JCMsuite C-API

Release 3.10.2

April 11, 2017

This document shows how to use `JCMsuite` from C/C++. We assume that the reader is familiar with the basic concepts of using `JCMsuite`, especially regarding the necessary input files and mesh generation and simulation flow.

The major concept of the C-API reflects the usage of `JCMsuite` via command line. First, the necessary input for the simulation has to be created, namely the content of the **|project.jcmp|**, `materials.jcm`, `sources.jcm`, `layout.jcm`, `boundary_conditions.jcm`. Instead of having those files in a folder on your hard drive, they are created on a so-called *Pinboard*, which can be understood as a virtual drive. Meshing is started by providing a handle to the layout file and creates a `grid.jcm` on the pinboard. Then, the solver can be started, providing a handle to the simulation file, usually **|project.jcmp|**. The simulation results can either be written to the hard drive, or they are created on the Pinbord as well. If they are created on the Pinboard, the C-API gives access to the content of the output files.

An initialization of the interface and finalization after usage takes care of creation and deletion of all global objects within the interface. Further, there is support for using `JCMsuite` simulatenously from different threads.

CHAPTER

ONE

GETTING STARTED

When using JCMsuite from your own C/C++ application the jcm_fetools.h header has to be included. When linking, your application jcm_fetools and jcm_geo dlls have to be provided.

To compile and link your application with JCMsuite you need to add $$JCMROOT/include to your include path and link against the two dlls jcm_fetools and jcm_geo.

When you are using the qmake tool for Makefile generation the following entries should be included in the corresponding .pro file:

```
INCLUDEPATH += $$JCMROOT/include
LIBS += -L$$JCMROOT/lib
LIBS += -ljcm_fetools -ljcm_geo
```

where JCMROOT points to the installation directory of JCMsuite and needs to be defined beforehand.

CHAPTER

TWO

IMPORTANT REMARKS

Pinboard objects

Following type of objects are available on the pinboard:

Data type	Description
Table	JCM table format. Output of many post processes, e.g. Fourier Transform
Grid	Finite Element mesh created by JCMgeo
FieldBag	Finite Element solution. Usually describes a tensor field on a Grid
DataTree	Input files of JCMgeo and JCMsolve. Syntax follows input schema described in the parameter reference
BlobFile	Internal file format

JCM input and DataTree objects

A JCM simulation setup is described by a number of input files for geometry, material properties, source terms, boundary conditions, and numerical settings. These files follow a given syntax specification, described in the Parameter Reference. Internally these files represented as DataTree objects, having data primitives and sub branches. Hence, DataTree objects on the pinboard correspond to JCM input files on the hard drive, and are needed for simulation setup within the C-API.

Indexing

All input and output parameters of the interface start with 0 by convention

CHAPTER

THREE

EXAMPLE APPLICATION

In the following we provide a basic example showing the simulation flow using the C-API. The archive (capi_example.zip) contains all sources together with example input files.

When using `qmake` you can use the `capi_example.pro` for Makefile generation. Before starting please adapt the `JCMROOT` path at the beginning of the `capi_example.pro` file.

When you are not using `qmake` please create a C++-project with the source `main.cpp`, with `$$JCMROOT/include` added to the include path and link against the two dlls `jcm_fetools` and `jcm_geo`.

After successfully compiling the program change to the sub-folder `capi_example` and start the program with

```
>> capi_examples testproject
```

The program reads JCM simulation input files from hard drive, creates the mesh, starts the simulation and prints some results to the console.

Note: You can use the C-API without any harddisk footprint and file reading. The input file reading in the example driver is only used for conveniences.

---

main routine

All files of the C-API start with JCM... Let us look at the main routine showing the main simulation flow:

```
int main(int argc, char* argv[])
{
  if (argc==1)
  {
    std::cout<<"*** Please provide path to JCM project directory"<<std::endl;
    return 1;
  }
```

```
//initialization of JCMsuite
bool initializeMPI=false;
JCMInitialize(true,initializeMPI);
JCMGeoInitialize(initializeMPI);

//read input files
std::string path(argv[1]);
std::vector<std::string> inputFiles;
std::vector<std::string> inputContent;
if (ReadJCMInputFile(path,inputFiles,inputContent))
  return 1;

//input files are loaded onto folder on pinboard
std::string pathPinboard=path+_SEP_+"virtual";

//create mesh
int interrupt=0;
const char* layoutString=inputContent[0].c_str();
std::string gridFileName=pathPinboard+_SEP_+"grid.jcm";

int gridHandle;
if (JCMCreateGrid(&gridHandle,
                  gridFileName.c_str(),
                  layoutString,
                  &interrupt))
{
  std::cerr<<"*** Mesh creation failed"<<std::endl;
  return 1;
}

//upload JCMsuite input files to pinboard
std::vector<int> inputFileHandles;
for (size_t iF=1;iF<inputFiles.size();iF++)
{
  int handleLoc;
  std::string filePath=pathPinboard+_SEP_+inputFiles[iF];
  if (JCMSetDataTreeHandle(&handleLoc,inputContent[iF].c_str(),filePath.c_str()))
  {
    std::cerr<<"*** File \""<<inputFiles[iF]<<"\" could not be created on pinboard"<<std::endl;
    return 1;
  }
  inputFileHandles.push_back(handleLoc);
}

//solve project
```

```
  std::vector<int> resultHandles_(3);
  for (size_t iR=0;iR<resultHandles_.size();iR++)
    resultHandles_[iR]=0;
  int* resultHandles=&resultHandles_[0];
  int nResultHandles;
  int projectHandle=inputFileHandles[0];
  if (JCMSolve(&resultHandles,&nResultHandles,projectHandle,&interrupt))
  {
    std::cerr<<"*** Solving failed"<<std::endl;
    return 1;
  }
  std::cout<<GetPinboardStatus()<<std::endl;


  //read results from pinboard
  for (int iR=0;iR<nResultHandles;iR++)
  {
    const char* fileName_;
    JCMSource(&fileName_,resultHandles_[iR]);

    std::string fileName(fileName_);
    if (fileName.compare(fileName.length()-6,6,"ft.jcm")==0)
      std::cout<<GetTableContent(resultHandles_[iR])<<std::endl;
  }

  //clean up handles
  JCMReleaseHandle(gridHandle);
  for (int iR=0;iR<nResultHandles;iR++)
    JCMReleaseHandle(resultHandles_[iR]);
  for (size_t iR=0;iR<inputFileHandles.size();iR++)
    JCMReleaseHandle(inputFileHandles[iR]);

  JCMFinalize(false);
  JCMGeoFinalize(false);
}
```

First, the C-API has to be initialized via *JCMInitialize* and *JCMGeoInitialize*, where also a license is checked and allocated. Next the input files are read from hard drive and their content and file name stored in C++ strings.

```
std::vector<std::string> inputFiles;
std::vector<std::string> inputContent;
if (ReadJCMInputFile(path,inputFiles,inputContent))
  return 1;
```

Of course this content can be created directly within your own application without file reading. Its content has to follow the JCM input syntax.

