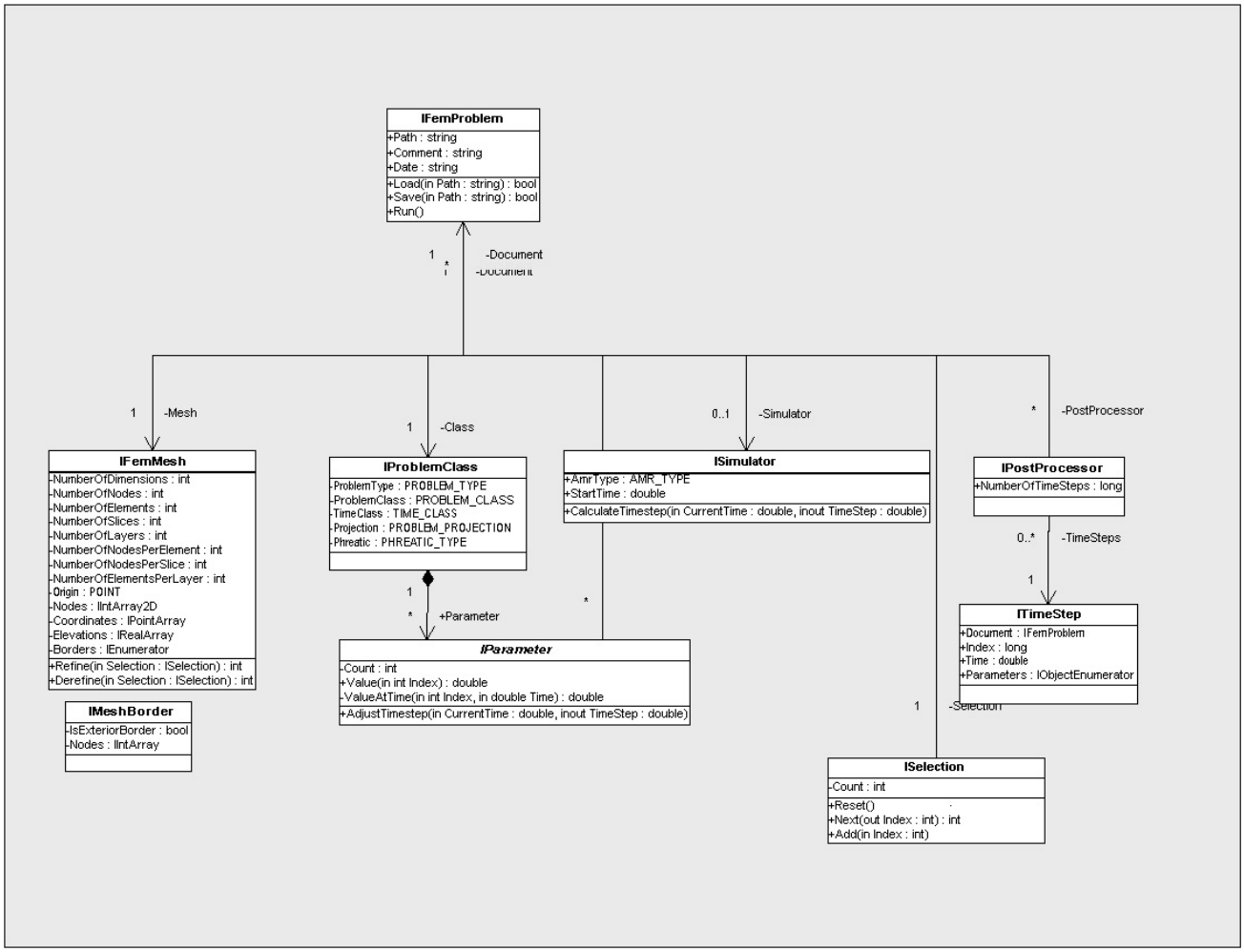

Abbildung 1: Struktur des FEFLOW Rechenkerns

Hierbei wird das FEM-Modell durch die Klasse IFemProblem beschrieben, welche aus einem FEM-Netz (FemMesh) und einer Problemklasse (ProblemClass) besteht. Weiterhin besitzt es Referenzen auf den FEFLOW Simulator und den Postprocessor. Die Problemklasse wiederum aggregiert die Parameter zur Beschreibung des FEM-Modells, deren Struktur in Abbildung 2: Struktur der FEM Parameter dargestellt ist.

Neben den Klassen und Schnittstellen zur Definition und zum Zugriff auf das FEM-Modell und den Simulator wurden noch eine Reihe von Hilfsklassen und Schnittstellen definiert, die z.B. den Zugriff auf die einzelnen Elemente und Elementgeometrien des Finite-Elemente-Netzes, Einheiten von Parametern oder auf verschiedene Ereignisse des Rechenkernes ermöglichen.

Eine vollständige Übersicht der FEFLOW Rechenkernschnittstellen ist in Form der zugehörigen Schnittstellenbeschreibungsdatei in Anhang A gegeben.

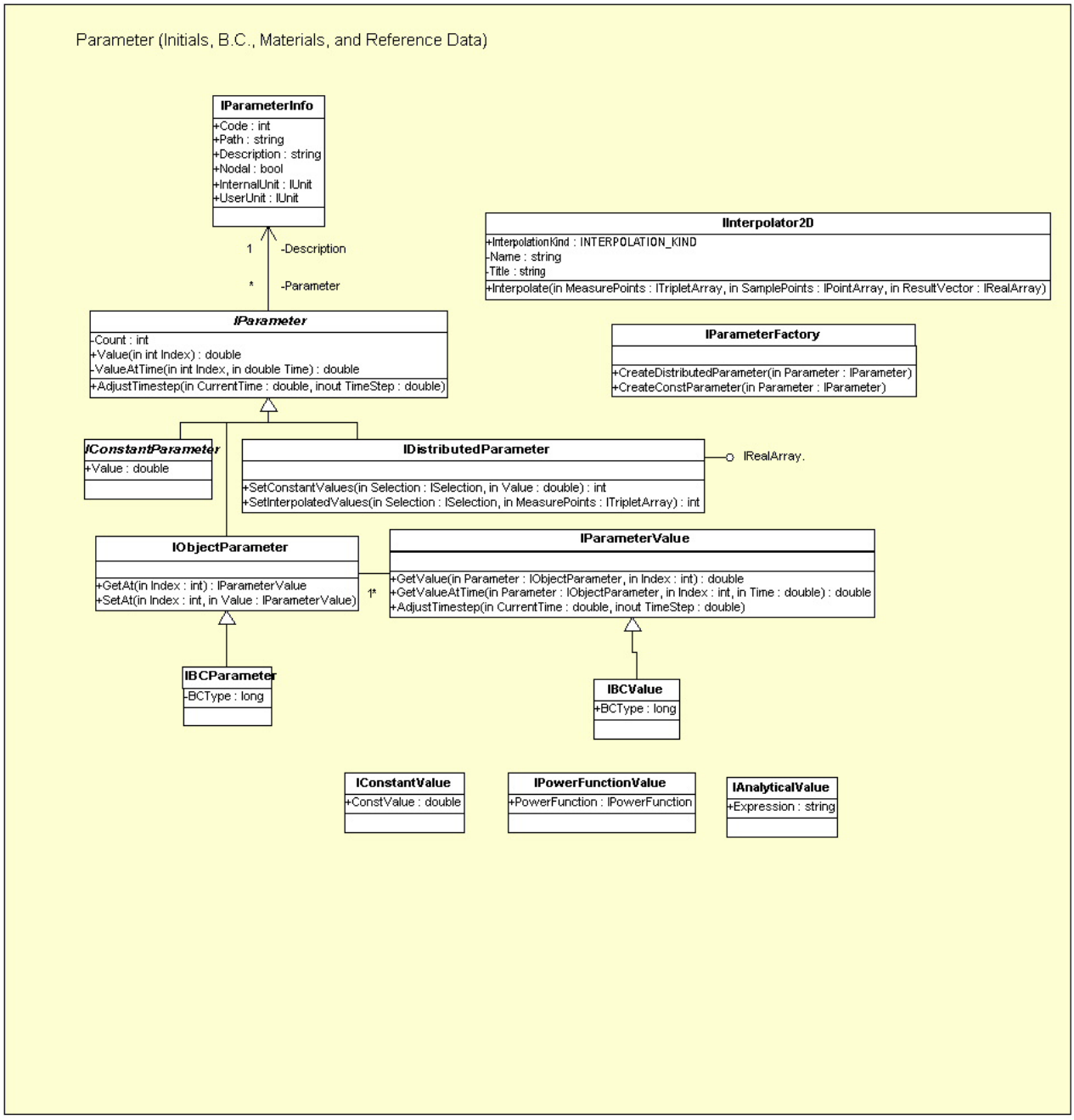

Abbildung 2: Struktur der FEM Parameter

2.2 Der FEFLOW Serializer

In der bisherigen monolithischen Anwendung FEFLOW erfolgte die Datei Ein- und Ausgabe der Modell- und Ergebnisdaten an verschiedenen Stellen unter Verwendung von Systemklassen und –funktionen. Im Rahmen der Komponentenzerlegung von FEFLOW wurde festgelegt, dass die künftige Komponentenbibliothek FEFLOW um eine Serializerkomponente ergänzt wird, die für die gesamte FEFLOW Datei Ein- und Ausgabe zuständig sein wird. Diese Komponente ermöglicht insbesondere einen vereinfachten Einsatz des Rechenkerns in verteilten Umgebungen, da so die Implementation einer Komponente ermöglich wird, die geeignete Netzwerkproto-

kolle unterstützt. Hierdurch kann die aufwendige Verteilung von (dateibasierten) Modelldaten deutlich erleichtert werden. Es ist so beispielsweise nicht mehr notwendig, ein verteiltes Dateisystem zur Verfügung zu haben. Ebenfalls werden die physischen Replikas von Modelldateien minimiert, wodurch sichergestellt werden kann, dass immer die aktuellste Modellversion in die Berechnungen einfließt. Ein weiterer Vorteil eines netzprotokollbasierten Serializer besteht darin, dass besonders in entfernten Ressourcen - wie sie für eine Gridanwendung typisch sind -, kein lokales Arbeitsverzeichnis und somit auch kein Benutzermapping erforderlich ist, da die Anwendung als reine Rechenressource betrachtet werden kann, in die Modelldaten direkt eingespeist und aus der Ergebnisse abgerufen werden, ohne diese temporär zwischenspeichern zu müssen. Hierdurch wird auch dem Sicherheitsaspekt in der Form Rechnung getragen, dass der Zugriff auf eigene Daten durch Dritte deutlich erschwert wird. Es wird lediglich ein lokales temporäres Verzeichnis zum Anlegen von M-Listen (Materiallisten) benötigt, sofern diese in dem Modell enthalten sind.

Die Serializerkomponente muss daher gewährleisten, dass sie:

- Datenstrukturen des FEFLOW Kerns in Datenströme (Streams) kodiert, wie z.B. Dateien oder Netzwerkverbindungen,
- Datenströme in FEFLOW Strukturen dekodiert,
- verschiedene Datenstromarten (binär und ASCII) behandelt,
- unterschiedliche Byte-Ausrichtungen (LSB, MSB) der verschiedenen Hardwareplatformen (z.B. Intel, Motorola) kompensiert,
- alle Datenstrukturen dynamisch verwaltet, d.h. es gibt keine programmseitig vorgegebenen Größenbeschränkungen,
- Dateiformate von Datenformaten entkoppelt,
- ein Versionsmanagement implementiert, das das Einlesen alter Formate in neue Strukturen und umgekehrt ermöglicht,
- zur Fehlerbehandlung die standardisierte Schnittstelle IErrorInfo unterstützt und
- die Möglichkeit der Reperatur defekter Datenströme bietet.

