I Overview
I Overview – Contents

Preface 4

Typographical Conventions 6

Contents of all MpCCI Manuals 22
Preface

MpCCI (Mesh-based parallel Code Coupling Interface) is the standard for simulation code coupling. MpCCI has been developed at the Fraunhofer Institute SCAI in order to provide an application independent interface for the coupling of different simulation codes.

Codes Supported by MpCCI

MpCCI 3.1 enables a direct communication between the coupled codes by providing adapters for a growing number of commercial codes. These code adapters make use of the already existing application programming interfaces (APIs) of the simulation tools. This technique allows for an easy installation of MpCCI at the end users site without changing the standard installation of the simulation codes.

A list of currently supported codes is given in the Release Notes.

Internal Architecture

The MpCCI 3.1 environment consists of several components:

- MpCCI Code Adapter allow to adapt MpCCI to commercial codes through their standard code APIs without any changes in the source of the simulation code.

- The MpCCI Graphical User Interface provides a comfortable way to define the coupling setup and to start the simulation - independent of the codes involved in the coupled application.

- The MpCCI Coupling Server is the ”heart” of the MpCCI system. Environment handling, communication between the codes, neighborhood computation, and interpolation are part of this kernel.

Standardized Quantities

One major advantage of having compatible code adapters for all codes supported by MpCCI is the standardization of coupling parameters and procedures independent from the used code pairing. MpCCI provides unified quantity definitions for

- Global quantities: time, iteration, residuals
- Mass source and sink: production species
- Momentum sources: e.g. Lorentz forces
- Energy sources: e.g. joule heat
- Material properties: e.g. electrical conductivity
• Boundary condition values: e.g. temperature or pressure
• Boundary condition gradients: e.g. heat flux density
• Grid data: nodal positions or displacements
• And chemical components: e.g. for reaction kinetics

**MpCCI Manuals**

The MpCCI documentation is split up into several manuals which are also called parts. Each part aims at a special kind of readers.

**Release Notes** The Release Notes contain information on changes versus prior versions of MpCCI. They are thus interesting for users, who have some experience with earlier versions of MpCCI.

**Installation Guide** The Installation Guide describes how to install MpCCI. It also contains information about the licensing.

**Getting Started** is intended for new users of MpCCI. The most important features of MpCCI are described by following a typical setup of a coupled simulation.

**User Manual** The User Manual contains a complete overview of the functions and features of MpCCI. This includes information on code coupling, command line options and functions of the MpCCI GUI.

**Codes Manual** The Codes Manual contains code-specific information. For each code which can be coupled with MpCCI a section is included.

**Tutorial** is a collection of examples which are explained in detail.

**Programmers Guide** The Programmers Guide is intended for users who want to write their own code adapters or who want to use MpCCI as a library.
Typographical Conventions

This manual adheres to a set of typographical conventions so that you can recognize actions and items. The following list illustrates each of the conventions:

- Text you enter from the keyboard or outputs is written in a typewriter font and surrounded by a gray box, e.g.: `mpcci gui`
- Filenames are enclosed in quotation marks, "example.txt". All paths are given with a slash (/) as directory separator, e.g. "mpcci/doc/pdf". On Windows systems this must be replaced by a backslash (\).
- Meta variables represent values and are enclosed in angle brackets as in `<MpCCI.home>`. They can appear everywhere and always should be replaced by appropriate values.
- Environment variables are always written in uppercase typewriter letters, like `VARIABLE`.
- Buttons in the MpCCI GUI look like buttons, e.g. [Next].
- Entries of the menu have a colored background, sub-menus are separated by an arrow. E.g. `File→Open Project` means the submenu `Open Project` of the `File` menu.
- Other options which can be selected are written like `Option`.
- Names of software are written in a sans-serif font, like `MpCCI`.
- Links can be clicked directly in the PDF and HTML versions of the manual and are marked blue there, this applies to links within the manual like `IV-2 Setting up a Coupled Simulation` or to web pages `www.scai.fraunhofer.de/mpcci`. 
# Contents of all MpCCI Manuals

## I Overview

Preface 4

Typographical Conventions 6

Contents of all MpCCI Manuals 22

## II Release Notes

1 Changes and New Features in the initial MpCCI 3.1 4
   1.1 Introduction ........................................ 4
   1.2 MpCCI 3.1.1-1 Release information .................. 4
   1.3 MpCCI 3.1.0-3 Patch information .................... 4
   1.4 MpCCI 3.1.0 Release information ..................... 5
   1.4.1 Installation, Configuration and Licensing ......... 5
   1.4.2 Coupling Server .................................. 5
   1.4.3 MpCCI Visualizer .................................. 6
   1.4.4 MpCCI GUI ........................................ 6
   1.4.5 Command Line Interface ............................ 6
   1.4.6 Abaqus .......................................... 6
   1.4.7 Flowmaster ....................................... 6
   1.4.8 FLUENT ........................................ 7
   1.4.9 ICEPAK .......................................... 7
   1.4.10 PERMAS ...................................... 7
   1.4.11 RadTherm ....................................... 7
   1.4.12 STAR-CD ....................................... 7

2 Prerequisites for MpCCI Installation 8

3 Supported Platforms in MpCCI 3.1.0 9
   3.1 Platforms Supported by the MpCCI 3.1.0 Server .... 9
   3.2 Codes Supported by MpCCI 3.1.0 on Different Platforms 11

4 Known Bugs and Limitations in MpCCI 3.1.0 16

## III Installation Guide
1 Installation Overview

2 Before the Installation
2.1 Downloading MpCCI .......................... 6
2.2 Where to Install .......................... 7
2.3 The Perl Interpreter ....................... 8
2.4 The Java Runtime Environment .......... 9
2.5 OpenSSH for Microsoft Windows .......... 10
2.6 MpCCI-RSH for Microsoft Windows ...... 11
2.7 MPICH for Microsoft Windows .......... 11

3 Installation of the MpCCI Software ............. 12
3.1 Multi-platform for UNIX, Linux and Microsoft Windows ................. 12
3.2 Local Microsoft Windows Installation with the MSI .......... 13

4 Immediately After the Installation - Quick Installation Tests without a License ............. 16
4.1 Your Home Directory under Microsoft Windows .......... 16
4.2 Testing the MpCCI Working Environment and Perl .......... 16
4.3 Testing whether MpCCI Finds Your Simulation Codes .......... 17

5 Licensing ............................................. 19
5.1 Request for a License File ................. 19
5.2 Installing and Activating a License .......... 20
5.2.1 Configure a License Manager as UNIX service .......... 21
5.2.2 Configure a License Manager as Windows service .......... 21
5.3 Defining the License Server .......... 24
5.4 Multiple License Servers .......... 26
5.5 Testing the License Server .......... 26

6 Configuring the MpCCI Users Environment ............. 27
6.1 Accessing Remote Hosts .......... 27
6.2 Configuring MpCCI via Environment Variables .......... 29

7 Testing the MpCCI Installation and Communication ............. 30

8 Troubleshooting ............................................. 32
8.1 Secure shell in general .......... 32
8.2 OpenSSH under Microsoft Windows .......... 32
8.3 rsh, rcp and rlogin under Microsoft Windows .......... 32
8.4 MPICH under Microsoft Windows .......... 35

9 Installing Perl ............................................. 37

10 Installing Java ............................................. 39
# IV  Getting Started

1  Multi-Physics Computation with MpCCI  
1.1  Multi-Physics  
1.2  Solution of Coupled Problems  
1.3  Code Coupling with MpCCI  

2  Setting up a Coupled Simulation  
2.1  A Simple Example  
2.2  Model Preparation  
2.2.1  CFD Model  
2.2.2  FE Model  
2.3  Starting the MpCCI GUI  
2.4  Models Step – Choosing Codes and Model Files  
2.5  Coupling Step – Definition of Coupling Regions and Quantities  
2.5.1  Define Coupling Regions  
2.5.2  Select the Components for each Interconnected Code  
2.5.3  Specify the Quantities which will be Exchanged  
2.6  Edit Step – Further Coupling Options  
2.7  Go Step – Starting Servers and Codes  
2.7.1  Configuring the Initial Exchange Mode  
2.7.2  Setting Option Parameters  
2.7.3  Starting the Coupled Simulation  
2.7.4  Interrupting the Computation  

3  Checking the Results  
3.1  The MpCCI Visualizer  
3.2  Post-Processing  

# V  User Manual

1  Introduction  
1.1  Basic Structure of MpCCI  

2  The MpCCI Software Package  
2.1  Introduction  
2.2  The MpCCI Home Directory  
2.3  Environment and Environment Variables  
2.3.1  MPCCI_ARCH - Architecture Tokens  
2.3.2  MPCCI_DEBUG - for Debugging  
2.3.3  MPCCI_INITIAL_EXCHANGE  
2.4  MpCCI Project and Output Files  

Contents of all MpCCI Manuals
## Contents of all MpCCI Manuals

### Overview

2.4.1 MpCCI Project Files .................................................. 16  
2.4.2 MpCCI Server Input Files .............................................. 16  
2.4.3 Log Files .............................................................. 16  
2.4.4 Tracefile .............................................................. 16  
2.5 The MpCCI Resource Directory ......................................... 17  
2.6 Temporary Files .......................................................... 18  
2.7 Third Party Software Used by MpCCI .................................. 20  
2.7.1 Perl ................................................................. 20  
2.7.2 Java ............................................................... 20  
2.7.3 MPI ............................................................... 20  
2.7.4 Remote Shell and Remote Copy .................................... 20  

### Code Coupling

3.1 Multi-Physics ............................................................ 21  
3.1.1 Physical Domains .................................................... 21  
3.1.2 Coupling Types ...................................................... 22  
3.2 Data Exchange .......................................................... 24  
3.2.1 Pre-Contact Search .................................................. 25  
3.2.2 Minimal Distance .................................................... 26  
3.2.3 Intersection ......................................................... 27  
3.2.4 Orphaned Nodes and Elements ..................................... 29  
3.2.5 Flux and Field Interpolation ....................................... 29  
3.3 Coupling Algorithms .................................................... 30  
3.3.1 Course of the Coupling Process ................................... 30  
3.3.2 Stationary Problems ................................................ 31  
3.3.3 Transient Problems ................................................ 31  
3.3.4 Exchange of Time Step Size ....................................... 36  
3.3.5 Subcycling .......................................................... 37  
3.3.6 Restarting a Coupled Simulation .................................. 38  
3.4 Running MpCCI in a Network .......................................... 39  
3.4.1 Client-Server Structure of MpCCI ................................. 39  
3.4.2 Hostlist files ....................................................... 41  
3.4.3 Remote Shell and Remote Copy ................................... 42  
3.4.4 Coupled Analysis in Batch Mode .................................. 42  
3.5 Mesh Checks ............................................................ 57  
3.5.1 Bounding Box Checks ............................................... 57  
3.5.2 Mesh Quality Checks ................................................. 57  

### Graphical User Interface

4.1 Starting and Exiting MpCCI GUI ....................................... 58  
4.1.1 Starting MpCCI GUI ................................................. 58  
4.1.2 Exiting MpCCI GUI ................................................. 59  
4.2 MpCCI GUI Menus ....................................................... 59
I Overview

4.2.1 File Menu ................................................................. 60
4.2.2 Batch Menu ............................................................... 61
4.2.3 License Menu ............................................................. 62
4.2.4 Tools Menu ............................................................... 63
4.2.5 Preferences Menu ....................................................... 63
4.2.6 Codes Menu .............................................................. 63
4.2.7 Help Menu ............................................................... 63
4.3 Models Step ................................................................. 64
4.3.1 Code Parameters ......................................................... 64
4.3.2 Requirements ............................................................ 64
4.4 Coupling Step ............................................................... 64
4.4.1 Generate Regions ....................................................... 65
4.4.2 Options Part .............................................................. 65
4.4.3 Quantity properties ..................................................... 65
4.4.4 Quantity Sender ........................................................ 66
4.4.5 Predefined Sets ........................................................ 66
4.4.6 Requirements ............................................................ 69
4.5 Edit Step ...................................................................... 71
4.5.1 Control ................................................................... 74
4.5.2 Contact .................................................................. 74
4.6 Go Step .................................................................. 76
4.6.1 Configuring the MpCCI Coupling Server .......................... 76
4.7 Remote File Browser ...................................................... 80
4.7.1 File Browser Handling ............................................... 80
4.7.2 How to mount a new file system ................................... 81

5 Command Line Interface .................................................. 84
5.1 Using the Command Line Interface ................................... 84
5.2 Overview of All Subcommands ........................................ 86
5.3 Starting MpCCI .............................................................. 88
5.3.1 mpcci gui .................................................................. 89
5.3.2 mpcci morpher .......................................................... 90
5.3.3 mpcci observe ........................................................... 93
5.3.4 mpcci pm ................................................................. 94
5.3.5 mpcci vis .................................................................. 95
5.3.6 mpcci xterm ............................................................. 96
5.4 Information and Environment ......................................... 97
5.4.1 mpcci arch ............................................................... 98
5.4.2 mpcci doc ............................................................... 99
5.4.3 mpcci info ............................................................... 100
5.4.4 mpcci env ............................................................... 102
5.4.5 mpcci home ............................................................. 104
5.4.6 mpcci where ............................................................ 105
5.5 Installation and Licensing ........................................ 106
  5.5.1 mpcci license .................................................. 107
  5.5.2 mpcci list ...................................................... 108
  5.5.3 mpcci lmutil ................................................... 109
  5.5.4 mpcci ssh ..................................................... 110
  5.5.5 mpcci test .................................................... 111
  5.5.6 mpcci update ................................................ 114
5.6 Job Control .......................................................... 115
  5.6.1 mpcci backup ................................................ 116
  5.6.2 mpcci batch ................................................... 117
  5.6.3 mpcci batch LSF ............................................. 119
  5.6.4 mpcci batch PBS .............................................. 120
  5.6.5 mpcci batch N1GE .......................................... 121
  5.6.6 mpcci batch LoadLeveler .................................. 122
  5.6.7 mpcci batch GLOBUS ....................................... 123
  5.6.8 mpcci clean ................................................ 124
  5.6.9 mpcci kill .................................................... 125
  5.6.10 mpcci ps .................................................... 127
  5.6.11 mpcci ptoi .................................................. 128
  5.6.12 mpcci server ............................................... 129
  5.6.13 mpcci top ................................................... 135

6   MpCCI Visualizer ...................................................... 136
  6.1 Using the MpCCI Visualizer ................................... 136
  6.1.1 Data Flow ................................................... 136
  6.1.2 Supported Platforms ...................................... 137
  6.1.3 Starting the Visualizer .................................. 137
  6.2 MpCCI Visualizer for .ccv ................................... 137
  6.2.1 Control Window .......................................... 137
  6.2.2 Viewer Window .......................................... 139
  6.3 MpCCI Visualizer for VTFx .................................. 144

7   MpCCI Grid Morpher ................................................ 145

8   MpCCI Project Manager ............................................. 146

VI Codes Manual ....................................................... 1
  1   Overview .......................................................... 9
  1.1 Common MpCCI Subcommands for Simulation Codes .......... 10
  1.2 Unit Systems .................................................. 12
## Overview

Contents of all MpCCI Manuals

### 2 Abaqus

2.1 Quick Information .......................................................... 13
2.1.1 Supported Coupling Schemes ............................................ 13
2.1.2 Supported Platforms and Versions ..................................... 13
2.1.3 References .............................................................. 14
2.1.4 Adapter Description ..................................................... 14
2.2 Coupling Process ........................................................... 15
2.2.1 Model Preparation ....................................................... 15
2.2.2 Models Step ............................................................. 15
2.2.3 Coupling Step ............................................................ 16
2.2.4 Go Step ................................................................. 17
2.2.5 Running the Computation ............................................... 18
2.2.6 Post-Processing ........................................................ 20
2.3 Code-Specific MpCCI Commands ......................................... 21
2.4 Code Environment .......................................................... 22
2.4.1 Prerequisites for a coupled simulation .................................. 22
2.5 Code Adapter Reference ................................................... 23
2.5.1 Patched Input File ....................................................... 23
2.6 Trouble shooting, open issues and known bugs .......................... 24

### 3 ANSYS

3.1 Quick Information ........................................................... 25
3.1.1 Supported Coupling Schemes ............................................ 25
3.1.2 Supported Platforms and Versions ..................................... 25
3.1.3 References .............................................................. 25
3.1.4 Adapter Description ..................................................... 26
3.2 Coupling Process ........................................................... 27
3.2.1 Model Preparation ....................................................... 27
3.2.2 APDL Script ............................................................. 30
3.2.3 Models Step ............................................................. 34
3.2.4 Coupling Step ............................................................ 35
3.2.5 Go Step ................................................................. 37
3.2.6 Running the Computation ............................................... 38
3.3 Code-Specific MpCCI Commands ......................................... 39
3.4 Code Adapter Reference ................................................... 40

### 4 FINE/Hexa

4.1 Quick Information ........................................................... 41
4.1.1 Supported Coupling Schemes ............................................ 41
4.1.2 Supported Platforms and Versions ..................................... 41
4.1.3 References .............................................................. 41
4.1.4 Adapter Description ..................................................... 41
4.2 Coupling Process ........................................................... 42
4.2.1 Model Preparation .................................................. 42
4.2.2 Models Step .......................................................... 44
4.2.3 Coupling Step ........................................................ 44
4.2.4 Go Step ................................................................. 45
4.2.5 Running the Computation ........................................... 45
4.2.6 Post-Processing ....................................................... 46
4.3 Code-Specific MpCCI Commands ...................................... 46
4.4 Code Environment ...................................................... 46
4.4.1 Prerequisites for a coupled simulation ......................... 46
4.5 Code Adapter Reference ................................................ 47

5 FINE/Turbo ................................................................. 48
5.1 Quick Information ....................................................... 48
5.1.1 Supported Coupling Schemes ...................................... 48
5.1.2 Supported Platforms and Versions ............................. 48
5.1.3 References ............................................................ 48
5.1.4 Adapter Description ................................................ 48
5.2 Coupling Process ........................................................ 49
5.2.1 Model Preparation .................................................... 49
5.2.2 Models Step .......................................................... 51
5.2.3 Coupling Step ......................................................... 51
5.2.4 Go Step ............................................................... 52
5.2.5 Running the Computation ......................................... 52
5.2.6 Post-Processing ....................................................... 52
5.3 Code-Specific MpCCI Commands ..................................... 52
5.4 Code Environment ...................................................... 53
5.4.1 Prerequisites for a coupled simulation ......................... 53
5.5 Code Adapter Reference ................................................ 53

6 Flowmaster ................................................................. 54
6.1 Quick Information ....................................................... 54
6.1.1 Supported Coupling Schemes ...................................... 54
6.1.2 Supported Platforms and Versions ............................. 54
6.1.3 References ............................................................ 54
6.1.4 Adapter Description ................................................ 55
6.2 Coupling Process ........................................................ 55
6.2.1 Model Preparation .................................................... 55
6.2.2 Models Step .......................................................... 58
6.2.3 Coupling Step ......................................................... 58
6.2.4 Go Step ............................................................... 59
6.3 Code-Specific MpCCI Commands ..................................... 60
6.4 Code Adapter Description ............................................. 61
# Overview of Contents of all MpCCI Manuals

## FLUENT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Information</td>
<td>62</td>
</tr>
<tr>
<td>Supported Coupling Schemes</td>
<td>62</td>
</tr>
<tr>
<td>Supported Platforms and Versions</td>
<td>62</td>
</tr>
<tr>
<td>References</td>
<td>62</td>
</tr>
<tr>
<td>Adapter Description</td>
<td>63</td>
</tr>
<tr>
<td>Coupling Process</td>
<td>63</td>
</tr>
<tr>
<td>Model Preparation</td>
<td>63</td>
</tr>
<tr>
<td>Models Step</td>
<td>65</td>
</tr>
<tr>
<td>Coupling Step</td>
<td>65</td>
</tr>
<tr>
<td>Go Step</td>
<td>68</td>
</tr>
<tr>
<td>Running the Computation</td>
<td>70</td>
</tr>
<tr>
<td>Code-Specific MpCCI Commands</td>
<td>75</td>
</tr>
<tr>
<td>Code Adapter Reference</td>
<td>76</td>
</tr>
<tr>
<td>The MpCCI UDF Library</td>
<td>76</td>
</tr>
<tr>
<td>UDF-Hooks</td>
<td>77</td>
</tr>
</tbody>
</table>

## FLUX

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Information</td>
<td>80</td>
</tr>
<tr>
<td>Supported Coupling Schemes</td>
<td>80</td>
</tr>
<tr>
<td>Supported Platforms and Versions</td>
<td>80</td>
</tr>
<tr>
<td>References</td>
<td>80</td>
</tr>
<tr>
<td>Adapter Description</td>
<td>80</td>
</tr>
<tr>
<td>Coupling Process</td>
<td>81</td>
</tr>
<tr>
<td>Model Preparation</td>
<td>81</td>
</tr>
<tr>
<td>pyFlux script</td>
<td>82</td>
</tr>
<tr>
<td>Models Step</td>
<td>86</td>
</tr>
<tr>
<td>Coupling Step</td>
<td>86</td>
</tr>
<tr>
<td>Go Step</td>
<td>88</td>
</tr>
<tr>
<td>Code-Specific MpCCI Commands</td>
<td>89</td>
</tr>
<tr>
<td>Code Environment</td>
<td>90</td>
</tr>
<tr>
<td>Prerequisites for a coupled simulation</td>
<td>90</td>
</tr>
<tr>
<td>Code Adapter Description</td>
<td>90</td>
</tr>
</tbody>
</table>

## MSC.Marc

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Information</td>
<td>92</td>
</tr>
<tr>
<td>Supported Coupling Schemes</td>
<td>92</td>
</tr>
<tr>
<td>Supported Platforms and Versions</td>
<td>92</td>
</tr>
<tr>
<td>References</td>
<td>92</td>
</tr>
<tr>
<td>Adapter Description</td>
<td>92</td>
</tr>
<tr>
<td>Coupling Process</td>
<td>93</td>
</tr>
<tr>
<td>Model Preparation</td>
<td>94</td>
</tr>
<tr>
<td>Models Step</td>
<td>94</td>
</tr>
</tbody>
</table>

---

**MpCCI 3.1.1-1**
# Contents of all MpCCI Manuals

## 12 STAR-CD

12.1 Quick Information ......................................................... 117
12.1.1 Supported Coupling Schemes ........................................ 117
12.1.2 Supported Platforms and Versions ................................. 118
12.1.3 References .................................................................. 118
12.1.4 Adapter Description ..................................................... 118
12.2 Coupling Process ............................................................ 118
12.2.1 Model Preparation ....................................................... 118
12.2.2 Models Step ............................................................... 121
12.2.3 Coupling Step ............................................................ 121
12.2.4 Go Step ..................................................................... 123
12.2.5 Running the Computation ............................................. 126
12.2.6 Post-Processing ........................................................... 127
12.3 Code-Specific MpCCI Commands ....................................... 129
12.4 Grid Morphing ................................................................. 130
12.4.1 MpCCI Grid Morpher .................................................... 130
12.4.2 Restart with MpCCI Grid Morpher ................................. 132
12.4.3 pro-STAR Grid Morpher .............................................. 133
12.5 Code Adapter Reference .................................................... 138
12.5.1 STAR-CD 3.26 ............................................................. 138
12.5.2 STAR-CD 4.0x ............................................................. 138
12.5.3 Automatic model preparation for STAR-CD 3.26 and STAR-CD 4.0x .................................................... 139
12.6 Trouble shooting, open issues and known bugs ................. 141

## VII Tutorial

1 Introduction ....................................................................... 6
2 Vortex-Induced Vibration of a Thin-Walled Structure ........... 8
   2.1 Problem Description ....................................................... 8
   2.2 Model Preparation ........................................................ 9
   2.2.1 Fluid Model ............................................................. 9
   2.2.2 Solid Model ............................................................ 13
   2.3 Models Step ................................................................. 14
   2.4 Coupling Step ............................................................. 18
   2.5 Edit Step .................................................................... 19
   2.6 Go Step .................................................................... 20
   2.7 Running the Computation ............................................. 23
   2.8 Discussion of Results ................................................... 25
3 Elastic Flap in a Duct

3.1 Problem Description .................................................. 26
3.2 Model Preparation ..................................................... 26
3.2.1 Solid Model ............................................................ 27
3.2.2 Fluid Model ............................................................ 28
3.3 Models Step ............................................................... 30
3.4 Coupling Step ............................................................ 34
3.5 Edit Step ................................................................. 38
3.6 Go Step ................................................................. 38
3.7 Running the Computation ............................................... 43
3.7.1 Starting the Simulation ............................................. 43
3.7.2 End of the Simulation ............................................... 45
3.8 Discussion of Results .................................................. 45

4 Exhaust Manifold

4.1 Problem Description .................................................. 47
4.2 Model Preparation ..................................................... 48
4.2.1 Solid Model ............................................................ 48
4.2.2 Fluid Model ............................................................ 49
4.2.3 Uncoupled Flow Simulation ...................................... 50
4.2.4 Prepare Models for Coupled Simulation ....................... 51
4.3 Models Step ............................................................... 52
4.4 Coupling Step ............................................................ 53
4.5 Edit Step ................................................................. 54
4.6 Go Step ................................................................. 57
4.7 Running the Computation ............................................... 61
4.8 Post-processing .......................................................... 62

5 Busbar System

5.1 Problem Description .................................................. 64
5.2 Model Preparation ..................................................... 65
5.2.1 Fluid Model ............................................................ 65
5.2.2 Electromagnetic Model .......................................... 66
5.3 Models Step ............................................................... 67
5.4 Coupling Step ............................................................ 70
5.5 Edit Step ................................................................. 74
5.6 Go Step ................................................................. 75
5.7 Running the Computation ............................................... 78
5.8 Discussion of Results .................................................. 79
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Code API</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>Code Integration and Simulation Code Requirements</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Data Exchange and Data Access</td>
<td>8</td>
</tr>
<tr>
<td>2.1.2</td>
<td>MpCCI Interface for Code Integration</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Code Integration with the MpCCI API Kit</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1</td>
<td>A Simple Example</td>
<td>11</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Step-by-Step Procedure for Code Integration</td>
<td>14</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Code Coupling with the Example</td>
<td>25</td>
</tr>
<tr>
<td>2.3</td>
<td>Code Configuration Directory</td>
<td>28</td>
</tr>
<tr>
<td>2.4</td>
<td>MpCCI GUI Configuration File gui.xcf</td>
<td>29</td>
</tr>
<tr>
<td>2.4.1</td>
<td>Code Information: <code>&lt;CodeInfo&gt;</code></td>
<td>29</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Codes Menu: <code>&lt;CodesMenuEntries&gt;</code></td>
<td>29</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Models Step: <code>&lt;ModelsMenuEntries&gt;</code></td>
<td>30</td>
</tr>
<tr>
<td>2.4.4</td>
<td>Component Types: <code>&lt;ComponentTypeDimensions&gt;</code></td>
<td>31</td>
</tr>
<tr>
<td>2.4.5</td>
<td>List of quantities: <code>&lt;SupportedQuantities&gt;</code></td>
<td>32</td>
</tr>
<tr>
<td>2.4.6</td>
<td>Go Step: <code>&lt;GoMenuEntries&gt;</code></td>
<td>33</td>
</tr>
<tr>
<td>2.4.7</td>
<td>Environments for Scanner, Starter, Stopper and Killer</td>
<td>35</td>
</tr>
<tr>
<td>2.4.8</td>
<td>General MpCCI GUI Elements</td>
<td>36</td>
</tr>
<tr>
<td>2.4.9</td>
<td>Testing gui.xcf</td>
<td>42</td>
</tr>
<tr>
<td>2.5</td>
<td>Perl Scripts</td>
<td>43</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Using Information from gui.xcf in Scripts</td>
<td>43</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Scanner.pm</td>
<td>43</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Starter.pm</td>
<td>45</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Stopper.pm</td>
<td>45</td>
</tr>
<tr>
<td>2.5.5</td>
<td>Info.pm</td>
<td>45</td>
</tr>
<tr>
<td>2.5.6</td>
<td>Subcmd.pm</td>
<td>46</td>
</tr>
<tr>
<td>2.5.7</td>
<td>Testing the Perl Scripts</td>
<td>47</td>
</tr>
<tr>
<td>2.6</td>
<td>MpCCI Coupling Manager Functions</td>
<td>48</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Definition of Output Functions: <code>MpCCI_Message_init</code></td>
<td>49</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Initialization: <code>MpCCI_Init</code></td>
<td>50</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Get Initial Exchange Mode: <code>MpCCI_Get_init_actions</code></td>
<td>52</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Data Exchange: <code>MpCCI_Transfer</code></td>
<td>53</td>
</tr>
<tr>
<td>2.6.5</td>
<td>End of Coupled Simulation: <code>MpCCI.Exit</code></td>
<td>54</td>
</tr>
<tr>
<td>2.6.6</td>
<td>Definition of Nodes: <code>MpCCI_Def_nodes</code></td>
<td>55</td>
</tr>
<tr>
<td>2.6.7</td>
<td>Definition of Elements: <code>MpCCI_Def_elems</code></td>
<td>56</td>
</tr>
<tr>
<td>2.7</td>
<td>MpCCI Driver Functions</td>
<td>58</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Description Values</td>
<td>60</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Methods Called before/after some Action</td>
<td>61</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Mesh Definitions</td>
<td>62</td>
</tr>
<tr>
<td>2.7.4</td>
<td>Data Exchange</td>
<td>63</td>
</tr>
<tr>
<td>2.8</td>
<td>Data Structures and Predefined Macros</td>
<td>64</td>
</tr>
<tr>
<td>2.8.1</td>
<td>Coupling Components</td>
<td>64</td>
</tr>
</tbody>
</table>
I Overview

1. Overview
2.8.2 Quantities
2.8.3 Loop Functions

3 MpCCI SDK Code Coupling Library

3.1 The MpCCI SDK Concepts
3.1.1 Communication levels
3.1.2 Coupling quantities
3.1.3 Synchronization concepts and data transfer
3.1.4 Coupling Regions
3.1.5 Neighborhood Search and Interpolation
3.1.6 The MpCCI SDK coupling server scheme

3.2 MpCCI SDK Functions
3.2.1 Naming Conventions and Terminology
3.2.2 MpCCI SDK Data Types
3.2.3 Initialization and Coupling Definition
3.2.4 Coupling Communication
3.2.5 Remeshing
3.2.6 Termination
3.2.7 Control
3.2.8 Miscellaneous functions
3.2.9 Overview of the MpCCI SDK functions

3.3 MpCCI Input File
3.3.1 Structure of the Input File
3.3.2 Code block
3.3.3 Quantities block
3.3.4 Control block
3.3.5 Contact block
3.3.6 Switches block
3.3.7 Jobs block
3.3.8 Parameters block
3.3.9 Coupling block
3.3.10 Additional block
3.3.11 Include Mechanism

3.4 An Example
3.4.1 Start-up and Initialization
3.4.2 Coupling Definition
3.4.3 Coupled Computation
3.4.4 Termination

IX Appendix

Quantity Reference
Contents of all MpCCI Manuals

I Overview

Literature 41
Glossary 42
Keyword Index 46
II Release Notes
## II Release Notes – Contents

1 Changes and New Features in the initial MpCCI 3.1
   1.1 Introduction .......................................................... 4
   1.2 MpCCI 3.1.1-1 Release information ................................. 4
   1.3 MpCCI 3.1.0-3 Patch information .................................... 4
   1.4 MpCCI 3.1.0 Release information .................................... 5
      1.4.1 Installation, Configuration and Licensing ....................... 5
      1.4.2 Coupling Server ................................................ 5
      1.4.3 MpCCI Visualizer .............................................. 6
      1.4.4 MpCCI GUI .................................................... 6
      1.4.5 Command Line Interface ....................................... 6
      1.4.6 Abaqus .......................................................... 6
      1.4.7 Flowmaster .................................................... 6
      1.4.8 FLUENT ........................................................ 7
      1.4.9 ICEPAK ........................................................ 7
      1.4.10 PERMAS ...................................................... 7
      1.4.11 RadTherm ...................................................... 7
      1.4.12 STAR-CD ..................................................... 7
2 Prerequisites for MpCCI Installation .................................. 8
3 Supported Platforms in MpCCI 3.1.0 .................................. 9
   3.1 Platforms Supported by the MpCCI 3.1.0 Server .................... 9
   3.2 Codes Supported by MpCCI 3.1.0 on Different Platforms ............ 11
4 Known Bugs and Limitations in MpCCI 3.1.0 .......................... 16
1 Changes and New Features in the initial MpCCI 3.1

1.1 Introduction

The release notes relate to MpCCI 3.1. The feature improvements and bug fixes from MpCCI 3.0.6 have been applied to MpCCI 3.1. MpCCI 3.1 represents a new release change over MpCCI 3.0.6. The communication protocol between the MpCCI GUI and the remote machine has been improved for a better file browsing. MpCCI 3.1 is available as an upgrade from an existing MpCCI 3.0.6 installation.