First the mesh is created using the layout description in the layoutString:

```
int gridHandle;
if (JCMCreateGrid(&gridHandle,
                  gridFileName.c_str(),
                  layoutString,
                  &interrupt))
{
  std::cerr<<"*** Mesh creation failed"<<std::endl;
  return 1;
}
```

Here, the gridHandle is an integer, which is used to refer to the grid on the pinboard when using corresponding functions of the C-API. The gridFileName defines, in which virtual folder the mesh should be created on the pinboard. The integer interrupt can be used from external to interrupt the mesh creation process. It is checked regularly, and mesh creation is stopped, when it is said to a value other than 0. All function of the C-API return a 1 if they fail which can be used checking their succesful execution.

Next, we upload all other simulation files to the pinboard:

```
if (JCMSetDataTreeHandle(&handleLoc,inputContent[iF].c_str(),filePath.c_str()))
{
  std::cerr<<"*** File \""<<inputFiles[iF]<<"\" could not be created on pinboard"<<std::endl;
  return 1;
}
```

Again, we pass the file content and the file path and obtain an integer handle, referring to the file on the pinboard. The filePath should be a location as it would be specified on your hard drive.

After uploding all files, we solve the project:

```
if (JCMSolve(&resultHandles,&nResultHandles,projectHandle,&interrupt))
{
  std::cerr<<"*** Solving failed"<<std::endl;
  return 1;
}
```

Here, we pass the integer handle projectHandle referring to the project file, when we uploaded it to the pinboard. Handles to all result files, which are created on the pinboard are written to the resultHandles array, their multitude is returned to nResultHandles. Again, the interupt integer is passed and can be set from the calling program to a non-zero value, to stop the simulation. Please note that for all post processes a Format can be specified. If this is set to Pinboard as shown here:

```
PostProcess {
  FourierTransform {
    OutputFileName = "$THIS_results/ft.jcm"
    FieldBagPath = "$THIS_results/fieldbag.jcm"
    NormalDirection = Y
    Format = Pinboard
  }
}
```

the output file is written to the pinboard and not to the hard drive. Note the $THIS tag in the file paths which automatically prepend the path of the simulation folder. In the project file

```
Project {
  ...
  StorageFormat = Pinboard
  ...
```

the StorageFormat can be set to Pinboard which writes the result fieldbag file to the pinboard instead of the hard drive.

In our C-API example we finally call a routine GetTableContent which accesses the result files and prints them to the console. Before exiting, we release all input file and result file handles from the pinboard

```
for (int iR=0;iR<nResultHandles;iR++)
  JCMReleaseHandle(resultHandles_[iR]);
for (size_t iR=0;iR<inputFileHandles.size();iR++)
  JCMReleaseHandle(inputFileHandles[iR]);
```

and finalize the C-API, releasing all of its memory and internal objects:

```
JCMFinalize(false);
JCMGeoFinalize(false);
```

Additional functionality

The function ReadJCMInputFile has no C-API functionality and is just one opportunity to generate the needed simulation description input, by reading files from hard drive.

The C++ subroutines GetTableContent and GetPinboardStatus include additional functionality of the C-API. Let us have a look at them.

The function GetTableContent shows how to obtain information of the result files from the pinboard and write the values of a table into the console:

```
std::string GetTableContent(int handle)
{
  std::stringstream pStream;

  const char* title;
  JCMTableGetTitle(&title,handle);
  int nColumns;
  JCMTableNColumns(&nColumns,handle);

  pStream<<"Table \""<<title<<"\":"<<std::endl;

  for (int iC=0;iC<nColumns;iC++)
  {
    const char* colName;
    JCMTableGetColumnName(&colName,handle,iC);
    pStream<<"Column "<<iC+1<<": "<<colName<<std::endl;
  }

  int nRows;
  JCMTableNRows(&nRows,handle);
  pStream<<"Content:"<<std::endl;
  for (int iR=0;iR<nRows;iR++)
  {
    pStream<<"Row "<<iR<<": ";
    for (int iC=0;iC<nColumns;iC++)
    {
      char type;
      JCMTableGetColumnType(&type,handle,iC);
      switch (type)
      {
      case 'I':
      {
        int entry;
        JCMTableGetIntEntry(&entry,handle,iR,iC);
        pStream<<entry;
        break;
      }
      case 'R':
      {
        double entry;
        JCMTableGetDoubleEntry(&entry,handle,iR,iC);
        pStream<<entry;
        break;
      }
      case 'C':
```

```
      {
        doublecomplex entry;
        JCMTableGetDoubleComplexEntry(&entry,handle,iR,iC);
        pStream<<entry.real<<"+i*"<<entry.imag;
        break;
      }
      default:
        break;
      }
      if (iC<nColumns-1)
        pStream<<",";
      pStream<<"\t";
    }
    pStream<<std::endl;
  }
  return pStream.str();
}
```

The values of the table are read with different functions *JCMTableGetIntEntry*, *JCMTableGetDoubleEntry*, *JCMTableGetDoubleComplexEntry* according to their type, which is obtained via *JCMTableGetColumnType*.