2.3 Die grafische Benutzerschnittstelle

Die enge Verknüpfung der grafischen Benutzeroberfläche (GUI) mit den unterschiedlichsten Funktionen des Rechenkernes stellt eine besondere Herausforderung in verteilten Umgebungen dar. Da die GUI der bisherigen monolithischen FEFLOW-Applikation auf OSF/Motif beruht und aus einem Großteil der Rechenkernfunktionen direkt aufgerufen wird, ist bisher ein XServer unabdingbar. Eine Ausführung der Applikation ohne Umleitung der Oberfläche auf einen XServer ist selbst im Batchbetrieb bisher nicht möglich. Bei verteilten Umgebungen - insbesondere Gridumgebungen – wird hierdurch der Installationsaufwand und die Wartung erschwert, da mindestens ein virtueller XServer zur Verfügung stehen muss, der die Grafikaufrufe bearbeitet. Darüberhinaus existieren in der monolithischen Anwendung Fehlerbehandlungen, die in From von modalen Dialogen gemeldet werden. Dies führt bei der Verwendung von beispielsweise virtuellen XServern dazu, dass es zum Stillstand der Anwendung kommen kann, da die Dialoge durch den Nutzer nicht einsehbar und somit nicht behandelbar sind. Ebenso führen diese Grafikroutinen zu einer Ver-

längerung der Rechenzeit, da sie zwar funktionslos sind, jedoch in jedem Fall ausgeführt werden.

In einer ersten Version der FEFLOW-Komponenten werden daher die grafischen Routinen im Rechenkern durch Callbacks ersetzt, die es gestatten die grafischen Routinen zur Laufzeit bzw. bei der Initialisierung durch unterschiedliche Implementierungen zu ersetzen. In der ersten Version stehen zwei Varianten zur Verfügung:

- die Implementierung zur Verwendung einer Oberfläche, bei der die Kommunikation zwischen Rechenkern und Oberfläche durch Nachrichten erfolgt, und
- eine Leerimplementierung, die bei reinen Grafikoperationen keine Funktion haben und bei Fehlerbehandlungen immer dafür sorgen, dass anstatt eines Fehlerdialogs der Fehler über die Schnittstellen weitergeleitet wird.

3 Anpassungsspezifikation FEFLOW

Die Anpassung, die an der monolithischen Anwendung FEFLOW vorgenommen werden müssen, um einen optimalen Einsatz als Grid-Anwendung zu gewährleisten, umfassen zwei Schritte.

Im ersten Schritt werden die COM-Schnittstellen zu dem Rechenkern spezifiziert und der Zugriff über diese Schnittstellen auf die Funktionen und Parameter des Rechenkerns ermöglicht. Die Realisierung hierbei erfolgt auf Basis der bisherigen Anwendung. Dies bedeutet, dass eine vollständige Auslösung der bisherigen grafischen Bibliothek (OSF/Motif) noch nicht erfolgt. Die zeichnenden Routinen des Rechenkerns werden jedoch über den in Kapitel 2.3 beschriebenen Mechanismus deaktiviert. Für den Betrieb des Rechenkerns muss in dieser Ausbaustufe noch ein XServer bereit stehen, auch wenn die grafischen Routinen keine Darstellungen mehr vornehmen. Ebenfalls erfolgt im ersten Schritt noch keine Auslösung der Datei Ein- und Ausgabe. Es wird der bisherige Mechanismus des Lesens von FEM-Dateien bzw. des Schreibens von Ergebnissen beibehalten. Diese Einschränkungen der ersten Anpassung wurden getroffen, um schon frühzeitig eine arbeitsfähige Lösung zu erhalten, die auf feststehenden Schnittstellen zum Rechenkern beruht.

Im zweiten Schritt erfolgt dann die vollständige Zerlegung von FEFLOW in die drei Komponenten entsprechend Kapitel 2. Das Ziel dieser Zerlegung ist neben der Bereitstellung bzw. Integration des Serializers ein Rechenkern, der vollständig ohne grafische Bibliotheken funktionsfähig ist und so ohne Bereitstellung eines XServers betrieben werden kann. Die Serializerkomponente wird zunächst die bisherigen dateibasierten Funktionalitäten zur Verfügung stellen. Über die Implementation einer Grid-spezifischen Serializerkomponente wird in der weiteren Entwicklung des Arbeitpaketes 2.5 nach Erfordernis entschieden.

Die unterschiedlichen Ausbaustufen der FEFLOW-Anpassungen sind in Abbildung 3: Schrittweise Überführung der FEFLOW Anwendung in ein Komponentenmodell dargestellt.

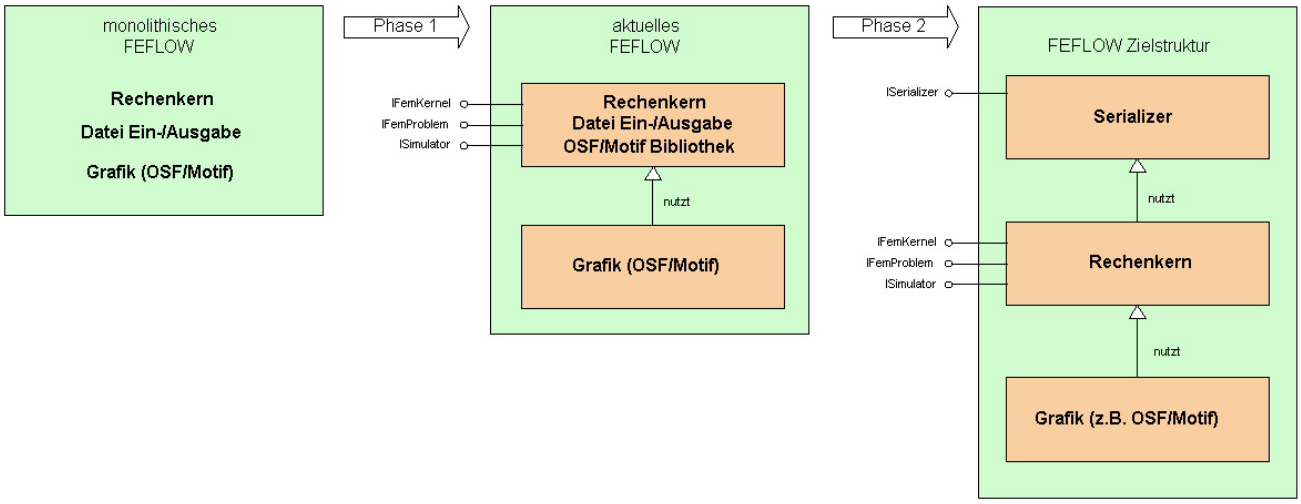

Abbildung 3: Schrittweise Überführung der FEFLOW Anwendung in ein Komponentenmodell

Neben der Bereitstellung des komponentenorientierten FEFLOWs, um den Einsatz in einer Grid-Infrastruktur zu erleichtern, sind noch weitere Bedingungen einzuhalten. Da es sich bei FEFLOW um eine vertriebene und komerziell genutzte Software han-

delt, die neben üblichen Fehlerbehebungen auch inhaltlich weiter entwickelt wird, ist eine parallele Grid-Variante weder wirtschaflich sinnvoll, noch in Bezug auf eine kommezielle Vermarktung der Grid-Infrastruktur zweckmäßig. Bei der Realisierung der Komponentenarchitektur wird daher auf eine projektspezifische Parallelentwicklung verzichtet. Dies bedeutet, dass alle Anpassungen Bestandteil der kommenden FEFLOW-Version werden, und somit die bisherigen Funktionen von FEFLOW uneingeschränkt sichergestellt werden müssen.

Desweiteren handelt es sich bei FEFLOW um eine Software, die für verschiedene Plattformen (z.B. Microsoft Windows, Linux, UNIX) zur Verfügung steht. Da das Microsoft Component Object Model (COM) sich für die Beschreibung und Realisierung der FEFLOW Komponentenarchitektur zwar eignet, in dieser Struktur jedoch nur für die MS Windows Architektur verfügbar ist, müssen die entsprechenden Mechanismen auf allen unterstützten Plattformen nachgebildet werden, um so ebenfalls eine Parallelentwicklung zu FEFLOW zu vermeiden.

4 Realisierungsstand der FEFLOW Spezifikation und Anpassung

Die Spezifikation der Schnittstellen zu dem FEFLOW Rechenkern ist abgeschlossen. In Anhang A ist die vollständige Beschreibung der Rechenkernschnittstellen in Form der Schnittstellenbeschreibungsdatei abgedruckt. Ergänzungen der Schnittstellenbeschreibung können im Zuge der Realisierung des Serializers erforderlich werden. Die aktuelle Struktur der Rechenkern-Schnittstellen wird jedoch bestehen bleiben. Es wird lediglich Erweiterungen geben, die entsprechend dem COM-Paradigma so erfolgen werden, dass die Funktion der bisherigen Schnittstellen gewährleistet bleibt.

Über die Schnittstellenbeschreibung hinaus ist ein komponentenbasiertes FEFLOW verfügbar, das entsprechend der Anforderungsspezifikation Kapitel 3 die erste Ausbauphase realisiert. In dieser Variante wird der FEFLOW-Rechenkern als Bibliothek (shared library) bereitgestellt. Durch diese Bibliothek wird der Rechenkern-Zugriff über die spezifizierten Schnittstellen realisiert. Ebenfalls realisiert wurde der COM-Mechanismus für nicht MS Windows Plattformen, da die aktuelle Zielplattform der Ingrid-Anwendung eine Linux-Architektur ist[1].

[1] Dies wurde für das Arbeitspaket 2.5 seitens der WASY GmbH festgelegt, da beispielsweise Ingrid-Partner wie die Universität Siegen als Betriebssystem ihres Clusters SUSE Linux nutzen, und die Kompatibilität der WASY Entwicklungen sichergestellt sein muss. Daher wird für das WASY Test- und Entwicklungsgrid als Betriebssystem SUSE Linux 10 genutzt.