1.2 MpCCI 3.1.1-1 Release information

FINE/Hexa and FINE/Turbo
- MpCCI 3.1.1 supports FINE/Hexa 2.10 and FINE/Turbo 8.8.
- Elastic flap and vortex vibration tutorials are included in the distribution.

MpCCI GUI
- Fix for browsing file using user group name with white spaces.

MpCCI Visualizer
- Visualizer displays displacements.
- Fix for Microsoft Windows platform. The distribution contains the missing zip tool.

MpCCI Microsoft Windows installer
- MpCCI Microsoft Windows installer download the current Perl version 5.8.9-827.

1.3 MpCCI 3.1.0-3 Patch information

Abaqus
- MpCCI 3.1.0 supports Abaqus 6.9.
- Time step size default value is set to 1e-5 to avoid zero value (fixes for Abaqus 6.9).

ANSYS
- Version 12 supported.
FLUENT
- Version 12.0.16 supported.
- Extension of the auto boundary conditions setting, needed for preparing a 1D-3D CFD coupling.

MSC.Marc
- Support for 2D axisymmetric model.
- Version 2008r1 supported.

RadTherm
- Add subcycling option.

MpCCI GUI
- Fix issue with DISPLAY connection error during start of code.

1.4 MpCCI 3.1.0 Release information

1.4.1 Installation, Configuration and Licensing

Installation
Users who have yet to upgrade to MpCCI 3.1.0 could install MpCCI over the MpCCI 3.0.6.

Licensing
Users will need an appropriate license file to access MpCCI 3.1.

1.4.2 Coupling Server

Interpolation Schemes
- Interpolation schemes for 1D and 3D coupling are improved.
- A new parameter InsideOnly has been added for the Minimal Distance algorithm. This option is available in the MpCCI GUI at the Edit Step (> V-4.5 Edit Step <). It will disable the shape function extrapolation. Only values inside an element are transferred.
1.4.3 MpCCI Visualizer

Under Microsoft Windows and Linux MpCCI now offers a new viewer for its tracefiles. The new viewer is automatically chosen by `mpcci vis` if available. Because the format it supports is VTFx the tracefile will be converted before calling the viewer. For more information the new viewer provides an online documentation.

The old viewer still remains available and is automatically chosen on the other platforms.

1.4.4 MpCCI GUI

MpCCI GUI is now using the native rsh/ssh client for communicating to the remote machine.

Batch Menu

A new batch menu can be accessed directly from MpCCI GUI for submitting, killing, status a job from a queuing system.

The job submission can be configured from MpCCI GUI and directly sent to the queuing system.

The list of supported batch queuing has been extended to: LSF, PBS, SGE, LoadLeveler.

Remote File Chooser

- The performance of the Remote File Chooser is improved.
- The communication protocol has been extended to rsh.

1.4.5 Command Line Interface

Batch

The job submission to a queuing system is also available from the command line interface (see \(\triangleright\) V-5.6 Job Control \(\triangleright\)).

1.4.6 Abaqus

- MpCCI 3.1.0 supports Abaqus 6.7 and Abaqus 6.8.
- Abaqus 6.8 can run in parallel on different hosts with MpCCI 3.1.0. Abaqus 6.7 notifies in such case that the number of CPUs exceed the number of CPUs available. Running Abaqus 6.7 in parallel is limited to the local machine.

1.4.7 Flowmaster

MpCCI 3.1.0 supports Flowmaster 7.5.x, Flowmaster 7.6.x.
1.4.8 FLUENT

- FLUENT 12.0.7 is now supported in addition to FLUENT 6.3.26.
- When running FLUENT in parallel a host file will be passed to FLUENT.
- FLUENT 6.2.16 has been removed from the distribution. The adapters may be available on demand.

1.4.9 ICEPAK

ICEPAK 4.4.8 is now supported in addition to ICEPAK 4.4.6.

1.4.10 PERMAS

PERMAS 12 is supported.

1.4.11 RadTherm

RadTherm 9.0.1 and RadTherm 9.1.0 are supported. Older RadTherm releases has been removed.

1.4.12 STAR-CD

STAR-CD 4.08.006 is supported in addition to STAR-CD 4.06.007 and STAR-CD 4.04.006.
2 Prerequisites for MpCCI Installation

Please see ⊳ 3 Supported Platforms in MpCCI 3.1.0 ⊲ for platform-specific prerequisites.

Required Disc Space
A full MpCCI installation (all platforms and all code adapters, plus a multi-platform Java JRE and the MpCCI-RSH and the OpenSSH and MPICH for Microsoft Windows) requires a free disc space of approx. 1.8 GB and is a collection of about 9000 files.

Third Party Software

Perl

Java

MpCCI-RSH for Windows is provided by MpCCI 3.1.0. This allows an access to all Windows operating system XP and Vista (See ⊳ III-2.6 MpCCI-RSH for Microsoft Windows ⊲).
## 3 Supported Platforms in MpCCI 3.1.0

Platform lists for supported simulation codes are given in the Codes Manual.

### 3.1 Platforms Supported by the MpCCI 3.1.0 Server

<table>
<thead>
<tr>
<th>Platform</th>
<th>Bits</th>
<th>MpCCI arch.</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP-UX 11.00 on PA-RISC</td>
<td>32/64</td>
<td>hpux11_parisc</td>
<td>OK</td>
</tr>
<tr>
<td>HP-UX 11.22 on Itanium I</td>
<td>32/64</td>
<td>hpux1122_ia64</td>
<td>OK</td>
</tr>
<tr>
<td>HP-UX 11.23 on Itanium II</td>
<td>32/64</td>
<td>hpux1123_ia64</td>
<td>OK</td>
</tr>
<tr>
<td>IBM AIX 5.1 on Power3 processor</td>
<td>32/64</td>
<td>aix51_power</td>
<td>OK use aix51_power</td>
</tr>
<tr>
<td>IBM AIX 5.2 on Power3 processor</td>
<td>32/64</td>
<td>aix52_power</td>
<td>use aix51_power</td>
</tr>
<tr>
<td>IBM AIX 5.3 on Power3 processor</td>
<td>32/64</td>
<td>aix53_power</td>
<td>use aix51_power</td>
</tr>
<tr>
<td>Linux with glibc 2.2 (RedHat 7.*, SuSE &lt;= 8.0)</td>
<td>32</td>
<td>linux_x86_glibc22</td>
<td>OK, no visualizer</td>
</tr>
<tr>
<td>Linux with glibc 2.3 (RedHat 9, SuSE &gt;= 8.1)</td>
<td>32</td>
<td>linux_x86</td>
<td>OK use linux_x86</td>
</tr>
<tr>
<td>Linux with glibc 2.3 on AMD Opteron 64 bit</td>
<td>32/64</td>
<td>linux_amd64</td>
<td>OK use linux_x86</td>
</tr>
<tr>
<td>Linux with glibc 2.3 on AMD64 or EM64T</td>
<td>32/64</td>
<td>linux_x64</td>
<td>OK use linux_x86</td>
</tr>
<tr>
<td>Linux with glibc 2.3 on Intel EM64T</td>
<td>32/64</td>
<td>linux_em64t</td>
<td>OK use linux_x86</td>
</tr>
<tr>
<td>Linux with glibc 2.3 on Itanium</td>
<td>32/64</td>
<td>linux_ia64</td>
<td>OK use linux_x86</td>
</tr>
<tr>
<td>Microsoft Vista 32 bit on Intel x86</td>
<td>32</td>
<td>vista_x86</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Vista 64 bit blend (AMD Opteron or EM64T)</td>
<td>32/64</td>
<td>vista_x64</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Vista 64 bit on AMD Opteron</td>
<td>32/64</td>
<td>vista_amd64</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Vista 64 bit on Intel EM64T</td>
<td>32/64</td>
<td>vista_em64t</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Windows XP on DEC alpha</td>
<td>64</td>
<td>mswin_alpha</td>
<td>not supported use vista_x86</td>
</tr>
<tr>
<td>Microsoft Windows XP on Intel x86</td>
<td>32</td>
<td>mswin_x86</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Windows XP64 blend (AMD Opteron or EM64T)</td>
<td>32/64</td>
<td>xp64_x64</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Windows XP64 on AMD Opteron</td>
<td>32/64</td>
<td>xp64_amd64</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Windows XP64 on Intel EM64T</td>
<td>32/64</td>
<td>xp64_em64t</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>Microsoft Windows XP64 on Intel Itanium</td>
<td>32/64</td>
<td>xp64_ia64</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>OSF1 V5.1 with Alpha processor</td>
<td>64</td>
<td>osf_alpha</td>
<td>OK, no visualizer use vista_x86</td>
</tr>
<tr>
<td>SGI IRIX64 6.5 on R10000</td>
<td>32/64</td>
<td>irix65_mips4</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>SUN Solaris &gt;= 2.7 on AMD64 compatible processor</td>
<td>32/64</td>
<td>solaris_amd64</td>
<td>not supported use vista_x86</td>
</tr>
<tr>
<td>SUN Solaris &gt;= 2.7 on Sparc processor</td>
<td>32/64</td>
<td>solaris_sparc</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>SUN Solaris &gt;= 2.7 on X86 compatible processor</td>
<td>32</td>
<td>solaris_x86</td>
<td>not supported use vista_x86</td>
</tr>
<tr>
<td>SUSE ES 10 with glibc 2.4 on AMD Opteron 64 bit</td>
<td>32/64</td>
<td>sles10_amd64</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>SUSE ES 10 with glibc 2.4 on AMD64 or EM64T</td>
<td>32/64</td>
<td>sles10_x64</td>
<td>OK use vista_x86</td>
</tr>
<tr>
<td>SUSE ES 10 with glibc 2.4 on Intel EM64T</td>
<td>32/64</td>
<td>sles10_em64t</td>
<td>OK use vista_x86</td>
</tr>
</tbody>
</table>
The above list is valid for MpCCI alone, i.e. the MpCCI executables. This does not automatically include all code adapters. Some simulation codes can only be coupled on a subset of the above platforms. A list of platforms for the supported codes is given in the following section and for each code in the corresponding chapter of the Codes Manual.
### 3.2 Codes Supported by MpCCI 3.1.0 on Different Platforms

Codes can only be supported on platforms they support themselves.

<table>
<thead>
<tr>
<th>Platform 1</th>
<th>Platform 2</th>
<th>Codes Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HP-UX 11.00 on PA-RISC</strong> (&lt;code&gt;hpux11.parisc&lt;/code&gt;)</td>
<td></td>
<td>ANSYS 90, 100, 110, 120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Abaqus 6.6, 6.7-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FLUENT 6.3.26, 12.0.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PERMAS 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RadTherm 9.1.2, 9.2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STAR-CD 3.26</td>
</tr>
<tr>
<td><strong>HP-UX 11.22 on Itanium I</strong> (&lt;code&gt;hpux1122.ia64&lt;/code&gt;)</td>
<td></td>
<td>ANSYS 90, 100, 110, 120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Abaqus 6.7-1, 6.7-2, 6.8-1, 6.8-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FLUENT 6.3.26, 12.0.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSC.Marc 2005r3, 2007r1, 2008r1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PERMAS 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STAR-CD 3.26</td>
</tr>
<tr>
<td><strong>HP-UX 11.23 on Itanium II</strong> (&lt;code&gt;hpux1123.ia64&lt;/code&gt;)</td>
<td></td>
<td>ANSYS 90, 100, 110, 120</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Abaqus 6.7-1, 6.7-2, 6.8-1, 6.8-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FLUENT 6.3.26, 12.0.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MSC.Marc 2005r3, 2007r1, 2008r1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PERMAS 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STAR-CD 3.26</td>
</tr>
<tr>
<td><strong>IBM AIX 5.1 on Power3 processor</strong> (&lt;code&gt;aix51.power&lt;/code&gt;)</td>
<td></td>
<td>FLUENT 6.3.26, 12.0.7, 12.0.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICEPAK 6.3.34</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PERMAS 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STAR-CD 3.26, 4.04.006, 4.06.007, 4.08.006</td>
</tr>
<tr>
<td><strong>Linux with glibc 2.2 (RedHat 7.*, SuSE &lt;= 8.0)</strong> (&lt;code&gt;linux_x86.glibc22&lt;/code&gt;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

MpCCI 3.1.1-1 II 11
### Supported Platforms in MpCCI 3.1.0

#### II Release Notes

**Linux with glibc 2.3 (RedHat 9, SuSE >= 8.1) (linux_x86)**

<table>
<thead>
<tr>
<th>Software</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>90, 100, 110, 120</td>
</tr>
<tr>
<td>Abaqus</td>
<td>6.6, 6.7-1, 6.7-2, 6.8-1, 6.8-2, 6.9-1</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>8.8-3</td>
</tr>
<tr>
<td>FLUENT</td>
<td>6.3.26, 12.0.7, 12.0.16</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>6.3.34, 6.3.36</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>2005r3, 2007r1, 2008r1</td>
</tr>
<tr>
<td>PERMAS</td>
<td>11</td>
</tr>
<tr>
<td>RadTherm</td>
<td>9.1.0, 9.1.2, 9.2.0</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>3.26, 4.04.006, 4.06.007, 4.08.006</td>
</tr>
</tbody>
</table>

**Linux with glibc 2.3 on AMD Opteron 64 bit (linux_amd64)**

<table>
<thead>
<tr>
<th>Software</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>90, 100, 110, 120</td>
</tr>
<tr>
<td>Abaqus</td>
<td>6.6</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>210.3</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>8.8-3</td>
</tr>
<tr>
<td>FLUENT</td>
<td>6.3.26, 12.0.7, 12.0.16</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>6.3.34, 6.3.36</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>2005r3, 2007r1, 2008r1</td>
</tr>
<tr>
<td>RadTherm</td>
<td>9.0.1, 9.1.0, 9.1.2, 9.2.0</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>3.26, 4.04.006, 4.06.007, 4.08.006</td>
</tr>
</tbody>
</table>

**Linux with glibc 2.3 on Intel EM64T (linux_em64t)**

<table>
<thead>
<tr>
<th>Software</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>100, 110</td>
</tr>
<tr>
<td>Abaqus</td>
<td>6.6</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>210.3</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>8.8-3</td>
</tr>
<tr>
<td>FLUENT</td>
<td>6.3.26, 12.0.7, 12.0.16</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>6.3.34, 6.3.36</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>2005r3, 2007r1, 2008r1</td>
</tr>
<tr>
<td>PERMAS</td>
<td>11, 12</td>
</tr>
<tr>
<td>RadTherm</td>
<td>9.0.1, 9.1.0, 9.1.2, 9.2.0</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>3.26, 4.04.006, 4.06.007, 4.08.006</td>
</tr>
</tbody>
</table>
Linux with glibc 2.3 on Itanium  

- **ANSYS**: 90, 100, 110, 120
- **Abaqus**: 6.7-1, 6.7-2, 6.8-1, 6.8-2, 6.9-1
- **FINE/Turbo**: 8.8-3
- **FLUENT**: 6.3.26, 12.0.7, 12.0.16
- **MSC.Marc**: 2005r3, 2008r1
- **RadTherm**: 9.1.0, 9.1.2, 9.2.0
- **STAR-CD**: 3.26, 4.04.006, 4.06.007, 4.08.006

Microsoft Vista 32 bit on Intel x86

Microsoft Vista 64 bit blend (AMD Opteron or EM64T)

Microsoft Vista 64 bit on AMD Opteron

Microsoft Vista 64 bit on Intel EM64T
Microsoft Windows XP on Intel x86 (mswin_x86)

ANSYS 90, 100, 110, 120
Abaqus 6.6, 6.7-1, 6.7-2, 6.8-1, 6.8-2, 6.9-1
FINE/Turbo 8.8-3
FLUENT 6.3.26, 12.0.7, 12.0.16
FLUX 10.2
Flowmaster 7.5, 7.6
ICEPAK 6.3.34
MSC.Marc 2005r3, 2007r1, 2008r1
RadTherm 9.1.0, 9.1.2, 9.2.0

Microsoft Windows XP64 blend (AMD Opteron or EM64T) (xp64_x64)

ANSYS 110, 120
Abaqus 6.6, 6.7-1, 6.7-2, 6.8-1, 6.8-2, 6.9-1
FINE/Hexa 210.3
FINE/Turbo 8.8-3
FLUENT 6.3.26, 12.0.7, 12.0.16
FLUX 10.2
ICEPAK 6.3.34
PERMAS 11

Microsoft Windows XP64 on AMD Opteron (xp64_amd64)

ANSYS 110, 120
Abaqus 6.6
FINE/Hexa 210.3
FINE/Turbo 8.8-3
FLUENT 6.3.26, 12.0.7, 12.0.16
ICEPAK 6.3.34

Microsoft Windows XP64 on Intel EM64T (xp64_em64t)

ANSYS 110, 120
Abaqus 6.6
FINE/Hexa 210.3
FINE/Turbo 8.8-3
FLUENT 6.3.26, 12.0.7, 12.0.16
ICEPAK 6.3.34

OSF1 V5.1 with Alpha processor (osf_alpha)

ANSYS 90, 100, 110, 120
MSC.Marc 2005r3, 2007r1
STAR-CD 3.26
SGI IRIX64 6.5 on R10000 (irix65_mips4)

ANSYS 90, 100, 110, 120
Abaqus 6.6
FLUENT 6.3.26
MSC.Marc 2005r3, 2007r1, 2008r1
STAR-CD 3.26

SUN Solaris >= 2.7 on Sparc processor (solaris_sparc)

ANSYS 90, 100, 110
FLUENT 6.3.26, 12.0.7, 12.0.16
ICEPAK 6.3.34
MSC.Marc 2005r3, 2007r1, 2008r1
RadTherm 9.1.0, 9.1.2, 9.2.0
STAR-CD 3.26, 4.04.006, 4.06.007, 4.08.006

SUSE ES 10 with glibc 2.4 on AMD Opteron 64 bit (sles10_amd64)

ANSYS 90, 100, 110, 120
FINE/Hexa 210.3
FINE/Turbo 8.8-3
FLUENT 6.3.26, 12.0.7, 12.0.16
ICEPAK 6.3.34, 6.3.36
MSC.Marc 2005r3, 2007r1, 2008r1
RadTherm 9.0.1, 9.1.0, 9.1.2, 9.2.0
STAR-CD 3.26, 4.04.006, 4.06.007, 4.08.006

SUSE ES 10 with glibc 2.4 on Intel EM64T (sles10_em64t)

ANSYS 100, 110
FINE/Hexa 210.3
FINE/Turbo 8.8-3
FLUENT 6.3.26, 12.0.7, 12.0.16
ICEPAK 6.3.34, 6.3.36
RadTherm 9.0.1, 9.1.0, 9.1.2, 9.2.0
STAR-CD 3.26, 4.04.006, 4.06.007, 4.08.006
4 Known Bugs and Limitations in MpCCI 3.1.0
III Installation Guide
III Installation Guide – Contents

1 Installation Overview ................................................. 5

2 Before the Installation ............................................... 6
   2.1 Downloading MpCCI ............................................. 6
   2.2 Where to Install .................................................. 7
   2.3 The Perl Interpreter ............................................ 8
   2.4 The Java Runtime Environment ............................. 9
   2.5 OpenSSH for Microsoft Windows ......................... 10
   2.6 MpCCI-RSH for Microsoft Windows ....................... 11
   2.7 MPICH for Microsoft Windows .............................. 11

3 Installation of the MpCCI Software ............................. 12
   3.1 Multi-platform for UNIX, Linux and Microsoft Windows .................................................. 12
   3.2 Local Microsoft Windows Installation with the MSI .................................................. 13

4 Immediately After the Installation - Quick Installation Tests without a License .......................... 16
   4.1 Your Home Directory under Microsoft Windows .................................................. 16
   4.2 Testing the MpCCI Working Environment and Perl .................................................. 16
   4.3 Testing whether MpCCI Finds Your Simulation Codes .................................................. 17

5 Licensing ............................................................... 19
   5.1 Request for a License File .................................... 19
   5.2 Installing and Activating a License ....................... 20
      5.2.1 Configure a License Manager as UNIX service .................................................. 21
      5.2.2 Configure a License Manager as Windows service .................................................. 21
   5.3 Defining the License Server .................................... 24
   5.4 Multiple License Servers ....................................... 26
   5.5 Testing the License Server ....................................... 26

6 Configuring the MpCCI Users Environment .................. 27
   6.1 Accessing Remote Hosts ....................................... 27
   6.2 Configuring MpCCI via Environment Variables ........... 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Testing the MpCCI Installation and Communication</td>
<td>30</td>
</tr>
<tr>
<td>8 Troubleshooting</td>
<td>32</td>
</tr>
<tr>
<td>8.1 Secure shell in general</td>
<td>32</td>
</tr>
<tr>
<td>8.2 OpenSSH under Microsoft Windows</td>
<td>32</td>
</tr>
<tr>
<td>8.3 rsh, rcp and rlogin under Microsoft Windows</td>
<td>32</td>
</tr>
<tr>
<td>8.4 MPICH under Microsoft Windows</td>
<td>35</td>
</tr>
<tr>
<td>9 Installing Perl</td>
<td>37</td>
</tr>
<tr>
<td>10 Installing Java</td>
<td>39</td>
</tr>
</tbody>
</table>
1 Installation Overview

⚠️ Please check the Release Notes for installation prerequisites and supported platforms first.

MpCCI installation does not require much, there are just a few steps you need to do. There are two ways to install MpCCI, the multi-platform installation which can be used for all platforms and the Windows installer.

Download Password

Before the installation you should contact the MpCCI team at mpcci@scai.fraunhofer.de to obtain a password for the software download.

Multi-platform Installation

1. **Download tar archive:** Visit the MpCCI website of Fraunhofer SCAI at www.scai.fraunhofer.de/mpcci, to download a compressed tar archive "mpcci-*.*.tar.gz" containing the MpCCI software.

2. **Extract MpCCI software:** Create a directory and unzip/untar the downloaded file within this directory. This new directory is the MpCCI root directory which is referred to as `<MpCCI_home>`.

3. **Add MpCCI to your PATH:** Append the MpCCI binaries directory "<MpCCI_home>/bin" to your PATH environment variable such that the command `mpcci` is available.

A detailed description of this procedure is given in ³ 3.1 Multi-platform for UNIX, Linux and Microsoft Windows ².

Windows Installer

1. **Download Windows installer:** Visit the MpCCI website of Fraunhofer SCAI at www.scai.fraunhofer.de/mpcci and download the Windows installer for your platform.

2. **Run the installer:** Start the installer, which also offers to install third-party software needed by MpCCI. The installer also sets up your environment.

See also ³ 3.2 Local Microsoft Windows Installation with the MSI ².

Licensing

MpCCI uses a FLEXlm license server, the setup is described in ³ 5 Licensing ².
2 Before the Installation

2.1 Downloading MpCCI

⚠️ For the MpCCI download you need a download password!

MpCCI is only available via internet download:

- Go to the MpCCI homepage [www.scai.fraunhofer.de/mpcci](http://www.scai.fraunhofer.de/mpcci).
- Select the point Download at the left.
- Enter username and password and press Login.
- Select the MpCCI version you want to download in the list appearing on the left.
- There are two download variants:
  - The Multi-platform Download which can be used for all platforms and file-server installations.
  - The Windows Installer Download which is only suitable for local installations for Windows XP systems.

Choose the appropriate variant and proceed.

Multi-platform Download

- Select the platforms. More information on the supported platforms is given in the Release Notes.
- Select modules. For coupling with commercial software you need the appropriate modules. Always select MpCCI itself!
- Select whether you want to include the JRE in your download. See 10 Installing Java ◄ for details. You can also select to download OpenSSH or MPICH for Windows XP.
- Press the Start Download button. The webserver now prepares a download file according to your choices, which may take some minutes. After that you can download the zipped download file "mpcci-<MpCCI-version>.tar.gz".
- Proceed with the installation as described in 3.1 Multi-platform for UNIX, Linux and Microsoft Windows ◄.

Windows Installer Download

- Please select your Windows platform and download the suitable installer "mpcci-<MpCCI-version>-<MpCCI_arch>.exe".
2.2 Where to Install

MpCCI is not just a multi-code software but also a multi-platform software. That means all kinds of UNIX, Linux and Microsoft Windows platforms can live together within in a single directory on any machine, whether it is a UNIX, Linux or Microsoft Windows system. The command `mpcci` runs on any platform.

We should mention that Microsoft Windows is used as a synonym for various Windows systems, Windows XP, Windows 64 and Windows Vista.

MpCCI uses architecture tokens to identify a platform. The architecture tokens is herein after referred to as `<MpCCI_arch>`. The architecture tokens are directory names and used to locate the binary executables.

A list of architecture tokens and currently supported platforms is given in the Release Notes.

You may later on rename the MpCCI home directory or move the MpCCI home directory to a different location or even a different file server since MpCCI never works with absolute pathnames nor is any file patched during the installation. Under Microsoft Windows MpCCI never depends on any registry key.

As long as you keep your PATH environment variable up-to-date MpCCI will work by just typing `mpcci`.

System dependent information is collected at runtime and the required system dependent settings are done automatically. There is no need to configure your MpCCI installation for a specific platform or for your local machine. Whatever operating system you are using, MpCCI behaves identical or at least similar.

The major difference between the UNIX world and Microsoft Windows is that under Microsoft Windows the X11 window system is not available per default and you can not redirect the graphical display output from your local machine to a remote UNIX or Microsoft Windows computer.

If you want to use the remote execution facilities of MpCCI, please ensure that the X11 forwarding mechanism works. The server must allow connections from remote computers and no firewall may block the port 6000.

If you have a computer network and an NFS file server or Samba is available in your local area network there is definitely no need to have a separate installation on each local desktop computer, whether it is a UNIX, Linux or Microsoft Windows system. Nevertheless, if you like you may install MpCCI locally on your desktop computer or copy the MpCCI home directory from the file server onto your local machine.

Microsoft Windows only Version and the Microsoft Windows Installer

For standalone Microsoft Windows systems without access to file servers we offer a separate MSI (Microsoft-Soft-Installer) version of MpCCI.

⚠️ The Microsoft Windows MSI versions of MpCCI are identical with the Microsoft Windows parts of the multi-platform version. The difference is the installation procedure itself, not the software.

During the MSI installation some Microsoft Windows registry entries are created which allows to uninstall MpCCI later on using the Microsoft Windows uninstaller. FurthermoreMpCCI is added to the Windows Start-Commands list.

Although you used the MSI version you may move or rename the MpCCI home directory as with the...
multi-platform installation. MpCCI will still work as before. However your Microsoft Windows uninstall information is then corrupted since all your links or your desktop icons are lost and the uninstaller will not be able to remove your MpCCI installation.

⚠️ In fact you do not need to have the MSI version if MpCCI can be installed on a file server. There is no advantage compared to the multi-platform installation. You may simply remove MpCCI from your Microsoft Windows system by deleting the MpCCI home directory.

Recommendations

Each computer in a network used to run MpCCI jobs needs to have access to an MpCCI installation.

We recommend to have a multi-platform file server installation and the MpCCI home directory on the file server is then mounted to your local computer.

Otherwise, in case of a compute cluster with local discs only you would need to copy the MpCCI home directory on each computer separately.

If your HOME directory is shared among the computers in a network you may also have your private MpCCI installation, e.g. under HOME/bin. In this case your HOME directory acts as a file server. This is the preferred method in case you are in trouble with your IT.

⚠️ Avoid a local installation if possible. A file server installation is the preferred way to use MpCCI in a heterogeneous network.

2.3 The Perl Interpreter

Since MpCCI is a multi-platform software you can never fire up any binary executable directly - these executables are wrapped by scripts. To avoid the maintenance of different script languages (Bourne shell "/bin/sh" under UNIX and Linux and ".BAT" files under Microsoft Windows) in fact all MpCCI commands are Perl scripts and MpCCI requires a working Perl installation.

Perl is a platform independent script language which allows to run identical scripts under UNIX as well as Microsoft Windows.

Perl was invented by Larry Wall. Perl is a public domain software distributed under the GNU Public License (GPL) and can be downloaded for free from the world wide web. Perl needs to be installed separately on your local computer.

⚠️ Perl is not part of the MpCCI installation.

If Perl is already installed on your system - this is true for nearly any UNIX and Linux system - you may check the Perl version by typing

```bash
perl -version
```
For MpCCI under UNIX and Linux you need at least Perl 5.6.

ActivePerl for Microsoft Windows

Perl is never part of your Microsoft Windows system. We recommend to have the ActivePerl 5.8.8 or higher installed. There are several issues with Perl versions before ActivePerl 5.8.0.

To be up to date with your ActivePerl you may follow the link

www.activestate.com/Products/ActivePerl.

ActivePerl is the best available Microsoft Windows port of Perl. A 64 bit version of ActivePerl for Windows 64 is available.

If you need to install Perl or upgrade your Perl installation, please see ▶9 Installing Perl ◄.

For MpCCI under Microsoft Windows you need at least Perl 5.8.

2.4 The Java Runtime Environment

The MpCCI GUI is a Java application. At least a JavaTM 2 Runtime Environment (JRE) or higher is required. This corresponds to Java 1.4.2. Anyway, there is no need to have the full Java development environment installed. For the MpCCI GUI the Java JRE is sufficient.

If you already have Java 1.4.2 or higher installed you are fine. Please check your current Java version by typing

```
java -version
```

If the output of the command above contains the string “gcj (GCC)” you are using GNU Java. This may lead to a library problem when the MpCCI GUI is started. In this case, you should use the Java JRE of the MpCCI installation.

A multi-platform Java JRE may be part of the MpCCI installation and of the Microsoft Windows MSI installation. There is no need to have Java installed before you install MpCCI.

If you still prefer to have your private full Java installation, please see ▶10 Installing Java ◄.

Recommendations

In case of a multi-platform file server installation you should always download the Java JRE together with MpCCI. You do not know in advance whether an MpCCI user has Java installed on his local machine or not, specially in a heterogeneous compute environment. If not MpCCI has the fallback possibility to use the Java JRE that comes with MpCCI.
However, you should not install Java on your local machine just because of \texttt{MpCCI}. Instead you should use the Java as part of the \texttt{MpCCI} installation.

### 2.5 OpenSSH for Microsoft Windows

OpenSSH is a free ssh distribution. OpenSSH is required under \texttt{Microsoft Windows} to execute commands on remote \texttt{Microsoft Windows} or UNIX systems from a \texttt{Microsoft Windows} or a UNIX system. OpenSSH makes \texttt{Microsoft Windows} interoperable with other \texttt{Microsoft Windows} systems and UNIX.

OpenSSH is part of the \texttt{MpCCI} installation and may be finally installed on the fly. You should not download OpenSSH for Microsoft Windows from the web and install it.

⚠️ In fact you should not have OpenSSH installed before the \texttt{MpCCI} installation.

#### Recommendations

Do never install OpenSSH separately. Please let \texttt{MpCCI} do this job.

⚠️ Together with \texttt{MpCCI} some bug fixes for the OpenSSH are installed and the OpenSSH is automatically configured.