There are a number of functions in the C-API in order to get information about the objects on the pinboard. Some functionality is shown in the routine GetPinboardStatus

```
std::string GetPinboardStatus()
{
  int nHandles;
  JCMPinboardSize(&nHandles);
  int* handles=new int[nHandles];
  JCMHandleList(&handles);

  std::stringstream pStream;
  pStream<<"Pinboard status:"<<std::endl;
  for (int iH=0;iH<nHandles;iH++)
  {
    int handle=handles[iH];
    const char* type;
    JCMWhatIs(&type,handle);
    const char* source;
    JCMSource(&source,handle);
    int nRef;
    JCMHandleReferenceCount(&nRef,handle);
    pStream<<"\""<<type<<"\" object "<<"<"<<source<<">  (NReferences="<<(nRef-1)<<")\n";
    JCMReleaseHandle(handle);
  }
```

```
  delete[] handles;
  return pStream.str();
}
```

where we obtain the number of handles on the pinboard *JCMPinboardSize* and their respective integer handles by *JCMHandleList*. We write access their type by *JCMWhatIs* and their file path location via *JCMSource*. Each time we ask for a handle, as here in *JCMHandleList* the reference count of the handle is increased. Because of that, we call *JCMReleaseHandle* in order to reduce the reference count by one, before leaving the function. If the reference count goes to 0, the object is deleted from the pinboard.

There are further function to obtain information of the mesh, evaluate a fieldbag, etc. All these function are explained in the *Function Reference*.

CHAPTER

FOUR

STRUCT REFERENCE

The following sections give the documentation of the C-API structs.

4.1 DataContent

Purpose

For each pinboard object, a raw data description *DataContent* can be obtained via JCMGetDataContent. This can be used to stream the object, e.g. via MPI, and upload it again onto a pinboard. E.g. JCMUploadGrid uploads the grid to the pinboard from its DataContent.

Struct members

```
struct DataContent
{
  char* data;
  size_t size;
};
```

Parameter	Description
data	array of raw data describing pinboard object
size	size of data

4.2 HeaderEntry

Purpose

A table in JCM format, includes a header with useful parameters besides it data. The entries of the header can be obtained via JCMTableGetMetaEntries. E.g. the header of a Fourier coefficient table includes information about permeability and permittivity values of the exterior domain.

Struct members

```
typedef HeaderEntry
{
  const char* key;
  const void* value;
  char type;
};
```

Parameter	Description
key	key of header entry
value	value of header entry
type	type of header entry: string['S'], integer['I'], real['R'], complex['C']

4.3 PatchHandle

Purpose

A grid contains of a collection of patches of given type and index. JCMsuite supports several patch types: Point, Edge, Triangle, Quadrilateral, Tetrahedron, Pyramid, Prism, and Brick.

Struct members

```
struct PatchHandle
{
  PatchType type;
  int index;
};
```

Parameter	Description
type	type of patch
index	index of patch

4.4 PatchPreview

Purpose

The *PatchPreview* contains general information about a patch of the grid: its location in the interior or exterior PML region, its domain id and isoparametric degree.

Struct members

```
struct PatchPreview
{
  int id;
  int infinityType;
  int isoparametrics[3];
};
```

Parameter	Description
id	domain id of patch
infinityType	infinity type of patch. 0 => interior domain, 1 => exterior domain, 2 => interior/exterior interface
isoparametrics	isoparametric degree of patch

4.5 FacePreview

Purpose

The *FacePreview* contains general information about faces of the grid and their two adjacent cell neighbours, which share the face. In 3D, the faces are Triangles and Quadrilateral, in 2D faces are Edges, in 1D Points.

Struct members

```
struct FacePreview
{
```

```
  int idNeighbours[2];
  int infinityType;
  int interiorTypeNeighbours[2];
  int boundaryType;
  int id;
  int isoparametrics[3];
};
```

Parameter	Description
idNeighbours	domain id of face neighbours
infinityType	infinity type of face. 0 => interior domain, 1 => exterior domain, 2 => interior/exterior interface
interiorTypeNeighbours	infinity type of face neighbours: 0 => interior patch, 1 => exterior PML patch
boundaryType	boundary type of face: -1 => identified, 0 => default, 1 => domain boundary
id	domain id of patch
isoparametrics	isoparametric degree of patch

CHAPTER

FIVE

FUNCTION REFERENCE

The following sections give the documentation of the C-API functions.

5.1 JCMCloseLogFiles

Purpose

closes log file of JCMsolve output which is opened via *JCMRedirectLogToFile*

Function signature

```
int JCMCloseLogFiles()
```

5.2 JCMCreateBlobFileHandle

Purpose

Creates a file handle on the pinboard pointing to a generic empty BlobFile. BlobFiles can be used for internal data storage, passing, etc. and take any data.

Function signature

```
int JCMCreateBlobFileHandle(int* handle, const char* fileName)
```

Parameter	in/output	Description
handle	output	pinboard handle of the created file
fileName	input	file name on the pinboard

5.3 JCMCreateGrid

Purpose

calls JCMgeo and creates mesh from layout description

Function signature

```
int JCMCreateGrid(int* handle, const char* fileName, const char* layoutContent, int* interrupt)
```

Parameter	in/output	Description
handle	output	pinboard handle of the created mesh
fileName	input	file name of created mesh on pinboard
layoutContent	input	JCM layout description
interrupt	input	external interrupt; if set to value unequal zero in calling program, meshing process is aborted

5.4 JCMCreateTableHandle

Purpose

Creates a file handle on the pinboard pointing to an empty Table.

Function signature

```
int JCMCreateTableHandle(int* handle, const char* fileName)
```

Parameter	in/output	Description
handle	output	pinboard handle of the created Table
fileName	input	file name on the pinboard

5.5 JCMFieldBagCartesianFieldValueReference

Purpose

returns pointer to complete field value matrix of Cartesian fieldbag

Function signature

```
int JCMFieldBagCartesianFieldValueReference(double** fieldValues, int handle, int iF)
```

Parameter	in/output	Description
fieldValues	output	pointer to field value matrix
handle	input	pinboard handle of fieldbag
iF	input	field index

5.6 JCMFieldBagCartesianGridDomainIdReference

Purpose

returns pointer to complete domain id matrix of Cartesian fieldbag

Function signature

```
int JCMFieldBagCartesianGridDomainIdReference(int** domainIds, int handle)
```

Parameter	in/output	Description
domainIds	output	pointer to domain id matrix
handle	input	pinboard handle of fieldbag

5.7 JCMFieldBagCartesianGridPoints

Purpose

returns grid coordinates of Cartesian fieldbag along specified direction

Function signature

```
int JCMFieldBagCartesianGridPoints(double** gridPoints, int handle, int iX)
```

Parameter	in/output	Description
gridPoints	output	grid coordinates
handle	input	pinboard handle of fieldbag
iX	input	direction: 0 => X, 1 => Y, 2 =>Z