Anhang A: Schnittstellenbeschreibung FEFLOW Rechenkern

```
#ifndef FEM_KERNEL_H
#define FEM_KERNEL_H

#if !defined KERNEL_EXPORTS && !defined INITGUID
# ifdef WIN32
#  ifdef _DEBUG
#   ifdef NO_HCL
#    pragma comment(lib, "FEMKERNELmd.lib")
#    pragma message("Automatically linking with FEMKERNELmd.dll")
#   else
#    pragma comment(lib, "FEMKERNELd.lib")
#    pragma message("Automatically linking with FEMKERNELd.dll")
#   endif
#  else
#   pragma comment(lib, "FEMKERNEL.lib")
#   pragma message("Automatically linking with FEMKERNEL.dll")
#  endif
# endif
#endif

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <wchar.h>

#include <wclcombase.h>
#include <wclsmartptr.h>

#ifndef NO_NAMESPACE
namespace femkernel {
#endif

#include "femtypes.h"

DEFINE_GUID(IID_IEnumerator,
    0xfc1ce53b,0x3c52,0x4a68,0xa6,0xa4,0x01,0x0e,0x1e,0x1e,0xbd,0x52);
DEFINE_GUID(IID_IIntEnumerator,
    0x92af420d,0xdfcd,0x4abc,0xa0,0x6b,0xc3,0x40,0x53,0x6d,0xa3,0xf0);
DEFINE_GUID(IID_IRealEnumerator,
    0x888d968e,0x3d1c,0x438d,0xb8,0x2d,0x4b,0xf3,0x90,0xe0,0xa0,0x80);
DEFINE_GUID(IID_IBoolEnumerator,
    0x3251824e,0xfc5c,0x4335,0x83,0x47,0x2d,0xed,0x05,0x3f,0x31,0x77);
DEFINE_GUID(IID_IPointEnumerator,
    0x9f684d7b,0x187a,0x4173,0xb2,0xdf,0x1f,0xc5,0x20,0xe6,0xaa,0x5d);
DEFINE_GUID(IID_IArrayEvents,
    0x1d820ebf,0xa23e,0x48d5,0x8b,0xde,0xf7,0xbf,0x65,0x97,0x89,0x22);
DEFINE_GUID(IID_IArray,
    0x17cfc8d7,0x0138,0x418c,0xb7,0xcc,0xf2,0x3d,0x3d,0xc1,0xe7,0x45);
DEFINE_GUID(IID_IArray2DEvents,
    0xa05de188,0xceac,0x47b4,0x99,0x42,0x38,0xbe,0xb7,0x53,0xe0,0xa3);
DEFINE_GUID(IID_IArray2D,
    0xbd953918,0x0ad4,0x4840,0x86,0x68,0xa1,0x90,0xf9,0xe7,0xc4,0x21);
DEFINE_GUID(IID_IIntArray,
    0xbadb23af,0x7ae1,0x4ce2,0xa4,0xcd,0x13,0xc4,0xae,0x48,0x2f,0x3c);
DEFINE_GUID(IID_IRealArray,
```

```
    0x7e7c64e6,0x8b38,0x416f,0xa9,0xc2,0x54,0x6e,0x3a,0xdd,0x99,0xdd);
DEFINE_GUID(IID_IPointArray,
    0x32373ba7,0x929b,0x4f5d,0x9c,0xbf,0x36,0xa1,0x63,0x16,0xe0,0xc3);
DEFINE_GUID(IID_IIntArray2D,
    0x2a75e979,0xe995,0x496f,0xbc,0xd8,0x3e,0xda,0xa6,0x84,0x55,0x38);
DEFINE_GUID(IID_IRealArray2D,
    0x516faf92,0x1f50,0x4a16,0xa0,0xb9,0x39,0xef,0xf9,0xeb,0x16,0x4c);
DEFINE_GUID(IID_IVarArray2D,
    0x084211c3,0x8047,0x4ef5,0xbc,0xe2,0xf2,0x12,0x50,0x12,0x58,0x99);
DEFINE_GUID(IID_IIntVarArray2D,
    0x7cfffdbb,0xb55e,0x4c09,0xa4,0xc5,0x38,0xa3,0x5e,0x24,0xd6,0x45);
DEFINE_GUID(IID_IUnitEvents,
    0x0fbec25f,0x34e5,0x442c,0x85,0x1b,0xce,0x2d,0x39,0x2d,0x44,0xa4);
DEFINE_GUID(IID_IUnit,
    0x94317b7c,0x931c,0x47aa,0x9e,0xf7,0xbc,0x40,0xe6,0xaf,0x7d,0x7f);
DEFINE_GUID(IID_ICreateUnit, // {21243BCE-3E7F-4d0e-9844-23A4AA3584CE}
    0x21243bce, 0x3e7f, 0x4d0e, 0x98, 0x44, 0x23, 0xa4, 0xaa, 0x35, 0x84, 0xce);
DEFINE_GUID(IID_IUnitClass,
    0x40f2383f,0x9a4c,0x4402,0x8b,0x58,0xa7,0x00,0x77,0x05,0x24,0x37);
DEFINE_GUID(IID_IUnitManager,
    0x6aee830a,0xbe46,0x4ef5,0x9b,0xf7,0x2f,0x04,0xb3,0x76,0xaa,0xb4);
DEFINE_GUID(IID_IEnumerable,
    0x208f38c5,0xdc78,0x4105,0x95,0x77,0x12,0x92,0x2e,0x96,0x23,0x54);
DEFINE_GUID(IID_IMonitorEvents,
    0x78ca9f45,0x4c64,0x4ca4,0xab,0x95,0xc0,0x7b,0x12,0x9e,0x11,0x1d);
DEFINE_GUID(IID_IFemProblemEvents,
    0x4c11904b,0xae17,0x410b,0x99,0xd1,0x7e,0x81,0x7c,0x85,0x38,0x5b);
DEFINE_GUID(IID_IFemProblem,
    0x1209783c,0xada7,0x41f3,0x8d,0x5c,0xf6,0xf0,0x15,0xd8,0x2b,0x32);
DEFINE_GUID(IID_IFemMesh,
    0x99642963,0x3b84,0x4e31,0x9d,0xc8,0xd4,0x9a,0x8d,0x7d,0x38,0xf8);
DEFINE_GUID(IID_IObjectEnumerator,
    0xbc8219b2,0xb777,0x42e1,0xa9,0x7e,0x80,0x82,0x7a,0xab,0xdb,0x91);
DEFINE_GUID(IID_ISpatialFilter,
    0x847ce16e,0x1485,0x467e,0xba,0x72,0x3e,0xe5,0x8e,0x2e,0xb6,0x90);
DEFINE_GUID(IID_IGeometry,
    0x1bf5cd52,0x23fe,0x4242,0x82,0x3d,0xfd,0xfc,0x23,0x72,0x58,0x6f);
DEFINE_GUID(IID_IEnvelope,
    0x0acc790b,0x1091,0x4c0a,0x95,0x90,0x04,0x3b,0x10,0xb5,0x85,0x92);
DEFINE_GUID(IID_IProblemClass,
    0x3dc69f3d,0x0ff1,0x4676,0xa6,0x11,0xf9,0x68,0x40,0x21,0x89,0x52);
DEFINE_GUID(IID_IParameter,
    0x94cce1a8,0x14d9,0x45c5,0x9c,0x7f,0x6f,0x1c,0xec,0x8a,0x92,0x39);
DEFINE_GUID(IID_ICreateParameter,   // {9E388FEB-B6B2-41ad-A22B-744875596E95}
    0x9e388feb, 0xb6b2, 0x41ad, 0xa2, 0x2b, 0x74, 0x48, 0x75, 0x59, 0x6e, 0x95);
DEFINE_GUID(IID_IVectorParameter,   // {4FC81D6E-CAF8-41c7-8690-6C04A2B88E29}
    0x4fc81d6e, 0xcaf8, 0x41c7, 0x86, 0x90, 0x6c, 0x4, 0xa2, 0xb8, 0x8e, 0x29);
DEFINE_GUID(IID_ICreateVectorParameter, // {EB8613EF-2695-4823-A3F9-98C2068BF359}
    0xeb8613ef, 0x2695, 0x4823, 0xa3, 0xf9, 0x98, 0xc2, 0x6, 0x8b, 0xf3, 0x59);
DEFINE_GUID(IID_IParameterInfo,
    0xf0523d85,0xea2d,0x49c2,0xa4,0x72,0x72,0x94,0x39,0x42,0x0f,0x28);
DEFINE_GUID(IID_ICreateParameterInfo,     // {AFEB095D-F38F-4010-9B25-
AF89D8CDC001}
    0xafeb095d, 0xf38f, 0x4010, 0x9b, 0x25, 0xaf, 0x89, 0xd8, 0xcd, 0xc0, 0x1);
DEFINE_GUID(IID_ISimulator,
    0xb9939b0c,0x8874,0x459d,0xae,0x4b,0x11,0xa9,0x06,0xaa,0x09,0xa6);
DEFINE_GUID(IID_IPostProcessor,
    0x638fc515,0xa1ad,0x40d1,0xaa,0x9a,0xc8,0xde,0x6f,0x1c,0x31,0x51);
DEFINE_GUID(IID_IFemMeshEvents,
```

```
    0x3f036303,0xf762,0x4cb3,0xb2,0x38,0x9a,0x01,0xc6,0xcc,0x4b,0x70);
DEFINE_GUID(IID_IRefinementEvents,        // {F185832B-C9FC-4a86-A087-
794BBD84A55F}
    0xf185832b, 0xc9fc, 0x4a86, 0xa0, 0x87, 0x79, 0x4b, 0xbd, 0x84, 0xa5, 0x5f);
DEFINE_GUID(IID_IMeshBorder,
    0x31e4330c,0xd9bb,0x4620,0x95,0x1c,0x0d,0xd2,0xd3,0x6d,0xdd,0x56);
DEFINE_GUID(IID_IProblemClassEvents,
    0x8244814a,0xa7d2,0x4e27,0xba,0x9c,0x45,0x59,0x19,0x16,0x53,0x2a);
DEFINE_GUID(IID_ISimulatorEvents,
    0xfb144bf4,0xd174,0x4c81,0xbf,0x7a,0x5c,0x00,0x9c,0xd8,0xbb,0x40);
DEFINE_GUID(IID_IPostProcessorEvents,
    0x1ee78ecf,0x6b88,0x43be,0xaf,0xd9,0x9e,0x8f,0xe0,0x95,0xac,0x97);
DEFINE_GUID(IID_IParameterEvents,
    0x5aed2d05,0xcc1e,0x4497,0x96,0x0f,0x93,0x2c,0x87,0xb9,0x89,0xd4);
DEFINE_GUID(IID_IConstantParameter,
    0x40afa614,0xa089,0x46fd,0x9f,0xa7,0xc6,0xfb,0xe8,0x86,0x68,0x94);
DEFINE_GUID(IID_IClonable,
    0x350a53c1,0x24a8,0x4b74,0xb9,0x44,0xe1,0xd1,0xec,0x83,0xa2,0xe5);
DEFINE_GUID(IID_IDistributedParameter,
    0x83cb87a3,0x7a89,0x4800,0x92,0xb4,0x6b,0xdd,0x88,0xb4,0x48,0xf3);
DEFINE_GUID(IID_IDistributedVectorParameter,// {65AEE05B-B48D-4258-A6E8-
14A936A84D82}
    0x65aee05b, 0xb48d, 0x4258, 0xa6, 0xe8, 0x14, 0xa9, 0x36, 0xa8, 0x4d, 0x82);
DEFINE_GUID(IID_IObjectParameterEvents,
    0x68923161,0x2302,0x4b61,0x93,0x60,0xa0,0x9a,0xce,0x57,0xed,0x06);
DEFINE_GUID(IID_IObjectParameter,
    0x9f8c3b08,0x09dc,0x455e,0x90,0x93,0xaf,0x9e,0x0c,0x31,0xb3,0x18);
DEFINE_GUID(IID_IParameterValue,
    0x2249dc69,0x9ee1,0x4fdc,0x9e,0x82,0x9a,0x92,0x9f,0xbd,0xe8,0x80);
DEFINE_GUID(IID_IBcParameter,
    0xec3d06ea,0xd078,0x42e9,0xac,0x01,0xa5,0xdd,0xa9,0xc6,0xbb,0x5f);
DEFINE_GUID(IID_IBcRefParameter,
    0xa11c5485,0xdfc3,0x4e60,0x96,0xad,0x75,0x1f,0xd1,0x93,0x31,0xfb);
DEFINE_GUID(IID_IConstantValue,
    0x0022d125,0x9695,0x4fb9,0x9e,0xb8,0x3f,0xbd,0xc9,0xab,0x0e,0xc1);
DEFINE_GUID(IID_IConstantBcValue,
    0xb64daebc,0x7839,0x48dd,0xae,0xf8,0x5d,0xb6,0xd5,0x5e,0x80,0x38);
DEFINE_GUID(IID_IBcValue,
    0xaacbc75f,0x2ccc,0x4fcb,0x92,0x44,0x9f,0xfb,0x4a,0xb6,0x00,0xb7);
DEFINE_GUID(IID_ITimeStep,          // {928CD667-9207-43e7-91D9-E2E70E5341E3}
    0x928cd667, 0x9207, 0x43e7, 0x91, 0xd9, 0xe2, 0xe7, 0xe, 0x53, 0x41, 0xe3);
DEFINE_GUID(IID_IFence,             // {2A74AAB7-BB75-4bbb-B834-77E57AEB26D8}
    0x2a74aab7, 0xbb75, 0x4bbb, 0xb8, 0x34, 0x77, 0xe5, 0x7a, 0xeb, 0x26, 0xd8);
DEFINE_GUID(IID_IMap,               // {28A25D93-1069-4040-BE05-7F10CDAEFCD6}
    0x28a25d93, 0x1069, 0x4040, 0xbe, 0x5, 0x7f, 0x10, 0xcd, 0xae, 0xfc, 0xd6);
DEFINE_GUID(IID_ISuperMesh,         // {D4596F54-FAC1-4ff7-ACE2-05C7E4854E84}
    0xd4596f54, 0xfac1, 0x4ff7, 0xac, 0xe2, 0x5, 0xc7, 0xe4, 0x85, 0x4e, 0x84);
DEFINE_GUID(IID_ISuperMeshEvents,   // {19376099-F4FF-410c-8910-2F1C02ED4242}
    0x19376099, 0xf4ff, 0x410c, 0x89, 0x10, 0x2f, 0x1c, 0x2, 0xed, 0x42, 0x42);
DEFINE_GUID(IID_ISuperElement,            // {1D9E4284-86D3-4664-8763-
7F327353C612}
    0x1d9e4284, 0x86d3, 0x4664, 0x87, 0x63, 0x7f, 0x32, 0x73, 0x53, 0xc6, 0x12);
DEFINE_GUID(IID_IFemKernel,         // {2578FCD1-CE23-4f59-968F-FA68A76DF444}
    0x2578fcd1, 0xce23, 0x4f59, 0x96, 0x8f, 0xfa, 0x68, 0xa7, 0x6d, 0xf4, 0x44);
DEFINE_GUID(IID_IFemKernelEvents,   // {76AC4DC0-407A-4dc5-A47A-273DE342BB9B}
    0x76ac4dc0, 0x407a, 0x4dc5, 0xa4, 0x7a, 0x27, 0x3d, 0xe3, 0x42, 0xbb, 0x9b);

// Class IDs to be used in FemCreateInstance()
```

```
DEFINE_GUID(CLSID_FemKernel,  // {2FBC6262-C2A8-42b0-A167-E67E86994E8D}
    0x2fbc6262, 0xc2a8, 0x42b0, 0xa1, 0x67, 0xe6, 0x7e, 0x86, 0x99, 0x4e, 0x8d);
DEFINE_GUID(CLSID_FemProblem,
    0xeaa207be,0x699d,0x44d3,0xbe,0xb7,0xc2,0x27,0x0b,0x92,0x53,0xa5);
DEFINE_GUID(CLSID_Envelope,
    0x57c230b7,0x2d49,0x4e83,0xa5,0x0f,0xae,0xca,0x00,0xc2,0xd8,0x78);
DEFINE_GUID(CLSID_SpatialFilter,
    0xbcd3e40a,0x1190,0x466d,0x9c,0x24,0x03,0x76,0x2d,0xf1,0xed,0x4e);
DEFINE_GUID(CLSID_ObjectParameter,
    0x2614f1eb,0x1227,0x4e04,0xac,0xea,0xe2,0x20,0xeb,0x2c,0xfd,0x06);
DEFINE_GUID(CLSID_BcParameter,
    0x6fdbce54,0xc4bc,0x47eb,0xb7,0xef,0x53,0xc0,0x98,0xc4,0x7b,0x9b);
DEFINE_GUID(CLSID_BcRefParameter,
    0xbaf25772,0x6867,0x49b7,0x83,0x24,0x39,0x4e,0x79,0x1c,0xaa,0x81);
DEFINE_GUID(CLSID_ConstantValue,
    0x6f79893f,0x0451,0x4336,0x84,0x87,0x08,0xc9,0x22,0x7e,0x11,0xae);
DEFINE_GUID(CLSID_ConstantBcValue,
    0xab285c51,0xfc5f,0x4eaa,0x8e,0xd9,0x8f,0x9e,0xcb,0x78,0x70,0x18);
DEFINE_GUID(CLSID_ParameterInfo,
    0x2edfc70a,0xb695,0x49b0,0x9e,0xf3,0xac,0xfa,0x8b,0xff,0x04,0x3a);