#### OpenSSH for Microsoft Windows is already installed

You are familiar with OpenSSH and already worked with OpenSSH.

⚠️ Please save your existing "\texttt{<Home>/.ssh}" directory - e.g. rename it ’.ssh’ - to avoid the lost of your already generated and working RSA-keys.

Uninstall OpenSSH. After \texttt{MpCCI} was successfully installed OpenSSH will be installed on the fly.

After the final OpenSSH installation remove the directory "\texttt{<Home>/.ssh}".

⚠️ Do not forget to rename your saved ’.ssh’ directory back to ’.ssh’.

#### OpenSSH for Microsoft Windows and cygwin

There are known issues if OpenSSH and cygwin are installed in parallel on the same machine, e.g. both come with an "\texttt{ls}" command accessing different and possibly incompatible versions of the same ".\texttt{dll}" files. In this case the OpenSSH may fail.

Best is to have either the OpenSSH installed or your existing cygwin installation includes a working ssh service. In the latter case the OpenSSH installation is not required.

If you need the OpenSSH and the cygwin installation in parallel then please make sure that the OpenSSH binaries directory is in front of your \texttt{PATH} environment variable.
2.6 MpCCI-RSH for Microsoft Windows

MpCCI comes with its own rshd and rlogind services for Windows. This package includes both, the services and the rsh and rcp commands. Although this software is part of MpCCI, is not necessarily bound to MpCCI. Therefore the service is - like the OpenSSH- a separate installation under Microsoft Windows.

MpCCI-RSH is required under Microsoft Windows to execute commands on remote Microsoft Windows or UNIX systems from a Microsoft Windows or a UNIX system. MpCCI-RSH makes Microsoft Windows interoperable with other Microsoft Windows systems and UNIX.

Using the MpCCI Microsoft Windows installer, the MpCCI-RSH package is automatically installed and started.

Otherwise if you have a computer network and an NFS file server or Samba is available in your local area network and need to install the MpCCI-RSH package, you can execute the installation program "<MpCCI_home> \mswin\mpccirsh\setup.exe".

After the installation of MpCCI-RSH you need to prepare the .rhosts file and rsh environment, see ▶8.3 Preparing the .rhosts file and the rsh environment ◄.

2.7 MPICH for Microsoft Windows

MPICH for Microsoft Windows is required to execute the MpCCI server process under Microsoft Windows.

MPICH is part of the MpCCI installation and may be installed on the fly. You do not need to download MPICH for Microsoft Windows and install it.

There will be no conflicts between the MPICH which comes with MpCCI and any existing MPICH. You may also have MPICH 1 and MPICH 2 installed in parallel without any conflicts.

Recommendations

Do never install MPICH separately. Please let MpCCI do this job.
3 Installation of the MpCCI Software

If you intend to install MpCCI under UNIX or Linux or on a file server you should have downloaded a multi-platform distribution file. For a local Microsoft Windows installation you download an "MSI" file.

3.1 Multi-platform for UNIX, Linux and Microsoft Windows

For all supported platforms you will find a distribution on the MpCCI download site. No matter on which systems you are logged on, UNIX, Linux or Microsoft Windows, the installation procedure is identical.

You need not to be an Microsoft Windows administrator or the UNIX root user if you do not extract the MpCCI distribution file within protected directories. MpCCI can be installed everywhere, e.g. in your private MpCCI home directory, "$HOME/MpCCI" or "%USERPROFILE%\MpCCI".

After downloading your MpCCI distribution file you have to

1. create your MpCCI home directory

2. change to your MpCCI home directory: `cd <MpCCI_home>`

3. extract the MpCCI files from the downloaded file with the command

   ```
   tar zxvf mpcci-<MpCCI-version>.tar.gz
   ```

   or, if the `z` option is not available, with the commands

   ```
   gunzip mpcci-<MpCCI-version>.tar.gz
   tar xvf mpcci-<MpCCI-version>.tar
   ```

   The `tar` command may not - we believe never - be available under Microsoft Windows.

The MpCCI software is now installed on your system.

Do not Forget to Set Your `PATH` Environment!

To use MpCCI without having trouble the MpCCI binaries directory "<MpCCI_home>/bin" needs to be in the PATH. If the MpCCI binaries directory is not in the PATH a remote connection to this machine from a remote host via any MpCCI software is impossible.

If you already have a previous release of MpCCI in the PATH, please append the newest release in front of the PATH and not at the end. Otherwise the `mpcci` command selected will always start the previous release.

According to your login shell under UNIX and Linux (see "/etc/passwd") you need to set:

Bourne shell ("/bin/sh") users
PATH=<MPCCI_HOME>/bin:$PATH
export PATH

Korn shell (ksh) and bash users

export PATH=<MPCCI_HOME>/bin:$PATH

c-shell (csh) and tc-shell (tcsh) users

setenv PATH <MPCCI_HOME>/bin:$PATH

and under Microsoft Windows

set PATH=<MPCCI_HOME>in;%PATH%

⚠️ Please keep in mind that a list separator is ‘:’ under UNIX and ‘;’ under Microsoft Windows.

### 3.2 Local Microsoft Windows Installation with the MSI

Log on as an administrator or make sure that you have administrative rights. You need administrative rights because of the required modification in the Microsoft Windows registry.

Please execute the downloaded self-extracting archive, e.g. "mpcci-<MpCCI-version>-mswin_x86.exe" for Windows XP 32-bit on Intel x86 processor.

This installation procedure builds a proper MpCCI software environment under Microsoft Windows.
You pick up those components, e.g. code adapters, you want to install from the components list.

You can provide the license server information, i.e. hostname and port number of your license server.
During the installation the prerequisites are tested and you have the chance to install the missing third party software (Java JRE, MPICH, OpenSSH, MpCCI-RSH, Perl) on the fly.

The MpCCI software is now installed on your system.

There is no need for any further configuration. The environment is automatically updated by the installer. If the installer has not been able to set the path to MpCCI automatically, please set it manually as described in the previous section.
4 Immediately After the Installation - Quick Installation Tests without a License

4.1 Your Home Directory under Microsoft Windows

Under UNIX your home directory is known to applications via the environment variable HOME, which is always defined.

Under Microsoft Windows the environment variable HOME may be defined, however the standard under Microsoft Windows is to use the variable USERPROFILE.

To make Microsoft Windows behave like a UNIX and vice versa MpCCI ignores any existing HOME variable and automatically (re)defines the variable HOME identical to either

- \texttt{HOME=%USERPROFILE%}
- \texttt{HOME=%HOMEDRIVE\%HOMEPATH%}
- \texttt{HOME=C:\}

At least HOMEDRIVE and HOMEPATH should always be defined on Microsoft Windows.

Your home directory is herein after referred to as \texttt{<Home>}. 

- Under Microsoft Windows MpCCI strictly ignores any environment variable HOME. MpCCI assigns the variable HOME.
- You need to have read-write-execute access rights for your \texttt{<Home>} directory.

4.2 Testing the MpCCI Working Environment and Perl

The first command you should execute is

\texttt{mpcci env}

If under Microsoft Windows the OpenSSH and MPICH are not installed, MpCCI will then do the post-installations. Please follow the installation instructions.

- If you are working under Microsoft Windows, you need to have administrative rights.

\texttt{mpcci env} should not fail and the output then shows the system information collected by MpCCI at runtime.

- If you are running under Microsoft Windows and a virus scanner is installed, the \texttt{mpcci env} may be quite slow at the first execution. During the collection of system information a lot of files are visited and executed and the virus scanner has to updated it’s cache for the first time. Once executed there will be no major delay any longer.
Testing whether Perl can really compile all MpCCI modules

Type in the command

```
mpcci test -modload
```

This test should not fail. The last output line you should see is:

```
Successfully loaded and compiled all Perl modules from the above list.
```

Testing the Perl/Tk installation

Launch the MpCCI project manager with the command

```
mpcci pm
```

This test may fail if the Perl/Tk toolkit is not installed. If you would like to use the MpCCI project manager GUI you need to install the Perl/Tk toolkit. Under Microsoft Windows Perl/Tk is part of the ActivePerl distribution.

Other Possible Tests

You may start further tests which do not require an MpCCI license. The command

```
mpcci test
```

will help you to test the MpCCI access and communication with remote systems. For help please just type `mpcci help test`.

4.3 Testing whether MpCCI Finds Your Simulation Codes

Depending on the code adapters you installed together with MpCCI you may now test whether MpCCI is able to find your code installations.

Please type just

```
mpcci
```
and check what codes are listed.

You may see the output

<table>
<thead>
<tr>
<th>Subcommands:</th>
<th>Tools related to code.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abaqus</td>
<td>Abaqus.</td>
</tr>
<tr>
<td>ANSYS</td>
<td>ANSYS.</td>
</tr>
<tr>
<td>FLUENT</td>
<td>FLUENT.</td>
</tr>
<tr>
<td>FLUX</td>
<td>FLUX.</td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Flowmaster.</td>
</tr>
<tr>
<td>IcePak</td>
<td>IcePak.</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>MSC.Marc.</td>
</tr>
<tr>
<td>PERMAS</td>
<td>PERMAS.</td>
</tr>
<tr>
<td>StarCD</td>
<td>StarCD.</td>
</tr>
<tr>
<td>codename</td>
<td>codename.</td>
</tr>
</tbody>
</table>

For each code listed you may type \texttt{mpcci <code name>} to get more help on the code specific commands and options.

To list the releases of a code that MpCCI needs to find on your system please type \texttt{mpcci <code name> -releases}, or in abbreviated form \texttt{mpcci <code name> rel}.

Under UNIX and Linux the executables for each code must be in the \texttt{PATH}. Under Microsoft Windows MpCCI reads the Microsoft Windows registry entry for a code. You need to have an ordinary installation of this code.

More detailed information may be listed using the command \texttt{mpcci <code name> -information}, or in abbreviated form \texttt{mpcci <code name> info}.

**Non standard code installation**

If you are a developer of a simulation code and you would like to test your development version of the code - we assume this is not a standard installation and confuses the part of MpCCI which collects information for your code - you may define an environment variable that lists the names of the executable files.

\texttt{MPCCI_CODEWORD_EXENAMES=executable1:executable2:...}

Currently this feature is implemented for Abaqus and MSC.Marc.
5 Licensing

After the installation of MpCCI you need to acquire a license file from Fraunhofer SCAI. MpCCI uses the FLEXlm based floating license mechanism. The FLEXlm license server has to be started on the license server host for which you require a license file.

5.1 Request for a License File

We need two pieces of information from you to generate a license file:

1. information about your license server host,
2. information about your desired MpCCI license features.

First decide on which host you would like to run the license server. This could be any host in a network or in case of a local installation it is your local machine. On this computer please type in the command `mpcci license -sysid` or abbreviated `mpcci lic sys`. You should see an output line like

```
SERVER hostname 00087519576d 47000
```

A FLEXlm license is bound to the MAC address of your network device. If you have multiple network devices installed (Ethernet card, Wireless LAN, Docking Station on a Notebook) you may see multiple hostids. For the MpCCI license server daemon the ID of the permanent integrated ethernet card address is the correct one.

Secondly, we need information about your desired MpCCI license features. MpCCI knows two basic FLEXlm features,

```
mpcci_sessions
mpcci_procs
```

which defines how many independent coupled sessions (mpcci_sessions) can be run simultaneously and how many MpCCI coupling processes are allowed in the sum of all simultaneous MpCCI coupling sessions (mpcci_procs).

The feature mpcci_sessions will be checked out and decremented by 1 during each single MpCCI session (each click on the Start button of the server in the Go panel of the MpCCI GUI or each call of mpcci server if you start the application manually).

The other feature mpcci_procs will be decremented by the number of MpCCI coupling processes (excluding the MpCCI control process) of the session. The minimum is 1 for mpcci_sessions and 2 for mpcci_procs, i.e. you can start MpCCI with e.g. one Abaqus process and one FLUENT process.
With 1 mpcci_sessions license you cannot start another coupling job as long as your MpCCI job is running. If you’d like to run two coupled FSI simulations simultaneously you will need a license with at least 2 mpcci_sessions and 4 mpcci_procs. If you have several FLUENT processes you will need more mpcci_procs accordingly.

In addition, there are MpCCI license features for the supported code adapters: if you couple code mycode using the MpCCI adapter for this code, MpCCI will look for a feature mpcci_adapter_myc ode on the license server. This feature must be contained in the license file, but it won’t be checked out or decremented.

Examples:

```plaintext
mpcci_adapter_abaqus
mpcci_adapter_ansys
mpcci_adapter_fluent
mpcci_adapter_starcd
```

The MpCCI morpher is also an extra licensed product and you need a license feature

```plaintext
mpcci_morpher
```

Please use the License Request Form Sheet on the SCAI download area (www.scai.fraunhofer.de/mpcci, go to Download, enter your username and password, then go to License Request).

Please fill out the form with all required data and submit. You will receive the required license file soon after via email from Fraunhofer SCAI.

### 5.2 Installing and Activating a License

Please copy your received license file into the file "<MpCCI_home>/license/mpcci.lic".

Then please start the FLEXlm license server, i.e. the server daemon and the FHGSCAI vendor daemon on the license server machine with the command

```plaintext
mpcci license -start or abbreviated: mpcci lic start
```

After the daemons were successfully started you can list the available licenses with the command

```plaintext
mpcci license -local or abbreviated: mpcci lic loc
```

If the license server is running with a valid license you will see an output e.g. like this:

```
License server: 47000@aquila
Vendor daemon : FHGSCAI v10.8, status "UP"
```
### 5.2.1 Configure a License Manager as UNIX service

On UNIX, edit the appropriate boot script, which may be "/etc/rc.boot", "/etc/rc.local", "/etc/rc2.d/Sxxx", "/sbin/rc2.d/Sxxxx", etc. Include commands similar to the following.

```shell
<MpCCI_HOME>/bin/mpcci license -start
```

The `<MpCCI_home>` is the full path of the MpCCI installation. This command will create a log file under the directory "<MpCCI_home>/license" and you have to ensure that the process could write in the directory.

### 5.2.2 Configure a License Manager as Windows service

Execute the "lmtools.exe" application from the MpCCI installation directory:
"<MpCCI_home>/license/<MpCCI_arch>/lmtools.exe"

`<MpCCI_arch>` corresponds to the output of the command:

```
mpcci arch
```
- Select in the Service/License File tab section the option Configuration using Services.

- Click the Config Services tab section.
• Enter a service name e.g. MpCCI license manager or MpCCI FLEXlm.

• Select the path of the program "lmgrd.exe" with the **Browse** button:
  
  "<MpCCI_home>/license/<MpCCI_arch>/lmgrd.exe"

• Select the license file "mpcci.lic" with the **Browse** button:
  
  "<MpCCI_home>/license/mpcci.lic"

• Activate the **Start Server at Power Up** option.

• Activate the **Use Services** option.

• You can optionally add a log file by providing a file name for the **Path to the debug log file** option.

• Click on the **Save Service** button.
Select in the Start/Stop/Reread tab section the license service.

Click on the Start Server button.

The license server is now running and configured to start at power up.

5.3 Defining the License Server

We assume that an MpCCI FLEXlm license server is already running.

Before you start your MpCCI session you have to know the hostname of the license server and the port where it is listening. The default port number is 47000, which can be changed by the administrator of the license server.

The location of the license server is defined via the variable MPCCI_LICENSE_FILE.

```
MPCCI_LICENSE_FILE=port@licenseserver
```

before starting any MpCCI session.

MpCCI has built in two mechanisms to check for the MPCCI_LICENSE_FILE variable:

At first it checks a FLEXlm resource file located in your <Home> directory: under UNIX and Linux this is "HOME/.flexlmrc" and under Microsoft Windows it is "%USERPROFILE%\flexlmrc". MpCCI scans the "flexlmrc" for a line containing

```
MPCCI_LICENSE_FILE=...
```
Secondly it reads the environment variable `MPCCI_LICENSE_FILE`. Supposed your license server name is `aquila` please set `MPCCI_LICENSE_FILE` as follows:

Bourne shell (`"/bin/sh"`) users

```
MPCCI_LICENSE_FILE=47000@aquila
export MPCCI_LICENSE_FILE
```

Korn shell (`ksh`) and `bash` users

```
export MPCCI_LICENSE_FILE=47000@aquila
```

c-shell (`csh`) and tc-shell (`tcsh`) users

```
setenv MPCCI_LICENSE_FILE 47000@aquila
```

and under Microsoft Windows

```
set MPCCI_LICENSE_FILE=47000@aquila
```

The values of both variables from the `".flexlmrc"` file and the environment variable `MPCCI_LICENSE_FILE` are merged.

**Recommendations**

We recommend to set the `MPCCI_LICENSE_FILE` environment variable in your resource file `".flexlmrc"` for FLEXlm. Environment variables are volatile, the `".flexlmrc"` file is not. Environment variables are sometimes not properly defined if a remote `MpCCI` command is started via `rsh` or `ssh`, the `".flexlmrc"` file is always accessible.

If you still prefer to use the environment variable `MPCCI_LICENSE_FILE` instead of the `".flexlmrc"` file:

- Under **UNIX** or Linux please store the setting in one of your login scripts (`".login", "profile", "cshrc", "bashrc", etc.).

- Under **Microsoft Windows** please make sure that `MPCCI_LICENSE_FILE` is a global environment variable for all users.

⚠️ **Under Microsoft Windows** you need to be the administrator to define global system wide variables.
5.4 Multiple License Servers

If you use several redundant FLEXlm license servers you will have several entries of the form `port@host` - one list entry for each license server.

⚠️ In this case the entries must be separated by a ";" under UNIX and Linux respectively by a ";" on Microsoft Windows. This is an example for Microsoft Windows:

```
set MPCCI_LICENSE_FILE=47000@aquila;47000@babbage;47000@zuse;...
```

5.5 Testing the License Server

After setting your `MPCCI_LICENSE_FILE` environment you should check whether licensing works.

```
mpcci license -servers
```

should list the value of `MPCCI_LICENSE_FILE`.

The command

```
mpcci license -mpcci
```

checks if the license server is running and can be reached from your local machine.
6 Configuring the MpCCI Users Environment

6.1 Accessing Remote Hosts

If all processes of a MpCCI job run on your local machine you may skip the following section.

General information on running MpCCI in a network is given in V-3.4 Running MpCCI in a Network.

If you intend to distribute the MpCCI job on multiple hosts in a network - the preferred solution - MpCCI needs to execute commands on remote hosts and may copy files to remote hosts. In this case MpCCI uses either the rsh commands (rsh, rcp) or the ssh commands (ssh, scp).

rsh and ssh must be properly configured to avoid any password request when executing commands on remote hosts.

Environment Variable MPCCI_RSHTYPE: Tell MpCCI to use either rsh or ssh

To find out whether to use the rsh command set (rsh, remsh etc., and rcp) or the secure shell commands (ssh, scp) MpCCI inspects the optional environment variable MPCCI_RSHTYPE. A non-empty value of MPCCI_RSHTYPE may either be “rsh” or “ssh”. Any other value is invalid and will result in an error.

If MPCCI_RSHTYPE is not defined or empty rsh is used as the default.

rsh Configuration under UNIX and Linux

You should have an "rhosts" file - used by rsh and ssh - and additionally may have an "hosts" file - used by ssh only - located in your <Home> directory. Your "<Home>/.rhosts" just lists - one hostname per line - all remote hosts from which you may log on the local host without a password request. You may test a working rsh environment via

```
rsh hostname ls
```

There should be no password request.

Please make sure that "<Home>/.rhosts" can be accessed from all hosts relevant for MpCCI.

ssh Configuration

If you prefer to use the ssh commands you need to generate your private RSA-keys to avoid a password request. Please use the command

```
ssh-keygen
```
Please read the OpenSSH documentation on how to distribute private and public keys to all hosts in a network.

You may test a working ssh environment via

```
ssh hostname ls
```

There should be no password request.

You may further run some tests on your ssh configuration using

```
mpcci ssh [options]
```

For help please enter

```
mpcci help ssh
```

**Testing MpCCI Access to Remote Hosts via rsh and ssh**

Even under Microsoft Windows you should create a "<Home>/.rhosts" file. You may then test the remote access to all host listed with

```
mpcci test .rhosts
```

The test tries to connect to all hosts, via rsh and/or ssh, prints a protocol in case of failures and finally writes a new “hostlist” file containing all successfully tested hosts.

To get a full help on mpcci test please enter

```
mpcci help test
```

For each remote host/hostfile listed on the command line it performs the following tasks:

- Test whether the remote hostname can be resolved by the Domain Name Service (DNS).
- Test whether the remote host is alive and reachable and ping gets a reply.
- Test possibility of rsh/ssh connections to the remote host.
- Brief test on the MpCCI installation on the remote host from the local host.
- Test server-client connection on ports 47001 and so on.
- Creation of a protocol host-file "mpcci.test.hostlist" which can be used as an MpCCI hostfile.
6.2 Configuring MpCCI via Environment Variables

The behavior of MpCCI can be influenced through environment variables. If MpCCI does not run properly on your system you may be able to fix problems by setting some variables as described in the following. A complete overview of all environment variables used by MpCCI is given in ▷ V-2.3 Environment and Environment Variables ◁.

**Environment Variable MPCCI_ARCH**

The variable MPCCI_ARCH holds the architecture token of MpCCI and is usually set by MpCCI automatically since MpCCI is capable to figure out the platform it is running on. Only if this mechanism fails or you definitely want to try running a different version of MpCCI, you should set this variable.

MPCCI_ARCH must always contain a valid architecture token as listed in the Release Notes.

Please read ▷ V-2.3.1 MPCCI_ARCH - Architecture Tokens ◁ for more information.

**Environment Variable MPCCI_DEBUG**

If MPCCI_DEBUG is set and not false (any value except empty or 0), then detailed logging output is produced to find out some pitfalls in case of failures. This has the same effect as startint MpCCI with the -debug option.
7 Testing the MpCCI Installation and Communication

A Simple Test on the Local Machine, "Hello World"

We assume that the above installation step was successful, that you got an MpCCI license file and already started the license server.

The MpCCI distribution contains a “hello world” like example which you should run in order to test the installation. Please run the example via

```
mpcci test -simple
```

First the MpCCI control process and the MpCCI servers are started. Then - after some delay - code 1 is started and connects itself to the MpCCI server 1. Finally after some delay code 2 is started and connects itself to the other MpCCI server.

You should see five windows: theMpCCI control window, two MpCCI server and two “hello world” application windows. On Microsoft Windows you will get only three windows because the output of MpCCI control and servers is mixed in a single window.

The resulting output of MpCCI-Control ends as follows:

```
[...]
CONTROL: MON message arrived ...
CONTROL: # double:1:1: terminate
CONTROL: MON message arrived ...
CONTROL: # single:2:2: terminate
CONTROL: # Received terminate messages from all processes.
CONTROL: # Quiting the monitoring loop, and calling CCL_Finalize.
CONTROL: Leaving monitor without errors.
CONTROL: ===============================================================
CONTROL: CCI_Finalize ... okay.
CONTROL: I am calling MPI_Finalize ...
```

The resulting output of MpCCI-Server-001 ends as follows:

```
[...]
double:1:0: ===============================================================
double:1:0: Entered CCI_Finalize.
double:1:0: CCI_Finalize ... okay.
double:1:0: ===============================================================
double:1:0: I am calling MPI_Finalize ...
```
The resulting output of code 1 ends as follows:

```plaintext
[...]  
Contacting MpCCI root server "47111@benz.scai.fhg.de" ...  
Contacting associated MpCCI server "47112@benz" ...  
Test code DOUBLE finished with success.
```

If the output looks like the above the test was successful.

On some platforms the control output ends with the additional line

```plaintext
Control process terminating normally.
```
8 Troubleshooting

8.1 Secure shell in general

Avoid any printout in your login script like ".login", ".profile", ".cshrc", ".bashrc", etc. This output may confuse the secure shell commands \texttt{scp} and \texttt{sftp} and may lead to strange and confusing error messages.

8.2 OpenSSH under Microsoft Windows

Together with OpenSSH an OpenSSH service daemon is installed. The name of the service is \texttt{opensshd}. If you can not login from a remote host onto the local Microsoft Windows PC you may have to restart the service daemon.

For a restart of the \texttt{opensshd} service please enter

\begin{verbatim}
net stop opensshd : Stop the service
net start opensshd : Start the service
\end{verbatim}

in a command shell.

⚠️ You need to have administrative rights to restart a service under Microsoft Windows.

8.3 \texttt{rsh}, \texttt{rcp} and \texttt{rlogin} under Microsoft Windows

By default Microsoft Windows does not have the UNIX \texttt{rshd} or \texttt{rlogind} services installed. There are additional packages available from Microsoft which include the \texttt{rshd} service. This is the so called Software for Unix Application package, SFU3.5 for Windows XP and the SUA package for Windows Vista (Windows Vista Enterprise only). If you have this package installed and the \texttt{rshd} service is properly installed and running you may skip the following section.

MpCCI comes with its own \texttt{rshd} and \texttt{rlogind} services for Windows. This package includes both, the services and the \texttt{rsh} and \texttt{rcp} commands. Although this software is part of MpCCI, is not necessarily bound to MpCCI. Therefore the service is - like the OpenSSH- a separate installation under Microsoft Windows.

After a successful installation of the MpCCI-RSH program (see \texttt{\textcopyright 2.6 MpCCI-RSH for Microsoft Windows \textcopyright}) under e.g. "C:\Program Files\MpCCI-RSH" you need to
• Install and start the rshd and optional rlogind services (▷8.3 Installing the MpCCI rshd service processes◁)
• Prepare you personal remote hosts file and rsh environment (▷8.3 Preparing the .rhosts file and the rsh environment◁)

Installing the MpCCI rshd service processes

If not already done during the installation of the MpCCI-RSH package you need to install the services in Microsoft Windows.

Change into the installation directory of the MpCCI rsh installation, e.g. "C:\Program Files\MpCCI-RSH\bin". Then type

```
rshd -install
rlogind -install
```

Now both services should be installed and started. The services are automatically restarted after a reboot of your computer.

You may stop and remove the services later on by either using the Microsoft Windows services panel or you simply type

```
rshd -remove
rlogind -remove
```

You can start/stop the services by either using the Microsoft Windows services panel or you simply type

```
rshd -stop or net stop mpcci_rshd
rshd -start or net start mpcci_rshd
rlogind -stop or net stop mpcci_rlogind
rlogind -start or net start mpcci_rlogind
```

⚠️ You need to have administrative rights to start or stop a service under Microsoft Windows.

Preparing the .rhosts file and the rsh environment

To give a remote user access to your local computer at first you need to create a "%USERPROFILE%\ .rhosts" file in which the trusted remote hosts and users are listed. This file may be a copy of your UNIX ".rhosts" file.
This file must be located in your "\%USERPROFILE\%" directory.

If you are a Windows domain account user please make sure that your "\%USERPROFILE\%" directory is not removed after you log off the Windows computer. Otherwise the rshd service is not capable to scan the required files before it creates a logon session for you.

The contents of the "\rhosts" file is a list of hostnames and user login names, one host per line.

```
host1 user1 user2 user3 ...
# additional users for host1
host1 myname friend
host2 myself
```

Empty lines are ignored.

Unlike with UNIX rshd at least one user login name per host is required. All host names must be fully qualified names (hostname.netname). IP4 names in dot notation are ignored.

For security reasons - e.g. a trojan horse may append new hosts and users to your "\rhosts" file - the rshd and rlogind services do never scan the "\rhosts" file directly but instead use an alternative keys file "\%USERPROFILE\%\rhosts.keys".

This file is created - with special access right assigned - from the "\rhosts" file by the command

```
    rsh -makekeys
    or
    rsh -k
```

The -k option will not check the hostname from the "\rhosts". If you have some trouble to connect to your local machine because of a DNS name resolution problem this option is recommended to be used.

If you add a new host in the "\rhosts" file you should create the "rhosts.keys" again.

Any modification of the "rhosts.keys" invalidates this file for use with rshd.

Do not modify the access rights of "\%USERPROFILE\%\rhosts.keys"

You should now be able to execute a remote command, e.g.

```
    rsh hostname dir  
    rsh hostname mpcci env
```

You may get further help for the rsh command with
Remote shell and the Microsoft Windows firewall

The rshd and rlogind services accept incoming connection requests on the TCP ports 513 and 514. The rsh command may ask the rshd to open an additional TCP socket on any unused port in the range 1023, 1022, ...514 and the rshd may then connect to the rsh command on any of these port.

During the installation the rshd, rlogind, rsh and rcp programs have been added to the Microsoft Windows firewall rules to accept incoming and outgoing connection. The listed ports above must not be blocked by the Windows firewall since the applications themselves have been added to the firewall rules.

8.4 MPICH under Microsoft Windows

MPICH starts processes on remote Microsoft Windows systems. Therefore you need to create an MPICH account with a valid Microsoft Windows account name and a valid login password. The MPICH account may/should be your personal account. However you could use an alternative account. You may e.g. create a new user named mpcci. However, it must be a valid account and you need to know the valid password for this account.

⚠️ Beware: Under Microsoft Windows an account name may differ from the login name!

Under Microsoft Windows your personal account name - not your login name - is stored in the environment variable \USERNAME. If your account is a domain account you need to specify the domain name plus the account name in the format userdomain\account. Under Microsoft Windows the name of your user domain is stored in the environment variable \USERDOMAIN.

Please enter

```
set username
set userdomain
```

to find out your user domain and your personal account name.

Then please

1. Open a command shell.

2. Change the current directory to the MPICH bin directory:
   ```
cd <MPICH_HOME>\mpd\bin
```
3. Execute the command `mpiregister.exe`

4. Enter your Windows account name `USERDOMAIN\ACCOUNTNAME`.

5. Enter your Microsoft Windows password for this account.

6. Accept this account to be persistent.

⚠️ Beware: An empty password is not allowed, so you can never run MPICH with an empty password!

⚠️ Whenever you change the password of the MPICH account you will need to `mpiregister` again.

To run MpCCI under Microsoft Windows the MPICH service must be installed. The name of the service is `mpich_mpd`.

It may happen sometimes that an account check fails or the `mpich_mpd` rejects to create MPICH processes. You may then have to restart the `mpich_mpd` service.

Please enter

```
net stop mpich_mpd : Stop the service
net start mpich_mpd : Start the service
```

in a command shell.

⚠️ You need to have administrative rights to restart a service under Microsoft Windows.
9 Installing Perl

UNIX and Linux

If you need to install or upgrade Perl under UNIX or Linux please either download the source code of Perl 5.8.8 from www.perl.com/download.csp and compile it on your local machine or use a binary distribution.

A binary distribution for all flavors of UNIX and Linux platforms can be downloaded from

- www.perl.com/CPAN/ports/index.html (all platforms)
- www.hp.com/go/perl (HP-UX port)

The installation is straightforward for a UNIX system administrator.

ActivePerl for Microsoft Windows

ActivePerl is the best available Microsoft Windows port of Perl. A 64 bit beta version for Windows 64 and Windows Vista 64 bit is available.

⚠️ Under Windows 64 please use the 64 bit version of Perl. Although the 32 bit Perl version works fine. and MpCCI is able to detect the true Windows version used in fact a 32 bit environment is prepared and you may not have the full 64 bit functionality available.

We recommend to use the ActivePerl 5.8.8. Versions before 5.8.8 had some problems under Microsoft Windows with signals, fork emulation, the backtick operator and whitespace in filename when executing external commands.

⚠️ Please do not use ActivePerl 5.10. This version comes without the TK modules required for the MpCCI project manager.

The latest version is ActivePerl 5.8.8.822. You may follow the link

www.activestate.com/Products/ActivePerl.

Please download the binary ActivePerl distribution from

aspn.activestate.com/ASPN/Downloads/ActivePerl.

We recommend to use the Microsoft Windows MSI installer version of ActivePerl since during the installation the MSI appends the Perl binaries directory to your PATH environment and properly modifies or sets the .pl file association in the HKEY_CLASSES_ROOT of the Microsoft Windows registry.