5.8 JCMFieldBagCartesianGridPointsReference

Purpose

returns pointer to grid coordinates of Cartesian fieldbag along specified direction

Function signature

```
int JCMFieldBagCartesianGridPointsReference(double** gridPoints, int handle, int iX)
```

Parameter	in/output	Description
gridPoints	output	pointer to grid coordinates
handle	input	pinboard handle of fieldbag
iX	input	direction: 0 => X, 1 => Y, 2 =>Z

5.9 JCMFieldBagDerivativeOrder

Purpose

returns order of highest derivative field in fieldbag

Function signature

```
int JCMFieldBagDerivativeOrder(int* derivativeOrder, int handle)
```

Parameter	in/output	Description
derivativeOrder	output	derivative order
handle	input	pinboard handle of fieldbag

5.10 JCMFieldBagDerivativeParameterName

Purpose

returns parameter names of derivatives

Function signature

```
int JCMFieldBagDerivativeParameterName(const char** paraName, int iDerivativeParameter, int handle)
```

Parameter	in/output	Description
paraName	output	parameter name
iDerivativeParameter	input	index of derivative
handle	input	pinboard handle of fieldbag

5.11 JCMFieldBagEvaluateOnCartesianGrid

Purpose

evaluates a number of fields in a fieldbag on a Cartesian mesh

Function signature

```
int JCMFieldBagEvaluateOnCartesianGrid(doublecomplex*** values, int handle, int lengthGridVectorX,
```

Parameter	in/output	Description
values	output	field values
handle	input	pinboard handle of fieldbag
lengthGrid-VectorX	input	size of gridVectorX
gridVectorX	input	x positions of evaluation points
lengthGrid-VectorY	input	size of gridVectorY
gridVectorY	input	y positions of evaluation points
lengthGrid-VectorZ	input	size of gridVectorZ
gridVectorZ	input	z positions of evaluation points
addSingu-larFields	input	specifies, if singular fields are added in case of a singular source. If false, only correction field values are returned
nFields	input	size of fieldIndices
fieldIndices	input	indices of fields which are evaluated

5.12 JCMFieldBagEvaluateOnExteriorPatch

Purpose

evaluates a fieldbag in exterior domain at given coordinates

Function signature

```
int JCMFieldBagEvaluateOnExteriorPatch(doublecomplex** values, int handle, const doublecomplex* coo
```

Param-eter	in/output	Description
values	output	field value
handle	input	pinboard handle of fieldbag
coordi-nates	input	evaluation point coordinates in exterior PML region
iField	input	field index
patchType	input	type of patch where field is evaluated if known. If this is not a valid patch, the patchType is determined from the given coordinates
patchIn-dex	input	index of patch where field is evaluated if known. If this is not a valid patch, the patchIndex is determined from the given coordinates

5.13 JCMFieldBagEvaluateOnPatch

Purpose

evaluates a fieldbag in interior domain at given coordinates

Function signature

```
int JCMFieldBagEvaluateOnPatch(doublecomplex** values, int handle, const double* coordinates, int i
```

Parameter	in/output	Description
values	output	field value
handle	input	pinboard handle of fieldbag
coordinates	input	evaluation point coordinates
iField	input	field index
patchType	input	type of patch where field is evaluated if known. If this is not a valid patch, the patchType is determined from the given coordinates
patchIndex	input	index of patch where field is evaluated if known. If this is not a valid patch, the patchIndex is determined from the given coordinates

5.14 JCMFieldBagFEDegree

Purpose

returns Finite Element Degree of fieldbag on given patch

Function signature

```
int JCMFieldBagFEDegree(int** degrees, int handle, PatchType patchType, int patchIndex)
```

Parameter	in/output	Description
degrees	output	Finite Element degree
handle	input	pinboard handle of fieldbag
patchType	input	type of patch
patchIndex	input	index of patch

5.15 JCMFieldBagIsCartesian

Purpose

returns if fieldbag is Cartesian

Function signature

```
int JCMFieldBagIsCartesian(int* isCartesian, int handle)
```

Parameter	in/output	Description
isCartesian	output	true, if fieldbag is Cartesian, false otherwise
handle	input	pinboard handle of fieldbag

5.16 JCMFieldBagIsComplex

Purpose

returns if fieldbag has complex values

Function signature

```
int JCMFieldBagIsComplex(bool* isComplex, int handle)
```

Parameter	in/output	Description
isComplex	output	true, if fieldbag is complex, false otherwise
handle	input	pinboard handle of fieldbag

5.17 JCMFieldBagIsCylindrical

Purpose

returns if fieldbag is Cylindrical

Function signature

```
int JCMFieldBagIsCylindrical(int* isCylindrical, int handle)
```

Parameter	in/output	Description
isCylindrical	output	true, if fieldbag is Cylindrical, false otherwise
handle	input	pinboard handle of fieldbag

5.18 JCMFieldBagIsTwisted

Purpose

returns if fieldbag has twisted coordinates

Function signature

```
int JCMFieldBagIsTwisted(int* isTwisted, int handle)
```

Parameter	in/output	Description
isTwisted	output	true, if fieldbag has twisted coordinates, false otherwise
handle	input	pinboard handle of fieldbag

5.19 JCMFieldBagNCartesianGridPoints

Purpose

returns number of mesh points of Cartesian fieldbag

Function signature

```
int JCMFieldBagNCartesianGridPoints(int* nGridPoints, int handle, int iX)
```

Parameter	in/output	Description
nGridPoints	output	number of mesh points
handle	input	pinboard handle of fieldbag
iX	input	direction: 0 => X, 1 => Y, 2 =>Z

5.20 JCMFieldBagNComponents

Purpose

returns number of field components of fieldbag. E.g. z-Polarized electric field has 1 component

Function signature

```
int JCMFieldBagNComponents(int* nComponents, int handle)
```

Parameter	in/output	Description
nComponents	output	number of components
handle	input	pinboard handle of fieldbag

5.21 JCMFieldBagNDerivativeParameters

Purpose

returns number of derivative parameters of fieldbag

Function signature

```
int JCMFieldBagNDerivativeParameters(int* nDerivativeParameters, int handle)
```

Parameter	in/output	Description
nDerivativeParameters	output	number of derivative parameters
handle	input	pinboard handle of fieldbag

5.22 JCMFieldBagNFields

Purpose

returns number of fields in fieldbag

Function signature

```
int JCMFieldBagNFields(int* nFields, int handle)
```

Parameter	in/output	Description
nFields	output	number of fields
handle	input	pinboard handle of fieldbag

5.23 JCMFieldBagNSubFields

Purpose

returns number of subfields in fieldbag.