DEFINE_GUID(CLSID_ConstantParameter,
    0xd10341e3,0x2d7c,0x47c7,0xa6,0xd8,0x82,0x45,0x69,0x35,0x98,0x1b);
DEFINE_GUID(CLSID_DistributedParameter,
    0xc2cf7e97,0x0222,0x4987,0xb4,0xe9,0xe6,0xba,0x22,0xce,0x78,0x76);
DEFINE_GUID(CLSID_IntArray,
    0x3fb1c9d0,0xd657,0x4444,0xbd,0x44,0xe8,0xfe,0x49,0xc0,0x69,0x93);
DEFINE_GUID(CLSID_RealArray,
    0x2d4624a3,0xf9d7,0x41d6,0x87,0x04,0x2e,0x9a,0xa6,0x2e,0x65,0xa4);
DEFINE_GUID(CLSID_PointArray,
    0x9cbf2599,0x86e0,0x4d2f,0xb4,0x93,0x7d,0x87,0x1c,0x5c,0x73,0x33);
DEFINE_GUID(CLSID_IntArray2D,
    0xdb4c01a0,0xd2dd,0x408a,0x93,0xac,0x53,0x18,0xb9,0xbd,0xf4,0xdb);
DEFINE_GUID(CLSID_RealArray2D,
    0xdd62e77a,0x919f,0x4e07,0xba,0x72,0xf4,0x03,0x89,0x22,0xac,0xdf);
DEFINE_GUID(CLSID_IntVarArray2D,
    0x591c29fc,0xffee,0x4878,0x92,0x8e,0x63,0x85,0x6f,0x12,0x98,0x9e);
DEFINE_GUID(CLSID_Unit,
    0xa0aa01e5,0x6292,0x4045,0x9b,0xef,0x31,0xec,0xec,0x0c,0xa1,0xad);
DEFINE_GUID(CLSID_UnitCollection,
    0x81c4d686,0x1da2,0x4e21,0xa4,0x28,0x2a,0xd6,0xad,0x87,0x35,0xa8);
DEFINE_GUID(CLSID_UnitManager,      // {2E22B3A5-A0A5-401f-BEDD-0CD241B4D693}
    0x2e22b3a5, 0xa0a5, 0x401f, 0xbe, 0xdd, 0xc, 0xd2, 0x41, 0xb4, 0xd6, 0x93);
DEFINE_GUID(CLSID_CreateUnit,   // {4C47A667-0A79-4355-B1F6-FDE000E1D78D}
    0x4c47a667, 0xa79, 0x4355,  0xb1, 0xf6, 0xfd, 0xe0, 0x0, 0xe1, 0xd7, 0x8d);

#ifndef INITGUID

#undef BEGIN_INTERFACE
#define BEGIN_INTERFACE(ifc,base)               \
  interface ifc : public base                   \
  {                                        \
    static const IID& GetIID() { return IID_##ifc; }
#undef END_INTERFACE
#define END_INTERFACE };

//
// Forward references and typedefs
//
```

```
interface IEnumerator;
interface IIntEnumerator;
interface IRealEnumerator;
interface IBoolEnumerator;
interface IPointEnumerator;
interface IArray;
interface IArray2D;
interface IIntArray;
interface IRealArray;
interface IPointArray;
interface IIntArray2D;
interface IRealArray2D;
interface IVarArray2D;
interface IIntVarArray2D;
interface IUnitEvents;
interface IUnit;
interface ICreateUnit;
interface IUnitClass;
interface IUnitManager;
interface IEnumerable;
interface IMonitorEvents;
interface IFemProblemEvents;
interface IFemProblem;
interface IFemMesh;
interface IObjectEnumerator;
interface ISpatialFilter;
interface IGeometry;
interface IEnvelope;
interface IProblemClass;
interface IParameter;
interface ICreateParameter;
interface IVectorParameter;
interface ICreateVectorParameter;
interface IParameterInfo;
interface ICreateParameterInfo;
interface ISimulator;
interface IPostProcessor;
interface IFemMeshEvents;
interface IRefinementEvents;
interface IMeshBorder;
interface IProblemClassEvents;
interface ISimulatorEvents;
interface IPostProcessorEvents;
interface IParameterEvents;
interface IConstantParameter;
interface IClonable;
interface IDistributedParameter;
interface IDistributedVectorParameter;
interface IObjectParameterEvents;
interface IObjectParameter;
interface IParameterValue;
interface IBcParameter;
interface IBcRefParameter;
interface IConstantValue;
interface IConstantBcValue;
interface IBcValue;
interface ITimeStep;
interface IFence;
interface IMap;
```

```
//interface ICoordinateDisplay;
interface ISuperElement;
interface ISuperMesh;
interface ISuperMeshEvents;
interface IFemKernel;
interface IFemKernelEvents;

//
// Smart pointer typedef declarations
//

WCL_SMARTPTR_TYPEDEF(IEnumerator);
WCL_SMARTPTR_TYPEDEF(IIntEnumerator);
WCL_SMARTPTR_TYPEDEF(IRealEnumerator);
WCL_SMARTPTR_TYPEDEF(IBoolEnumerator);
WCL_SMARTPTR_TYPEDEF(IPointEnumerator);
WCL_SMARTPTR_TYPEDEF(IArray);
WCL_SMARTPTR_TYPEDEF(IArray2D);
WCL_SMARTPTR_TYPEDEF(IIntArray);
WCL_SMARTPTR_TYPEDEF(IRealArray);
WCL_SMARTPTR_TYPEDEF(IPointArray);
WCL_SMARTPTR_TYPEDEF(IIntArray2D);
WCL_SMARTPTR_TYPEDEF(IRealArray2D);
WCL_SMARTPTR_TYPEDEF(IVarArray2D);
WCL_SMARTPTR_TYPEDEF(IIntVarArray2D);
WCL_SMARTPTR_TYPEDEF(IUnitEvents);
WCL_SMARTPTR_TYPEDEF(IUnit);
WCL_SMARTPTR_TYPEDEF(ICreateUnit);
WCL_SMARTPTR_TYPEDEF(IUnitClass);
WCL_SMARTPTR_TYPEDEF(IUnitManager);
WCL_SMARTPTR_TYPEDEF(IEnumerable);
WCL_SMARTPTR_TYPEDEF(IMonitorEvents);
WCL_SMARTPTR_TYPEDEF(IFemProblemEvents);
WCL_SMARTPTR_TYPEDEF(IFemProblem);
WCL_SMARTPTR_TYPEDEF(IFemMesh);
WCL_SMARTPTR_TYPEDEF(IObjectEnumerator);
WCL_SMARTPTR_TYPEDEF(ISpatialFilter);
WCL_SMARTPTR_TYPEDEF(IGeometry);
WCL_SMARTPTR_TYPEDEF(IEnvelope);
WCL_SMARTPTR_TYPEDEF(IProblemClass);
WCL_SMARTPTR_TYPEDEF(IParameter);
WCL_SMARTPTR_TYPEDEF(ICreateParameter);
WCL_SMARTPTR_TYPEDEF(IVectorParameter);
WCL_SMARTPTR_TYPEDEF(ICreateVectorParameter);
WCL_SMARTPTR_TYPEDEF(IParameterInfo);
WCL_SMARTPTR_TYPEDEF(ICreateParameterInfo);
WCL_SMARTPTR_TYPEDEF(ISimulator);
WCL_SMARTPTR_TYPEDEF(IPostProcessor);
WCL_SMARTPTR_TYPEDEF(IFemMeshEvents);
WCL_SMARTPTR_TYPEDEF(IRefinementEvents);
WCL_SMARTPTR_TYPEDEF(IMeshBorder);
WCL_SMARTPTR_TYPEDEF(IProblemClassEvents);
WCL_SMARTPTR_TYPEDEF(ISimulatorEvents);
WCL_SMARTPTR_TYPEDEF(IPostProcessorEvents);
WCL_SMARTPTR_TYPEDEF(IParameterEvents);
WCL_SMARTPTR_TYPEDEF(IConstantParameter);
WCL_SMARTPTR_TYPEDEF(IClonable);
WCL_SMARTPTR_TYPEDEF(IDistributedParameter);
WCL_SMARTPTR_TYPEDEF(IDistributedVectorParameter);
```

```
WCL_SMARTPTR_TYPEDEF(IObjectParameterEvents);
WCL_SMARTPTR_TYPEDEF(IObjectParameter);
WCL_SMARTPTR_TYPEDEF(IParameterValue);
WCL_SMARTPTR_TYPEDEF(IBcParameter);
WCL_SMARTPTR_TYPEDEF(IBcRefParameter);
WCL_SMARTPTR_TYPEDEF(IConstantValue);
WCL_SMARTPTR_TYPEDEF(IConstantBcValue);
WCL_SMARTPTR_TYPEDEF(IBcValue);
WCL_SMARTPTR_TYPEDEF(ITimeStep);
WCL_SMARTPTR_TYPEDEF(IFence);
WCL_SMARTPTR_TYPEDEF(IMap);
WCL_SMARTPTR_TYPEDEF(ISuperElement);
WCL_SMARTPTR_TYPEDEF(ISuperMesh);
WCL_SMARTPTR_TYPEDEF(ISuperMeshEvents);
WCL_SMARTPTR_TYPEDEF(IFemKernel);
WCL_SMARTPTR_TYPEDEF(IFemKernelEvents);

extern HRESULT FemCreateInstance(REFCLSID rclsid, IUnknown* pUnkOuter,
                         REFIID riid, void** ppv) /*throw()*/;

template <class T>
void WclCreateInstance(REFCLSID rclsid, IUnknown* pUnkOuter, T** ppv)
throw(_com_error)
{
  HRESULT hr = FemCreateInstance(rclsid, pUnkOuter, T::GetIID(), reinter-
pret_cast<void**>(ppv));
  if (FAILED(hr))
    _com_issue_error(hr);
}

// Create global FEM problem instance
//extern HRESULT COMAPI CreateFemProblem(IFemProblem** pVal) /*throw()*/;
// Retrieve global FEM problem instance
//extern HRESULT COMAPI GetFemProblem(IFemProblem** pVal) /*throw()*/;

//
// Type library items
//

BEGIN_INTERFACE(IEnumerator, IUnknown)
    //
    // Wrapper methods for error-handling
    //
    void Reset(); // throw(_com_error)
    bool MoveNext(); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(raw_Reset) ( ) PURE;
    STDMETHOD(raw_MoveNext) (
        VARIANT_BOOL * pVal ) PURE;
END_INTERFACE

template <class T>
interface ITypedEnumerator : public IEnumerator
{
    //
    // Wrapper methods for error-handling
```

```
    //
    T GetCurrent(); // throw(_com_error)
    void PutCurrent(T newVal); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Current) (
        T* pVal ) PURE;
    STDMETHOD(put_Current) (
        T newVal ) PURE;
};

BEGIN_INTERFACE(IIntEnumerator, ITypedEnumerator<int>)
END_INTERFACE
BEGIN_INTERFACE(IRealEnumerator, ITypedEnumerator<double>)
END_INTERFACE
BEGIN_INTERFACE(IBoolEnumerator, ITypedEnumerator<bool>)
END_INTERFACE
BEGIN_INTERFACE(IPointEnumerator, ITypedEnumerator<RPOINT>)
END_INTERFACE

BEGIN_INTERFACE(IObjectEnumerator, IEnumerator)
    //
    // Wrapper methods for error-handling
    //
    IUnknownPtr GetCurrent(); // throw(_com_error)
    void PutCurrent(IUnknown* newVal); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Current) (
        IUnknown** pVal ) PURE;
    STDMETHOD(put_Current) (
        IUnknown* newVal ) PURE;
END_INTERFACE


BEGIN_INTERFACE(IArray, IUnknown)
    //
    // Wrapper methods for error-handling
    //
    int  GetCount(); // throw(_com_error)
    void Resize(int nCount); // throw(_com_error)
    void RemoveAt(int nIndex); // throw(_com_error)
    void RemoveAll(); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Count) (
        int * pVal ) PURE;
    STDMETHOD(raw_Resize) (
        int Count ) PURE;
    STDMETHOD(raw_RemoveAt) (
        int nIndex) PURE;
    STDMETHOD(raw_RemoveAll) ( ) PURE;
END_INTERFACE
```

```
BEGIN_INTERFACE(IArray2D, IUnknown)
    //
    // Wrapper methods for error-handling
    //
    int      GetRows(); // throw(_com_error)
    int  GetColumns(); // throw(_com_error)
    void Resize(int Rows, int Columns); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Rows) (
        int * pVal ) PURE;
    STDMETHOD(get_Columns) (
        int * pVal ) PURE;
    STDMETHOD(raw_Resize) (
        int Rows,
        int Columns ) PURE;
END_INTERFACE

template <class T>
interface ITypedArray : public IArray
{
    //
    // Wrapper methods for error-handling
    //
    T  GetValue(int nIndex); // throw(_com_error)
    void PutValue(int nIndex, T newVal); // throw(_com_error)
    int      Add(T newVal); // throw(_com_error)
    void InsertAt (int nIndex, T newVal); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Value) (
        int nIndex,
        T * pVal ) PURE;
    STDMETHOD(put_Value) (
        int nIndex,
        T pVal ) PURE;
    STDMETHOD(raw_Add) (
        T    newVal,
        int* nIndex ) PURE;
    STDMETHOD(raw_InsertAt) (
        int nPos,
        T newVal ) PURE;
};

BEGIN_INTERFACE(IIntArray, ITypedArray<int>)
END_INTERFACE
BEGIN_INTERFACE(IRealArray, ITypedArray<double>)
END_INTERFACE
BEGIN_INTERFACE(IPointArray, ITypedArray<RPOINT>)
END_INTERFACE

template <class T>