After downloading the distribution file ("ActivePerl-5.8.x.msi"), please execute the ActivePerl installation program and follow the instructions. The installation is a typical Microsoft Windows MSI installation and is straightforward.

Please make sure that the .pl filename extension is properly associated with the newest Perl version installed. You may validate this by typing
assoc .pl

You should see an association pattern like `perl` and use this token as the argument for the `ftype` command

ftype perl

You should now see a line like

"D:\program files\perl\bin\perl.exe" "%1" %*

and type in the fully qualified pathname to get the version of Perl:

"D:\program files\perl\bin\perl.exe" -version
10 Installing Java

If you do not have a working Java or Java JRE installed or your current Java version is GNU Java or not up-to-date (§ 2.4 The Java Runtime Environment) you have two choices:

- You may download a multi-platform Java runtime environment as part of the MpCCI distribution.
- You may download the Java JRE from the world wide web and install it on your local computer.

Java as part of MpCCI: Multi-platform

In this case you should select the Include JRE checkbox while downloading the multi-platform MpCCI. MpCCI JRE is not really installed on your local computer. The JRE is just a portion of software within the MpCCI home directory.

The Java JRE needs not to be configured for a specific platform. There are no conflicts between the MpCCI JRE and existing Java installations on your local machine.

MpCCI locates any existing JRE installation at runtime, whether installed locally, within MpCCI or on a file server as part of MpCCI.

Java as part of MpCCI: Microsoft Windows with MSI

If you need to install JRE under Microsoft Windows and you use the MpCCI MSI version we recommend to download and install Java together with MpCCI.

After starting the Microsoft Windows installer for MpCCI please select the Include JRE checkbox. Having got the Java distribution, execute the installation program and follow the instructions. The Microsoft Windows installation is straightforward.

Java web download and installation

The JRE for many platforms is available from Sun Microsystems at java.sun.com/j2se/1.5.0/download.jsp.

For some platforms, e.g. IBM or HP, you will have to look at the vendor specific sites. The installation is straightforward for a UNIX system administrator.
IV Getting Started
IV Getting Started – Contents

1 Multi-Physics Computation with MpCCI
   1.1 Multi-Physics ................................................................. 4
   1.2 Solution of Coupled Problems ........................................... 5
   1.3 Code Coupling with MpCCI ............................................... 6

2 Setting up a Coupled Simulation ........................................... 8
   2.1 A Simple Example .......................................................... 8
   2.2 Model Preparation .......................................................... 8
      2.2.1 CFD Model .............................................................. 9
      2.2.2 FE Model .............................................................. 10
   2.3 Starting the MpCCI GUI .................................................. 10
   2.4 Models Step – Choosing Codes and Model Files ..................... 11
   2.5 Coupling Step – Definition of Coupling Regions and Quantities 13
      2.5.1 Define Coupling Regions ........................................... 14
      2.5.2 Select the Components for each Interconnected Code ........ 15
      2.5.3 Specify the Quantities which will be Exchanged ............... 15
   2.6 Edit Step – Further Coupling Options ................................ 16
   2.7 Go Step – Starting Servers and Codes ................................ 18
      2.7.1 Configuring the Initial Exchange Mode ......................... 19
      2.7.2 Setting Option Parameters ........................................ 20
      2.7.3 Starting the Coupled Simulation ................................. 20
      2.7.4 Interrupting the Computation ..................................... 23

3 Checking the Results ......................................................... 24
   3.1 The MpCCI Visualizer ..................................................... 24
   3.2 Post-Processing ............................................................. 25
1 Multi-Physics Computation with MpCCI

1.1 Multi-Physics

The purpose of MpCCI is to perform multi-physics simulations. The systems under consideration are known as coupled systems. A coupled system consists of two or more distinct systems. Each system is governed by a characteristic set of differential equations, but both systems share some variables and cannot be solved separately (see also Zienkiewicz and Taylor [2000] for a more precise definition).

\[
\begin{align*}
\text{domain A} & : \quad f(a,b,c,d) = 0 \\
\text{domain B} & : \quad g(c,d,e,f) = 0 \\
\end{align*}
\]

Figure 1: Coupled system: Two domains and a coupling region. Both systems share the variables c and d in the coupling region

Figure 1 shows two coupled systems. In the coupling region, both systems share some variables and the governing equations of both systems must be solved. Depending on the dimension of the coupling region, *surface coupling* and *volume coupling* are distinguished. The shared variables (c and d in Figure 1) are called *coupling quantities*.

Typical multi-physics simulations are:

**Fluid-Structure Interaction (FSI):**
- First system: Fluid flow (Navier-Stokes equations)
- Second system: Solid mechanics (equilibrium)
- Quantities: Pressure (1 → 2), deformation (2 → 1)

**Thermomechanical coupling**
- First system: Solid mechanics (equilibrium)
- Second system: Heat conduction (Fourier’s law)
- Quantities: Temperature (2 → 1), deformation (1 → 2)

**Electrothermal coupling**
1.2 Solution of Coupled Problems

To find a solution for a coupled problem, all governing equations, which can be combined in a large system, must be solved. The solution in this way is called \textit{strong coupling}. However, solving a system with strong coupling is often difficult as different approaches are necessary to solve the sub-problems.

An alternative approach is through \textit{weak coupling}. Here each problem is solved separately and some variables are exchanged and inserted into the equations of the other problem. This procedure usually yields a less exact solution compared to \textit{strong coupling}. The advantages of the weak coupling are that the sub-problems can be solved faster than the complete system and that specialized solvers can be used for each.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{staggered_algorithm.png}
\caption{Staggered algorithm for solution of a coupled problem.}
\end{figure}

The staggered method is sketched in Figure 2, which is one of the weak coupling approaches of MpCCI: Code A computes one step, sends data to code B, which then computes one step and sends data back to code A and so on. In addition to the staggered approach, MpCCI supports parallel execution of both codes. The selection of the coupling algorithm is described in \ref{go:step}. An important issue is the data exchange. The quantities must be transferred from one code to the other. This process can be divided into two parts:

\textbf{Association}: Each point and/or element of the components of a coupling region is linked to a partner in
the other system. The process of finding partners is also called neighborhood search, see V-3.2 Data Exchange ◢.

**Interpolation:** The quantities must be transferred to the associated partner on the other mesh. In this process, the different mesh geometries, data distributions and the conservation of fluxes must be considered.

MpCCI fully supports the data exchange between non-conforming meshes, i.e. the meshes of each subsystem can be defined to optimize the solution of the subsystem.

### 1.3 Code Coupling with MpCCI

Running a co-simulation with MpCCI requires the following steps:

**Preparation of Model Files.** Before starting a co-simulation, each domain must be modeled separately, i.e. a model file must be created for each simulation code. The models must contain a definition of the coupling regions. The preparation of a model file depends on the simulation codes, a detailed description is given in the corresponding sections of the Codes Manual.

**Definition of the Coupling Process.** Simulation codes and corresponding model files must be selected. The coupled regions, quantities and a coupling algorithm must be selected, further coupling options can be given. This step is completely supported by the MpCCI GUI.

**Running the Co-simulation.** After starting the MpCCI server, both coupled codes are started. Each code computes its part of the problem while MpCCI controls the quantity exchange.

**Post-Processing.** After the co-simulation, the results can be analyzed with the post-processing tools of each simulation code, with the MpCCI Visualizer or with general-purpose post-processing tools.

The application of MpCCI requires a good knowledge of the employed simulation codes. Therefore, it is recommended to use those codes for a co-simulation the user has already some experience with.

The data exchange between two simulation codes requires a “code adapter”, which constitutes a “code plug-in” for data transfer. Therefore only codes which are supported by MpCCI can be used. It is also possible to develop special code adapters, more details are given in the Programmers Guide.

An overview of the complete co-simulation process is given in Figure 3.
Figure 3: Co-simulation with MpCCI: Overview of the simulation process
2 Setting up a Coupled Simulation

2.1 A Simple Example

Let us look at a simple example to clarify the approach, which was described in the preceding chapter. Figure 1 shows a model of a valve which consists of a flexible flap which can be open or closed depending on the direction of the flow. The purpose of the simulation is to investigate the behavior of the valve depending on inlet and outlet pressure.

The computation of the pressure distribution and the flow rates is achieved using *Computational Fluid Dynamics* (CFD), whereas the deformation of the flap can be computed applying numerical structural mechanics via a *Finite Element* (FE) code. Each domain, has a significant influence on the other, thus the problems cannot be solved separately. The CFD simulation requires the deformation of the flap as a boundary condition, whereas the FE simulation requires the fluid pressure as external load. Thus during a co-simulation, these quantities must be exchanged as shown in Figure 2.

![Figure 1: A simple coupled system](image)

![Figure 2: Exchange of quantities in the example FSI simulation.](image)

2.2 Model Preparation

Before starting the coupled simulation, the models must be prepared in each code. In our example, this requires a model of the flow region and a model of the flap structure.
Both models are created in the undeformed condition, i.e. the flap is straight. The computation of such a problem requires a CFD-code which can handle moving/deformable meshes.

It is recommended that you keep your FE and CFD model files in separate directories (see Figure 3) because of the following reasons:

- Clear storage and maintenance of simulation data.
- Ensuring that the codes do not overwrite files of other codes, when identical file names are used.
- Simplification of the porting of the analysis when running the simulation on different platforms.

### 2.2.1 CFD Model

To model the fluid domain, define the geometry in the CFD pre-processor and create a mesh. Figure 4 shows a possible mesh for the fluid domain. (Please note that the displayed mesh is rather coarse and would probably not yield good results).
All boundary conditions must be defined as well. They comprise the flap surface, walls, inlet and outlet in Figure 4. Also for the coupling region boundary conditions may be set - this depends on the CFD code applied, please check the Codes Manual for details.

It is recommended to run the fluid problem on its own before starting a coupled simulation to be sure that the model setup is correct. In the present example the coupling region could be defined as a rigid wall for this purpose. If the model is not appropriate for computations with the CFD code alone, it is most likely that a co-simulation will either not run.

### 2.2.2 FE Model

![Figure 5: Finite Element model of the flap](image)

Compared to the CFD model which covers the fluid domain, the FE model covers the flap itself. Figure 5 shows a mesh of the flap with quadrilateral elements. The element sizes do not need to correspond to those of the fluid domain in any way as MpCCI can handle non-conforming meshes.

The flap is connected to the top wall, i.e. the top nodes of the mesh must be fixed. The surface of the flap must be defined as a boundary to which the pressure load can be applied in the co-simulation. Although the system will not show any deformation without an external load, it is recommended to check the validity of the mesh and boundary conditions (i.e. the fixed nodes here) by running the FE problem alone and/or by performing a problem check if it is available in the FE code. To obtain a deformation you can replace the fluid forces by a similar load.

### 2.3 Starting the MpCCI GUI

The MpCCI GUI is started by executing the command `mpcci gui` from a shell console.

The MpCCI GUI guides you through the steps to interconnect the FE and CFD models and to run the coupled simulation.
### 2.4 Models Step – Choosing Codes and Model Files

In the Models Step, the following steps should be accomplished:

- Select the analysis codes to couple.
- Specify their model files.
- Set some code specific parameters if desired.
- Scan the model files to determine the potential coupling regions.

![Figure 6: Models Step for CFD-code](image)

For each code, `Code_1` and `Code_2` (Figure 6), a field for selecting and configuring the analysis code is provided in the MpCCI GUI.

**For the CFD-Code:** Select the desired CFD-Code from the pull-down menu providing the available codes. After your selection the parameters and settings of the picked solver are shown underneath (see Figure 6).

1. Select the CFD model file by clicking on the **Browse** button in order to choose the file via the file browser.
2. You may also configure some additional parameters like the CFD version or release.

3. Scan the model file by clicking on the Start Scanner button.

![Figure 7: Models Step for FE-code](image)

For the **FE-Code**: Select the desired FE-Code from the list provided by the pull-down menu. After your selection the corresponding parameters and settings are shown underneath (see Figure 7).

1. Select the FE model file via the file browser by clicking on the Browse button.
2. Select the unit system.
3. If the unit system is set to variable an additional parameter will be shown where you can select the grid length unit.
4. In addition you may change the settings for the FE release or the scan method.
5. Scan the model file by clicking on the Start Scanner button.

For each scanned model file the status is assigned to the corresponding analysis code: ✔️ Done if the extraction of the interface regions was successful.
if the scanner encountered problems during the scan which may depend on the parameters set for the analysis code.

By clicking on the status button, you get the information that has been extracted from the model file. If the scanning has failed the error message from the scanner is displayed in a pop-up window.

After performing a successful scan of all model files you may continue with the next step of the coupled simulation setup by clicking on the Next button.

### 2.5 Coupling Step – Definition of Coupling Regions and Quantities

In the Coupling Step the following steps need to be done:

- Define the coupling regions.
- Select the components for each interconnected code.
- Specify the quantities which will be exchanged.

The components are automatically sorted by their element type. There are basically two types of components:

**Global Variables** These components are data structures that are not related to the CFD or FEM grids. They contain global quantities like time or time step size. These components can be found under the Global (0D) element type label.

**Element Components** These components comprise collections of elements. They contain model parts and the related grid based quantities, e.g. nodal positions, heat values or forces. To build up the interconnection, elements should be gathered that are part of the coupling region. A collection of 1D elements will be found under the Line (1D) label, a collection of 2D elements under the Face (2D) label, and a collection of 3D elements under the Volume (3D) label.

To navigate between the different element types you have to click on the corresponding tab: Global (0D), Line (1D), Face (2D) or Volume (3D). Inactive element types are grayed out (see Figure 8).

In our example only the Face (2D) label is activated and already selected.

For each element type the MpCCI GUI window is divided into three fields:

- **Regions** with a list of the created coupling regions.
- **Components** with the lists of code components which may be selected into the coupled components lists, which define the coupling region.
- **Quantities** with a list of available quantities which may be selected and configured to be transferred from one application to the other on the selected coupling region.
2.5.1 Define Coupling Regions

In the field “Regions” there is already the default region Face.1 defined. To change its name use Rename from the menu which pops up when you click with the right mouse button onto the region name. To add further coupling regions use the Add button at the top of the regions list (Figure 9).
2.5.2 Select the Components for each Interconnected Code

In the Components part for each code a list with available components for that particular code is shown.

According to the valve example with the flexible flap this will imply: *For CFD-code* select the component flap-surface by double clicking or using drag & drop (see Figure 10).

*For FE-code* select the component outside in the same way as mentioned above.

2.5.3 Specify the Quantities which will be Exchanged

In the quantities list, select the quantities to be transferred by clicking on the checkbox near the name of the quantity (Figure 11). In MpCCI the names of the quantities are standardized. In our example we want to transfer *pressure* and *deformation*. In MpCCI the pressure is handled by the quantity *RelWallForce* and the deformation by *NPosition* which stands for “nodal position”. For further information on the meaning of quantities see the Quantity Reference in the Appendix and ▷ V-3.1.1 Physical Domains ◁. For our example select and configure the following quantities:

*RelWallForce* for exchanging the pressure (see Figure 11). Sender is CFD-code. Its pressure unit is not editable because the unit system for the CFD-code is fixed to the “SI” unit system. On the other hand you may select the appropriate unit for the FE-code because of the “variable” unit system setting in the model step (see Figure 7).

*NPosition* for exchanging the deformation (see Figure 12). Sender is FE-code. The unit for the deformation may be set as described above.

For further information have a look at ▷ V-4.4 Coupling Step ◁.

The definition of the coupling interface is completed, please click on the [Next>] button to continue.
2.6 Edit Step – Further Coupling Options

In the Edit Step you may modify some MpCCI control parameters like
• parameters for algorithms, e.g. search algorithms or mesh quality checks,
• switches controlling the output level for debugging or
• information about the coupling algorithm.

An overview of all settings is given in ▷V-4.5 Edit Step◁.

Figure 13: Edit Step

The MpCCI GUI provides default values for all settings which can be modified. The Edit Step window displays a tree of parameters on the left. On the right side the corresponding parameters are shown in a table as depicted in Figure 13. This table provides information such as

• the parameter name displayed with an orange background,
• the editable parameter value displayed with a yellow background,
• and a parameter description.

To select a parameter click on the parameter name in the parameters tree. That followed edit the value by clicking the cell with the parameter and select or type in the desired value for this parameter.
For most cases the default values are appropriate, but for our example you may edit the name of the tracefile (Control → TraceFile → Name) which is the input file for the MpCCI Visualizer. Additionally you may edit the output level (Control → OutputLevel → Global) which tells MpCCI how much output will be written during the simulation process.

Now click on the [Next>] button to proceed to the start-up window for the coupled simulation.

### 2.7 Go Step – Starting Servers and Codes

![Go Step](image)

**Figure 14: Go step**

In the Go Step each application including the MpCCI coupling server is shown in its own frame (see Figure 14). Before starting the coupled simulation you have to configure the start-up of the applications. For the MpCCI coupling server you usually may retain the default values. For each analysis code you have to configure the initial exchange mode and set some option parameters if necessary.
2.7.1 Configuring the Initial Exchange Mode

In our coupled valve problem we consider that the pressure will initiate the deformation of the flap. Therefore the CFD-code will be the initiator and the FE-code in our example will request the pressure solution before starting its computation. After both codes have performed the initial data exchange they run the analysis computation until a coupled target time is reached. At this target time both analysis codes will exchange their solution quantities.

For the CFD-code, set the Initial quantities transfer mode to send as in Figure 15.

![Figure 15: CFD-code settings](image)

For the FE-code, set the Initial quantities transfer mode to receive (see Figure 16).
With this *Initial quantities transfer* configuration for the CFD- and FE-code, the coupled simulation will use a parallel coupling algorithm (see Figure 17).

### 2.7.2 Setting Option Parameters

For the CFD-code select the *Run parallel* option. Now the parameters for running the CFD-code in parallel are added to the window as shown in Figure 15. Set the number of processes to be applied e.g. to 4. If no hosts are specified the CFD-code will run in parallel on the local machine.

For the FE-code no more options need to be set.

### 2.7.3 Starting the Coupled Simulation

Before starting a coupled simulation you have to establish a project by using the action *Save As* from the File menu. This is due to the fact that MpCCI needs the project file to generate its input file and to get information about the settings which were defined in the MpCCI GUI.

Server and each application have their own *Start* button. But only the *Start* button from the MpCCI server is enabled because the server has to be launched in advance. After the server has been started its *Start* button will be replaced by a running symbol, all settings will be locked and the *Start* buttons for the next applications will be enabled one after the other.

For our valve example the start-up procedure will be:
1. Click on **Start** to launch the coupling server. Now the server waits to get response from all client codes.

2. The **Start** button of the next application (CFD-code) is enabled. Now click on **Start** to launch the CFD analysis code.

3. After that the last application (FE-code) has to be launched by a click on its **Start** button.

Now the client codes are started and the coupling server starts the initialization phase as follows:

1. Initial handshaking: the CFD-code and the FE-code contact the server.

   
   
   ```
   [MpCCI SETUP] Reading general setup data from file "/home/COUPLED/.cciclientrc" ..  
   [MpCCI SETUP] Contacting leading cciserver on host nemo, port 47111 ...  
   [MpCCI SETUP] Got a connection to the leading mpcci server ...  
   [MpCCI SETUP] Got the answer "SOCKET 47112 nemo"  
   [MpCCI SETUP] "SLEEPTIME 0.200000 LOGFILE mpccirun.clientlog.1"  
   ==> CLIENT with socket connection to remote host nemo, port 47112
   ```

2. Exchange of the interface topology: The coupling server requests the interface topology from each client. The clients send their interface topology.

   ```
   cfd:1.0: Exchanging partitions with local code ...  
   cfd:1.0:  
   cfd:1.0: Exchanging partitions with remote codes ...  
   
   fe:2.0: Exchanging partitions with local code ...  
   fe:2.0:  
   fe:2.0: Exchanging partitions with remote codes ...
   ```

3. MpCCI coupling server performs neighborhood searches and computes the mapping between CFD-code and FE-code meshes.

   ```
   cfd:1.0: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
   cfd:1.0: NEIGHBORHOOD COMPUTATION  
   cfd:1.0: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
   cfd:1.0: I’m calculating neighborhood between mesh 1 of the local code  
   cfd:1.0: and mesh 1 of remote code 2 ...  
   
   fe:2.0: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
   fe:2.0: NEIGHBORHOOD COMPUTATION  
   fe:2.0: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~  
   fe:2.0: 
   ```
In our example the parallel coupling algorithm is used with send (CFD) and receive (FE) as initial exchange mode (see Figure 17 bottom). The CFD-code is represented by “code B” and the FE-code by “code A”. Before starting the computation the CFD-code sends the surface forces RelWallForce to the FE-code (1). Then both codes compute in parallel their solution quantities pressure and deformation until the coupled target time is reached (2). Now the solution quantities are exchanged (3). Possible coupling algorithms are described in detail in V-3.3 Coupling Algorithms §.

Figure 17: Initial exchange and resulting coupling algorithms for two codes with “exchange before solution”
When an application exits normally, a window displays information collected from the application output. Generally there are two types of windows: One for *information* and another one for *errors*.

### 2.7.4 Interrupting the Computation

While the applications are running, the [Kill] and [Stop] buttons at the bottom of the window are enabled and provide following functionalities:

- **[Kill]** to terminate all applications by sending a kill signal.
- **[Stop]** to terminate all applications by sending a stop signal. The stop call may not immediately take effect, because it depends on the implementation of the stop procedure in the simulation code.

To modify parameter settings you have to terminate the applications by pressing the [Kill] button. Otherwise the editing of parameter values is disabled.
3 Checking the Results

3.1 The MpCCI Visualizer

The MpCCI Visualizer is suitable for a quick check whether the coupling process was successful. The coupling region, orphaned nodes and exchanged quantities can be checked to ensure that the specified coupling has really occurred. Here only a short introduction is presented, a concise description of the MpCCI Visualizer is given in V-6 MpCCI Visualizer.

During the coupling process, an additional control process can be started, which writes a “tracefile”, i.e. a collection of the exchanged data (see Figure 3 and 2.7 Go Step – Starting Servers and Codes). To start a control process and obtain a tracefile, the checkbox Start additional control process must be selected in the Go Step. The default name of the tracefile is "tracefile.ccv", it can be changed in the Edit Step of the MpCCI GUI.

After a simulation the MpCCI Visualizer can be started by selecting Tools→Visualizer from the MpCCI menu or by entering the command mpcci vis. Tracefiles can even be opened with the MpCCI Visualizer before the completion of a computation.

Figure 1: Control window (left) and viewer window (right) of the MpCCI Visualizer

The visualizer starts with the control window, which is depicted in Figure 1 on the left. To open a tracefile, choose File→Open. After selecting a file, the viewer window pops up, which displays the coupling region.
In the control window the data to display can be selected including the exchanged quantities. In Figure 1 the quantity \textit{NPosition} is selected and displayed in the viewer window. Both the quantities which are sent, i.e. before the interpolation, and the quantities which are received are displayed. As sent and received data are normally defined at the same locations, both parts can be separated with the arrow buttons in the viewer window.

Transient analyses consist of several steps, the step number can be selected in the control window.

The MpCCI Visualizer can not replace a post-processing tool, as the tracefile only obtains information from MpCCI, which solely covers the coupling region. Information on other regions of the analysis is not available.

### 3.2 Post-Processing

During a coupled simulation both codes should write their results to appropriate files. The visualization of these results can be carried through with built-in tools of the simulation codes. Unfortunately the built-in post-processing tools solely allow the visualization of one part of the problem. In our example this means, that you can display the fluid properties with one tool and the structural properties with the other tool.

There is general post-processing software, which can read output data from different standardized formats and combine results from different files. Describing these tools is however beyond the scope of this manual.
V User Manual
MpCCI is a registered trademark of Fraunhofer SCAI

www.scai.fraunhofer.de/mpcci

Fraunhofer Institute
Algorithms and
Scientific Computing

Fraunhofer Institute for Algorithms and Scientific Computing SCAI
Schloss Birlinghoven, 53754 Sankt Augustin, Germany

Abaqus and SIMULIA are trademarks or registered trademarks of Dassault Systèmes
ANSYS is a registered trademark of Ansys, Inc.
FINE/Hexa is a registered trademark of NUMECA International
FINE/Turbo is a registered trademark of NUMECA International
FLUENT is a registered trademark of Fluent Inc.
FLUX is a registered trademark of Cedrat of Grenoble, France
ICEPAK is a registered trademark of Fluent Inc.
MSC.Marc is a registered trademark of MSC.Software Corporation
PERMAS is a registered trademark of Intes GmbH
STAR-CD is a registered trademark of CD adapco Group
RadTherm is a registered trademark of ThermoAnalytics Inc.

FLEXlm is a registered trademark of Macrovision
Perl has a copyright by Larry Wall and others
ActivePerl has a Community License Copyright of Active State Corp.
OpenSSH has a copyright by Tatu Ylonen, Espoo, Finland
Java is a registered trademark of Sun Microsystems, Inc.
MPICH-1 and MPICH-2 has a copyright by University of Chicago and Mississippi State University
Linux is a registered trademark of Linus Torvalds
UNIX is a registered trademark of The Open Group
Windows, Windows XP and Windows Vista are registered trademarks of Microsoft Corp.
# V User Manual – Contents

## 1 Introduction
1.1 Basic Structure of MpCCI .......................................................... 8

## 2 The MpCCI Software Package
2.1 Introduction ............................................................................. 10
2.2 The MpCCI Home Directory .................................................... 11
2.3 Environment and Environment Variables ................................. 12
   2.3.1 MPCCI_ARCH - Architecture Tokens ................................. 13
   2.3.2 MPCCI_DEBUG - for Debugging ........................................ 14
   2.3.3 MPCCI_INITIAL_EXCHANGE ......................................... 15
2.4 MpCCI Project and Output Files .............................................. 16
   2.4.1 MpCCI Project Files ..................................................... 16
   2.4.2 MpCCI Server Input Files .............................................. 16
   2.4.3 Log Files ..................................................................... 16
   2.4.4 Tracefile ..................................................................... 16
2.5 The MpCCI Resource Directory .............................................. 17
2.6 Temporary Files ..................................................................... 18
2.7 Third Party Software Used by MpCCI ..................................... 20
   2.7.1 Perl .......................................................................... 20
   2.7.2 Java .......................................................................... 20
   2.7.3 MPI .......................................................................... 20
   2.7.4 Remote Shell and Remote Copy ................................. 20

## 3 Code Coupling
3.1 Multi-Physics ................................................................. 21
   3.1.1 Physical Domains ..................................................... 21
   3.1.2 Coupling Types .......................................................... 22
3.2 Data Exchange ..................................................................... 24
   3.2.1 Pre-Contact Search ...................................................... 25
   3.2.2 Minimal Distance ........................................................ 26
   3.2.3 Intersection .............................................................. 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.4 Orphaned Nodes and Elements</td>
<td>29</td>
</tr>
<tr>
<td>3.2.5 Flux and Field Interpolation</td>
<td>29</td>
</tr>
<tr>
<td>3.3 Coupling Algorithms</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1 Course of the Coupling Process</td>
<td>30</td>
</tr>
<tr>
<td>3.3.2 Stationary Problems</td>
<td>31</td>
</tr>
<tr>
<td>3.3.3 Transient Problems</td>
<td>31</td>
</tr>
<tr>
<td>3.3.4 Exchange of Time Step Size</td>
<td>36</td>
</tr>
<tr>
<td>3.3.5 Subcycling</td>
<td>37</td>
</tr>
<tr>
<td>3.3.6 Restarting a Coupled Simulation</td>
<td>38</td>
</tr>
<tr>
<td>3.4 Running MpCCI in a Network</td>
<td>39</td>
</tr>
<tr>
<td>3.4.1 Client-Server Structure of MpCCI</td>
<td>39</td>
</tr>
<tr>
<td>3.4.2 Hostlist files</td>
<td>41</td>
</tr>
<tr>
<td>3.4.3 Remote Shell and Remote Copy</td>
<td>42</td>
</tr>
<tr>
<td>3.4.4 Coupled Analysis in Batch Mode</td>
<td>42</td>
</tr>
<tr>
<td>3.5 Mesh Checks</td>
<td>57</td>
</tr>
<tr>
<td>3.5.1 Bounding Box Checks</td>
<td>57</td>
</tr>
<tr>
<td>3.5.2 Mesh Quality Checks</td>
<td>57</td>
</tr>
<tr>
<td>4 Graphical User Interface</td>
<td>58</td>
</tr>
<tr>
<td>4.1 Starting and Exiting MpCCI GUI</td>
<td>58</td>
</tr>
<tr>
<td>4.1.1 Starting MpCCI GUI</td>
<td>58</td>
</tr>
<tr>
<td>4.1.2 Exiting MpCCI GUI</td>
<td>59</td>
</tr>
<tr>
<td>4.2 MpCCI GUI Menus</td>
<td>59</td>
</tr>
<tr>
<td>4.2.1 File Menu</td>
<td>60</td>
</tr>
<tr>
<td>4.2.2 Batch Menu</td>
<td>61</td>
</tr>
<tr>
<td>4.2.3 License Menu</td>
<td>62</td>
</tr>
<tr>
<td>4.2.4 Tools Menu</td>
<td>63</td>
</tr>
<tr>
<td>4.2.5 Preferences Menu</td>
<td>63</td>
</tr>
<tr>
<td>4.2.6 Codes Menu</td>
<td>63</td>
</tr>
<tr>
<td>4.2.7 Help Menu</td>
<td>63</td>
</tr>
<tr>
<td>4.3 Models Step</td>
<td>64</td>
</tr>
<tr>
<td>4.3.1 Code Parameters</td>
<td>64</td>
</tr>
</tbody>
</table>
4.3.2 Requirements .............................................. 64
4.4 Coupling Step .................................................... 64
  4.4.1 Generate Regions ............................................ 65
  4.4.2 Options Part .................................................. 65
  4.4.3 Quantity properties .......................................... 65
  4.4.4 Quantity Sender .............................................. 66
  4.4.5 Predefined Sets ............................................. 66
  4.4.6 Requirements ................................................ 69
4.5 Edit Step ......................................................... 71
  4.5.1 Control ........................................................ 71
  4.5.2 Contact ......................................................... 74
4.6 Go Step ........................................................... 76
  4.6.1 Configuring the MpCCI Coupling Server ...................... 76
4.7 Remote File Browser ............................................ 80
  4.7.1 File Browser Handling ....................................... 80
  4.7.2 How to mount a new file system ............................ 81

5 Command Line Interface 84
  5.1 Using the Command Line Interface .............................. 84
  5.2 Overview of All Subcommands .................................. 86
  5.3 Starting MpCCI ................................................... 88
    5.3.1 mpcci gui ................................................ 89
    5.3.2 mpcci morpher ............................................ 90
    5.3.3 mpcci observe ............................................ 93
    5.3.4 mpcci pm .................................................. 94
    5.3.5 mpcci vis ................................................ 95
    5.3.6 mpcci xterm .............................................. 96
  5.4 Information and Environment ................................... 97
    5.4.1 mpcci arch ............................................... 98
    5.4.2 mpcci doc ................................................. 99
    5.4.3 mpcci info ............................................... 100
    5.4.4 mpcci env ................................................. 102
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>MpCCI Grid Morpher</td>
<td>145</td>
</tr>
<tr>
<td>8</td>
<td>MpCCI Project Manager</td>
<td>146</td>
</tr>
</tbody>
</table>
1 Introduction

This user manual shall give an overview of the basic structures and features of MpCCI. All code-specific issues are discussed in the chapters of the Codes Manual.

1.1 Basic Structure of MpCCI

MpCCI is a software environment which enables the exchange of data between the meshes of two simulation codes in a multi-physics simulation. The architecture of this coupling process is shown in Figure 1.

MpCCI enables a direct communication between the coupled codes by providing adapters for each code. These code-adapters make use of already existing application programming interfaces (APIs) of the simulation tools. This technique allows an easy installation of MpCCI at the end users site without changing the standard installation of the simulation codes.