Function signature

```
int JCMFieldBagNSubFields(int* nSubFields, int handle)
```

Parameter	in/output	Description
nSubFields	output	number of sub-fields
handle	input	pinboard handle of fieldbag

5.24 JCMFieldBagPhaseFactor

Purpose

returns phase factor of fieldbag, i.e. Bloch-vector and frequency

Function signature

```
int JCMFieldBagPhaseFactor(doublecomplex* phaseFactor, int handle, int iField, int iSF, int ix)
```

Parameter	in/output	Description
phaseFactor	output	phase factor
handle	input	pinboard handle of fieldbag
iField	output	field index
iSF	input	sub field index
ix	input	0,1,2 => kx, ky, kz; 3 => omega

5.25 JCMFieldBagSubFieldPolarization

Purpose

returns polarization of field

Function signature

```
int JCMFieldBagSubFieldPolarization( const char** polarization, int handle, int iSF)
```

Parameter	in/output	Description
polarization	output	polarization, e.g. z or xyz
handle	input	pinboard handle of fieldbag
iSF	input	sub field index

5.26 JCMFieldBagSubFieldTensorRank

Purpose

returns tensor rank of field

Function signature

```
int JCMFieldBagSubFieldTensorRank(int* rank, int handle, int iSF)
```

Parameter	in/output	Description
rank	output	tensor rank
handle	input	pinboard handle of fieldbag
iSF	input	sub field index

5.27 JCMFieldBagSubFieldType

Purpose

returns type of field

Function signature

```
int JCMFieldBagSubFieldType(const char** type, int handle, int iSF)
```

Parameter	in/output	Description
type	output	type of field
handle	input	pinboard handle of fieldbag
iSF	input	sub field index

5.28 JCMFinalize

Purpose

finalizes JCMsolve. All internal objects are deleted. Should be called, before exiting calling application if *JCMInitialize* was called

Function signature

```
void JCMFinalize(bool finalizeMPI)
```

Parameter	in/output	Description
finalizeMPI	input	true if MPI interface should also be finalized

5.29 JCMGeoCloseLogFiles

Purpose

closes log file of JCMgeo output which is opened via *JCMGeoRedirectLogToFile*

Function signature

```
int JCMGeoCloseLogFiles()
```

5.30 JCMGeoFinalize

Purpose

finalizes JCMgeo. All internal objects are deleted. Should be called, before exiting calling application if *JCMGeoInitialize* was called

Function signature

```
void JCMGeoFinalize(bool finalizeMPI)
```

Parameter	in/output	Description
finalizeMPI	input	true, if MPI interface should also be finalized

5.31 JCMGeoGetSchema

Purpose

returns raw pointer to JCMgeo input schema

Function signature

```
int JCMGeoGetSchema(void** schema)
```

Parameter	in/output	Description
schema	output	pointer to JCMgeo input schema

5.32 JCMGeoGetThreadId

Purpose

returns id of the current thread in which this function is called. Thread has to be registered via *JCMGeoRegisterThread* in advance

Function signature

```
int JCMGeoGetThreadId(int* threadId)
```

Parameter	in/output	Description
threadId	output	id of thread

5.33 JCMGeoInitialize

Purpose

This function has to be called before first usage of any JCMGeo API functions.

Function signature

```
int JCMGeoInitialize(bool initializeMPI)
```

Parameter	in/output	Description
initializeMPI	input	spectifies if the MPI interface should be initialized within JCMgeo

5.34 JCMGeoInitializeInternal

Purpose

for interal use

Function signature

```
int JCMGeoInitializeInternal(FILE* _stdout, FILE* _stderr, const char* applicationFile, const char*
```

Parameter	in/output	Description
_stdout		
_stderr		
applicationFile		
fatherPIDs		
initializeMPI		
nProcessesUser_		

5.35 JCMGeoRedirectLogToFile

Purpose

redirects JCMgeo output to log file

Function signature

```
int JCMGeoRedirectLogToFile(const char* fileStdOut, const char* fileStdErr, const char* mode)
```

Parameter	in/output	Description
fileStdOut	input	file name for stdout output
fileStdErr	input	file name for stderr output
mode	input	file opening mode: a => append, w => open or create

5.36 JCMGeoRegisterThread

Purpose

this function should be called, if JCMgeo is called from a new thread. Enables redirecting of output from different threads via *JCMGeoRedirectLogToFile* and makes thread id available via *JCMGeoGetThreadId*

Function signature

```
int JCMGeoRegisterThread()
```

5.37 JCMGeoSetNumThreads

Purpose

sets number of threads used by JCMgeo

Function signature

```
int JCMGeoSetNumThreads(int nThreads)
```

Parameter	in/output	Description
nThreads	input	number of threads

5.38 JCMGeoUnRegisterThread

Purpose

unregisters thread, registered by *JCMGeoRegisterThread*. Should be called before exit, if thread was registered.

Function signature

```
int JCMGeoUnRegisterThread()
```

5.39 JCMGetDataContent

Purpose

retrieves data content of object on pinboard

Function signature

```
int JCMGetDataContent(const DataContent** content, int handle)
```

Parameter	in/output	Description
content	output	data content of object on pinboard
handle	input	pinboard handle of object

5.40 JCMGetHandle

Purpose

retrieves the handle of an object on the pinboard from its file path location on pinboard

Function signature

```
int JCMGetHandle(int* handle, const char* fileName)
```

Parameter	in/output	Description
handle	output	handle of object on pinboard
fileName	input	file path to object on pinboard

5.41 JCMGetSchema

Purpose

returns raw pointer to JCMsolve input schema

Function signature

```
int JCMGetSchema(void** schema)
```

Parameter	in/output	Description
schema	output	pointer to JCMsolve input schema

5.42 JCMGetThreadId

Purpose

returns id of the current thread in which this function is called. Thread has to be registered via *JCMRegisterThread* in advance

Function signature

```
int JCMGetThreadId(int* threadId)
```

Parameter	in/output	Description
threadId	output	id of thread

5.43 JCMGridBoundingBox

Purpose

returns bounding box of mesh without exterior PML region

Function signature

```
int JCMGridBoundingBox(double* minX, double* maxX, double* minY, double* maxY, double* minZ, double
```