interface ITypedArray2D : public IArray2D
{
```

```
    typedef T element_t;
    typedef T& element_ref_t;
    typedef const T& const_element_ref_t;

    //
    // Wrapper methods for error-handling
    //
    T GetValue(int nRow, int nColumn); // throw(_com_error)
    void PutValue(int nRow, int nColumn, T newVal); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Value) (
        int nRow,
        int nColumn,
        T* pVal ) PURE;
    STDMETHOD(put_Value) (
        int nRow,
        int nColumn,
        T   newVal ) PURE;
};

BEGIN_INTERFACE(IIntArray2D, ITypedArray2D<int>)
END_INTERFACE

BEGIN_INTERFACE(IRealArray2D, ITypedArray2D<double>)
END_INTERFACE

BEGIN_INTERFACE(IVarArray2D, IUnknown)
    //
    // Wrapper methods for error-handling
    //
    int  GetRows(); // throw(_com_error)
    int  GetColumns(int nRow); // throw(_com_error)
    void Resize(int nRows, int * pColumns); // throw(_com_error)
    void ResizeRow(int nRow, int nColumns); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Rows) (
        int * pVal ) PURE;
    STDMETHOD(get_Columns) (
        int nRow,
        int * pVal ) PURE;
    STDMETHOD(raw_Resize) (
        int nRows,
        int * pColumns ) PURE;
    STDMETHOD(raw_ResizeRow) (
        int nRow,
        int nColumns ) PURE;
END_INTERFACE

template <class T>
interface ITypedVarArray2D : public IVarArray2D
{
    typedef T element_t;
    typedef T& element_ref_t;
```

```
    typedef const T& const_element_ref_t;

    //
    // Wrapper methods for error-handling
    //
    T GetValue(int nRow, int nColumn); // throw(_com_error)
    void PutValue(int nRow, int nColumn, T newVal); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
    STDMETHOD(get_Value) (
        int nRow,
        int nColumn,
        T * pVal ) PURE;
    STDMETHOD(put_Value) (
        int nRow,
        int nColumn,
        T newVal ) PURE;
};

BEGIN_INTERFACE(IIntVarArray2D, ITypedVarArray2D<int>)
END_INTERFACE

BEGIN_INTERFACE(IUnit, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IUnitClassPtr GetClass ( ); // throw(_com_error)
    _bstr_t GetName ( ); // throw(_com_error)
    _bstr_t GetTitle ( ); // throw(_com_error)
    VARIANT_BOOL GetReadOnly ( ); // throw(_com_error)
    enum ORIGIN_TYPE GetOrigin ( ); // throw(_com_error)
    double GetFactor ( ); // throw(_com_error)
    double GetShift ( ); // throw(_com_error)
    double ConvertFrom (
        double inVal,
        struct IUnit * pSourceUnit ); // throw(_com_error)
    HRESULT ConvertValues (
        double* pVal,
      int   nVal,
        struct IUnit * pSourceUnit ); // throw(_com_error)
    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Class) (
        struct IUnitClass ** pVal ) PURE;
    STDMETHOD(get_Name) (
        BSTR * pVal ) PURE;
    STDMETHOD(get_Title) (
        BSTR * pVal ) PURE;
    STDMETHOD(get_ReadOnly) (
        VARIANT_BOOL * pVal ) PURE;
    STDMETHOD(get_Origin) (
        enum ORIGIN_TYPE * pVal ) PURE;
    STDMETHOD(get_Factor) (
        double * pVal ) PURE;
```

```
    STDMETHOD(get_Shift) (
        double * pVal ) PURE;
    STDMETHOD(raw_ConvertFrom) (
        double inVal,
        struct IUnit * pSourceUnit,
        double * pOutVal ) PURE;
    STDMETHOD(raw_ConvertValues) (
        double* pVal,
      int   nVal,
        struct IUnit * pSourceUnit) PURE;
END_INTERFACE

BEGIN_INTERFACE(ICreateUnit, IUnit)
    //
    // Wrapper methods for error-handling
    //

    void PutName (
        _bstr_t pVal ); // throw(_com_error)
    void PutTitle (
        _bstr_t pVal ); // throw(_com_error)
    void PutReadOnly (
        VARIANT_BOOL pVal ); // throw(_com_error)
    void PutOrigin (
        enum ORIGIN_TYPE pVal ); // throw(_com_error)
    void PutFactor (
        double pVal ); // throw(_com_error)
    void PutShift (
        double pVal ); // throw(_com_error)
    //
    // Raw methods provided by interface
    //

    STDMETHOD(put_Name) (
        BSTR pVal ) PURE;
    STDMETHOD(put_Title) (
        BSTR pVal ) PURE;
    STDMETHOD(put_ReadOnly) (
        VARIANT_BOOL pVal ) PURE;
    STDMETHOD(put_Origin) (
        enum ORIGIN_TYPE pVal ) PURE;
    STDMETHOD(put_Factor) (
        double pVal ) PURE;
    STDMETHOD(put_Shift) (
        double pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IUnitClass, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    enum UNIT_CLASS GetClassID ( ); // throw(_com_error)
    _bstr_t GetName ( ); // throw(_com_error)
    _bstr_t GetTitle ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetUnits ( ); // throw(_com_error)
    IUnitPtr GetReferenceUnit ( ); // throw(_com_error)
    IUnitPtr FindUnit (
        _bstr_t UnitString ); // throw(_com_error)
```

```
    HRESULT AddUnit (
        struct IUnit * newVal ); // throw(_com_error)
    HRESULT RemoveUnit (
        struct IUnit * pUnit ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_ClassID) (
        enum UNIT_CLASS * pVal ) PURE;
    STDMETHOD(get_Name) (
        BSTR * pVal ) PURE;
    STDMETHOD(get_Title) (
        BSTR * pVal ) PURE;
    STDMETHOD(get_ReferenceUnit) (
        struct IUnit * * pVal ) PURE;
    STDMETHOD(get_Units) (
        struct IObjectEnumerator * * pVal ) PURE;
    STDMETHOD(raw_FindUnit) (
        BSTR UnitString,
        struct IUnit * * pVal ) PURE;
    STDMETHOD(raw_AddUnit) (
        struct IUnit * newVal ) PURE;
    STDMETHOD(raw_RemoveUnit) (
        struct IUnit * pUnit ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IUnitManager, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IObjectEnumeratorPtr GetUnitClasses ( ); // throw(_com_error)

    IUnitClassPtr FindUnitClass (
        int UnitClassID ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_UnitClasses) (
        struct IObjectEnumerator * * pVal ) PURE;
    STDMETHOD(raw_FindUnitClass) (
        int UnitClassID,
        struct IUnitClass * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IEnumerable, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IEnumeratorPtr GetEnumerator ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //
```

```
    STDMETHOD(get_Enumerator) (
        struct IEnumerator * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IMonitorEvents, IUnknown)

    // Show computational progress
    virtual void OnProgress(int nPercent) PURE; // throw(_com_error)
    // Show message string
    virtual void OnMessage(BSTR pszMessage) PURE; // throw(_com_error)
    // Show formatted additional information text
    virtual void OnInfoText(BSTR pszMessage) PURE; // throw(_com_error)
    // Check for aborting computation
    virtual bool OnCheckAbort() PURE; // throw(_com_error)

END_INTERFACE

BEGIN_INTERFACE(IFemProblem, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    _bstr_t GetPath ( ); // throw(_com_error)
    void PutPath (_bstr_t pVal); // throw(_com_error)
    _bstr_t GetTitle (); // throw(_com_error)
    void PutTitle (_bstr_t pVal ); // throw(_com_error)
    DATE GetDate ( ); // throw(_com_error)
    void PutDate (DATE pVal ); // throw(_com_error)
    int GetFileVersion ( ); // throw(_com_error)
    enum FEMFILE_TYPE GetFileType ( ); // throw(_com_error)
    IFemMeshPtr GetFemMesh ( ); // throw(_com_error)
    ISuperMeshPtr GetSuperMesh ( ); // throw(_com_error)
    IProblemClassPtr GetProblemClass ( ); // throw(_com_error)
    ISimulatorPtr GetSimulator ( ); // throw(_com_error)
    IPostProcessorPtr GetPostProcessor ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetFences ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetMaps ( ); // throw(_com_error)
    HRESULT RequestLoadDocument (_bstr_t Path, long nID = 0); //
throw(_com_error)
    HRESULT RequestSaveDocument (_bstr_t Path, int FileVersion, enum FEMFILE_TYPE
FileType, long nID = 0 ); // throw(_com_error)
    HRESULT CloseDocument ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Path) (BSTR * pVal) PURE;
    STDMETHOD(put_Path) (BSTR pVal) PURE;
    STDMETHOD(get_Title) (BSTR * pVal) PURE;
    STDMETHOD(put_Title) (BSTR pVal) PURE;
    STDMETHOD(get_Date) (DATE * pVal) PURE;
    STDMETHOD(put_Date) (DATE pVal) PURE;
    STDMETHOD(get_FileVersion) (int * pVal) PURE;
    STDMETHOD(get_FileType) (enum FEMFILE_TYPE * pVal) PURE;
    STDMETHOD(get_FemMesh) (struct IFemMesh * * pVal) PURE;
    STDMETHOD(get_SuperMesh) (struct ISuperMesh * * pVal) PURE;
    STDMETHOD(get_ProblemClass) (struct IProblemClass * * pVal) PURE;
```

```
    STDMETHOD(get_Simulator) (struct ISimulator * * pVal) PURE;
    STDMETHOD(get_PostProcessor) (struct IPostProcessor * * pVal) PURE;
    STDMETHOD(get_Fences) (struct IObjectEnumerator * * pVal) PURE;
    STDMETHOD(get_Maps) (struct IObjectEnumerator * * pVal) PURE;
    STDMETHOD(raw_RequestLoadDocument) (BSTR Path, long nID) PURE;
    STDMETHOD(raw_RequestSaveDocument) (BSTR Path, int FileVersion, enum
FEMFILE_TYPE FileType, long nID) PURE;
    STDMETHOD(raw_CloseDocument) () PURE;
END_INTERFACE

BEGIN_INTERFACE(IFemProblemEvents, IUnknown)

    // Notify that the document was loaded completely
    virtual void OnDocumentLoaded (IFemProblem* pDoc, RESULT_TYPE nResult, long
nID) PURE; // throw(_com_error)
    // Notify that the document saving has finished
    virtual void OnDocumentSaved (IFemProblem* pDoc, RESULT_TYPE nResult, long
nID) PURE; // throw(_com_error)

END_INTERFACE

BEGIN_INTERFACE(IFemMesh, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    int GetNumberOfDimensions ( ); // throw(_com_error)
    int GetNumberOfNodes ( ); // throw(_com_error)
    int GetNumberOfElements ( ); // throw(_com_error)