Since the meshes belonging to different simulation codes are not compatible in general, MpCCI performs an interpolation. In case of parallel codes MpCCI keeps track on the distribution of the domains onto different processes. MpCCI allows the exchange of nearly any kind of data between the coupled codes; e.g. energy and momentum sources, material properties, mesh definitions, or global quantities. The details of the data exchange are hidden behind the concise interface of MpCCI.

The MpCCI environment consists of several components:

- MpCCI code-adapters allow to adapt MpCCI to commercial codes through their standard code APIs without any changes in the source of the simulation code.
- The MpCCI GUI provides a comfortable way to define the coupling setup and to start the simulation - independent of the codes involved in the coupled application, it is described in §4 Graphical User Interface <
- The MpCCI coupling server is the “heart” of the MpCCI system. Environment handling, communication between the codes, neighborhood computation, and interpolation are part of this kernel.
Figure 1: MpCCI architecture
2 The MpCCI Software Package

2.1 Introduction

MpCCI is not a single executable, but a complex software package, which consists of the following parts:

**Coupling server and clients.** The “heart” of MpCCI, which is responsible for the code coupling process, see ▶ 3 Code Coupling ↳.

**MpCCI Visualizer.** The MpCCI visualizer is a tool for checking the coupling process. Meshes and transferred quantities can be displayed. See ▶ 6 MpCCI Visualizer ↳ for more information.

**Collection of small tools.** A variety of small tools is included in the MpCCI software package. These are useful for obtaining information on the computing environment, installation and licensing issues, and job control. A description of these tools is given in ▶ 5 Command Line Interface ↳.

**MpCCI Grid Morpher.** In some coupled analyses the coupling surface is moving during the simulation. If one of the coupled codes does not provide a morphing tool itself, the MpCCI Grid Morpher can be used, see ▶ 7 MpCCI Grid Morpher ↳.

**MpCCI Project Manager.** The project manager, which is described in ▶ 8 MpCCI Project Manager ↳, is useful to organize your coupling projects. It lists all project in your HOME-directory and gives easy access to basic MpCCI tasks.

These parts of MpCCI can basically be accessed in two ways:

- Through the graphical user interface, MpCCI GUI, which offers a convenient way to define a coupled simulation, starting and controlling jobs. Most of the MpCCI tools can be started directly from the MpCCI GUI. An introduction to using the MpCCI GUI can be found in Getting Started. A concise description of all MpCCI GUI features is given in ▶ 4 Graphical User Interface ↳.

- All parts of MpCCI can be run directly from the command line (i.e. by typing commands in a shell in UNIX/Linux or the DOS shell in Windows). Besides starting the main parts of MpCCI, a collection of small tools is offered. These tools can be used to gather important information, deal with licensing issues and to control the coupling process. All available commands are described in ▶ 5 Command Line Interface ↳.
2.2 The MpCCI Home Directory

MpCCI resides completely within one directory with subdirectories. This MpCCI home directory is referred to as MPCCI_HOME.

This directory is created during the installation of MpCCI. For selection of the path, please see the Installation Guide. The only requirement, which must be met to run MpCCI properly is that the MpCCI executable, which is located in "MPCCI_HOME/bin" must be included in your PATH environment variable.

The MPCCI_HOME directory contains a number of subdirectories:

"bin" The MpCCI binaries, which contains the MpCCI main program "mpcci" or "mpcci.exe" and the MpCCI shell "mpccish" or "mpccish.exe". The MpCCI shell is executed by MpCCI on remote machines.

"codes" contains all code-specific files, in separate subdirectories for each code and one for the MpCCI server. Each subdirectory includes information for the MpCCI GUI ("gui.xcf"), and some Perl scripts ("*.pm") for scanning of input files, starting and stopping the coupled codes. These are called by MpCCI during the coupling process.

"dist" contains distribution information, which is required for patch updates.

"doc" contains the MpCCI documentation, which can be accessed with the mpcci_doc command, see 5.4.2 mpcci doc on page 99

"gui" is the base directory of the MpCCI GUI, containing configuration files and the ".jar" archives of the MpCCI GUI.

"include" contains header files, which are necessary for developing code adapters, see Programmers Guide.

"lib" contains libraries, which are necessary for developing code adapters, see Programmers Guide.

"license" contains the license manager FLEXlm, which is used by MpCCI. All license files should also be placed in this directory.

"perlmod" contains various Perl modules, mainly small helper tools.

"tutorial" contains the input files needed for trying the examples described in the Tutorial.

⚠️ Do not edit any of the files in the MPCCI_HOME directory unless you know what you do. Improper changing of files may destroy your MpCCI distribution, which makes a new download necessary.
2.3 Environment and Environment Variables

MpCCI uses environment variables to transport information between subprocesses and processes created on remote hosts. Most of the environment variables are volatile, some are not. You may display theMpCCI related environment variables with the command `mpcci env`.

MpCCI distinguishes two categories of variables:

Control variables are named _MPCCI_. . . and may be defined before startingMpCCI to control the behavior of MpCCI. An overview of these variables is given below.

Internal variables begin with an underscore _MPCCI_. . . are volatile environment variables used internally by MpCCI and should not be set by the user. Internal variables which are related to a specific code are also named accordingly, i.e. _MPCCI_<code name>_ . . .

⚠️ All internal variables are intended for internal use only and are subject to change without notice. Changing these variables may yield malfunction of MpCCI, they should not be used by any external application.

MpCCI Control Variables

The MpCCI control variables are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPCCI_ARCH</td>
<td>Contains the MpCCI architecture token, which is used to identify the platform</td>
<td>2.3.1 MPCCI_ARCH - Architecture Tokens</td>
</tr>
<tr>
<td>MPCCI_DEBUG</td>
<td>If defined, MpCCI runs in debug mode</td>
<td>2.3.2 MPCCI_DEBUG - for Debugging</td>
</tr>
<tr>
<td>MPCCI_HOSTLIST_FILE</td>
<td>Path to file which contains a list of possible hosts for parallel runs</td>
<td>3.4.1 Client-Server Structure of MpCCI</td>
</tr>
<tr>
<td>MPCCI_INITIAL.Exchange</td>
<td>Passed to the code adapter to determine the coupling algorithm</td>
<td>2.3.3 MPCCI_INITIAL.Exchange</td>
</tr>
<tr>
<td>MPCCI_LICENSE_FILE</td>
<td>Location of the MpCCI license server</td>
<td>III-5.3 Defining the License Server</td>
</tr>
<tr>
<td>MPCCI_MODE</td>
<td>Selects 32 or 64 Bit mode</td>
<td></td>
</tr>
<tr>
<td>MPCCI_REMOTE_HOSTNAME</td>
<td>Name of the host seen by a remote system</td>
<td>3.4 Running MpCCI in a Network</td>
</tr>
<tr>
<td>MPCCI_RSHTYPE</td>
<td>Type of remote shell used by MpCCI</td>
<td>3.4.3 Remote Shell and Remote Copy</td>
</tr>
<tr>
<td>MPCCI_TMPDIR</td>
<td>Path to temporary directory</td>
<td>2.6 Temporary Files</td>
</tr>
</tbody>
</table>

System Variables

In addition to the control variables, MpCCI evaluates some system variables:
### General variables (all OS)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAVA_BINDIR</td>
<td>These variables are evaluated by MpCCI to find the Java installation. Please ensure that it is set correctly.</td>
<td>2.7.2 Java</td>
</tr>
<tr>
<td>JAVA_HOME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAVA_ROOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JDK_HOME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRE_HOME</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PATH</td>
<td>The list of directories which is searched for executables. &quot;MPCCI_HOME/bin&quot; must be included in the path.</td>
<td>III-2 Before the Installation</td>
</tr>
</tbody>
</table>

### Windows variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLUSERSPROFILE</td>
<td>location of All Users Profile</td>
<td></td>
</tr>
<tr>
<td>COMSPEC</td>
<td>secondary command interpreter</td>
<td></td>
</tr>
<tr>
<td>HOMEDRIVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOMEPATH</td>
<td>These variables define the home directory, which is referred to as HOME in this manual.</td>
<td></td>
</tr>
<tr>
<td>PROCESSOR_ARCHITECTURE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROCESSOR_ARCHITEW6432</td>
<td>The MpCCI architecture token is defined according to the values of these variables</td>
<td></td>
</tr>
<tr>
<td>USERDOMAIN</td>
<td>The domain of the user, needed for MPI.</td>
<td>2.7.3 MPI on page 20</td>
</tr>
<tr>
<td>USERPROFILE</td>
<td>Used to determine the home directory of a user on a remote machine.</td>
<td>2.7.4 Remote Shell and Remote Copy on page 20</td>
</tr>
</tbody>
</table>

### UNIX/Linux variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>The home directory, which is referred to as HOME in this manual.</td>
<td></td>
</tr>
</tbody>
</table>

### 2.3.1 MPCCI_ARCH- Architecture Tokens

MpCCI uses architecture tokens to identify a platform. A list of architecture tokens and currently supported platforms is given in the Release Notes.

Before the setting of this variable is really required you should get a list of all installed MpCCI platforms. The command `mpcci list archs` shows all available MpCCI platforms installed on a file server.

The variable `MPCCI_ARCH` holds the architecture token of MpCCI and is usually set by MpCCI automatically since MpCCI is capable to figure out the platform it is running on. This variable should never be set by the user.

However there may be a situation where this mechanism fails:
• MpCCI is confused to find out the platform

• MpCCI selects a wrong architecture

• MpCCI can not find the installation for your platform

**MPCCI_ARCH** must be set to a valid architecture token.

Examples:

• You are running HP-UX 11.23 with Itanium II, but only the **hpux1122_ia64** is installed and architecture **hpux1123_ia64** can not be located:

  MPCCI_ARCH=hpux1122_ia64

  solves your problems since 11.22 software can run under 11.23

• You are running under AIX 5.3, but only the **aix51_power** is installed. MpCCI will complain:

  MPCCI_ARCH=aix51_power

  solves your problems since 5.1 software can run under 5.3.

• You know you have a glibc 2.2 installed, but MpCCI thinks it is 2.3:

  MPCCI_ARCH=linux_x86_glibc22

  solves your problem.

• You have a Linux system on a Itanium processor, however **linux_ia64** is not installed:

  MPCCI_ARCH=linux_x86

  solves your problem since Itanium is 32 bit x86 compatible.

• You have a Linux system on a EM64T processor, however your Linux is not a Scientific Linux:

  MPCCI_ARCH=linux_x86

  solves your problem since EM64T is 32 bit x86 compatible.

### 2.3.2 **MPCCI_DEBUG** - for Debugging

This variable is optional, it needs not be defined.

If **MPCCI_DEBUG** is defined and its value is not false (any value except empty or 0), then detailed logging output is produced to find out some pitfalls in case of failures.

Whenever the **-debug** option appears on the command-line of the **mpcci** command

```
mpcci arg1 arg2 ... -debug ... argn
```

**MPCCI_DEBUG** will be set to 1.
2.3.3 MPCCI INITIAL EXCHANGE

This variable is required and is only used by the adapter for the application. It is set automatically within the MpCCI GUI before starting an application.

If you use the `mpcci server ...` command to start the server and invoke the applications "by hand", please do not forget to set this variable to:

```
MPCCI_INITIAL_EXCHANGE=send     or
MPCCI_INITIAL_EXCHANGE=receive   or
MPCCI_INITIAL_EXCHANGE=exchange  or
MPCCI_INITIAL_EXCHANGE=skip
```

before you start the applications.
2.4 MpCCI Project and Output Files

2.4.1 MpCCI Project Files

The MpCCI project files are named "*.csp" and contain all relevant data for a coupled computation in XML format.

2.4.2 MpCCI Server Input Files

The MpCCI server input files are named "*.cci" and generated from the project files before the MpCCI server is started.

⚠️ The contents of the server input files are subject to change without notice. Do not write or modify "*.cci" files on your own!

2.4.3 Log Files

In addition to the log files of the coupled simulation codes, MpCCI writes its own files:

"mpccirun.serverlog.*" For each coupling server, a separate log file is written. The servers are numbered, all internal calls of the code coupling interface are logged.

"mpccirun.errorlog" If errors appear in any server, the error messages are written to the error log files. Please keep in mind that errors can appear in any server or client, so the relevant error messages are often found in other log files.

"mpccirun.stdout.*" If you choose the option Create server log files in the Go Step of the MpCCI GUI (see ▶ 4.6 Go Step ◄) and switch off the option Run server processes inside xterm, the output of the MpCCI server processes will be written to “stdout” files. They contain the same information which is displayed in the terminal windows.

The stdout files contain very useful information for debugging. If you set the OutputLevel in the Edit Step to 3 (see ▶ 4.5 Edit Step ◄), all quantities which are exchanged are written to the stdout files.

2.4.4 Tracefile

The tracefile "*.ccv" is used for the MpCCI Visualizer, see ▶ 6 MpCCI Visualizer ◄.
2.5 The MpCCI Resource Directory

For storing permanent files related to your personal account MpCCI uses the subdirectory ".mpcci" within your home directory.

If there is no directory "<Home>/mpcci" MpCCI will create one whenever you start MpCCI.

"<Home>/mpcci" contains the following MpCCI related files:

- "mpcci.hosts" a hostfile listing possible remote MpCCI hosts
- "Projects" a list of your MpCCI projects for the MpCCI Project Manager
- "tmp" The temporary MpCCI directory

You should avoid to delete this directory once it was created and contains files.
2.6 Temporary Files

At runtime MpCCI needs to store intermediate information and creates temporary scripts - shell scripts under UNIX or `.bat`-files under Microsoft Windows - which wraps commands or are copied from the local host to a remote system via the remote copy commands `rcp` or `scp`. Some of the temporary files are created in the current working directory (where the MpCCI application runs or where the MpCCI server was started) and are removed after successful use, others are kept for the duration of the MpCCI session.

MpCCI maintains a list of these long term temporary files created during a session and tries to remove them at the end. Long living temporary files from the last category are stored per default in the directory "<HOME>/mpcci/tmp".

In some cases it might happen that MpCCI is killed or dies because of a signal or an exception. To eliminate all temporary files in such a case you may either delete the complete "tmp" directory or use the MpCCI command

```
mpcci clean
```

which removes all temporary files.

⚠️ Never remove temporary files as long as a simulation is running.

An MpCCI temporary directory will be created whenever you start MpCCI.

**Environment Variable MPCCI_TMPDIR: Shared HOME and Alternative Directory for Temporary Files**

Instead of using the default directory for temporary files you may want to assign a different directory to MpCCI as a temporary directory, e.g.

- "/tmp" or "/var/tmp" under UNIX
- "C:\WINDOWS\Temp" under Microsoft Windows

There are some reasons to redirect the temporary directory:

Your home directory may be physically located on a remote system. Then "<Home>/mpcci/tmp" is also located on this remote file-server.

Under UNIX this may be an NFS based directory, which is mounted automatically by the automounter (NFS is the Network File System). Under Microsoft Windows your "<Home> \.mpcci\tmp" may be either on an Microsoft Windows file-server or a UNIX file-server mounted to your Microsoft Windows PC via the Samba tool.

In the latter case accessing temporary files in "<Home>/mpcci/tmp" should be quite fast, but also NFS may sometime lead to out-of-sync delays for a few seconds. This delay may interrupt process communication if the timeout-limit has been defined with a too small value (of less than 10 seconds).
If your \texttt{MpCCI} job runs into trouble with timeouts and missing files and if you know about the fact that your home directory is on a remote host and mounted automatically, then it would be better to use a temporary directory on a locally mounted disc.

You may specify the pathname of an alternative temporary directory by setting the environment variable

\begin{verbatim}
MPCCI_TMPDIR=\texttt{path\_to\_a\_temporary\_directory}
\end{verbatim}


to a valid directory.

\textcolor{red}{⚠️} Note that you need to have write access to that directory before starting \texttt{MpCCI}.

If the environment variable \texttt{MPCCI_TMPDIR} is not defined before you start \texttt{MpCCI}, then \texttt{MpCCI} uses the default \texttt{"<Home>/.mpcci/tmp"} and sets \texttt{MPCCI_TMPDIR} automatically.

If the environment variable \texttt{MPCCI_TMPDIR} is defined before you start \texttt{MpCCI}, then the value of this variable defines the temporary files directory.
2.7 Third Party Software Used by MpCCI

2.7.1 Perl

A large part of the platform-independent functionality of MpCCI is realized with Perl scripts. Perl is available for all platforms supported by MpCCI and already included in Linux distributions.

The installation of Perl is described in III-9 Installing Perl.

For more information on Perl please visit the Perl website www.perl.org.

2.7.2 Java

The MpCCI GUI is based on Java, which is a trademark of Sun Microsystems. To run the MpCCI GUI a Java virtual machine is required.

For more information on Java installation please see III-10 Installing Java.

2.7.3 MPI

MpCCI uses MPI for parallelization. For UNIX and Linux platforms, MPI is already included in the MpCCI distribution, only for Windows a separate installation is required, which is included in the MpCCI download and is installed automatically. See also III-2.7 MPICH for Microsoft Windows.

The MPI version used by MpCCI does not conflict with native platform MPI’s or MPI’s used by other simulation codes.

2.7.4 Remote Shell and Remote Copy

MpCCI uses the remote shell rsh or secure shell ssh to start processes on remote computers. For copying files rcp or scp are used. See 3.4 Running MpCCI in a Network for a detailed description of remote computing.

On Microsoft Windows systems, no remote shell is included in the operating system, therefore the MpCCI-RSH,OpenSSH must be installed for MpCCI, see III-2.6 MpCCI-RSH for Microsoft Windows, III-2.5 OpenSSH for Microsoft Windows.
### 3 Code Coupling

MpCCI is a tool to perform multi-physics computations by coupling independent simulation codes. A general introduction to this approach is already given in \(\text{IV-1 Multi-Physics Computation with MpCCI}\). Each of the coupled codes is independent. The coupling is achieved by exchanging data in the coupling region. This procedure can be split up into two basic issues: How data is transferred and when data is transferred. The first issue is the topic of \(\text{3.2 Data Exchange}\), possible coupling algorithms, which determine the exchange time, are discussed in \(\text{3.3 Coupling Algorithms}\).

This section only gives a short overview of the methods used in MpCCI and explains the options which can be chosen. The description mainly considers a surface coupling, line and volume coupling work analogously.

#### 3.1 Multi-Physics

##### 3.1.1 Physical Domains

A physical domain is usually characterized by a set of unknowns equations which describe certain properties of a material.

The simulation code can be classified according to the physical domains, i.e. the problems which they can solve. The domains known by MpCCI are:

<table>
<thead>
<tr>
<th>Domain</th>
<th>MpCCI Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solid mechanics</td>
<td>SolidStructure</td>
</tr>
<tr>
<td>Fluid mechanics</td>
<td>Fluid</td>
</tr>
<tr>
<td>Acoustics</td>
<td>SolidAcoustics</td>
</tr>
<tr>
<td>Fluid pipe systems</td>
<td>network</td>
</tr>
<tr>
<td>Solid heat transfer</td>
<td>SolidThermal</td>
</tr>
<tr>
<td>Fluid heat transfer</td>
<td>FluidThermal</td>
</tr>
<tr>
<td>Heat radiation</td>
<td>Radiation</td>
</tr>
<tr>
<td>Electromagnetism</td>
<td>ElectroMagnetism</td>
</tr>
</tbody>
</table>

In MpCCI physical domains are rather chosen to fit typical capabilities of simulation codes than to clearly distinguish branches of physics.

**Solid Mechanics**

Solid mechanics describes the behavior of solid bodies when exposed to external load. Usually deformation, stresses and strains of structures are computed using the Finite Element Method (FEM), thus structural mechanics codes are often simply known as Finite Element codes, although the method is not limited to solid mechanics. The governing equation for solid mechanics problem are the mechanical equilibrium and Newton’s laws of motion. These are completed by material equations which describe the behavior of different materials.
**Fluid Mechanics**

Fluid mechanics deals with the behavior of fluids, e.g. flows of liquids or gases. A common term for the numerical simulation of fluid mechanics is Computation Fluid Dynamics (CFD), thus fluid mechanics codes are known as CFD-codes. The governing equations are the Navier-Stokes equations and the continuum hypothesis.

**Acoustics**

Acoustics studies sound effects, i.e. wave propagation in solids and fluids. Usually this is coupled with vibration analyses. Physically this is a subbranch of solid or fluid mechanics, but the simulation methods are different.

(The name **SolidAcoustics** in MpCCI refers to solid mechanics codes which can also compute acoustic problems).

**Solid Heat Transfer**

Heat transfer in solids is based on heat conduction. Usually boundary temperatures and heat sources are given and the heat distribution shall be computed. The governing law of heat conduction can usually be solved by solid mechanics codes. Heat transfer via radiation is discussed separately below.

**Fluid Heat Transfer**

Heat transfer in fluids differs from that in solids as heat is transferred not only by heat conduction but also by convection. Most fluid codes can also compute heat transfer.

**Heat Radiation**

Heat transfer by radiation is separated from solid and fluid heat transfer because specialized simulation codes exist for this issue. They compute the heat distribution caused by electromagnetic radiation (e.g. infrared light), which is emitted by bodies or fluids.

**Electromagnetism**

Electromagnetism deals with the computation of electric and magnetic fields. This includes the computation of electric currents and resulting forces. The governing equation are Maxwell’s equations.

### 3.1.2 Coupling Types

Among the many possible combinations of physical domains, there are some pairs which are often used in multi-physics simulations together with a typical set of quantities which are exchanged. These combinations of domains and quantities are called *coupling types* in MpCCI.

The MpCCI GUI offers some predefined coupling types to select, see ▷4.4 Coupling Step◁. Some typical coupling types will be described in the following.
Fluid-Structure Interaction (FSI)

In an FSI simulation, usually a structure deforms due to forces caused by a fluid flow while the deformation changes the fluids boundary. The deformation must be transferred to the CFD code, which corresponds to the quantities “Nodal displacement” (NDisplacement) or “Nodal position” (NPosition), while forces are sent from CFD to the structural code, e.g. “Boundary absolute force vector” (WallForce), “Boundary relative force vector” (RelWallForce) or the “Relative pressure” (OverPressure).

CFD codes usually use a reference pressure, i.e. the atmospheric pressure, which is not considered in structural codes. For a coupled simulation it is therefore recommended to transfer the relative pressure only, i.e. RelWallForce or Overpressure instead of WallForce and AbsPressure which include the reference pressure. Please consult the appropriate section of the Codes Manual for more information on the reference pressure in a specific simulation code.

For Fluid-Structure Interaction there are two predefined coupling types, Absolute pressure based Fluid-Structure Interaction and Gauge pressure based Fluid-Structure Interaction. In some cases a one-way coupling is sufficient, which corresponds to the types One way force mapping and One way pressure mapping.

FSI is used in the following tutorials:
▷ VII-2 Vortex-Induced Vibration of a Thin-Walled Structure
▷ VII-3 Elastic Flap in a Duct
▷ VII-6 Pipe Nozzle

Thermal Coupling

Thermal coupling is applied if heat transfer is to be computed in as system consisting of a structure and a fluid. At the boundary, i.e. the surface of the structure, both share the same temperature and thermal energy is exchanged as heat flux.

There are two basic combinations of quantities which can be exchanged:

- Exchange of temperature (Temperature or WallTemp) and the heat flux (WallHeatFlux). The temperature is typically sent by the structural (or radiation) code, while the fluid code is sender of the heat flux. This corresponds to the predefined coupling type Transient surface heat transfer.

- Exchange of temperature, film temperature (FilmTemp) and the heat coefficient (WallHTCoeff). As above, the wall temperature is sent by the structural code. The fluid code sends the film temperature and wall heat transfer coefficient, a combination which can be directly applied as boundary condition by most structural codes. The heat transfer coefficient $h$ is computed from the heat flux $q$ and the difference between the wall temperature $T_w$ and the adjacent fluid cell temperature $T_c$ in the adapter of the fluid code:

$$h = \frac{q}{T_w - T_c}$$

The coupling types Steady state surface heat transfer and Steady state radiative heat transfer correspond to this procedure.
The second approach is recommended because it is more stable. Thermal coupling is used in the tutorial VII-4 Exhaust Manifold.

**Electrothermal Analysis**

The resistive loss of electric currents corresponds to a heat source while the temperature influences the resistivity of conductors. This problem is addressed with an electrothermal analysis, which combines electromagnetism with heat transfer. The quantities which are actually should be exchanged are the electric resistivity and the temperature. However, most heat transfer codes cannot compute the resistivity directly, which is therefore done using user-defined functions. The quantity combination depends on the decision in which code these additional computations are carried through.

A special application in this area is electric arc computation. A simple example is shown in tutorial VII-5 Busbar System.

### 3.2 Data Exchange

![](data_exchange.png)

Figure 1: Data exchange between two non-matching grids (distance exaggerated)

Data exchange does not mean that values are shared, as would be the case in a monolithic code, but each quantity is determined by one code, which then sends it to the other. In the sending code, the data is defined on a mesh of some kind and shall be transferred to the mesh of the receiving code. These meshes describe the same geometric entity, but typically differ in element size and node location, which is referred to as “nonmatching grids”, see Figure 1.

The exchange procedure can be split up into three steps:

**Pre-Contact Search** Before the actual contact search the elements of the meshes are sorted in a way to speed up the association process.
**Association** For each node or element of one grid, the “partners” on the other grid must be found. The data will then be exchanged between associated nodes and/or elements. This process is also called neighborhood search.

**Interpolation** The data which shall be transferred must be adapted to the target mesh, while e.g. conserving the sum of forces.

There are two basic approaches for association and interpolation, the *Minimal Distance* algorithm and the *Intersection* algorithm. Both are discussed in the following sections. All parameters for association and interpolation can be set in the Edit Panel of the MpCCI GUI, see >4.5 Edit Step<.

### 3.2.1 Pre-Contact Search

![Figure 2: Pre-contact search: a) Decomposition of a quadrilateral element into two triangles. b) Bounding boxes of the first triangle for values 1.0 and 1.5 of BboxExpansion. c) Buckets, to which the triangle is associated.](image)

To simplify the contact search, all elements are decomposed into triangles for two-dimensional meshes or tetrahedra for three-dimensional meshes. Based on this decomposition, the same search algorithms can be applied for all element types.

The search for elements can be carried through with different algorithms, which can be selected in the Edit Panel. The default algorithm is a bucket algorithm, which is the only selection which is recommended, all other algorithms will be removed in future releases.

In the pre-contact search, each triangle is surrounded by a bounding box. This bounding box fully contains the triangle and has equal lengths in all space dimensions. The size of the bounding box can be changed by setting the parameter BboxExpansion in the Edit Panel. A value of 1.0 corresponds to the smallest cube containing the triangle. The length of the cube is multiplied with BboxExpansion.
The space which covers the elements is split up into small squares or cubes, the so-called *buckets*. Each bucket has a list of triangles, which are close to it. The bounding box of a triangle lies within a number of buckets, as depicted in Figure 2 c). The triangle is added to the lists of all these buckets.

The size of the buckets is chosen automatically, but it can also be altered by changing the parameter `BucketExpansion`. It is recommended to keep it set to 1.0. Setting a minimum bucket size (`MinBboxSize`) is also not necessary.

### 3.2.2 Minimal Distance

The minimal distance algorithm is based on point-element relationships, it is therefore also called “PE” for “point-element”.

Each node of one mesh lies within a bucket of the other meshes. After the pre-contact search, the bucket contains a list of triangles. These triangles belong to elements, which are regarded as candidates for a contact.

![Diagram](image)

**Figure 3:** Selecting an element: a) Barycentric distance $d_b$ around an element. b) vertical distance $d_n$ and in-plane distance $d_t$

From this set of elements, the best element must be selected. This is achieved by finding the best triangle and then selecting the corresponding element. This process is based on the relative position of the triangle and the node P. First, the projection P’ of the point onto the surface of each triangle is computed, as can be seen in Figure 3 b).

Three distances are evaluated to determine the best element. The first is the barycentric distance of the point P’

$$d_b = |u| + |v| + |w| - 1$$  \hspace{1cm} (1)

with the barycentric coordinates $u$, $v$ and $w$. The value of $d_b$ is zero within the element, but as soon as the point lies outside the element, the value increases as depicted in Figure 3 a).
The second value is the distance $d_t$ of $P'$ to the nearest point of the triangle and the third the distance of
$P$ to the plane, i.e. the distance between $P$ and $P'$.

The final measure to select an element is

$$\tau = \theta_1 d_b + \theta_2 \frac{d_t^2}{A} + \theta_3 \frac{d_n^2}{A},$$

where $A$ is the area of the triangle. The weights $\theta_1$, $\theta_2$ and $\theta_3$, which can be set in the Edit Panel, determine
which distances are regarded. The element with the smallest value of $\tau$ is selected. If $\tau$ exceeds a certain
value, which can be set by the parameter Rejection, the element is not selected, which may mean that no
element is found at all, see $\triangleright$3.2.4 Orphaned Nodes and Elements $\triangleleft$.

The interpolation of values is then achieved by evaluating the shape functions of the element to find the
value at the point $P'$, which is then transferred to the other mesh, i.e. the node $P$.

$i$ The minimal distance algorithm is not recommended if flux-values are transferred from coarse to fine
meshes. In this case you may obtain local peaks of quantities. Please use the Intersection method
instead.

**InsideOnly Option**

If a point is matched to an element as shown in Figure 3 b), it may receive data from its projection $P'$. This may be a problem if $P'$ lies outside of the element, because the value is interpolated using the element shape functions. Even if all nodal values of the element are positive, one may get a negative value outside the element, which is then transferred to the partner code.

For certain quantities, e.g. temperatures, negative values are not allowed. If the InsideOnly option is enabled, $P$ receives its data not from $P'$ but from the closest point within the element.

$!$ Do not use InsideOnly to transfer the node coordinates NPosition! Because in this case a node $P$ would
not be moved onto the surface ($P'$), but inside the element, which may yield an invalid mesh.

### 3.2.3 Intersection

As an alternative to the Minimal Distance algorithm, MpCCI offers an algorithm which is based on an
element-element association, also referred to as “EE” for “element-element”.

Instead of searching a partner for a node or point, partners for an element are searched. Again all elements
are decomposed into triangles. To find the partners for one triangle, the bounding box is constructed, this
time without additional expansion. The bounding box intersects the bucket space of the other model. If
the intersecting buckets contain elements from the pre-contact search they are combined to form a list of
contact candidates.

All candidate triangles are now projected onto the plane defined by the searching triangle as depicted in
Figure 4 a). For each triangle, the area of the common area with the searching triangle is computed and
Figure 4: Intersection algorithm: a) Projection of triangles, one triangle does not intersect the target triangle (distance exaggerated). b) Intersection areas

For each element a weight $w_i$ is defined by

$$w_i = \frac{A_i}{\sum A_i}$$  \hspace{1cm} (3)

where $A_i$ are the intersection areas, the sum of which corresponds to the area of the searching triangle. Note that the sum of all weights is 1.

These weights are finally used to interpolate the data, i.e. the value for the target element is a function of the values of the source elements and the corresponding weights.

Figure 5: Intersection algorithm: a) Projection distance and projection angle b) Overlapping matching areas

Not all elements are automatically considered to match an element on the other mesh. The projection
distance and projection angle as sketched in Figure 5 a) can be limited by options in the Edit Step. Elements with too large angles or distances remain orphaned.

In addition there is a rejection parameter. Elements which share only a very small common area are not considered. See ▷ 4.5 Edit Step ◁ for how to choose the minimal area fraction for matching elements.

For the intersection algorithm, an additional check may be chosen by the parameter PerformTest. If the projection of elements overlap as shown in Figure 5 b), or some areas do not obtain any projection areas, warnings are printed to the log file if PerformTest is activated.

### 3.2.4 Orphaned Nodes and Elements

A node is called “orphaned” if it is not associated to an element during the contact search. This also means that this node can neither send nor receive any data from the other mesh. Instead, MpCCI sends the default values, which can be defined for each quantity (see ▷ 4.4 Coupling Step ◁).

Besides assigning default values it is possible to extrapolate values from non-orphaned areas. The extrapolated values of the orphans will lie in the range of the lowest and highest values of the bordering non-orphaned areas.