Parameter	in/output	Description
minX	output	minimum x coordinate of mesh
maxX	output	maximum x coordinate of mesh
minY	output	minimum y coordinate of mesh
maxY	output	maximum y coordinate of mesh
minZ	output	minimum z coordinate of mesh
maxZ	output	maximum z coordinate of mesh
handle	input	pinboard handle of grid

5.44 JCMGridCompare

Purpose

compares two grids for equality

Function signature

```
int JCMGridCompare(int* equal, int handle1, int handle2)
```

Parameter	in/output	Description
equal	output	true, if grid are equal
handle1	input	handle to first grid
handle2	input	handle to second grid

5.45 JCMGridCoordinateTransformation

Purpose

returns linear coordinate transformation of grid

Function signature

```
int JCMGridCoordinateTransformation(double** rotation, double** translation, int handle)
```

Parameter	in/output	Description
rotation	output	rotation of grid
translation	output	translation of grid
handle	input	pinboard handle of grid

5.46 JCMGridDisplacementParameterName

Purpose

returns name of grid displacement/derivative parameter

Function signature

```
int JCMGridDisplacementParameterName(const char** name, int handle, int iParameter)
```

Parameter	in/output	Description
name	output	name of parameter
handle	input	pinboard handle of grid
iParameter	output	index of parameter

5.47 JCMGridExtendedBoundingBox

Purpose

returns bounding box of mesh with exterior PML region

Function signature

```
int JCMGridExtendedBoundingBox(double* minX, double* maxX, double* minY, double* maxY, double* minZ
```

Parameter	in/output	Description
minX	output	minimum x coordinate of mesh
maxX	output	maximum x coordinate of mesh
minY	output	minimum y coordinate of mesh
maxY	output	maximum y coordinate of mesh
minZ	output	minimum z coordinate of mesh
maxZ	output	maximum z coordinate of mesh
handle	input	pinboard handle of grid

5.48 JCMGridExteriorPointCoordinatesInRefinedPatch

Purpose

for internal use

Function signature

```
int JCMGridExteriorPointCoordinatesInRefinedPatch(doublecomplex** coordinates, int handle, PatchTyp
```

Parameter	in/output	Description
coordinates		
handle	input	pinboard handle of grid
patchType		
patchIndex		
refinementLevel		
displacementParameter		
simplexGrid		

5.49 JCMGridFacePreview

Purpose

returns preview of face in grid

Function signature

```
int JCMGridFacePreview(FacePreview* faceview, int handle, PatchType patchType, int patchIndex)
```

Parameter	in/output	Description
faceview	output	preview of grid face
handle	input	pinboard handle of grid
patchType	output	patch type of face
patchIndex	output	patch index of face

5.50 JCMGridGetConnectedComponentsOfNeighbours

Purpose

for internal use

Function signature

```
int JCMGridGetConnectedComponentsOfNeighbours(int** connectedComponentInside, int** connectedCompon
```

Parameter	in/output	Description
connectedComponentInside		
connectedComponentOutside		
faces		
nFaces		
handle	input	pinboard handle of grid

5.51 JCMGridGetPatchId

Purpose

returns domain id of patch

Function signature

```
int JCMGridGetPatchId(int* id, int handle, PatchType patchType, int patchIndex)
```

Parameter	in/output	Description
id	output	domain id
handle	input	pinboard handle of grid
patchType	input	type of patch
patchIndex	input	index of patch

5.52 JCMGridManifoldDim

Purpose

returns manifold dimension of grid

Function signature

```
int JCMGridManifoldDim(int* manifoldDim, int handle)
```

Parameter	in/output	Description
manifoldDim	output	manifold dimension
handle	input	pinboard handle of grid

5.53 JCMGridNDisplacementParameters

Purpose

returns number of displacement/derivative parameters of grid

Function signature

```
int JCMGridNDisplacementParameters(int* n, int handle)
```

Parameter	in/output	Description
n	output	number of displacement parameters
handle	input	pinboard handle of grid

5.54 JCMGridNPatches

Purpose

returns number of patches of given type in grid

Function signature

```
int JCMGridNPatches(int* nPatches, int handle, PatchType patchType)
```

Parameter	in/output	Description
nPatches	output	number of patches
handle	input	pinboard handle of grid
patchType	input	type of patches

5.55 JCMGridNSubPatchesInRefinedPatch

Purpose

for internal use

Function signature

```
int JCMGridNSubPatchesInRefinedPatch(int* nSubPatches, int handle, PatchType patchType, int subPatc
```

Parameter	in/output	Description
nSubPatches		
handle	input	pinboard handle of grid
patchType		
subPatchType		
refinementLevel		
simplexGrid		

5.56 JCMGridOutlineEdgesInRefinedPatch

Purpose

for internal use

Function signature

```
int JCMGridOutlineEdgesInRefinedPatch(bool** isOutline, int handle, int patchType, int subPatchInde
```

Parameter	in/output	Description
isOutline		
handle	input	pinboard handle of grid
patchType		
subPatchIndex		
refinementLevel		
simplexGrid		

5.57 JCMGridPatchPointIndices

Purpose

returns point indices of corner points of patch

Function signature

```
int JCMGridPatchPointIndices(int** pointIndices, int handle, PatchType patchtType, int patchIndex)
```

Parameter	in/output	Description
pointIndices	output	point indices of corners
handle	input	pinboard handle of grid
patchtType	input	type of patch
patchIndex	input	index of patch

5.58 JCMGridPatchPreview

Purpose

returns preview of patch

Function signature

```
int JCMGridPatchPreview(PatchPreview* patchview, int handle, PatchType patchType, int patchIndex)
```

Parameter	in/output	Description
patchview	output	preview of patch
handle	input	pinboard handle of grid
patchType	input	type of patch
patchIndex	input	index of patch

5.59 JCMGridPointBarycentricCoordinatesInRefinedPatch

Purpose

for internal use

Function signature

```
int JCMGridPointBarycentricCoordinatesInRefinedPatch(double** bary, int handle, int patchType, int
```

Parameter	in/output	Description
bary		
handle	input	pinboard handle of grid
patchType		
patchIndex		
refinementLevel		
displacementParameter		
simplexGrid		