    int GetNumberOfSlices ( ); // throw(_com_error)
    int GetNumberOfLayers ( ); // throw(_com_error)
    int GetNumberOfNodesPerElement ( ); // throw(_com_error)
    int GetNumberOfNodesPerSlice ( ); // throw(_com_error)
    int GetNumberOfElementsPerLayer ( ); // throw(_com_error)
    int GetNumberOfNodesPerElement2D ( ); // throw(_com_error)
    int GetSliceIndex (
      int nSlice ); // throw(_com_error)
    int GetLayerIndex (
      int nLayer ); // throw(_com_error)
    struct RPOINT GetOrigin ( ); // throw(_com_error)
    void PutOrigin (
        struct RPOINT pVal ); // throw(_com_error)
    IIntArray2DPtr GetNodes ( ); // throw(_com_error)
    IIntVarArray2DPtr GetElements ( ); // throw(_com_error)
    IPointArrayPtr GetCoordinates ( ); // throw(_com_error)
    IRealArray2DPtr GetElevations ( ); // throw(_com_error)
    IFemProblemPtr GetDocument ( ); // throw(_com_error)
    struct RRECT GetExtent ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetBorders ( ); // throw(_com_error)
    HRESULT QueryElementCoordinates (
        int nIndex,
        int nPoints,
        struct RPOINT * points ); // throw(_com_error)
    HRESULT QueryElementCoordinatesF (
        int nIndex,
        int nPoints,
        struct FPOINT * points ); // throw(_com_error)
    HRESULT QueryElementNodes (
        int nIndex,
```

```
    int nNodes,
    int* nodes); // throw(_com_error)
IIntEnumeratorPtr SearchElements (
    struct ISpatialFilter * pFilter ); // throw(_com_error)
enum ELEV_STAT GetSliceType (
  int nSlice ); // throw(_com_error)
HRESULT QuerySliceElevations (
    enum PARAMETER_TYPE nType,
    int nSlice,
    int nNodes,
    double* elev ); // throw(_com_error)

//
// Raw methods provided by interface
//

STDMETHOD(get_NumberOfDimensions) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfNodes) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfElements) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfSlices) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfLayers) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfNodesPerElement) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfNodesPerSlice) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfElementsPerLayer) (
    int * pVal ) PURE;
STDMETHOD(get_NumberOfNodesPerElement2D) (
    int * pVal ) PURE;
STDMETHOD(get_Origin) (
    struct RPOINT * pVal ) PURE;
STDMETHOD(put_Origin) (
    struct RPOINT pVal ) PURE;
STDMETHOD(get_Nodes) (
    struct IIntArray2D * * pVal ) PURE;
STDMETHOD(get_Elements) (
    struct IIntVarArray2D * * pVal ) PURE;
STDMETHOD(get_Coordinates) (
    IPointArray * * pVal ) PURE;
STDMETHOD(get_Elevations) (
    struct IRealArray2D * * pVal ) PURE;
STDMETHOD(get_Document) (
    struct IFemProblem * * pVal ) PURE;
STDMETHOD(get_Extent) (
    struct RRECT * pVal ) PURE;
STDMETHOD(get_Borders) (
    struct IObjectEnumerator * * pVal ) PURE;
STDMETHOD(raw_QueryElementCoordinates) (
    int nIndex,
    int nPoints,
    struct RPOINT * points ) PURE;
STDMETHOD(raw_QueryElementCoordinatesF) (
    int nIndex,
    int nPoints,
```

```
        struct FPOINT * points ) PURE;
    STDMETHOD(raw_QueryElementNodes) (
        int nIndex,
        int nNodes,
        int* nodes) PURE;
    STDMETHOD(raw_SearchElements) (
        struct ISpatialFilter * pFilter,
        struct IIntEnumerator * * pVal ) PURE;
    STDMETHOD(get_SliceType) (
      int nSlice,
      enum ELEV_STAT * pVal ) PURE;
    STDMETHOD(raw_QuerySliceElevations) (
        enum PARAMETER_TYPE nType,
        int nSlice,
        int nNodes,
        double* elev) PURE;
    STDMETHOD(get_SliceIndex) (
      int nSlice,
        int * pVal ) PURE;
    STDMETHOD(get_LayerIndex) (
      int nLayer,
        int * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IFemMeshEvents, IUnknown)

    // Notify geometry changes
    STDMETHOD(raw_OnGeometryChanged) (
        struct IFemProblem * pDoc ) PURE;

END_INTERFACE

BEGIN_INTERFACE(IRefinementEvents, IUnknown)

    // Notify starting of refinement process
    STDMETHOD(PreRefineProblem) (
      struct IFemProblem * pDoc,
      int nNodeCount2D,
      int nElemCount2D ) PURE;
    // Notify finishing of refinement process
    STDMETHOD(PostRefineProblem) (
      struct IFemProblem * pDoc,
      VARIANT_BOOL bDiscard ) PURE;
    // Inherit element attributes from previous mesh
    STDMETHOD(OnRefineElementAttributes) (
      struct IFemProblem * pDoc,
        int nNewCount, int* pNewElem,
        int nOldCount, int* pOldElem ) PURE;
    // Inherit node attributes from previous mesh
    STDMETHOD(OnRefineNodeAttributes) (
      struct IFemProblem * pDoc,
      int nNewNode,
      int nNodeFrom,
      int nNodeTo ) PURE;
END_INTERFACE

BEGIN_INTERFACE(ISpatialFilter, IUnknown)
    //
    // Wrapper methods for error-handling
```

```
    //

    IGeometryPtr GetGeometry ( ); // throw(_com_error)
    void PutRefGeometry (
        struct IGeometry * pVal ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Geometry) (
        struct IGeometry * * pVal ) PURE;
    STDMETHOD(putref_Geometry) (
        struct IGeometry * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IGeometry, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    enum GEOMETRY_TYPE GetShape ( ); // throw(_com_error)
    VARIANT_BOOL GetIsEmpty ( ); // throw(_com_error)
    IEnvelopePtr GetEnvelope ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Shape) (
        enum GEOMETRY_TYPE * pVal ) PURE;
    STDMETHOD(get_IsEmpty) (
        VARIANT_BOOL * pVal ) PURE;
    STDMETHOD(get_Envelope) (
        struct IEnvelope * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IEnvelope, IGeometry)
    //
    // Wrapper methods for error-handling
    //

    double GetXmin ( ); // throw(_com_error)
    void PutXmin (
        double pVal ); // throw(_com_error)
    double GetYmin ( ); // throw(_com_error)
    void PutYmin (
        double pVal ); // throw(_com_error)
    double GetXmax ( ); // throw(_com_error)
    void PutXmax (
        double pVal ); // throw(_com_error)
    double GetYmax ( ); // throw(_com_error)
    void PutYmax (
        double pVal ); // throw(_com_error)
    RRECT GetRect ( ); // throw(_com_error)
    void PutRect (
        RRECT newVal ); // throw(_com_error)

    //
```

```
    // Raw methods provided by interface
    //

    STDMETHOD(get_Xmin) (
        double * pVal ) PURE;
    STDMETHOD(put_Xmin) (
        double pVal ) PURE;
    STDMETHOD(get_Ymin) (
        double * pVal ) PURE;
    STDMETHOD(put_Ymin) (
        double pVal ) PURE;
    STDMETHOD(get_Xmax) (
        double * pVal ) PURE;
    STDMETHOD(put_Xmax) (
        double pVal ) PURE;
    STDMETHOD(get_Ymax) (
        double * pVal ) PURE;
    STDMETHOD(put_Ymax) (
        double pVal ) PURE;
    STDMETHOD(get_Rect) (
        RRECT * pVal ) PURE;
    STDMETHOD(put_Rect) (
        RRECT newVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IProblemClass, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IFemProblemPtr GetDocument ( ); // throw(_com_error)
    enum PROBLEM_TYPE GetProblemType ( ); // throw(_com_error)
    enum PROBLEM_CLASS GetProblemClass ( ); // throw(_com_error)
    enum TIME_CLASS GetTimeClass ( ); // throw(_com_error)
    enum PROBLEM_PROJECTION GetProjection ( ); // throw(_com_error)
    enum AMR_TYPE GetAmrType ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetParameters ( ); // throw(_com_error)
    IParameterPtr GetParameter (
        int nID ); // throw(_com_error)
    HRESULT Configure (
        enum PROBLEM_TYPE ic0,
        enum PROBLEM_CLASS ic1,
        enum TIME_CLASS ic2,
        enum PROBLEM_PROJECTION proj,
        enum PHREATIC_TYPE phreatic ); // throw(_com_error)
    IObjectEnumeratorPtr FindParameter (
        struct IParameter * pParam ); // throw(_com_error)
    IObjectEnumeratorPtr FindID (
        int nID ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Document) (
        struct IFemProblem * * pVal ) PURE;
    STDMETHOD(get_ProblemType) (
        enum PROBLEM_TYPE * pVal ) PURE;
    STDMETHOD(get_ProblemClass) (
```

```
        enum PROBLEM_CLASS * pVal ) PURE;
    STDMETHOD(get_TimeClass) (
        enum TIME_CLASS * pVal ) PURE;
    STDMETHOD(get_Projection) (
        enum PROBLEM_PROJECTION * pVal ) PURE;
    STDMETHOD(get_AmrType) (
        enum AMR_TYPE * pVal ) PURE;
    STDMETHOD(get_Parameters) (
        struct IObjectEnumerator * * pVal ) PURE;
    STDMETHOD(get_Parameter) (
        int nID,
        struct IParameter * * pVal ) PURE;
    STDMETHOD(raw_Configure) (
        enum PROBLEM_TYPE ic0,
        enum PROBLEM_CLASS ic1,
        enum TIME_CLASS ic2,
        enum PROBLEM_PROJECTION proj,
        enum PHREATIC_TYPE phreatic ) PURE;
    STDMETHOD(raw_FindParameter) (
        struct IParameter * pParam,
        struct IObjectEnumerator * * pVal ) PURE;
    STDMETHOD(raw_FindID) (
        int nID,
        struct IObjectEnumerator * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IProblemClassEvents, IUnknown)
    // Notify problem class changes
    STDMETHOD(ProblemClassChanged) (
        struct IProblemClass * pProblemClass ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IParameter, IUnknown)
    //
    // Wrapper methods for error-handling
    //
    IFemProblemPtr GetDocument ( ); // throw(_com_error)
    IParameterInfoPtr GetDescription ( ); // throw(_com_error)
    int GetCount ( ); // throw(_com_error)
    double GetValue (
        int nIndex ); // throw(_com_error)
    HRESULT QueryValues (
        int nIndex,
        int nValues,
        double * pVal ); // throw(_com_error)