In many cases, however, orphaned nodes can yield severe problems, e.g. if nodal displacements are transferred, orphaned nodes are not moved along with the rest of the mesh. This can yield severe distortions, which lead to an abortion of the computation even if extrapolation is turned on.

If orphaned nodes are present, they are listed in the output of the coupling servers:

```
FE:1:0: The nodes with the following ids did not find a match
(local_number[global_number]):
FE:1:0: 8[7]
FE:1:0: 90[89]
FE:1:0: 92[91]
FE:1:0: 277[276]
FE:1:0: 278[277]
FE:1:0: 279[278]
```

To identify the regions where orphaned nodes appear, it is recommended to use the MpCCI Visualizer, which can display all orphaned nodes in the coupling region, see ▷ 6 MpCCI Visualizer ◁.

For element-element relations as used in the intersection interpolation, there are also orphaned elements for which no corresponding elements could be found in the other mesh.

### 3.2.5 Flux and Field Interpolation

The interpolation of quantities requires the distinction of two different cases, which can be characterized as flux and field interpolation.
In flux interpolation, the value is adapted to the element sizes to preserve the integral. Flux interpolation is e.g. used for forces.

In field interpolation, the values are kept to ensure a conservative transfer. This is used e.g. for are pressures or densities.

In MpCCI a third case, flux density, is considered. This is a different term for field values, thus the interpolation is carried through in the same way.

The interpolation type is already defined for each quantity, see the quantities list in the Appendix.

### 3.3 Coupling Algorithms

#### 3.3.1 Course of the Coupling Process

The coupling process consists of three main phases:

**Initialization** The codes initialize their data and MpCCI is initialized. MpCCI establishes a connection between the codes and the association or neighborhood search is carried through.

**Iteration** Each code computes its part of the problem and data is exchanged at certain moments.

**Finalization** The computation is ended by disconnecting the codes and stopping all codes and MpCCI.

During the iteration, which also consists of several time steps in a transient problem, the data is exchanged several times. MpCCI cannot control the simulation processes of the coupled codes. Therefore, it is important that whenever one side sends data, the other side should be ready to receive. The data is also not associated with certain time steps or iterations.

A consequence of this is also that the possible coupling algorithms are mainly determined by the capabilities of the simulation codes and the corresponding code adapters. The code adapters call send- and/or receive-functions. These calls can appear at different states of the computation:

- At the beginning of each time step, i.e. before the iteration of the time step.
- At the end of each time step, i.e. after the iteration.
- Before or after an iteration step.
- On direct demand of the user.

Most codes only offer exchanges at the beginning or end of a time step, which limits the choice of coupling algorithms. Exchange on direct demand usually is only used in stationary computations. Most simulation codes do not support exchanges during iterations.

Please check the Codes Manual for information on the supported exchange times of the codes.

MpCCI only supports weak coupling, as strong coupling can only be realized in two ways:
• The equations of each domain of the coupled systems are combined in one large system of equations, which is solved to obtain the solution. This contradicts the approach of MpCCI, which couples two codes, each of which is highly specialized in its field.

• Each system is solved separately, but data is exchanged during each iteration step, i.e. both codes compute one time step (the only one in a stationary computation) at the same time. If one code has finished a step of the iteration, it sends its data to the other code, which uses it in its own iteration. Both iterations are continued until both converge, yielding a state which fulfills the equation of both physical domains. The only disadvantage in comparison to weak coupling is that convergence is slower, thus the computations are slower, but yield more precise results.

The latter approach is aspired in future versions of MpCCI, which can only be realized if codes support exchanges during iterations. In MpCCI the selection of a coupling algorithm is based on the selection of the initial exchange. There are four choices, which are defined in the Go Step of the MpCCI GUI (see §4.6 Go Step): exchange, receive, send and skip. The setting of this option defines what happens if the exchange routine is called for the first time. All further calls of the exchange routine will yield a complete exchange, i.e. sending and receiving of data.

⚠️ It is important to keep in mind, that sending of data is always possible: The data is stored temporarily by MpCCI, i.e. it can be received later by the other code. MpCCI can store several sets of data, which are then send to the other code in the order in which they were sent.

If one code is receiving data, it waits until data is available from the other code.

### 3.3.2 Stationary Problems

For stationary problems, it is assumed that there is exactly one solution of the coupled problem, which shall be found. The coupling algorithm does not have a big influence on the solution in this case.

### 3.3.3 Transient Problems

Two general cases can be distinguished: Either both codes send and receive data, or one code only sends while the other receives only.

In any case it is important to know whether each code performs the data exchange at the beginning of the time step before the iteration, which we call “exchange before solution”, or at the end of a time step after the iteration, “exchange after solution”.

#### 3.3.3.1 Construction of Coupling algorithms

Coupling algorithms can be constructed in a straight-forward way. The three basic steps are shown in Figure 6. It is important to check first, whether the codes exchange before or after solution. In this example, code A exchanges after the solution, code B before the solution.
Figure 6: Construction of a coupling algorithm: Code A exchanges after solution, while code B exchanges before solution.

The construction of the algorithm works as follows:

1. Draw time lines for both codes, including boxes which indicate the data exchanges. For code A, the box is at the end of the steps, for code B always at the beginning. The first box in each code represents the initial transfer, which can be chosen in the MpCCI GUI. We assume, that for A exchange (marked by “x”) and for B receive (marked by “r”) were selected. All further exchanges always get an “x” as data is received and sent in all further steps.

2. Now, the data transfers are constructed, we proceed from the left to the right. Find the first sending operation in code A, which is found in the first box (x = send and receive). Thus a transfer line starts here. Now find the first receiving operation in code B, which is also the first box. This is the end of the line, which yields the first transfer.

Now we proceed with the next sending operation, which is found in the second box in code B, which is connected to the first box of code A.

In this way all arrows can be found.
3. The only thing missing now is the order of operations. For this we remember:

- “receive” always waits for data
- “send” never waits (data is buffered)
- exchanges “x” can be carried through in any order: Send first or receive first, as necessary.

In our example, this means code A can start the computation (1), as it does not need to wait, while code B receives data, thus it will wait. After code A has finished computing the first time step, it can send data (2) and then waits to receive data. Now, code B got its data and can start computing (3), then sends data to code A (4), which can then proceed (5), sends data (6) and code B continues (7).

In this way we obtain the final algorithm depicted in Figure 6, which can be identified as a sequential coupling algorithm. Note that not all combinations of the initial transfer yield reasonable coupling algorithms.

### 3.3.3.2 Standard Case: Bidirectional Transfer

In most coupled simulations, data is transferred between the codes in both directions. The coupling algorithms depend on the order of solution and transfer and the selected initial transfer.

There is only a limited number of reasonable settings. Three cases can be regarded:

- Both codes exchange before iteration. The possible combinations are sketched in Figure 7 a).
- Both codes exchange after the iteration. The only difference to the previous case is that no data is transferred immediately before the first solution step. Otherwise the algorithms remain the same. One example is shown in Figure 7 b).
- One code exchanges after the iteration, and one before the iteration, which yields a wide variety of algorithms. However, only few can actually be recommended for use. To add further algorithms, one can allow a “dummy step” in the code with exchange after solution, i.e. the “clocks” of both codes are not synchronous. One code is always one time step ahead. Possible algorithms are depicted in Figure 8.
a) Both codes exchange before iteration

1. A: receive
2. 3 4 5 6
3. 7 8
4. 9
5. serial coupling
6. B: exchange
7. 1 2
8. 4
9. 5
10. parallel coupling
11. A: exchange
12. 2
13. 4
14. B: exchange
15. 2
16. 4
17. parallel coupling
18. A: receive
19. 2
20. 4
21. B: send
22. 2
23. 4
24. parallel coupling

b) Both codes exchange after iteration

1. A: receive
2. 1 3
3. 7
4. 10
5. serial coupling
6. B: exchange
7. 1 5
8. 9

Figure 7: Coupling algorithms for different combinations of Initial quantities transfer. The difference between cases a) and b) is only the additional step of both codes before the initial exchange in case b).
Figure 8: Coupling algorithms for code A with exchange after solution and code B with exchange before solution. The algorithms at the top on the right are obtained if a dummy step is introduced.
### 3.3.3.3 Special Case: Unidirectional Transfer

In some coupling applications, only unidirectional transfer is used, i.e. one code only sends data to the other code, which only receives. This reduces the possible coupling algorithms, there is only one basic coupling algorithm which should be used if possible. If A is the sending code and B the receiving code, this algorithm can be described by the algorithm in Figure 9.

![Figure 9: Coupling algorithm for unidirectional transfer](image)

Depending on the coupling properties of the codes, the user should select the following settings:

<table>
<thead>
<tr>
<th>Exchange before/after solution</th>
<th>Initial Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 A: before, B: before</td>
<td>A: skip, B: receive</td>
</tr>
<tr>
<td>2 A: before, B: after</td>
<td>A: skip, B: skip with dummy step</td>
</tr>
<tr>
<td>3 A: after, B: before</td>
<td>A: send, B: receive</td>
</tr>
<tr>
<td>4 A: after, B: after</td>
<td>A: skip, B: receive</td>
</tr>
</tbody>
</table>

In case 2, the coupling algorithm can only be realized with a dummy step in code B. The best combinations are 3 and 4.

### 3.3.4 Exchange of Time Step Size

Instead of using a fixed coupling time step size, it is also possible to use adaptive time stepping. In this case the time step size is determined by one code and sent to the partner code, which changes its own time step to the received value. MpCCI regards the time step as a global quantity without any special meaning, its name is “PhysicalTime” (see quantity reference in the Appendix).

The exchange of time step sizes may yield unwanted effects:

**Time Shift:** It depends on when exactly each code determines the time step size and sends it to the partner code. It may happen that the partner code uses the time step size of the previous step. This problem cannot always be solved, try to use a different coupling algorithm.

**Convergence Problems:** If the time step size is determined by one code it may be simply too large for the partner code. Please choose carefully which code shall determine the size and set reasonable limits.

⚠️ It is strongly recommended to always define a default time step size in each simulation code, because the value of “PhysicalTime” may not be defined at the beginning of a simulation! This depends on the initial exchange settings and how exactly each code evaluates the received value. The default value should be the same for both partner codes.
3.3.5 Subcycling

The idea of subcycling is the definition of extra iteration steps inside a coupling step, which is schematically depicted in Figure 10. The ability of subcycling depends on the respective applied software. There are limited possibilities to implement subcycling by configuring MpCCI and the employed solvers. The motivation for subcycling may be the implementation of optimal settings for each solver concerning computational costs, stability and accuracy. For a transient simulation one solver could require smaller time steps for an accurate and/or stable solution process. Forcing the other code to apply the rather small time step might yield an inefficient process. Furthermore, in steady-state calculations for thermal problems, heat propagation in fluids is subject to much faster transportation processes than in solid bodies, which will call for subcycling procedures in the fluid domain.

![Diagram of coupling algorithms with subcycling](image)

Figure 10: Three examples for coupling algorithms with subcycling: a) constant subcycling steps $\Delta t_s$ and exact coupling steps sizes of both codes, b) not necessarily constant subcycling steps $\Delta t_s$ and exact coupling step sizes enforced, c) not necessarily constant subcycling steps $\Delta t_s$ and exact coupling step sizes not enforced.

A tutorial where subcycling is applied for a thermal simulation is → VII-4 Exhaust Manifold ←.
### 3.3.6 Restarting a Coupled Simulation

One should distinguish between an actual restart and a complex start.

**Restart** means the simulation is interrupted by killing the simulation or reaching a target time. Then the simulation is restarted seamlessly, i.e. continued as if it had never been interrupted.

**Complex Start** means that the coupled simulation consists of different analysis steps, which differ in e.g. boundary conditions, simulation algorithms (transient/steady state). The second step is then based on the results of the first one.

When restarting a coupled simulation, this consists of three parts. Each of the simulation codes must be restarted separately – the same way it is done without coupling – and MpCCI must be restarted. For MpCCI it can be chosen to either save the neighborhood information into a restart file by setting appropriate options in the Edit step under “Restart file”. The parameter “name” defines the name of the file, and “Access” must be set to “write” in the first run, see also ▷4.5 Edit Step◁.

If the neighborhood information for MpCCI is read from a restart file, the same neighborhood is used as in the first run, which is recommended. If this is not done, the new neighborhood search is performed on deformed meshes and may yield different results in comparison to the first search, i.e. the restart may not be really seamless.

Restarting a coupled simulation is an advanced analysis procedure. It is difficult to choose the initial exchanges such that the time in both codes is still synchronous after a restart. This is also code-dependent, there is information on restarts for some codes in the Codes Manual.
3.4 Running MpCCI in a Network

The use of MpCCI in a network is almost unlimited. MpCCI can couple couple simulation codes which run on different computers with different operation systems. It is also possible to run one or both of the simulation codes in parallel and to start the coupled simulation on a cluster using a queuing system.

3.4.1 Client-Server Structure of MpCCI

a) Coupling without parallel execution

![Diagram of client-server structure without parallel execution]

b) Coupling with parallel execution of simulation codes

![Diagram of client-server structure with parallel execution]

Figure 11: Client-server structure of MpCCI
The code coupling of MpCCI is based on a client-server model: The MpCCI processes act as servers while the simulation codes – more precisely the code adapters of the simulation codes – act as clients.

Additionally an optional control process can be started to write a tracefile.

If codes are coupled without parallel execution, one server is started for communication with each simulation code, see Figure 11 a).

If a simulation code is running in parallel, there are two possible ways of communicating:

- All data which is exchanged with the simulation code is passed via a main process, MpCCI only starts one server for this code. This is depicted in Figure 11 b) on the left.
- Each subprocess of a simulation code communicates with one MpCCI server, i.e. MpCCI has to start several servers as shown in Figure 11 b) on the right.

**Distributing MpCCI Servers on Different Machines**

The MpCCI servers use MPI for communication (which is also socket-based). They can be run on different machines which use the same endianness. (The “endianness” is the order in which bytes are stored or transmitted, which depends on the processor.)

You can check the architecture by running `mpcci arch`. If the computers you want to use have the same architecture tokens, they also have the same endianness.

The hosts for the MpCCI servers can be chosen in the Go Step of the MpCCI GUI. Select the option Distribute server on remote hosts and enter the names of remote computers (with domain name if necessary) into the field Optional 'host host ...' to be used. See >4.6 Go Step for details.

**Distributing Simulation Codes on Different Machines**

The communication between the simulation code clients and MpCCI servers is purely socket-based.

If a simulation code shall be started on a remote computer, the input file must be placed there and selected in the remote file browser (>4.7 Remote File Browser). If the code shall additionally be started in parallel, please select the appropriate options in the Go Step, as described in the Codes Manual.

**Port Numbers for Socket Communication**

For a coupled simulation MpCCI uses a sequence of port numbers for the different socket connections. This sequence is defined by the basic port number, which can be selected in the MpCCI GUI (>4.6.1 Configuring the MpCCI Coupling Server) or when starting the server with `mpcci server` (>5.6.12 mpcci server).

In the example sketched in Figure 11 a) one port is used as basic port and one for each connection to a client:

- 47111 Basic port
- 47112 Connection to simulation code A
- 47113 Connection to simulation code B

For the parallel run of Figure 11 b) a total of 5 ports are needed:
47111 Basic port
47112 Connection to main process of simulation code A
47113 Connection to process 1 of simulation code B
47114 Connection to process 2 of simulation code B
47115 Connection to process 3 of simulation code B

⚠️ Once all connections are established, the ports may be re-used, even if a simulation is still running. Only if you want to start several coupled simulations at the same time, you should use different sets of ports.

If you need to change the ports due to firewall restrictions, please ensure that a sufficient number of subsequent ports can be accessed.

### 3.4.2 Hostlist files

If you run coupled simulation on different computers frequently, it is recommended to use a hostlist file, which contains a list of computers. Each line in the list corresponds to one remote machine. There are two possible line formats, the standard format:

```
[user@]hostname [#rsh] [#ssh] [#home:path] [#arch:token]
```

or extended (".rhost" type) format:

```
hostname [user user user] [#rsh] [#ssh] [#home:path] [#arch:token]
```

Lines starting with a `#` are treated as comments. The entries `#rsh` and `#ssh` are used to select the network communication method (see ▶ 3.4.3 Remote Shell and Remote Copy below), the `#home:` option is used to specify the path to the home directory (only needed if not standard directory) and the `#arch:` option can be used to explicitly choose an architecture, see ▶ 2.3.1 MPCCI_ARCH - Architecture Tokens for details.

Thus a hostlist file might look like:

```
fred@jupiter #ssh #home:/u/fred
saturn.example.com fred barney #linux_x86
# this line is a comment
mars
```

The standard MpCCI hostlist file is "\(<Home>/mpcci/mpcci.hosts\)", which is located in the MpCCI Resource Directory (▶ 2.5 The MpCCI Resource Directory). If you want to use a different default hostlist file, you can set the environment variable `MPCCI_HOSTLIST_FILE` to the path of the file.

You may list the contents of the hostlist file with `mpcci list -hosts`. 
3.4.3 Remote Shell and Remote Copy

To start processes on remote machines and copy files, \texttt{MpCCI} can use either the \texttt{rsh} or the \texttt{ssh} family of commands. The environment variable \texttt{MPCCI\_RSHTYPE} defines which family of remote commands is used:

<table>
<thead>
<tr>
<th>Remote Shell</th>
<th>File Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{rsh}</td>
<td>ftp or \texttt{rcp}</td>
</tr>
<tr>
<td>\texttt{ssh}</td>
<td>scp or \texttt{sftp}</td>
</tr>
</tbody>
</table>

If \texttt{MPCCI\_RSHTYPE} is not defined or not set the default is \texttt{rsh} under UNIX and Linux and \texttt{ssh} under Microsoft Windows.

It depends on your network configuration which method should be used.

Please also read \ref{III-6.1 Accessing Remote Hosts} for testing the remote access facilities.

3.4.4 Coupled Analysis in Batch Mode

3.4.4.1 General Approach

To start a batch job with \texttt{MpCCI}, i.e. a simulation without graphical interface, a project file "*.csp" must already be present. The following procedure is recommended:

1. Set up a coupled simulation on a computer with graphical interface. Proceed up to the Go Step in the \texttt{MpCCI} GUI and save the project file.

   If a code has to run in parallel, you have to enable the parallel option as described in the \texttt{Codes Manual}

2. Start the coupled analysis with \texttt{mpcci -batch }\texttt{<project file>} from the \texttt{MpCCI} project directory.

The \texttt{mpcci -batch} command will start the simulation and distributes the codes as configured in the project file.

3.4.4.2 Job Scheduler Environment

To run a coupled computation with a job scheduler (queuing system) on a computing cluster \texttt{MpCCI} provides a set of command to submit, status, kill a job to/from a job scheduler.

\texttt{MpCCI} supports batch-systems like LSF, PBS, SGE, LoadLeveler, GLOBUS.

When preparing the project files, you cannot already know on which nodes of a cluster your processes will be started. \texttt{MpCCI} automatically detects that it is started by a job scheduler and changes the entries in the project files accordingly, i.e. the jobs are spread onto the reserved nodes by \texttt{MpCCI}. \texttt{MpCCI} will create a new project file containing the new associated hosts for each code. This project file name adds the prefix "\texttt{batch}_" to the original project file name.
The following procedure is recommended in order to prepare the project file:

1. Start the MpCCI GUI on a computer with graphical interface.

2. Set up the coupled analysis as a local job. This means the model files are selected on the local file system.

3. Proceed up to the Go Step in the MpCCI GUI and save the project file.

- To distribute the server on remote hosts (see 4.6.1 Configuring the MpCCI Coupling Server), you have to enable the option and may let the host list information empty.
- To configure a code to run in parallel, you have to enable the parallel option as described in the Codes Manual for the code and specify the number of processes.

Until this step you have prepared your project file.

In case that the computing cluster has a different file system than the workstation where you have setup the project file, it is recommend to copy the project directory including all input files for the simulation codes and the MpCCI project file "*.csp" to the compute head node.

Usually also the MpCCI servers are run on different cluster nodes, thus you should consider the number of required MpCCI server processes when determining the number of necessary nodes for the job resource allocation. The MpCCI GUI may assist you to submit the job in the queue and configure the host allocation for each code if necessary or you can submit the job by using the MpCCI command line.

Submit a job with MpCCI GUI

To submit the MpCCI job with the MpCCI GUI you have to start the MpCCI GUI on the head node of your cluster. Usually the batch queue commands are available on the compute head node.

You have to open your project and select the in the Batch→Submit in the menu (which is enable in the Go Step). A dialog will pop up and you are able to configure the batch options.

By clicking on the submit button MpCCI prepares for the current project a script having the project name ("<project name>.[sh|bat]") to be submitted to the batch system. This generated script contains all the options provided by the MpCCI GUI for the selected batch system and the call of mpcci -batch <project file>.

Below follows the description of the configuration dialog for each batch system.
Options Details for a LSF system

![Submit Batch Job Window](image)

**Figure 12: LSF Options**

**Job name**  you can provide a job name for the coupled analysis. This setting represents the option `-J` of the submit command.

**Queue name**  you can provide a queue name to submit the job. This setting represents the option `-q` of the submit command.

**Name for the stdout output file**  you can provide a filename for the stdout collected by the batch system. This setting represents the option `-o` of the submit command.

**Name for the stderr output file**  you can provide a filename for the stderr collected by the batch system. This setting represents the option `-e` of the submit command.
**Number of host/processor** you can provide the number of host/processor to be reserved for the job. This setting represents the option -n of the submit command.

**Host selection (option -m)** you can provide a set of candidate hosts for running the coupled analysis. This setting represents the option -m of the submit command.

**Resource requirements (option -R)** you can provide the resource requirement setting for the coupled analysis. This setting represents the option -R of the submit command.

**Additional command options** you can provide a list of submit options that are not listed here. These options will be passed to the submit command.
Options Details for a PBS system

![OpenPBS Options](image)

The options are identical for the PBS, PBSPro, OpenPBS, Torque batch systems.

**Job name**  you can provide a job name for the coupled analysis. This setting represents the option -N of the submit command.

**Queue name**  you can provide a queue name to submit the job. This setting represents the option -q of the submit command.

**Name for the stdout output file**  you can provide a filename for the stdout collected by the batch system. This setting represents the option -o of the submit command.
**Name for the stderr output file**  you can provide a filename for the stderr collected by the batch system. This setting represents the option `-e` of the submit command.

**Join both standard output and standard error**  select this option to merge the standard output and standard error. This setting represents the option `-j oe` of the submit command.

**Keep both standard output and standard error**  select this option to keep both standard output and standard error. This setting represents the option `-k oe` of the submit command.

**Defines the resources that are required by the job (option: `-l`)**  you can define a resource requirement for running the coupled analysis. This setting represents the option `-l` of the submit command.

**Additional command options**  you can provide a list of submit options that are not listed here. These options will be passed to the submit command.
Options Details for a SGE system

The options are identical for the SGE, N1GE, SGEEE batch systems.

**Job name**  you can provide a job name for the coupled analysis. This setting represents the option -J of the submit command.

**Queue name**  you can provide a queue name to submit the job. This setting represents the option -q of the submit command.

**Name for the stdout output file**  you can provide a filename for the stdout collected by the batch system. This setting represents the option -o of the submit command.
Name for the stderr output file  you can provide a filename for the stderr collected by the batch system. This setting represents the option `-e` of the submit command.

Change to the working directory of call  select this option in order to request the batch system to change to the working directory of the submit call. This setting represents the option `-cwd` of the submit command.

Resource requirements (option `-l`) you can provide the resource requirement setting for the coupled analysis. This setting represents the option `-l` of the submit command.

Define the parallel environment option (option: `-pe`) you can provide the parameter for the parallel environment. This setting represents the option `-pe` of the submit command.

Additional command options  you can provide a list of submit options that are not listed here. These options will be passed to the submit command.
Options Details for a **LoadLeveler** system

![LoadLeveler Options](image)

**Figure 15: LoadLeveler Options**

**Job name**  you can provide a job name for the coupled analysis. This setting represents the option `job_name` of the submit command.

**Job type**  you can specify the type of the job by selecting the option `pvm3`, `parallel`, `serial`. This setting represents the option `job_type` of the submit command.

**Name for the stdout output file**  you can provide a filename for the stdout collected by the batch system. This setting represents the option `output` of the submit command.

**Name for the stderr output file**  you can provide a filename for the stderr collected by the batch system. This setting represents the option `error` of the submit command.
**Number of nodes [min,max]** you can provide the number of nodes to be reserved for the job by specifying an interval [min,max]. This setting represents the option `node` of the submit command.
Options Details for a GLOBUS system

![GLOBUS Options](image)

**Hostname to submit the job**  you can provide the hostname of the GLOBUS factory for job submission. This setting represents the option `-F` of the submit command.

**Select the batch system to use**  you can select the type of scheduler to be used by GLOBUS. This setting represents the option `-Ft` of the submit command.

**Name for the stdout output file**  you can provide a filename for the stdout collected by the batch system. This setting represents the option `stdout` of the submit command.

**Name for the stderr output file**  you can provide a filename for the stderr collected by the batch system. This setting represents the option `stderr` of the submit command.
**Number of nodes**  you can provide the number of nodes to be reserved for the job. This setting represents the option `count` of the submit command.

**Job type**  you can provide the type of the job. This setting represents the option `jobtype` of the submit command.

**Queue name**  you can provide the queue name to submit the coupled analysis job. This setting represents the option `queue` of the submit command.

**Additional command options**  you can provide a list of submit options that are not listed here. These options will be passed to the submit command.
Submit a job with MpCCI command line

The syntax of the command line is the following:

```
MpCCI-batch <BATCH_NAME> submit [BATCH-OPTIONS] <project file>
```

with the arguments meaning:

- `<BATCH_NAME>` represents the name of the batch system e.g. PBS, SGE, etc.
- `[BATCH-OPTIONS]` represents a list of the original options of the submit command.
- `<project file>` is the project file or a shell script.

⚠️ If the `<project file>` is a shell script this will be simply passed to the submit command.

By using the MpCCI batch command line you have to define yourself the host allocation for each code. You can pass as `[BATCH-OPTIONS]` argument the host allocation definition. The variables:

- `MPCCI_<CODENAME-1> _HOSTS="host-a1 host-a2 host-a3..."`
- `MPCCI_<CODENAME-2> _HOSTS="host-b1 host-b2 host-b3 ..."`
- `MPCCI_SERVER_HOSTS="host-c1 host-c2 host-c3 ..."`

should be given as arguments. (see also ▶3.4.4.4 Application Specific Code to Host Allocation ◂).

After submitting the job the job id is saved by MpCCI in the directory "<HOME>/mpcci/batch". You can retrieve the list of submitted with this command: `mpcci list jobs` which returns the job id with the associated batch system:

<table>
<thead>
<tr>
<th>Batch System</th>
<th>Job ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGE</td>
<td>1532</td>
</tr>
<tr>
<td>PBS</td>
<td>8470</td>
</tr>
</tbody>
</table>

This information can be used to status or kill the job.

Here is an example of job submission for SGE:

```
$> mpcci -batch PBS submit -J coupled_test -cwd -pe cre 8 MPCCI_SERVER_HOSTS=./.* MPCCI_FEM_HOSTS=./.* MPCCI_CFD_HOSTS=./.* test.csp
```

and this is the generated script "test.sh" that is given to the submit command:

```bash
#!/bin/sh
#$ -S /bin/sh
#$ -v MPCCI_LICENSE_FILE,PATH
#$ -cwd
```
3.4 Running MpCCI in a Network

```bash
#$ -N coupled_test
#$ -pe cre 8
MPCCI_CFD_HOSTS=/>.*/
MPCCI_SERVER_HOSTS=/>.*/
MPCCI_FEM_HOSTS=/>.*/
export MPCCI_CFD_HOSTS MPCCI_SERVER_HOSTS MPCCI_FEM_HOSTS
mpcci batch test.csp
```

Status a job with MpCCI command line

Each batch system has a status command:

```bash
mpcci batch <BATCH_NAME> status <JOB ID>
```

The job id may be retrieved with the `mpcci list jobs`.

⚠️ The status command has to be executed on the machine where the job has been submitted.

Kill a job with MpCCI command line

Each batch system has a kill command:

```bash
mpcci batch <BATCH_NAME> kill <JOB ID>
```

The job id may be retrieved with the `mpcci list jobs`.

⚠️ The kill command has to be executed on the machine where the job has been submitted.

### 3.4.4.3 Host management under a Job Scheduler Environment

At startup of a coupled job MpCCI checks if corresponding environment variables

- LSB_MCPU_HOSTS
- PBS_NODEFILE
- PE_HOSTFILE
- LOADL_PROCESSOR_LIST

are defined. MpCCI then uses the listed host names to start MpCCI coupling server and both coupled codes on these hosts.

### 3.4.4.4 Application Specific Code to Host Allocation

MpCCI allows to specify in more detail the allocation of hosts for specific codes.

If environment variables
• MPCCI_<CODENAME-1> _HOSTS = "host-a1 host-a2 host-a3..."

• MPCCI_<CODENAME-2> _HOSTS = "host-b1 host-b2 host-b3 ..."

• MPCCI_SERVER_HOSTS = "host-c1 host-c2 host-c3 ...

are set by then the MpCCI startup procedure will allocate the listed hosts for the code represented by the name <CODENAME> or for the server with MPCCI_SERVER_HOSTS. It is up to the user to define these environment variables if the MpCCI batch command line is used. Otherwise by submitting the job from MpCCI GUI these environment variables are written in the batch script.

Instead of a list of real hostnames a regular Perl-pattern might be used to define the selection of code-specific hosts extracted from the defined PBS, LSF, SGE or LoadLeveler environment variables:

MPCCI_<CODENAME-1> _HOSTS = "/pattern/".

CODENAME-1 must be a ”word” character.
3.5 Mesh Checks

MpCCI offers two sorts of checks for the coupled meshes:

- Bounding box checks to detect errors in selecting the coupling regions
- Mesh quality checks to identify misshaped elements which may cause problems for data transfer and simulation.

3.5.1 Bounding Box Checks

The bounding box checks can be selected in the Edit Step of the MpCCI GUI (see 4.5 Edit Step). For each of the coupled meshes in a coupling region, the bounding box is computed. This box is a rectangular box with sides parallel to the coordinate axes, which contains the whole mesh. The bounding boxes thus represent the total size of the meshes.

Three checks are performed if selected:

**Overlap check.** It is checked whether the bounding boxes of the two meshes of a coupling region intersect each other.

**Scale check.** It is checked whether the sizes of the bounding boxes of the two meshes are roughly the same. This check is meant to detect errors in unit systems.

**Coupling region match check.** It is checked whether the dimensions of the bounding boxes differ by more than 10% in size and position.

3.5.2 Mesh Quality Checks

The mesh quality checks are performed if the appropriate option is selected in the Edit Step (see 4.5 Edit Step).

There are two kinds of checks:

**Check for doubly defined nodes.** It is checked whether all nodes on one partition are distinct from each other. If that is not the case, an error message is given.

**Check for degenerate elements.** The exact check which is performed depends on the element type.
4 Graphical User Interface

4.1 Starting and Exiting MpCCI GUI

4.1.1 Starting MpCCI GUI

When you create a coupled simulation project, the MpCCI GUI generates a set of files containing the definition of the coupled simulation model, e.g. the MpCCI input file and the MpCCI log files. Consequently, before you start the MpCCI GUI, you should move to a directory where you have permission to create files. You execute MpCCI GUI by running the mpcci executable with the gui subcommand and the desired options.

Usage:
    mpcci gui [OPTIONS] [project] [OPTIONS]

Synopsis:
    'mpcci gui' is used to launch the MpCCI GUI.

Options:
    -chwd <PATH>  Replace the symbolic working directory $(CWD) used inside the projectfile by the absolute pathname specified in the <PATH> argument.
    -help         This screen.
    -new          Start the GUI with a new project.
    -nolic        Do not check for a license before starting the GUI. This option may be used when no license is available but you would like to prepare a job.
    -norsa        Do not ask for ssh assistance if an rsa key file does not exist.