5.60 JCMGridPointCoordinates

Purpose

returns coordinates of point

Function signature

```
int JCMGridPointCoordinates(double** coordinates, int handle, int pointIndex)
```

Parameter	in/output	Description
coordinates	output	coordinates
handle	input	pinboard handle of grid
pointIndex	input	index of point

5.61 JCMGridPointCoordinatesInRefinedPatch

Purpose

for internal use

Function signature

```
int JCMGridPointCoordinatesInRefinedPatch(double** coordinates, int handle, PatchType patchType, in
```

Parameter	in/output	Description
coordinates		
handle	input	pinboard handle of grid
patchType		
patchIndex		
refinementLevel		
displacementParameter		
simplexGrid		

5.62 JCMGridPointIndicesInRefinedPatch

Purpose

for internal use

Function signature

```
int JCMGridPointIndicesInRefinedPatch(int** pointIndices, int handle, PatchType patchType, int subP
```

Parameter	in/output	Description
pointIndices		
handle	input	pinboard handle of grid
patchType		
subPatchType		
subPatchIndex		
refinementLevel		
simplexGrid		

5.63 JCMGridSpaceDim

Purpose

returns spatial dimension of grid

Function signature

```
int JCMGridSpaceDim(int* spaceDim, int handle)
```

Parameter	in/output	Description
spaceDim	output	spatial dimension of grid
handle	input	pinboard handle of grid

5.64 JCMHandleList

Purpose

returns list of all handles in pinboard. Increases reference count of each handle by 1. The number of handles is obtained via *JCMPinboardSize* and should be used to resize handles list

Function signature

```
int JCMHandleList(int** handles)
```

Parameter	in/output	Description
handles	output	list of handles

5.65 JCMHandleReferenceCount

Purpose

returns the reference count of a pinboard object

Function signature

```
int JCMHandleReferenceCount(int* nReferences, int handle)
```

Parameter	in/output	Description
nReferences	output	reference count of pinboard object
handle	input	pinboard handle of object

5.66 JCMIncreaseReferenceCount

Purpose

increases the reference count of the handle on the pinboard

Function signature

```
int JCMIncreaseReferenceCount(int handle)
```

Parameter	in/output	Description
handle	input	pinboard handle of object

5.67 JCMInitialize

Purpose

This function has to be called before first usage of any other JCM API functions.

Function signature

```
int JCMInitialize(int allocate_license, int initializeMPI)
```

Parameter	in/output	Description
allocate_license	input	false, if only license free functionality will be used, else true
initializeMPI	input	spectifies if the MPI interface should be initialized within JCM

5.68 JCMInitializeInternal

Purpose

for internal use

Function signature

```
int JCMInitializeInternal(FILE* _stdout, FILE* _stderr, int allocate_license, const char* applicati
```

Parameter	in/output	Description
_stdout		
_stderr		
allocate_license		
applicationFile		
fatherPIDs		
initializeMPI		

5.69 JCMIsOnPinboard

Purpose

checks if object with given file path is loaded on pinboard

Function signature

```
int JCMIsOnPinboard(int* isOnPinboard, const char* fileName)
```

Parameter	in/output	Description
isOnPinboard	output	true, if object is loaded on pinboard
fileName	input	

5.70 JCMMain

Purpose

for internal use. Call JCMsolve to start solver via C-API.

Function signature

```
int JCMMain(int argc, char* argv[])
```

Parameter	in/output	Description
argc		
argv[]		

5.71 JCMPinboardSize

Purpose

return number of handles on pinboard

Function signature

```
int JCMPinboardSize(int* size)
```

Parameter	in/output	Description
size	output	number of handles on pinboard

5.72 JCMPostProcess

Purpose

computes post processes of given project. Corresponds to JCMsolve –post_process ... command

Function signature

```
int JCMPostProcess(int** resultHandles, int* nResultHandles, int projectHandle, int* interrupt)
```

Parameter	in/output	Description
resultHandles	output	result handles from post processes
nResultHandles	output	number of created result handles
projectHandle	input	handle to project data tree object containing post processes
interrupt	input	external interrupt; if set to value unequal zero in calling program, meshing process is aborted

5.73 JCMPrincipalAxes

Purpose

returns the principal axes of given 3x3 matrix

Function signature

```
int JCMPrincipalAxes(double*** axes, double matrix[3][3])
```

Parameter	in/output	Description
axes	output	principal axes
matrix[3][3]	input	3x3 matrix values

5.74 JCMRedirectLogToFile

Purpose

redirects JCMsolve output to log file

Function signature

```
int JCMRedirectLogToFile(const char* fileStdOut, const char* fileStdErr, const char* mode)
```

Parameter	in/output	Description
fileStdOut	input	file name for stdout output
fileStdErr	input	file name for stderr output
mode	input	file opening mode: a => append, w => open or create

5.75 JCMRegisterThread

Purpose

this function should be called, if JCMsolve is called from a new thread. Enables redirecting of output from different threads via *JCMRedirectLogToFile* and makes thread id available via *JCMGetThreadId*

Function signature

```
int JCMRegisterThread()
```

5.76 JCMReleaseHandle

Purpose

decreases reference count of pinboard object by 1

Function signature

```
int JCMReleaseHandle(int handle)
```

Parameter	in/output	Description
handle	input	handle of pinboard object

5.77 JCMSetBlobFileContents

Purpose

sets content of a blob file created by *JCMCreateBlobFileHandle*

Function signature

```
int JCMSetBlobFileContents(int handle, const char* text, int sizeText, const char* _Owner, const ch
```

Parameter	in/output	Description
handle	input	pinboard handle of blob file
text	input	file content
sizeText	input	size of text
_Owner	input	owner specifier
_BLOBType	input	type of blob file
hasHead	input	true, if file header should be created

5.78 JCMSetDataTreeHandle

Purpose

creates data tree handle on pinboard with given content. This has to be used to create the input files on the pinboard needed by JCMSolve.