    double GetMinValue (); // throw(_com_error)
    double GetMaxValue (); // throw(_com_error)
    HRESULT RecalcBounds ( ); // throw(_com_error)
    HRESULT AdjustTimeStep (
        double fCurrentTime,
        double * fTimeStep ); // throw(_com_error)
    HRESULT SetTime (
        double fCurrentTime ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Document) (
```

```
        struct IFemProblem * * pVal ) PURE;
    STDMETHOD(get_Description) (
        struct IParameterInfo * * pVal ) PURE;
    STDMETHOD(get_Count) (
        int * pVal ) PURE;
    STDMETHOD(get_Value) (
        int nIndex,
        double * pVal ) PURE;
    STDMETHOD(get_MinValue) (
        double * pVal ) PURE;
    STDMETHOD(get_MaxValue) (
        double * pVal ) PURE;
    STDMETHOD(raw_RecalcBounds) ( ) PURE;
    STDMETHOD(raw_AdjustTimeStep) (
        double fCurrentTime,
        double * fTimeStep ) PURE;
    STDMETHOD(raw_SetTime) (
        double fCurrentTime ) PURE;
    STDMETHOD(raw_QueryValues) (
        int nIndex,
        int nValues,
        double * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(ICreateParameter, IParameter)
    //
    // Wrapper methods for error-handling
    //

    void PutDocument (
        struct IFemProblem * pVal ); // throw(_com_error)
    void PutDescription (
        struct IParameterInfo * pVal ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(put_Document) (
        struct IFemProblem * pVal ) PURE;
    STDMETHOD(put_Description) (
        struct IParameterInfo * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IVectorParameter, IParameter)
    //
    // Wrapper methods for error-handling
    //
    double GetComponent (int nDim, int nIndex ); // throw(_com_error)
    double GetComponentMin (int nDim); // throw(_com_error)
    double GetComponentMax (int nDim); // throw(_com_error)

    double GetValueX (
        int nIndex ); // throw(_com_error)
    double GetValueY (
        int nIndex ); // throw(_com_error)
    double GetValueZ (
        int nIndex ); // throw(_com_error)
    double GetMinValueX (); // throw(_com_error)
```

```
    double GetMaxValueX (); // throw(_com_error)
    double GetMinValueY (); // throw(_com_error)
    double GetMaxValueY (); // throw(_com_error)
    double GetMinValueZ (); // throw(_com_error)
    double GetMaxValueZ (); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Component) (
        int nDim,
        int nIndex,
        double * pVal ) PURE;
    STDMETHOD(get_ComponentMin) (
        int nDim,
        double * pVal ) PURE;
    STDMETHOD(get_ComponentMax) (
        int nDim,
        double * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IParameterInfo, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    int GetID ( ); // throw(_com_error)
    _bstr_t GetTreePath ( ); // throw(_com_error)
    _bstr_t GetDescription ( ); // throw(_com_error)
    enum PARAMETER_TYPE GetParamType ( ); // throw(_com_error)
    IUnitPtr GetInternalUnit ( ); // throw(_com_error)
    IUnitPtr GetUserUnit ( ); // throw(_com_error)
    void PutUserUnit (struct IUnit * pVal ); // throw(_com_error)
    double GetDefaultValue ( ); // throw(_com_error)
    int GetDimension( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_ID) (
        int * pVal ) PURE;
    STDMETHOD(get_TreePath) (
        BSTR * pVal ) PURE;
    STDMETHOD(get_Description) (
        BSTR * pVal ) PURE;
    STDMETHOD(get_ParamType) (
        enum PARAMETER_TYPE * pVal ) PURE;
    STDMETHOD(get_InternalUnit) (
        struct IUnit * * pVal ) PURE;
    STDMETHOD(get_UserUnit) (
        struct IUnit * * pVal ) PURE;
    STDMETHOD(put_UserUnit) (
        struct IUnit * pVal ) PURE;
    STDMETHOD(get_DefaultValue) (
        double * pVal ) PURE;
    STDMETHOD(get_Dimension) (
        int * pVal ) PURE;
```

```
END_INTERFACE

BEGIN_INTERFACE(ICreateParameterInfo, IParameterInfo)
    //
    // Wrapper methods for error-handling
    //

    void PutID (
        int pVal ); // throw(_com_error)
    void PutTreePath (
        _bstr_t pVal ); // throw(_com_error)
    void PutDescription (
        _bstr_t pVal ); // throw(_com_error)
    void PutParamType (
        enum PARAMETER_TYPE pVal ); // throw(_com_error)
    void PutInternalUnit (
        struct IUnit * pVal ); // throw(_com_error)
    void PutDefaultValue (
        double pVal ); // throw(_com_error)
    void PutDimension (
        int pVal ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(put_ID) (
        int pVal ) PURE;
    STDMETHOD(put_TreePath) (
        BSTR pVal ) PURE;
    STDMETHOD(put_Description) (
        BSTR pVal ) PURE;
    STDMETHOD(put_ParamType) (
        enum PARAMETER_TYPE pVal ) PURE;
    STDMETHOD(put_InternalUnit) (
        struct IUnit * pVal ) PURE;
    STDMETHOD(put_DefaultValue) (
        double pVal ) PURE;
    STDMETHOD(put_Dimension) (
        int pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(ISimulator, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IFemProblemPtr GetDocument ( ); // throw(_com_error)
    double GetCurrentTime ( ); // throw(_com_error)
    double GetFinalTime ( ); // throw(_com_error)
    double GetCurrentTimeIncrement ( ); // throw(_com_error)

    enum SIMULATOR_STATE GetState( ); // throw(_com_error)
    HRESULT Start ( ); // throw(_com_error)
    HRESULT Pause ( ); // throw(_com_error)
    HRESULT Stop ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
```

```
    //

    STDMETHOD(get_Document) (
        struct IFemProblem * * pVal ) PURE;
    STDMETHOD(get_CurrentTime) (
        double * pVal ) PURE;
    STDMETHOD(get_FinalTime) (
        double * pVal ) PURE;
    STDMETHOD(get_CurrentTimeIncrement) (
        double * pVal ) PURE;
    STDMETHOD(get_State) (
        enum SIMULATOR_STATE * pVal ) PURE;
    STDMETHOD(raw_Start) ( ) PURE;
    STDMETHOD(raw_Pause) ( ) PURE;
    STDMETHOD(raw_Stop) ( ) PURE;
END_INTERFACE

BEGIN_INTERFACE(ITimeStep, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IFemProblemPtr GetDocument ( ); // throw(_com_error)
    double GetTime ( ); // throw(_com_error)
    double GetDuration ( ); // throw(_com_error)
    int GetIndex ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Document) (
        struct IFemProblem * * pVal ) PURE;
    STDMETHOD(get_Time) (
      double * pVal ) PURE;
    STDMETHOD(get_Duration) (
      double * pVal ) PURE;
    STDMETHOD(get_Index) (
      int * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IPostProcessor, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IFemProblemPtr GetDocument ( ); // throw(_com_error)
    int GetNumberOfTimeSteps ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetTimeSteps ( ); // throw(_com_error)
    ITimeStepPtr GetCurrentTimeStep (); // throw(_com_error)
    HRESULT RequestLoadTimeStep (struct ITimeStep * pVal, long nID = 0); //
throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Document) (
        struct IFemProblem * * pVal ) PURE;
```

```
    STDMETHOD(get_NumberOfTimeSteps) (
      int * pVal) PURE;
    STDMETHOD(get_TimeSteps) (
      struct IObjectEnumerator * * pVal) PURE;
    STDMETHOD(get_CurrentTimeStep) (
      struct ITimeStep * * pVal) PURE;
    STDMETHOD(raw_RequestLoadTimeStep) (
      struct ITimeStep * pVal,
      long nID) PURE;
END_INTERFACE

BEGIN_INTERFACE(IPostProcessorEvents, IUnknown)
    // Notify that time step loading has finished
    STDMETHOD(raw_OnTimeStepLoaded) (
        struct IFemProblem * pDoc,
      struct ITimeStep* pStep,
      RESULT_TYPE nResult,
      long nID) PURE;
END_INTERFACE

BEGIN_INTERFACE(IMeshBorder, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    bool IsExteriorBorder ( ); // throw(_com_error)
    IIntArrayPtr GetNodes ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_IsExteriorBorder) (
        VARIANT_BOOL * pVal ) PURE;
    STDMETHOD(get_Nodes) (
        struct IIntArray * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(ISimulatorEvents, IUnknown)

    virtual bool EnterSimulator(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool LeaveSimulator(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PreSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PostSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PreTimeStep(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PostTimeStep(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PreFlowSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PostFlowSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PreMassSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PostMassSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PreHeatSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)
    virtual bool PostHeatSimulation(IFemProblem* pDoc) PURE; // throw(_com_error)

END_INTERFACE

BEGIN_INTERFACE(IConstantParameter, ICreateParameter)
    //
    // Wrapper methods for error-handling
    //
```

```
    double GetConstValue ( ); // throw(_com_error)
    void PutConstValue (
        double pVal ); // throw(_com_error)
    HRESULT Resize (
        int newVal ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_ConstValue) (
        double * pVal ) PURE;
    STDMETHOD(put_ConstValue) (
        double pVal ) PURE;
    STDMETHOD(raw_Resize) (
        int newVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IClonable, IUnknown)
    //
    // Wrapper methods for error-handling
    //
    IClonablePtr Clone(); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(raw_Clone) (
        struct IClonable * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IDistributedParameter, ICreateParameter)