To start an MpCCI job in batch mode with a well prepared coupled simulation project use:

    mpcci -batch <project name>

If you start the MpCCI GUI without any options like

    mpcci gui
the following default checks are performed:

- Check and preparation of your environment.
- A check of your secure shell rsa/dsa key files. If no rsa/dsa key files are found you will be asked to create them.
- Check for an existing project file in the current directory in order to load that project to the MpCCI GUI. If more than one project file exists no project will be loaded.
- Check of the license environment. MpCCI GUI tries to check for a license e.g. `MPCCI_LICENSE_FILE` and also informs you when your license will soon expire. In order to disable it use the option `-nolic`.

If you do not include the `<project name>` to the `mpcci gui` the MpCCI GUI starts at the initial step.

### 4.1.2 Exiting MpCCI GUI

You can exit the MpCCI GUI session at any time by selecting `File → Exit` from the main menu bar, by using the key shortcut `Alt + x` or by closing the main window of the MpCCI GUI. If you made any changes to the current coupled simulation project, the MpCCI GUI asks if you want to save the changes before exiting the session. MpCCI GUI then closes the current coupled simulation project file and exits the session.

### 4.2 MpCCI GUI Menus

Before talking about the menus let’s have a look at Figure 1 the title of the MpCCI GUI. The title of the main window of the MpCCI GUI contains the following information:

- the version of MpCCI you are running,
- the name of the currently coupled simulation project or “noname” if no project is specified and
- the steps of the coupling process with the currently worked on step marked by angle brackets.

![Figure 1: MpCCI GUI title and main menus](image)

Below the title we have the MpCCI GUI menus. They are available all the time while MpCCI GUI is running. The main menus, also shown in Figure 2 are:
4 Graphical User Interface

**File** for creating, opening, saving and exiting a coupled simulation project.

**Batch** for submitting, querying status and killing a coupled simulation project from a batch queuing system like LSF, PBS, SGE, GLOBUS, etc. (see also 3.4.4 Coupled Analysis in Batch Mode)

**License** for managing the MpCCI licenses and displaying detailed license information.

**Tools** for launching various MpCCI commands directly from the MpCCI GUI.

**Preferences** for changing the Look & Feel of the MpCCI GUI.

**Codes** for providing code specific commands. If at most three codes are available the codename menus are directly shown in the menu bar and the Codes menu disappears.

**Help** for requesting help and for getting information about the MpCCI GUI.

---

![Menu Entries](image.png)

The menu entries are:

### 4.2.1 File Menu

**File→New Project** creates a new project. The MpCCI GUI will be initialized with its default application values and the first step which is the models step.

**File→Open Project** opens a project file. A file chooser pops up and you are able to select one of your previously saved projects. The project will be opened at the last step you were. 

*File-Types:* “Coupled Simulation Project” - Files ("*.csp")

**File→Save Project** saves the project. The current project settings are saved under the current project name. This command is available when a project configuration has been opened or if the project already has been saved via File→Save Project As.

**File→Save Project As** saves the project settings under a specified name. The MpCCI GUI displays the Save As dialog box. The active project can be renamed and saved to a new directory.

**File→Exit** ends the MpCCI GUI session. MpCCI GUI prompts to save the project with unsaved changes before terminating the application.
4.2.2 Batch Menu

**Batch → Submit** submits the current project to the batch queuing system. A configuration dialog pops up and shows the available batch system in a list. By selecting one of the batch system you will be able to configure some options for the job. The configuration dialog is split in three parts (see Figure 3):

- **Application specific host allocation**: For each application you are able to provide either a list of hostnames or a regular Perl-pattern. This Pattern will extract automatically the hosts from the batch environment.

- **Batch system selection**: You can select a queuing system to use.

- **Specific batch options**: You find a list of option to setup for example the queue, the job name, the output filenames, etc.

![Figure 3: Batch Submit Configuration Dialog](image-url)
**Batch→Status** queries the status of a batch job. You are able to select one of your previously submitted jobs referenced by its job id (see Figure 4).

![Figure 4: Batch Status Dialog](image)

**Batch→Kill** kills a batch job. You are able to select one of your previously submitted jobs referenced by its job id (see Figure 5).

![Figure 5: Batch Kill Dialog](image)

### 4.2.3 License Menu

- **License→Display license servers** lists all defined license servers.
- **License→Display license files** lists all local license files defined.
- **License→Check the MpCCI license status** displays MpCCI license features and available tokens.
- **License→Check all available licenses** displays all license features and available tokens.
- **License→MpCCI license expiration date** displays the expiration date of the MpCCI license.
- **License→Start MpCCI license server** starts the FHGSCAI vendor daemon on the local host.
License→Stop MpCCI license server stops the FHGSCAI vendor daemon running on the local host.

### 4.2.4 Tools Menu

Tools→Print environment variables shows a dialog box with the environment used by MpCCI.

Tools→Display the current process list shows a dialog box with a list of the current processes running on the local system.

Tools→Display the process list shows the processes running on the local machine. On Windows it runs the taskmgr command and on UNIX systems it runs the top command.

Tools→Test→MpCCI simple test runs a simple test-case delivered with MpCCI.

Tools→Visualizer starts the MpCCI Visualizer.

### 4.2.5 Preferences Menu

Preferences→Look & Feel allows to change the Look & Feel of the MpCCI GUI. Depending on the underlying Window Manager the available supported Look & Feels can be selected e.g. Metal, CDE/Motif, Windows or Windows Classic.

### 4.2.6 Codes Menu

Codes→Codename Provides code specific commands which are defined in the respectively code configuration file (see IX-2.4.2 Codes Menu: <CodesMenuEntries>).</p>

### 4.2.7 Help Menu

Help→MpCCI Documentation displays a help window with the MpCCI documentation.

Help→About displays product information and third party products integrated in the MpCCI GUI.
4.3 Models Step

The MpCCI Models Step is the first of four steps for setting up a coupled simulation. Its aim is

- to select the codes involved in the coupling,
- to choose their corresponding model files and
- to start the scanner for each code which will extract the components needed to build the coupling regions in the next step, the Coupling Step.

4.3.1 Code Parameters

Additionally some codes may provide some further parameters to be set by the user: e.g. the version or release of the code to run or the unit system to be used for the grid length and quantities. For a description of the code specific parameters have a look at the Codes Manual.

4.3.2 Requirements

In order to proceed with the Coupling Step the following requirements have to be fulfilled in this Models Step:

- selection of at least two codes for coupling
- setting of all required parameters for each code especially the model files
- successful run of the scanner for each code
- usage of the same model dimension by the coupled codes

All these criteria are checked when clicking on the next button and a dialog with an appropriate message will pop up if the requirements are not fulfilled. In case of a warning the user may decide to continue with the next step although it is not recommended by MpCCI. If an error occurs the user can’t go on before having corrected the complained items.

4.4 Coupling Step

In the MpCCI coupling step you create and configure your coupling regions by

1. Defining coupling regions.
2. Selecting the components for each interconnected code.
3. Specifying the quantities which will be exchanged.

These basic steps are described in ▷ IV-2.5 Coupling Step – Definition of Coupling Regions and Quantities ◁. In addition you may use some more features which are described as follows:

### 4.4.1 Generate Regions

If you have equally named components between your coupled codes you can automatically generate coupling regions with the equally named components in it. The **Generate** button at the bottom of the **Regions** area issues one region for each matching component pair. Now you still have to specify the quantities for each generated region.

### 4.4.2 Options Part

The **Options** part below the coupled components list (see Figure 6) provides

**Synchronized scrolling** useful for scrolling large component lists when you are looking for similar named components in both codes. If enabled and you scroll one component list the list with the other code components will be automatically scrolled to the same alphabetical position.

**Component Name Matching** which configures the behavior of the automatic component selection. Automatic component selection means that if you select a component of one code, components of the other code with matching names will be automatically selected, too. Choose **None** to disable the automatic selection, **Similar** to automatically select components with similar names (names starting with equal initial letters) and **Exact** to automatically select components with the exact name of the manually selected component.

![Options](image)

Figure 6: Options for choosing coupled components

### 4.4.3 Quantity properties

In addition to specifying the exchanged quantities you may change some quantity properties in the quantity configuration part of the coupling step (see Figure 7):

default value is used by MpCCI when no code specific value is available (see \textit{3.2.4 Orphaned Nodes and Elements}).

interpolation type specifies the type of interpolation MpCCI uses for that quantity. For global quantities the provided interpolation types are \textit{min}, \textit{max}, \textit{sum} and \textit{prod}. For all other quantities MpCCI provides \textit{field}, \textit{flux} and \textit{fluxdens} (see also \textit{3.2.5 Flux and Field Interpolation}).

extrapolate orphans assigns MpCCI to extrapolate values into orphaned regions instead of using the default value (see \textit{3.2.4 Orphaned Nodes and Elements}).

### 4.4.4 Quantity Sender

Sometimes a quantity can be sent and received by two or more codes. In that case the user has the opportunity to select the sender of the quantity in the \textbf{Sender} part of the quantity configuration as shown in figure Figure 8.

Figure 8: Selection of the Sender for a Quantity

### 4.4.5 Predefined Sets

Furthermore the MpCCI GUI provides some \textit{Predefined Sets} of already preconfigured quantities at the top of the quantities list (see Figure 9). The predefined quantity settings are selectable from a list. To apply a coupling specification you have to select a \textbf{Coupling Type} or a \textbf{region} and click on the \textbf{Set} button. The associated quantities with their sender and receiver codes will be automatically selected.

\textbf{Copy from Region} offers the possibility to copy a set of quantity settings from another already defined region of this element type. You may use this option if you have specified the quantities set yourself and want to propagate the configuration to other coupled regions.
Select Coupling Type provides different coupling types according to the type of the codes you couple. Each provided Coupling Type is characterized by some properties like:

- the dimension which corresponds to the coupling dimension.
- the quantities which will be sent by special types of codes. Each code may have assigned more than one type (see IX-2.4 MpCCI GUI Configuration File gui.xcf). The known code types are:
  - CFD.
  - ElectroMagnetism.
  - FluidPlasma.
  - FluidThermal.
  - InjectionMoulding.
  - Radiation.
  - SolidStructure.
  - SolidThermal.

For at least one code type a list of quantities to be sent is associated.

Only those coupling types are shown in the list which fit to the coupled codes and supported quantities of the coupled simulation setup. This means that each code type in a coupling type has to match at least one coupled code and that all quantities listed for the code type have to be supported as send-quantity by the matching code and as receive-quantity by the other coupled codes. Below the different coupling types are listed.

**Transient electric arc coupling: Method 1**

Dimension of coupling region: 3

<table>
<thead>
<tr>
<th>Code type</th>
<th>Quantities to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>FluidPlasma</td>
<td>ElectrRes3</td>
</tr>
<tr>
<td>ElectroMagnetism</td>
<td>LorentzForce JouleHeat</td>
</tr>
</tbody>
</table>
Transient electric arc coupling: Method 2
Dimension of coupling region: 3

<table>
<thead>
<tr>
<th>Code type</th>
<th>Quantities to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>FluidPlasma</td>
<td>ElectrRes3</td>
</tr>
<tr>
<td>ElectroMagnetism</td>
<td>MagneticField CurrentDensity</td>
</tr>
</tbody>
</table>

One way force mapping
Dimension of coupling region: 2

<table>
<thead>
<tr>
<th>Code type</th>
<th>Quantities to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD</td>
<td>RelWallForce</td>
</tr>
</tbody>
</table>

One way pressure mapping
Dimension of coupling region: 2

<table>
<thead>
<tr>
<th>Code type</th>
<th>Quantities to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFD</td>
<td>OverPressure</td>
</tr>
</tbody>
</table>

Steady state radiative heat transfer
Dimension of coupling region: 2

<table>
<thead>
<tr>
<th>Code type</th>
<th>Quantities to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radiation</td>
<td>WallTemp</td>
</tr>
<tr>
<td>FluidThermal</td>
<td>FilmTemp WallHTCoeff</td>
</tr>
</tbody>
</table>

Steady state surface heat transfer
Dimension of coupling region: 2

<table>
<thead>
<tr>
<th>Code type</th>
<th>Quantities to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>SolidThermal</td>
<td>WallTemp</td>
</tr>
<tr>
<td>FluidThermal</td>
<td>FilmTemp WallHTCoeff</td>
</tr>
</tbody>
</table>

Transient MHD Coupling
Dimension of coupling region: 3

<table>
<thead>
<tr>
<th>Code type</th>
<th>Quantities to send</th>
</tr>
</thead>
<tbody>
<tr>
<td>FluidPlasma</td>
<td>ElectrRes3</td>
</tr>
<tr>
<td>ElectroMagnetism</td>
<td>LorentzForce</td>
</tr>
</tbody>
</table>
4.4.6 Requirements

In the Coupling Step the following requirements have to be fulfilled:

At least

- one coupling region with at least
- two components of at least
- two different codes with at least
- one quantity applied
has to be set up.
If one or more of the above mentioned items is missing a click on the next button will cause a dialog popping up with an appropriate message and will inhibit the next step.
4.5 Edit Step

The options in the Edit Step are the control parameters for MpCCI. They are grouped into control and contact sections.

4.5.1 Control

In this section you may specify some general steering parameters used by MpCCI.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckBoundingBox</td>
<td>MpCCI will check if the models have the same geometry unit and the same position. This is done by comparing the bounding boxes of the models, see ▶ 3.5.1 Bounding Box Checks ◁ for details. (Default: on)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Tracefile</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Specifies the name of the MpCCI tracefile to be created. If no name or an empty name is specified no tracefile will be written. Note: The MpCCI control process is needed for writing a tracefile (see \textgreater{} 4.6.1 Configuring the MpCCI Coupling Server\textless{}). Default: &quot;tracefile.ccv&quot;</td>
</tr>
<tr>
<td>CloseAfterWriting</td>
<td>If the switch is set (default) MpCCI will close the tracefile each time after writing data into it.</td>
</tr>
<tr>
<td>ImplicitCouplingSteps</td>
<td>Tells MpCCI whether or not to generate a new coupling step in the tracefile during a call of \texttt{CCI.Check_convergence}. Default: set</td>
</tr>
<tr>
<td>TraceCommValues</td>
<td>If the switch is set (default) in addition to the geometry data, MpCCI will also write communication values into the tracefile.</td>
</tr>
<tr>
<td>TraceMeshValues</td>
<td>If the switch is set (default) MpCCI will write communication values of coupling type \texttt{mesh} into the tracefile unless the parameter TraceCommValues is switched off.</td>
</tr>
<tr>
<td>TraceMeshChangeValues</td>
<td>If the switch is set (default) MpCCI will write communication values of coupling type \texttt{mesh_change} into the tracefile unless the parameter TraceCommValues is switched off.</td>
</tr>
<tr>
<td><strong>Output level</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Global</strong></td>
<td>The global output level determines how much output MpCCI writes:</td>
</tr>
<tr>
<td></td>
<td>0  no output,</td>
</tr>
<tr>
<td></td>
<td>1  message when entering/leaving a subroutine. (default setting)</td>
</tr>
<tr>
<td></td>
<td>2  additional input and output arrays (needed for Playback Tool).</td>
</tr>
<tr>
<td></td>
<td>3  maximal output.</td>
</tr>
</tbody>
</table>
### Mesh quality

**checkMeshQuality**

If the switch is set, MpCCI performs some mesh quality checks during `CCI.Close_setup()` and gives a warning if there are badly shaped elements or other inconsistencies of the mesh. The following checks are performed:

- **doubly defined nodes** It is checked whether all nodes on one partition are distinct from each other. If that is not the case, an error message is given.

- **degenerate elements** For each element it is checked whether its node numbers are distinct from each other. If that isn’t the case, a warning is given.

By default `CheckMeshQuality` is turned on. This setting may cause an MpCCI internal problem if the coordinates of the nodes are big. This problem can be avoided by turning the checks off. More details on the checks are given in §3.5.2 Mesh Quality Checks.<

### Restart file

With the restart file settings the neighborhood relations can be stored in files to be reused and skip the neighborhood search which can be time-consuming. This is described in §3.3.6 Restarting a Coupled Simulation.<

- **Name**
  
  Name of the restart file(s) keeping the neighborhood relations. If this is `xxx` e.g., the restart files will be named `xxx.1`, `xxx.2`, ... for process 1, 2, ...etc. The default name is "restart.rst".

- **Access**
  
  Defines whether the restart files should be ignored, written or read:

  - **none** ignore restart files (default setting)
  - **read** read restart files
  - **write** write restart files

### Misc

- **MonitorSleepTime**
  
  Sets the sleep time for the control process during its monitor loop. The default value is 0.04s.

- **AllowCoreDump**
  
  Defines whether or not a core-dump file is written in case of abnormal termination. On default this flag is set.
4.5.2 Contact

In this section you specify the attributes of the grids and define the contact detection algorithm and matching criterion to use. The algorithms are described in \textit{\ref{3.2 Data Exchange}}.

![Figure 11: Contact Options](image)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchAlgorithm</td>
<td>Specifies the contact search algorithm to use:</td>
</tr>
<tr>
<td></td>
<td>\textbf{Minimal distance} uses the minimal distance algorithm (default).</td>
</tr>
<tr>
<td></td>
<td>\textbf{intersection} uses a neighborhood search and interpolation based on</td>
</tr>
<tr>
<td></td>
<td>intersection methods.</td>
</tr>
<tr>
<td>Overlap</td>
<td>Defines the overlap between the codes.</td>
</tr>
<tr>
<td></td>
<td>\textbf{full} means that the neighborhood search in MpCCI tries to find</td>
</tr>
<tr>
<td></td>
<td>contact for each point and if it is not possible in the first attempt,</td>
</tr>
<tr>
<td></td>
<td>MpCCI uses brute force linear search. (default)</td>
</tr>
<tr>
<td></td>
<td>\textbf{partial} allows that the two coupling regions have only a partial</td>
</tr>
<tr>
<td></td>
<td>overlap.</td>
</tr>
<tr>
<td>OrphanedPointsWarning</td>
<td>Print warning messages and lists of orphaned nodes into the output files.</td>
</tr>
<tr>
<td></td>
<td>Note: if the overlap is partial this switch should be set off.</td>
</tr>
<tr>
<td></td>
<td>(default: on)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Overlap</td>
<td>Defines the overlap between the codes.</td>
</tr>
<tr>
<td></td>
<td><strong>partial</strong> allows that the two coupling regions have only a partial overlap. If this value is set the OrphanedPointsWarning should be switched off.</td>
</tr>
<tr>
<td></td>
<td><strong>full</strong> means that the neighborhood search in MpCCI tries to find contact for each point and if it is not possible in the first attempt, MpCCI uses brute force linear search. This is the default value.</td>
</tr>
<tr>
<td>OrphanedPointsWarning</td>
<td>Print warning messages and lists of orphaned nodes into the output files if this switch is set (default). If the overlap is partial this switch should be set off.</td>
</tr>
<tr>
<td>PreContactSearch</td>
<td>MpCCI generates pairs of points and elements (non-matching grids) and pairs of points (matching grids) which are located close to each other based on the following settings:</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Specifies the search algorithm to use.</td>
</tr>
<tr>
<td></td>
<td><strong>none</strong> No pre-contact search algorithm. This yields brute force linear search for all points.</td>
</tr>
<tr>
<td></td>
<td><strong>all</strong> Pre-contact algorithm that generates all pairs of points and elements. Equivalent to brute force linear search.</td>
</tr>
<tr>
<td></td>
<td><strong>bucket</strong> A bucket search algorithm is used (default).</td>
</tr>
<tr>
<td></td>
<td><strong>bucket_sphquad</strong> A bucket search algorithm for a pair of meshes on the sphere consisting of spherical quadrilaterals is used.</td>
</tr>
<tr>
<td>BboxExpansion</td>
<td>Specifies the size of the bounding box. Default is 1.1</td>
</tr>
<tr>
<td>MinBboxSize</td>
<td>Specifies the minimal bucket size. Default is 0.</td>
</tr>
<tr>
<td>BucketExpansion</td>
<td>Specifies the size of the bucket. Default is 1.0</td>
</tr>
<tr>
<td>MinimalDistance</td>
<td>Defines the matching criterion for the minimal distance based on local coordinates computation. Elements that not fulfill ( \Theta_1 \cdot d_b + \Theta_2 \cdot d_t^2/A + \Theta_3 \cdot d_n^2/A &lt; \text{Rejection} ) are rejected.</td>
</tr>
<tr>
<td>Theta1</td>
<td>Weight for the closest barycentric distance ( d_b ) to the element. The default value is 0.0.</td>
</tr>
<tr>
<td>Theta2</td>
<td>Weight for the relative in-plane distance ( d_t ) to the element. The default value is 1.0.</td>
</tr>
<tr>
<td>Theta3</td>
<td>Weight for the out-of-plane (normal) distance ( d_n ) to the element. The default value is 0.0.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Rejection</td>
<td>Defines the rejection criterion. The default value is 0.001.</td>
</tr>
<tr>
<td>InsideOnly</td>
<td>Enables the shape function extrapolation. Only values from inside an element are transferred. This option avoids negative values when exchanging temperature etc. Default is not set. See ▷ 3.2.2 Minimal Distance ◄ for details.</td>
</tr>
<tr>
<td>Intersection</td>
<td>defines the matching criteria for the intersection algorithm:</td>
</tr>
<tr>
<td>PerformTest</td>
<td>If this value is set (which is the default) MpCCI prints warnings if elements fail to match elements on the partner mesh.</td>
</tr>
<tr>
<td>ScaleResult</td>
<td>If this value is set (which is the default) MpCCI scales the size of the overlap for partially intersecting border elements to element size.</td>
</tr>
<tr>
<td>MaxProjectionDistance</td>
<td>Defines the maximal projection distance in meters. A negative value means that all distances are accepted. The default value is -1.0. Note that the projection distance is also limited by the bucket size.</td>
</tr>
<tr>
<td>MaxProjectionAngle</td>
<td>Defines the maximal angle between source and target triangle. The default value is 30.0°.</td>
</tr>
<tr>
<td>Rejection</td>
<td>Defines the rejection criterion for negligible intersection areas. Elements with a smaller intersection area are not considered in the projection. The default value is 0.001.</td>
</tr>
</tbody>
</table>

### 4.6 Go Step

In the Go Step you will configure the start-up of the applications. Following is a detailed description of the MpCCI coupling server parameters. The configuration for the analysis codes is described in the appropriate code section of the Codes Manual. How to start, stop and kill the coupled simulation can be found in ▷ IV-2.7 Go Step – Starting Servers and Codes ◄.

#### 4.6.1 Configuring the MpCCI Coupling Server

For the MpCCI coupling server the following parameters can be configured (see also Figure 12):

- **Filename prefix for log files** is used as prefix to all files generated by MpCCI like "mpccirun.inputfile".

- **Create server log files** is used to save all the output from the MpCCI server-processes to files. The log files are named "<Filename prefix>.stdout.n" where n is a number from 0 up to the number of server processes - 1.

- **Start additional control process** is used to start a process which logs the communication activities of the coupled applications. The logging information is written into the previously mentioned log file with...
number 0. This parameter is also used to start a process which writes the “tracefile” as mentioned in ▶ IV-2.6 Edit Step – Further Coupling Options ◀ and ▶ IV-3.1 The MpCCI Visualizer ◀.

Create preview tracefile only and exit is used to preview the coupled regions for the involving codes. After all codes have started and sent its coupled regions definition to the MpCCI server, a tracefile is written and the coupled simulation will terminate. The tracefile may be read by the MpCCI Visualizer.

Auto force exit jobs on termination tells MpCCI to automatically force an exit of remaining running jobs when one code already exited by itself. Perhaps the remaining jobs are blocked when one job already ended. Usually used in batch mode where the user can’t interactively stop or kill running jobs. If this parameter is set the panel expands and shows some more options (see Figure 13):

Timeout in seconds specifies the time to wait between one job is exited and the remaining jobs will
be forced to exit. The default value is 60 seconds.

Select 32/64 bit server is used to select the server mode to run on the target platform.

- **32 bit preferred** use the 32 bit server mode if it is supported on this platform.
- **64 bit preferred** use the 64 bit server mode if it is supported on this platform.
- **auto select** use automatically the appropriate server mode for the current platform. If both server modes 32/64 bit exist the 64 bit mode will be selected as default.

Select server precision is used to define the computation precision.

- **double** use the double precision server.
- **single** use the single precision server.
- **auto select** use automatically the appropriate server precision for the current platform. If both server precisions single/double exist the double precision server will be selected as default.

⚠️ Using a double precision server will require more memory for the coupling system.

Run server processes inside xterm It lets the MpCCI GUI open an xterm window for each MpCCI process on UNIX platforms. On Windows the server will always run inside an xterm window independent of the selection of this option. If this parameter is set the panel expands and shows some more options for configuring the xterm (see Figure 14):

⚠️ Using a double precision server will require more memory for the coupling system.

Auto close xterm at exit automatically closes the xterm when the server exits. If not set the xterm stays open even if the server isn’t running anymore and the user can see the messages output by the server.
Select xterm background color provides colors to be used as background for the xterm which sometimes make it easier to find the right window.

Distribute server on remote hosts Configures MpCCI to run in a network. (see Figure 15):

Don’t run first server process on the local host is used to start the first server process on a different host than the local host. This first server process is responsible for listening to the client connection on a specified port.

Shared file system (no file copy) means that the underlying file system is a shared file system and file copy isn’t necessary.

Optional DISPLAY variable sets the DISPLAY variable to redirect the display to the target host. This value must be given as hostname:0.0.

Optional local host alias sets a name which is chosen to identify the local host.

Optional 'host host ...' to be used provides a list of hosts on which the server processes will be distributed.

Optional hostlist file is also used to get a list of hosts on which the server processes will be distributed.

Use default hostfile use the hostfile defined by the MPCCI_HOSTLIST_FILE environment variable.

Preferred remote shell type is used to select the type of the remote shell to start the MpCCI server.

rsh (classic rsh) uses the classic remote shell command rsh.
**ssh (secure shell)** uses the secure remote shell command `ssh`.

**Main server port address** specifies the port address on which the MpCCI server listens for the client connection. The default port is in most cases acceptable, modifications are only required if firewalls block connections.

### 4.7 Remote File Browser

#### 4.7.1 File Browser Handling

By clicking on the button **Browse** the remote file browser opens. After having selected up a file the file name is displayed in the text field. If you point the text field with the mouse pointer a tip appears and indicates the location of the file. This information contains the hostname and the absolute file path. If the field is empty the tip shows the message **no file specified**. In the text field you may enter a file name.

If the name is relative the directory and host of the old file parameter will be taken or if no old file exists the current working directory on the local computer will be taken. Then the file will be located relative to your current working directory.

The remote file browser (Figure 16) displays the files of your current directory in the center and on the right it shows a list of different file systems mounted to this remote file browser. The current directory is represented by the working directory where the MpCCI GUI is running. After you have selected a first file for your application the path of this file represent the current directory for your selected code. Each code has its own working directory.

The default file system view mounted is your local file system represented by the hostname where the MpCCI GUI is running.

If you have more than one file system mounted the current file system is highlighted with a blue background and has the name of the remote machine.

The navigation is performed by using the mouse double click.

The file selection is performed by doing a double click or a simple click to select it, followed by a click on the **Open** button.
To save a file you have to select the directory by using the mouse and finally enter the file name in the provided text field followed by a click on the **Save** button.

### 4.7.2 How to mount a new file system

On the right side of the remote file browser there are two buttons to **Connect** and **Disconnect** a file system. To disconnect a file system from the file browser, select the corresponding file system in the list and click on the **Disconnect** button.

The local file system cannot be disconnected and is represented by the figure **Figure 17**

![Local File System Symbol](image)

*Figure 17: Local File System Symbol*
To mount a new file system:

1. You have to open the connection dialog by clicking on the **Connect** button.

2. You have to enter the hostname of the remote machine and some user information.

![Connection Setting](image.png)

**Figure 18: File Browser Connection Dialog**

- Select your remote machine.
  At the initialization phase of the **MpCCI GUI** a list of hosts is searched in the files ".rhosts" and ".shosts" from the user home directory. This list is used to provide some predefined hosts to select.

  You may write the name of the remote machine on the **Host** text field. If you type the hostname on the input field the **MpCCI GUI** tries to figure out the host name you want from the host list. Otherwise you may directly pick up the machine name to connect to from the combo box.

- Configure the protocol of the remote connection.
  You may use **ssh** service or **rsh**.

- Specify the user name.
  You may modify the user to be used for the connection to the remote machine or let the predefined name.

After configuring the connection you must click on the **[OK]** button to establish the connection. If the connection to the remote host is successfully established, you will see the name of the new remote machine in the list of the file system view and the remote file system. Each remote file system is characterized by the type of the protocol used for the connection. On figure **Figure 19** you have the icon for the secure and non secure connection.

3. You have a view on the remote file system and you may navigate to select a file.
Figure 19: Secure Connection Icon (left) and a non secure Connection Icon (right)
5 Command Line Interface

5.1 Using the Command Line Interface

MpCCI has an extensive command line interface, which offers a lot of functionality beyond the actual coupling process, including license information, job control etc.

To use the command line interface, MpCCI must be installed properly. Especially the PATH environment variable must be set correctly and contain the binary directory "MPCCI_HOME/bin", see the Installation Guide for details.

To obtain a quick help on the MpCCI commands, it is also possible to enter the command followed by a “help key”. If you want to get further information e.g. on starting the MpCCI GUI, which is normally done by entering `mpcci gui`, type

```
mpcci gui -help
```

or

```
mpcci gui ?
```

MpCCI is very flexible in interpreting the command line options. In general dashes (-) can be omitted and commands or options can be abbreviated as long as the abbreviation is unique. Further, command interpretation is not case sensitive. The following commands are thus all equivalent, and show the expiry date of your license:

```
mpcci license -expire
mpcci License EXPIRE
mpcci lic -exp
mpcci lic e
```

If less subcommands of options than required are entered, a list of available options is printed. Entering only the command `mpcci` without further options consequently yields basic information:

- Version information.
- Basic usage as described in this section.
- A list of all available commands.

The general description looks like:

```
*************************************************
MpCCI 3.1.1-1, Build (Fri Aug 27 14:46:58 2010)
```
Usage:
    mpcci SUBCMD [OPTIONS] [ARGS] [HELPKEY] ...

Synopsis:
    ’mpcci’ is the root for all MpCCI related commands. You need
to specify at least one subcommand (SUBCMD) on the commandline.

    SUBCMD, OPTIONS (not file name ARGS!) may be typed in lowercase
or UPPERCASE letters or may even be abbreviated as long as there
is no ambiguity.

    Since the command and help system is dynamically configured at
runtime some SUBCMDS and OPTIONS may not always appear in the list
below or may suddenly become ambiguous as they are activated only
under certain circumstances (existing file/installation etc.)
If you use ’mpcci’ in scripts you should never use the
abbreviated form of SUBCMD or OPTION.

    You get online help for most of the SUBCMDS and OPTIONS if the
HELPKEY ([-]help, [-/]? ...) appears on the commandline.

    Please type either

    "mpcci SUBCMD HELPKEY"  or
    "mpcci HELPKEY SUBCMD"

    to get more detailed help on the subcommands.

The list of commands is omitted here, a concise list is given in the following section.
## 5.2 Overview of All Subcommands

In addition to subcommands which are related to the general functions of MpCCI, further code-specific commands are offered, which are described in the corresponding sections of the Codes Manual.