Function signature

```
int JCMSetDataTreeHandle(int* handle, const char* content, const char* fileName)
```

Parameter	in/output	Description
handle	output	handle of created data tree object
content	input	tree content. This should have the same JCM syntax as a usual input file on hard drive
fileName	input	file path, under which the data tree object should be loacted on the pinboard

5.79 JCMSetNumThreads

Purpose

sets number of threads used by JCMsolve

Function signature

```
int JCMSetNumThreads(int nThreads)
```

Parameter	in/output	Description
nThreads	input	number of threads

5.80 JCMSetTableContents

Purpose

sets content of a table created by *JCMCreateTableHandle*

Function signature

```
int JCMSetTableContents(int handle, const char* content, int contentLength)
```

Parameter	in/output	Description
handle	input	pinboard handle of table
content	input	table content
contentLength	input	size of content

5.81 JCMSolve

Purpose

starts JCMsolve for solution of given project. Corresponds to JCMsolve –solve ... command

Function signature

```
int JCMSolve(int** resultHandles, int* nResultHandles, int projectHandle, int* interrupt)
```

Parameter	in/output	Description
resultHandles	output	result handles of fieldbag and from post processes
nResultHandles	output	number of created result handles
projectHandle	input	handle to project data tree object containing project description
interrupt	input	external interrupt; if set to value unequal zero in calling program, meshing process is aborted

5.82 JCMSource

Purpose

returns file path of pinboard object

Function signature

```
int JCMSource(const char** fileName, int handle)
```

Parameter	in/output	Description
fileName	output	file path under which it is located on pinboard
handle	input	handle of object on pinboard

5.83 JCMTableGetColumnName

Purpose

returns name of table column

Function signature

```
int JCMTableGetColumnName(const char** columnName, int handle, int col)
```

Parameter	in/output	Description
columnName	output	name of column
handle	input	pinboard handle of table
col	input	index of column

5.84 JCMTableGetColumnType

Purpose

returns type of table column

Function signature

```
int JCMTableGetColumnType(char* type, int handle, int col)
```

Parameter	in/output	Description
type	output	type of column
handle	input	pinboard handle of table
col	input	index of column

5.85 JCMTableGetDoubleColumn

Purpose

returns pointer to values of real column

Function signature

```
int JCMTableGetDoubleColumn(const double** entry, int handle, int col)
```

Parameter	in/output	Description
entry	output	pointer to column values
handle	input	pinboard handle of table
col	input	index of column

5.86 JCMTableGetDoubleComplexColumn

Purpose

returns pointer to values of complex column

Function signature

```
int JCMTableGetDoubleComplexColumn(const doublecomplex** entry, int handle, int col)
```

Parameter	in/output	Description
entry	output	pointer to column values
handle	input	pinboard handle of table
col	input	index of column

5.87 JCMTableGetDoubleComplexEntry

Purpose

returns table entry of complex type

Function signature

```
int JCMTableGetDoubleComplexEntry(doublecomplex* entry, int handle, int row, int col)
```

Parameter	in/output	Description
entry	output	complex table entry
handle	input	pinboard handle of table
row	input	index of row
col	input	index of column

5.88 JCMTableGetDoubleEntry

Purpose

returns table entry of real type

Function signature

```
int JCMTableGetDoubleEntry(double* entry, int handle, int row, int col)
```

Parameter	in/output	Description
entry	output	real table entry
handle	input	pinboard handle of table
row	input	index of row
col	input	index of column

5.89 JCMTableGetIntColumn

Purpose

returns pointer to values of integer column

Function signature

```
int JCMTableGetIntColumn(const int** entries, int handle, int col)
```

Parameter	in/output	Description
entries	output	pointer to column values
handle	input	pinboard handle of table
col	input	index of column

5.90 JCMTableGetIntEntry

Purpose

returns table entry of integer type

Function signature

```
int JCMTableGetIntEntry(int* entry, int handle, int row, int col)
```

Parameter	in/output	Description
entry	output	integer table entry
handle	input	pinboard handle of table
row	input	index of row
col	input	index of column

5.91 JCMTableGetMetaEntries

Purpose

returns pointer to header entries. Note: memory is allocated dynamically for "headerEntries" parameter and has to be free'd by *JCMTableReleaseMetaEntries*

Function signature

```
int JCMTableGetMetaEntries(HeaderEntry** headerEntries, int* nEntries, int handle)
```

Parameter	in/output	Description
headerEntries	output	pointer to header entries
nEntries	output	size of headerEntries
handle	input	pinboard handle of table

5.92 JCMTableGetTitle

Purpose

return title of table

Function signature

```
int JCMTableGetTitle(const char**, int handle)
```

Parameter	in/output	Description
char**	output	title of table
handle	input	pinboard handle of table

5.93 JCMTableNColumns

Purpose

returns number of columns of table

Function signature

```
int JCMTableNColumns(int* ncol, int handle)
```

Parameter	in/output	Description
ncol	output	number of columns
handle	input	pinboard handle of table

5.94 JCMTableNRows

Purpose

returns number of rows of table

Function signature

```
int JCMTableNRows(int* nrow, int handle)
```

Parameter	in/output	Description
nrow	output	number of rows
handle	input	pinboard handle of table

5.95 JCMTableReleaseMetaEntries

Purpose

releases memory, allocated by *JCMTableGetMetaEntries* for “headerEntries”

Function signature

```
int JCMTableReleaseMetaEntries(HeaderEntry* headerEntries)
```

Parameter	in/output	Description
headerEntries	input	pointer to header entries

5.96 JCMUnRegisterThread

Purpose

unregisters thread, registered by *JCMRegisterThread*. Should be called before exit, if thread was registered.

Function signature

```
int JCMUnRegisterThread()
```

5.97 JCMUploadGrid

Purpose

uploads grid to pinboard. Grid is described by its data content which is obtained via *JCMGetDataContent* function. This function can by used to serialize grid, transfer it as data stream and upload it onto (another) pinobard.

Function signature

```
int JCMUploadGrid(int* handle, const char* fileName, const char* content, int contentLength)
```

Parameter	in/output	Description
handle	output	pinboard handle of the created mesh
fileName	input	file name of created mesh on pinboard
content	input	data content of grid object on pinboard
contentLength	input	size on content

5.98 JCMWhatIs

Purpose

returns data type of pinboard object

Function signature

```
int JCMWhatIs(const char** type, int handle)
```

Parameter	in/output	Description
type	output	data type
handle	input	handle of object on pinboard

5.99 JCMWriteData

Purpose

writes pinboard object to hard drive

Function signature

```
int JCMWriteData(const char* fileName, bool binary, int handle)
```

Parameter	in/output	Description
fileName	input	output file name
binary	input	flag, if data should be written in binary mode
handle	input	handle of object on pinboard

CHAPTER

SIX

LEGAL INFORMATION