    //
    // Wrapper methods for error-handling
    //

    double GetDistValue (
        int nIndex ); // throw(_com_error)
    void PutDistValue (
        int nIndex,
        double pVal ); // throw(_com_error)
    HRESULT Resize (
        int newVal ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_DistValue) (
        int nIndex,
        double * pVal ) PURE;
    STDMETHOD(put_DistValue) (
        int nIndex,
        double pVal ) PURE;
    STDMETHOD(raw_Resize) (
        int newVal ) PURE;
END_INTERFACE
```

```
BEGIN_INTERFACE(IDistributedVectorParameter, IVectorParameter)
    //
    // Wrapper methods for error-handling
    //

    void PutDocument (
        struct IFemProblem * pVal ); // throw(_com_error)
    void PutDescription (
        struct IParameterInfo * pVal ); // throw(_com_error)
    void PutComponent (
      int nDim,
        int nIndex,
        double pVal ); // throw(_com_error)
    HRESULT Resize (
      int nDim,
        int nVal ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(put_Document) (
        struct IFemProblem * pVal ) PURE;
    STDMETHOD(put_Description) (
        struct IParameterInfo * pVal ) PURE;
    STDMETHOD(put_Component) (
        int nDim,
        int nIndex,
        double pVal ) PURE;
    STDMETHOD(raw_Resize) (
      int nDim,
        int nVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IObjectParameter, ICreateParameter)
    //
    // Wrapper methods for error-handling
    //

    IParameterValuePtr GetObjectValue (
        int nIndex ); // throw(_com_error)
    void PutObjectValue (
        int nIndex,
        struct IParameterValue * pVal ); // throw(_com_error)
    HRESULT UpdateBounds (
        double newVal ); // throw(_com_error)
    HRESULT Resize (
        int newVal ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_ObjectValue) (
        int nIndex,
        struct IParameterValue * * pVal ) PURE;
    STDMETHOD(put_ObjectValue) (
        int nIndex,
```

```
        struct IParameterValue * pVal ) PURE;
    STDMETHOD(raw_UpdateBounds) (
        double newVal ) PURE;
    STDMETHOD(raw_Resize) (
        int newVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IParameterValue, IUnknown)
    STDMETHOD(GetValue) (
        struct IObjectParameter * pParam,
        int nIndex,
        double * pVal ) PURE;
    STDMETHOD(GetValueAtTime) (
        struct IObjectParameter * pParam,
        int nIndex,
        double time,
        double * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IBcParameter, IObjectParameter)
    STDMETHOD(get_BcType) (
        int nIndex,
        int * pVal ) PURE;
    STDMETHOD(SetContraints) ( ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IBcRefParameter, IBcParameter)
    STDMETHOD(get_BcParameter) (
        struct IBcParameter * * pVal ) PURE;
    STDMETHOD(put_BcParameter) (
        struct IBcParameter * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IConstantValue, IUnknown)
    STDMETHOD(GetConstValue) (
        struct IObjectParameter * pParam,
        double * pVal ) PURE;
    STDMETHOD(PutConstValue) (
        struct IObjectParameter * pParam,
        double newVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IConstantBcValue, IUnknown)
    STDMETHOD(PutBcType) (
        struct IBcParameter * pParam,
        int newVal ) PURE;
    STDMETHOD(PutBcValue) (
        struct IBcParameter * pParam,
        double newVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IBcValue, IParameterValue)
    STDMETHOD(GetBcType) (
        struct IBcParameter * pParam,
        int nIndex,
        int * pVal ) PURE;
    STDMETHOD(GetBcValue) (
        struct IBcParameter * pParam,
        int nIndex,
```

```
        double * pVal ) PURE;
    STDMETHOD(GetBccType) (
        struct IBcParameter * pParam,
        int nIndex,
        int * pVal ) PURE;
    STDMETHOD(HasBccMax) (
        struct IBcParameter * pParam,
        int nIndex,
        VARIANT_BOOL * pVal ) PURE;
    STDMETHOD(HasBccMin) (
        struct IBcParameter * pParam,
        int nIndex,
        VARIANT_BOOL * pVal ) PURE;
    STDMETHOD(GetBccMaxValue) (
        struct IBcParameter * pParam,
        int nIndex,
        double * pVal ) PURE;
    STDMETHOD(GetBccMinValue) (
        struct IBcParameter * pParam,
        int nIndex,
        double * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IFence, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IPointArrayPtr GetPoints ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Points) (
        struct IPointArray * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(IMap, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    _bstr_t GetPath ( ); // throw(_com_error)
    enum MAP_TYPE GetType ( ); // throw(_com_error)
    enum SHAPE_TYPE GetShape ( ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Path) (
        BSTR * pVal ) PURE;
    STDMETHOD(get_Type) (
        enum MAP_TYPE * pVal ) PURE;
    STDMETHOD(get_Shape) (
        enum SHAPE_TYPE * pVal ) PURE;
END_INTERFACE
```

```
// Interface of a super element
BEGIN_INTERFACE(ISuperElement, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    ISuperMeshPtr GetMesh ( ); // throw(_com_error)
    struct RRECT GetExtent ( ); // throw(_com_error)
    int GetNumberOfNodes ( ); // throw(_com_error)
    int GetNode (int nIndex); // throw(_com_error)
    double GetX (int nIndex); // throw(_com_error)
    double GetY (int nIndex); // throw(_com_error)
    enum EDGE_TYPE GetEdgeType (int nIndex); // throw(_com_error)
    HRESULT QueryCoordinates (
        int* nPoints,
        struct RPOINT * points,
      VARIANT_BOOL bExpand ); // throw(_com_error)
    HRESULT QueryCoordinatesF (
        int* nPoints,
        struct FPOINT * points,
      VARIANT_BOOL bExpand ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Mesh) (
      ISuperMesh * * pVal ) PURE;
    STDMETHOD(get_Extent) (
      struct RRECT * pVal ) PURE;
    STDMETHOD(get_NumberOfNodes) (
      int * pVal ) PURE;
    STDMETHOD(get_Node) (
      int nIndex, int * pVal ) PURE;
    STDMETHOD(get_X) (
      int nIndex, double * pVal ) PURE;
    STDMETHOD(get_Y) (
      int nIndex, double * pVal ) PURE;
    STDMETHOD(get_EdgeType)(
      int nIndex, enum EDGE_TYPE * pVal ) PURE;
    STDMETHOD(raw_QueryCoordinates) (
        int* nPoints,
        struct RPOINT * points,
      VARIANT_BOOL bExpand ) PURE;
    STDMETHOD(raw_QueryCoordinatesF) (
        int* nPoints,
        struct FPOINT * points,
      VARIANT_BOOL bExpand ) PURE;
END_INTERFACE

// Interface of a super-element mesh
BEGIN_INTERFACE(ISuperMesh, IUnknown)
    //
    // Wrapper methods for error-handling
    //

    IFemProblemPtr GetDocument ( ); // throw(_com_error)
    struct RPOINT GetOrigin ( ); // throw(_com_error)
    void PutOrigin (
```

```
        struct RPOINT pVal ); // throw(_com_error)
    struct RRECT GetExtent ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetBorders ( ); // throw(_com_error)
    int GetNumberOfElements ( ); // throw(_com_error)
    IObjectEnumeratorPtr GetElements ( ); // throw(_com_error)
    IObjectEnumeratorPtr SearchElements (
        struct ISpatialFilter * pFilter ); // throw(_com_error)

    //
    // Raw methods provided by interface
    //

    STDMETHOD(get_Document) (
        struct IFemProblem * * pVal ) PURE;
    STDMETHOD(get_Origin) (
        struct RPOINT * pVal ) PURE;
    STDMETHOD(put_Origin) (
        struct RPOINT pVal ) PURE;
    STDMETHOD(get_NumberOfElements) (
        int * pVal ) PURE;
    STDMETHOD(get_Elements) (
        struct IObjectEnumerator * * pVal ) PURE;
    STDMETHOD(get_Extent) (
        struct RRECT * pVal ) PURE;
    STDMETHOD(get_Borders) (
        struct IObjectEnumerator * * pVal ) PURE;
    STDMETHOD(raw_SearchElements) (
        struct ISpatialFilter * pFilter,
        struct IObjectEnumerator * * pVal ) PURE;
END_INTERFACE

BEGIN_INTERFACE(ISuperMeshEvents, IUnknown)

    // Notify geometry changes of the supermesh
    STDMETHOD(OnGeometryChanged) (
        struct IFemProblem * pDoc ) PURE;

END_INTERFACE

BEGIN_INTERFACE(IFemKernel, IUnknown)

  // Start FEFLOW kernel in a separate thread
    virtual bool StartKernel ( int argc, char* argv[] ) /*throw()*/ PURE;
    // Tell FEFLOW to terminate its own thread
    virtual void ShutdownKernel ( ) /*throw()*/ PURE;
    // Retrieve current kernel version, e.g. 5211 means 5.2(p11)
    virtual int  GetKernelVersion ( ) /*throw()*/ PURE;
    // Update the display
    virtual void UpdateDisplay() /*throw()*/ PURE;

END_INTERFACE

BEGIN_INTERFACE(IFemKernelEvents, IUnknown)

  // Notification that the kernel is up and ready to operate.
  virtual void OnKernelUp (IFemKernel* pKernel) PURE; // throw(_com_error)
  // Notification that the kernel is shut down.
  virtual void OnKernelDown (IFemKernel* pKernel) PURE; // throw(_com_error)
  // Print intro message onto the splash screen.
```

```
  virtual void OnIntroMessage (BSTR msg) PURE; // throw(_com_error)
  // Handle kernel assertion failures (optional)
  virtual void OnAssert(BSTR expr, BSTR file, unsigned int line) PURE; //
throw(_com_error)
  // Print a LOG_{ERROR|WARNING|INFO|DEBUG} log message (optional)
  virtual void OnLogMessage (LOG_TYPE nSeverity, BSTR pszMessage) PURE; //
throw(_com_error)
  // Show alert box with given buttons (optional)
  virtual int OnAlert (BSTR pszButtons, int nDefButton, BSTR pszText) PURE; //
throw(_com_error)

END_INTERFACE

#include "femkernel.inl"

#endif // !INITGUID

#ifndef NO_NAMESPACE
}
#endif

#endif // FEM_KERNEL_H
```