The MpCCI commands are discussed in more detail in following sections, which are sorted by the purpose of the commands. To find a command by its name, the following list shows all MpCCI commands in alphabetical order:

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Short Description</th>
<th>Discussed in</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch [-n]</td>
<td>Print the MpCCI base architecture without a newline or with a newline [-n] at the end and exit. This is different from <code>mpcci info arch</code> which prints the used architecture.</td>
<td>▶5.4.1 mpcci arch ◄ on page 98</td>
</tr>
<tr>
<td>backup &lt;file ...&gt;</td>
<td>Make a backup copy of a list of files.</td>
<td>▶5.6.1 mpcci backup ◄ on page 116</td>
</tr>
<tr>
<td>batch &lt;project&gt;</td>
<td>Start an MpCCI batch job with project file &lt;project&gt;.</td>
<td>▶5.6.2 mpcci batch ◄ on page 117</td>
</tr>
<tr>
<td>clean</td>
<td>Remove all files from the temporary MpCCI directory $&lt;Home&gt;/mpcci/tmp$.</td>
<td>▶5.6.8 mpcci clean ◄ on page 124</td>
</tr>
<tr>
<td>doc</td>
<td>View the MpCCI documentation.</td>
<td>▶5.4.2 mpcci doc ◄ on page 99</td>
</tr>
<tr>
<td>env</td>
<td>Print out the environment used by MpCCI in various formats for further processing.</td>
<td>▶5.4.4 mpcci env ◄ on page 102</td>
</tr>
<tr>
<td>gui</td>
<td>Launch the MpCCI GUI.</td>
<td>▶5.3.1 mpcci gui ◄ on page 89</td>
</tr>
<tr>
<td>home [-n]</td>
<td>Print the MpCCI home directory without a newline or with a newline [-n] at the end and exit. This is in fact a shortcut for <code>mpcci info home</code>.</td>
<td>▶5.6.5 mpcci home ◄ on page 104</td>
</tr>
<tr>
<td>info</td>
<td>Print general information about MpCCI.</td>
<td>▶5.4.3 mpcci info ◄ on page 100</td>
</tr>
<tr>
<td>kill</td>
<td>Platform independent process kill based on command line pattern matching.</td>
<td>▶5.6.9 mpcci kill ◄ on page 125</td>
</tr>
<tr>
<td>license</td>
<td>Manage the license server and print license related information.</td>
<td>▶5.5.1 mpcci license ◄ on page 107</td>
</tr>
<tr>
<td>list</td>
<td>List information about the MpCCI installation and the supported codes.</td>
<td>▶5.5.2 mpcci list ◄ on page 108</td>
</tr>
<tr>
<td>lmutil</td>
<td>Run the FLEXlm <code>lmutil [OPTIONS]</code> command delivered with MpCCI. Avoid running the lmutil command installed by codes other than MpCCI.</td>
<td>▶5.5.3 mpcci lmutil ◄ on page 109</td>
</tr>
<tr>
<td>monitor</td>
<td>Launch the MpCCI online monitor.</td>
<td>?? ?? ◄ on page ??</td>
</tr>
<tr>
<td>morpher</td>
<td>Launch the MpCCI grid morpher.</td>
<td>▶5.3.2 mpcci morpher ◄ on page 90</td>
</tr>
<tr>
<td>observe</td>
<td>Start the MpCCI file observer.</td>
<td>▶5.3.3 mpcci observe ◄ on page 93</td>
</tr>
<tr>
<td>pm</td>
<td>Launch the MpCCI project manager.</td>
<td>▶5.3.4 mpcci pm ◄ on page 94</td>
</tr>
<tr>
<td>Subcommand</td>
<td>Short Description</td>
<td>Discussed in</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>ps</strong></td>
<td>Unix <code>ps -ef</code> compatible ps for all platforms.</td>
<td>▶️ 5.6.10 mpcci ps &lt;/a&gt; on page 127</td>
</tr>
<tr>
<td><strong>ptoi</strong></td>
<td>Convert an MpCCI project file into anMpCCI input file.</td>
<td>▶️ 5.6.11 mpcci ptoi &lt;/a&gt; on page 128</td>
</tr>
<tr>
<td><strong>server</strong></td>
<td>Start the MpCCI server.</td>
<td>▶️ 5.6.12 mpcci server &lt;/a&gt; on page 129</td>
</tr>
<tr>
<td><strong>ssh</strong></td>
<td>Check/fix your ssh installation.</td>
<td>▶️ 5.5.4 mpcci ssh &lt;/a&gt; on page 110</td>
</tr>
<tr>
<td><strong>test</strong></td>
<td>Run some install/communication tests.</td>
<td>▶️ 5.5.5 mpcci test &lt;/a&gt; on page 111</td>
</tr>
<tr>
<td><strong>top</strong></td>
<td>Display process top list / Launch the taskmanager.</td>
<td>▶️ 5.6.13 mpcci top &lt;/a&gt; on page 135</td>
</tr>
<tr>
<td><strong>update</strong></td>
<td>Update check on the MpCCI installation.</td>
<td>▶️ 5.5.6 mpcci update &lt;/a&gt; on page 114</td>
</tr>
<tr>
<td><strong>vis</strong></td>
<td>Launch the MpCCI visualizer.</td>
<td>▶️ 5.3.5 mpcci vis &lt;/a&gt; on page 95</td>
</tr>
<tr>
<td><strong>where &lt;CMD&gt;</strong></td>
<td>Find all locations of the executable &lt;CMD&gt; in the PATH.</td>
<td>▶️ 5.4.6 mpcci where &lt;/a&gt; on page 105</td>
</tr>
<tr>
<td><strong>xterm</strong></td>
<td>Start a process inside an xterm.</td>
<td>▶️ 5.3.6 mpcci xterm &lt;/a&gt; on page 96</td>
</tr>
</tbody>
</table>
5.3 Starting MpCCI

The commands in this section start the different parts of MpCCI. Besides the actual coupling engine, MpCCI offers several helper applications.

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Short Description</th>
<th>Discussed in</th>
</tr>
</thead>
<tbody>
<tr>
<td>gui</td>
<td>Launch the MpCCI GUI.</td>
<td>5.3.1 mpCCI gui on page 89</td>
</tr>
<tr>
<td>morpher</td>
<td>Launch the MpCCI grid morpher.</td>
<td>5.3.2 mpCCI morpher on page 90</td>
</tr>
<tr>
<td>observe</td>
<td>Start the MpCCI file observer.</td>
<td>5.3.3 mpCCI observe on page 93</td>
</tr>
<tr>
<td>pm</td>
<td>Launch the MpCCI project manager.</td>
<td>5.3.4 mpCCI pm on page 94</td>
</tr>
<tr>
<td>vis</td>
<td>Launch the MpCCI visualizer.</td>
<td>5.3.5 mpCCI vis on page 95</td>
</tr>
<tr>
<td>xterm</td>
<td>Start a process inside an xterm.</td>
<td>5.3.6 mpCCI xterm on page 96</td>
</tr>
</tbody>
</table>
5.3.1 mpcci gui

Usage:
mpcci gui [OPTIONS] [project] [OPTIONS]

Synopsis:
’mpcci gui’ is used to launch the MpCCI GUI.

Options:
- chwd <PATH> Replace the symbolic working directory $(CWD)
  used inside the projectfile by the absolute
  pathname specified in the <PATH> argument.
- help This screen.
- new Start the GUI with a new project.
- nolic Do not check for a license before starting the GUI.
  This option may be used when no license is available
  but you would like to prepare a job.
- norsa Do not ask for ssh assistance if an
  rsa key file does not exist.

The MpCCI GUI is started with this command, i.e. the MpCCI GUI pops up, which is described in 4 Graphical User Interface.
5.3.2 mpcci morpher

Usage:

mpcci morpher [LAUNCH-OPTIONS] <model-name> [OPTIONS]

Synopsis:

'mpcci morpher' is used to start an MpCCI grid morpher daemon.

LAUNCH-OPTIONS:

- help This screen.
- lhost:[user@]hostname Launch the morpher on the remote host.
- lprec:d Use double precision version of the morpher.
- lprec:s Use single precision version of the morpher.

Grid morpher daemon 2.0, single precision
Build Dec 8 2008, 12:40:23
Copyright (c) 2004-2008, FhG SCAI.

License checkout...
Your license will expire in 7431 days.

Usage:

mpcci_morpher-32s.exe model [OPTIONS]

Options: #d=decimal, #f=float, #s=string, #c=char

Options to control the deformation of edges

- enocheck Skip all edge checks
- rlen[gth] #f #f Min/max allowed relative length change of an edge
  0.0 < Min length < 1.0
  1.0 < Max length < ?
- alen[gth] #f #f Min/max allowed absolute edge length

Options to control the deformation of faces

- fnocheck Skip all face checks
- rar[ea] #f #f Min/max allowed relative change of face area
  0.0 < Min area < 1.0
  1.0 < Max area < ?
Options to control the deformation of cells
- cnocheck Skip all cell checks
- rvolume #f #f Min/max allowed relative change of cell volume
  
  0.0 < Min volume < 1.0
  1.0 < Max volume < ?
- avolume #f #f Min/max allowed absolute cell volume
- caspect #f Max. allowed cell aspect ratio [1..50]
- cskeew #f Max. allowed skewness of cells [0.5..0.99]

Options to control the handling of boundaries
- dblayers #d Deformed boundary layer level: [0...512]
- fblayers #d Fixed boundary layer level: [0...512]
- fixregion #d Add non default fixed boundary regions
  #d is the region number
- fltregion #d Add non default floating boundary regions
  #d is the region number
- corner #f Angle to make node floating along boundaries
  #f [0.0...20.0] is the angle in degree
  the angle should not be larger than 5.0 deg

Options to control the morpher
- once #s #s Morph only once: infile.vrt outfile.vrt
- steps #d No. of steps for once morphing
- damp[ing] #f Level damping factor [0.0...0.5]
- diag[onal] Take care for shear in quad faces
  and hexahedral cells
- iter[ations] #d Max. no. of iterations: [0...?]
- tol[erance] #f Convergence tolerance: ]0.0...0.3]
- mrelax #f Morphing relaxation factor: ]0.0...2.0]
- local Avoid a global distribution of deformations
- minangle #f Min. angle allowed in faces and cells
  #f [5.0...30.0] is the angle in degrees

Options to control the smoother
- smooth #d No. of smooth sweeps
- srelax #f Smoothing relaxation factor: ]0.0...2.0]

Auxiliary options
<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-port</td>
<td>#d</td>
<td>Port number for socket communications</td>
</tr>
<tr>
<td>-debug</td>
<td></td>
<td>Save a -morph.vrt file after morphing</td>
</tr>
<tr>
<td>-help</td>
<td></td>
<td>This screen and exit</td>
</tr>
<tr>
<td>-output</td>
<td>#c</td>
<td>Information output level: [f</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f = full, verbose + displacements received</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v = verbose</td>
</tr>
<tr>
<td></td>
<td></td>
<td>q = quiet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d = default between quiet and verbose</td>
</tr>
<tr>
<td>-setup</td>
<td></td>
<td>List setup and exit</td>
</tr>
<tr>
<td>-scale</td>
<td>#f</td>
<td>Scale vertices read by factor #f</td>
</tr>
<tr>
<td>-wait</td>
<td>#d</td>
<td>Set waiting timeout in seconds, 0=never timeout</td>
</tr>
<tr>
<td>-noparamcheck</td>
<td></td>
<td>Do not check the limits of any parameter</td>
</tr>
</tbody>
</table>
5.3.3 mpcci observe

Usage:
mpcci observe [-help] file file file ....

Synopsis:
'mpcci observe' is used to launch the file observer. The file observer waits until the file exists and then displays the tail of the file in a separate xterm. Just try the observer.
5.3.4 mpcci pm

Usage:
    mpcci pm [-help]

Synopsis:
    'mpcci pm' is used to launch the project MpCCI manager.
    Just try the pm.

There are no further options:

The MpCCI Project Manager is a tool to keep an overview of the MpCCI projects in your home-directory. It offers possibilities to edit files, create, start and delete projects, etc.

The MpCCI Project Manager is described in \textit{\textgreater{}8 MpCCI Project Manager\textless{}}.
5.3.5 mpcci vis

Usage:
    mpcci vis [tracefile]

Synopsis:
    'mpcci vis' is used to launch the MpCCI tracefile visualiser.

Options:
    -help    This screen.

The MpCCI Visualizer is suitable for quickly checking whether the coupling process was successful. The coupling region, orphaned nodes and exchanged quantities can be checked to ensure that a coupling has really occurred. A description of the MpCCI Visualizer is given in 6 MpCCI Visualizer <.
5.3.6 mpcci xterm

Usage:
   mpcci xterm [[xterm]OPTIONS] -cmd <cmd ...>

Synopsis:
   'mpcci xterm' is in fact a wrapper for the standard X11 xterm command which is used to run a command <cmd ...> inside a new window, the xterm.

   Unlike the X11 'xterm [OPTIONS] -e <commandline>' the xterm is launched as a background process and the xterm remains opened after the command exited. 'mpcci xterm' is also available under MS Windows.

   The input stream to the <cmd ...> may be redirected and be read from the file -input <FILE>.

   In addition, if -log <FILE> is not empty or "-", a copy of the xterm output is logged into the file <FILE>.

   The command <cmd ...> may contain several arguments, therefore the item -cmd <cmd ...> must be the last one.

Options: (all options not listed below are passed to the xterm command):
-cmd <cmd ...> Command to fire up.
-display <DISPLAY> Set DISPLAY before.
-help This screen.
-home <HOME> Set the start directory to <HOME>.
-input <FILE> Redirect stdin to <FILE>.
-log <FILE> Redirect stdout via |tee <FILE>.
-rev Use reverse colors.
-title <TITLE> Define the title of the xterm.
5.4 Information and Environment

The commands in this section can be used to obtain information about MpCCI and the environment.

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Short Description</th>
<th>Discussed in</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch [-n]</td>
<td>Print the MpCCI base architecture without a newline or with a newline [-n] at the end and exit. This is different from <code>mpcci info arch</code> which prints the used architecture.</td>
<td>➤ 5.4.1 mpcci arch &lt; on page 98</td>
</tr>
<tr>
<td>doc</td>
<td>View the MpCCI documentation.</td>
<td>➤ 5.4.2 mpcci doc &lt; on page 99</td>
</tr>
<tr>
<td>env</td>
<td>Print out the environment used by MpCCI in various formats for further processing.</td>
<td>➤ 5.4.4 mpcci env &lt; on page 102</td>
</tr>
<tr>
<td>home [-n]</td>
<td>Print the MpCCI home directory without a newline or with a newline [-n] at the end and exit. This is in fact a shortcut for <code>mpcci info home</code>.</td>
<td>➤ 5.4.5 mpcci home &lt; on page 104</td>
</tr>
<tr>
<td>info</td>
<td>Print general information about MpCCI.</td>
<td>➤ 5.4.3 mpcci info &lt; on page 100</td>
</tr>
<tr>
<td>where &lt;CMD&gt;</td>
<td>Find all locations of the executable &lt;CMD&gt; in the PATH.</td>
<td>➤ 5.4.6 mpcci where &lt; on page 105</td>
</tr>
</tbody>
</table>
5.4.1 mpcci arch

mpcci arch is simply a shortcut for mpcci info arch, see also 5.4.3 mpcci info on page 100. mpcci arch prints the MpCCI architecture token of the platform on which it is run. A list of architecture tokens is given in the Release Notes.
5.4.2 mpcci doc

This command can be used to view MpCCI documentation, the available documentation can be listed with the option `-list`. The documentation is located in the "<MpCCI_home>/doc" directory and can be accessed directly as well.

Currently two kinds of documentation are available:

- `mpcci doc Flexlm_enduser_Guide` The FLEXlm End Users Guide
- `mpcci doc MpCCI doc` The MpCCI documentation.
5.4.3 mpcci info

The `mpcci info` command is used to obtain information on the environment that is used by MpCCI:

Usage:

```
mpcci info [-]OPTIONS
```

Synopsis:

`'mpcci info'` is used to print single tokens to `stdout` for the further use in scripts or `.bat` files...

Options:

- `--arch` MpCCI basic architecture token.
- `--arch32` MpCCI 32 bit architecture token.
- `--arch64` MpCCI 64 bit architecture token.
- `--build` Build date of the installed MpCCI release.
- `--help` This screen.
- `--home` MpCCI home path.
- `--java` Java command used by MpCCI.
- `--javaver` Java version of the java command.
- `--jobid` Jobid used by MpCCI.
- `--liba32` Pathname of the 32 bit libmpcci.a (if available).
- `--liba64` Pathname of the 64 bit libmpcci.a (if available).
- `--make` Make command used by MpCCI.
- `--patches` Patchlevel of the installed MpCCI release.
- `--perl` Pathname of the perl command used by MpCCI.
- `--perlinc` @INC list used by Perl.
- `--perlver` Version of the running Perl.
- `--release` Full MpCCI release token.
- `--remcp` Pathname of remote copy command used by MpCCI.
- `--remsh` Pathname of remote shell command used by MpCCI.
- `--rshtype` MpCCI server remote shell type used.
- `--userid` Userid used by MpCCI.
- `--version` X.Y version number of the installed MpCCI release.

The options `-arch`, `-arch32`, and `-arch64` list the architecture token of the machine. This token is used to distinguish between different hardware and operating systems. MpCCI has its own tokens for identification, which may differ from those used by the coupled codes. A list of the architecture tokens is given in the Release Notes.
Information on the releases and version number of MpCCI are obtained with \texttt{-build}, \texttt{-release} and \texttt{-version}.

The option \texttt{-userid} prints the user which is running MpCCI, i.e. your current user name. This can be useful to check user names on remote machines.

The job id can be obtained with \texttt{-jobid}, it is composed of the user name and a time-dependent value, thus changes with every call of MpCCI, but is kept during one run.

MpCCI uses external software installed on your system. Sometimes several versions are installed, thus it is important to know which was found by MpCCI:

\textbf{Java} is needed for the MpCCI GUI (see \texttt{4 Graphical User Interface} ), the full path to the Java executable is obtained with \texttt{-java}, the version of java which was found is obtained with \texttt{-javaver}. See also \texttt{III-10 Installing Java}.

\textbf{Perl} The path to the Perl executable is obtained with \texttt{-perl}, the version with \texttt{-perlver}, and \texttt{-perlinc} lists the \texttt{@INC} of Perl, which is a list of directories, which is searched for Perl modules. See also \texttt{III-9 Installing Perl}.

\textbf{rsh} \texttt{-rsh} gives the path to the remote shell rsh. There are two types of remote shells rsh and ssh. The type which is currently used by MpCCI is shown by \texttt{-rshtype}. The remote shell can be changed by setting the \texttt{MPCCI_RSHTYPE} to either type. See \texttt{2.7.4 Remote Shell and Remote Copy} for more information on remote shells.

\textbf{rcp} Gives the path to the rcp command, see also \texttt{2.7.4 Remote Shell and Remote Copy}. 
### 5.4.4 mpcci env

`mpcci env` is a lookup-function for environment variables. Its sole purpose is to print a list of all environment variables, which are relevant for MpCCI. More information on these environment variables is given in [2.3 Environment and Environment Variables](#). This command is useful for debugging.

The list can be formatted in various formats for use in shell scripts.

**Usage:**
```
mpcci env [-]FORMAT
```

**Synopsis:**
The MpCCI environment is set up at runtime and contains all informations about your system required by MpCCI.

'mpcci env' is used to print out the MpCCI environment in various formats for the further processing in shell scripts or in a MS Windows batch file.

'mpcci env' is used internally by MpCCI to fetch information about the MpCCI configurations on remote hosts.

**Examples:**
To save the MpCCI environment in a file which may be sent for support reasons type

```
"mpcci env pretty > mpcci_env.txt"
```

**Supported formats:**
- `-bash` UNIX bash format
- `-bat` MS-DOS .BAT format
- `-csh` UNIX csh format
- `-help` this screen
- `-java` Java properties format
- `-ksh` UNIX ksh format
- `-perl` Perl expression format
- `-plain` plain format
- `-pretty` human readable format (default)
- `-sh` UNIX sh format
- `-tcsh` UNIX tcsh format
-xml  XML similar format
5.4.5 mpcci home

Prints the full path of the MpCCI home directory. All files which belong to the MpCCI distribution are located in this directory. With

```
mpcci home
```

the path is given without a trailing newline character, whereas

```
mpcci home -n
```

yields the same path followed by a newline character.

The path to the MpCCI home directory is stored in the environment variable MPCCI_HOME during a run of MpCCI.
5.4.6 \texttt{mpcci where}

Usage:

\texttt{mpcci where [-help] command ...}

Synopsis:

'\texttt{mpcci where}' is used to list all commands found by investigating the "PATH" environment variable.

This is useful to find out whether MpCCI catches the correct executable file from the "PATH". Sometimes the "PATH" should be reordered to help MpCCI find the command really needed.

For some important commands MpCCI has build in alternative methods to find the correct executable. In this case the result of \texttt{where} does not show the executable selected by MpCCI.

Examples:

To find the location of the mpcci command please type

\texttt{"mpcci where mpcci"}
5.5 Installation and Licensing

The commands in this section are needed to check the validity of an installation, also including license information.

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Short Description</th>
<th>Discussed in</th>
</tr>
</thead>
<tbody>
<tr>
<td>license</td>
<td>Manage the license server and print license related information.</td>
<td>▶ 5.5.1 mpcci license &lt; on page 107</td>
</tr>
<tr>
<td>list</td>
<td>List information about the MpCCI installation and the supported codes.</td>
<td>▶ 5.5.2 mpcci list &lt; on page 108</td>
</tr>
<tr>
<td>lmutil</td>
<td>Run the FLEXlm lmutil [OPTIONS] command delivered with MpCCI. Avoid running the lmutil command installed by codes other than MpCCI.</td>
<td>▶ 5.5.3 mpcci lmutil &lt; on page 109</td>
</tr>
<tr>
<td>ssh</td>
<td>Check/fix your ssh installation.</td>
<td>▶ 5.5.4 mpcci ssh &lt; on page 110</td>
</tr>
<tr>
<td>test</td>
<td>Run some install/communication tests.</td>
<td>▶ 5.5.5 mpcci test &lt; on page 111</td>
</tr>
<tr>
<td>update</td>
<td>Update check on the MpCCI installation.</td>
<td>▶ 5.5.6 mpcci update &lt; on page 114</td>
</tr>
</tbody>
</table>
5.5.1 mpcci license

Usage:

mpcci license [-pN] [-tT] [-]OPTION ...

Synopsis:

‘mpcci license’ is used to manage the MpCCI licenses and to display detailed license information.

Options:

- all List all features of all available licenses.
- avail Brief display the no. of MpCCI sessions and processes available.
- clean Remove all local MpCCI license logfiles.
- expire Display the expiration date of the MpCCI license.
- files List all local license files defined.
- help This screen.
- info Print MpCCI related summary.
- local MpCCI feature overview for licenses on the local host.
- log Display the MpCCI related license logfiles.
- mpcci MpCCI feature overview for all hosts in a network.
- pN Redefines the port number N used with the FHGSCAI license server.
  The current port for FHGSCAI is "-p47000".
  If used this option must be the first on the commandline.
- restart Restart the FHGSCAI license server on the local host.
- servers List all defined license servers "[port]@host".
- start Start the FHGSCAI license server on the local host.
- stop Stop the FHGSCAI license server running on the local host.
- sysid MpCCI system ID used for generating a license file.
- tT Set license request timeout to T seconds.
  If used this option must be the first/second on the commandline.
- vars List the relevant environment variables xxx_LICENSE_FILE.
## 5.5.2 mpcci list

**Usage:**

```
mpcci list [-]OPTIONS
```

**Synopsis:**

'mpcci list' is used to list information about the MpCCI installation.

**Options:**

- `-archs` List all installed MpCCI architectures.
- `-batchsystems` List all installed batch systems.
- `-codes` List installed simulation codes found by MpCCI.
- `-help` This screen.
- `-hosts` List default hostlist entries.
- `-jobs` List all submitted batch jobs.
- `-writers` List all available MpCCI4 trace file writers.
5.5.3 mpcci lmutil

lmutil - Copyright (c) 1989-2006 Macrovision Europe Ltd. and/or Macrovision Corporation. All Rights Reserved.
usage: lmutil lmborrow -status
    lmutil lmborrow -clear
    lmutil lmborrow {all|vendor} dd-mmm-yyyy:[time]
    lmutil lmborrow -return [-c licfile] [-d display_name] [-fqdn] feature
    lmutil lmdiag [-c licfile] [-n]
    lmutil lmhostid [-internet|-user|-display|-n]
                    -hostname|-hostdomain|-string|-long]
    lmutil lminstall [-i infile] [-o outfile]
                    [-overfmt {2, 3, 4, 5, 5.1, 6, 7.1, 8}]
                    [-odecimal] [-maxlen n]
    lmutil lmnewlog [-c licfile] vendor new-file, or
    lmutil lmnewlog [-c licfile] feature new-file
    lmutil lmpath -status
    lmutil lmpath -override {all | vendor } path
    lmutil lmpath -add {all | vendor } path
    lmutil lmremove [-c licfile] feature user host display
    lmutil lmremove [-c licfile] -h feature host port handle
    lmutil lmreread [-c licfile] [-vendor name] [-all]
    lmutil lmswitchr [-c licfile] vendor new-file, or
    lmutil lmswitchr [-c licfile] feature new-file
    lmutil lmstat [-c licfile] [lmstat-args]
    lmutil lmswitch [-c licfile] vendor new-file, or
    lmutil lmswitch [-c licfile] feature new-file
    lmutil lmver flexlm_binary
    lmutil -help (prints this message)
    lmutil utility_name -help (display detailed usage information)
## 5.5.4 mpcci ssh

**Usage:**

```bash
mpcci ssh [-]OPTIONS
```

**Synopsis:**

'mpcci ssh' is used to check/update the secure shell (ssh) installation and settings. Some ssh settings should be done to avoid password requests for each MpCCI process launched on the local or remote hosts when using the secure shell (ssh/scp) set of commands for remote shell and remote file copy.

If you prefer to use the classic rsh/rcp set of commands please make sure that in your remote hosts file

"<Home>/.rhosts"

the local and all the remote hosts used by MpCCI are listed.

**Options:**

- `-all` Run all the checks below.
- `-env` Update the MpCCI environment variables in the ssh login environment file.
- `-help` This screen.
- `-keygen` Check/generate an ssh key for the local host to avoid password requests.
- `-mode` Test the ssh daemon configuration for "StrictModes".
5.5.5 mpcci test

Usage:

mpcci test [OPTIONS] hostname ... hostlist ... hostfile ... [-simple]

Synopsis:

'mpcci test' is used to perform various tests:
- MpCCI local and remote installation
- Remote host connections
- Perl environment
- etc.

The remote hostnames may be specified in various formats.

hostname: [user@]host
hostlist: [user@]host[:[user@]host[:...]]...
hostfile: filename with hostnames (like .rhosts)

For each host ...
- test whether the hostname is resolved by the DNS.
- test whether the host is alive and reachable.
- test possibility of rsh/ssh connections to the remote host.
- brief test on MpCCI installation on the remote host from the local host
- test server-client connection on ports 47111 ++
- write a protocol hostfile "mpcci.hostlist" which can be used as an MpCCI hostfile.

Examples:

Run a simple testcase delivered with the MpCCI installation:

on the local host:  mpcci test -simple
on various hosts :  mpcci test testhost mpcci@server.com -simple

Test the MpCCI access and communication with remote systems:

mpcci test [OPTIONS] [ [user@]host[:[user@]host[:...]] ] | hostfile ... ]
mpcci test fred@flintstone.family: wilma@flintstone.family
mpcci test fred@trex.farm bmw@stone-cars.manufactory
mpcci test flintstone.hostlist aquarium whale@waterworld.future:shark@zoo
mpcci test ~/.rhosts ~/.shosts brontosaurus@jurassic-park.vision
Options:

- `connect <port@hostname>`

  INTERNAL USE ONLY: for remote communication test.

- `help`

  This screen.

- `known`

  Test if each host is already known to ssh.

- `listen <port>`

  INTERNAL USE ONLY: for remote communication test.

- `modload`

  Load/compile all eventually MpCCI used Perl modules for a validation and exit. This test may help while you are integrating your code into MpCCI or to figure out if some common Perl modules are not installed on your system.

- `port <port>`

  Set the client/server communication port no.
  (default=47111).

- `remote <hostname>`

  For cases with VPN connections, IPSEC tunneling, and NAT translation:
  `<hostname>` is the name or dotaddress of the local host seen from the remote hosts. Redefine the local
hostname if necessary.
e.g.  -rem 192.168.2.16
or    -rem myvpnhost.company.com

-rsh

Test only rsh type connections.

-sharedfs

When launching the MpCCI server processes and distributed domain
solver applications on multiple hosts certain configuration files
need to be accessed from the remote hosts.
If there is a common (network) file system and your home directory
or the job directory is a shared directory there is no need to
distribute any file to the remote hosts.

-simple

Run a simple testcase delivered with MpCCI.
-simple must be the last option of the commandline.

-ssh

Test only ssh type connections.

-wait

Wait after each error or warning.
5.5.6 mpcci update

Usage:
   mpcci update [OPTION]

Synopsis:
   'mpcci update' is used to check or update the MpCCI installation from the MpCCI web-server.

   Before you download any file first run a check. If the list of files is large it may be faster to download a complete new version or patchlevel of MpCCI.

   Please make sure that you have write access rights to your local MpCCI installation directory and to all files before you start the download of new files.

Options:
   -count is used to count the number of updated files which might be required by your local MpCCI installation.

   -download is used to download the latest updated files.
   An MpCCI user name is required for the download.

   -list is used to query the MpCCI web-server for a list of updated files which might be required by your local MpCCI installation.
## 5.6 Job Control

During a coupled analysis it is often necessary to keep control of the different jobs which are started. The commands in this section help with starting, controlling and interrupting calculations.

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Short Description</th>
<th>Discussed in</th>
</tr>
</thead>
<tbody>
<tr>
<td>backup &lt;file ...&gt;</td>
<td>Make a backup copy of a list of files.</td>
<td>▷ 5.6.1 mpcci backup ▷ on page 116</td>
</tr>
<tr>
<td>batch &lt;project&gt;</td>
<td>Start an MpCCI batch job with project file &lt;project&gt;.</td>
<td>▷ 5.6.2 mpcci batch ▷ on page 117</td>
</tr>
<tr>
<td>clean</td>
<td>Remove all files from the temporary MpCCI directory &lt;Home&gt;/mpcci/tmp.</td>
<td>▷ 5.6.8 mpcci clean ▷ on page 124</td>
</tr>
<tr>
<td>kill</td>
<td>Platform independent process kill based on command line pattern matching.</td>
<td>▷ 5.6.9 mpcci kill ▷ on page 125</td>
</tr>
<tr>
<td>ps</td>
<td>Unix ps -ef compatible ps for all platforms.</td>
<td>▷ 5.6.10 mpcci ps ▷ on page 127</td>
</tr>
<tr>
<td>ptoi</td>
<td>Convert an MpCCI project file into an MpCCI input file.</td>
<td>▷ 5.6.11 mpcci ptoi ▷ on page 128</td>
</tr>
<tr>
<td>server</td>
<td>Start the MpCCI server.</td>
<td>▷ 5.6.12 mpcci server ▷ on page 129</td>
</tr>
<tr>
<td>top</td>
<td>Display process top list / Launch the taskmanager.</td>
<td>▷ 5.6.13 mpcci top ▷ on page 135</td>
</tr>
</tbody>
</table>
5.6.1 mpcci backup

Usage:
   mpcci backup file [file [file ...]]

Synopsis:
   'mpcci backup' is used to copy one or more files into backup files adding an additional free suffix ".bakNNN". This is a system independent backup copy command.

Examples:
   mpcci backup path/to/file.ext => path/to/file.ext.bak000
5.6.2 mpcci batch

In the usage output you can see if a queuing system has been detected by MpCCI. To access the specific queuing system command help, execute `mpcci batch <Batch Name>`.

Usage:

```
mpcci batch [OPTIONS] <projectname>
mpcci batch TORQUE ...
mpcci batch OPENPBS ...
mpcci batch N1GE ...
mpcci batch LSF ...
mpcci batch PBS ...
mpcci batch PBSPRO ...
mpcci batch GLOBUS ...
mpcci batch LOADLEVELER ...
```

Synopsis:

'mpcci batch' is used to start an MpCCI job in batch mode.
'mpcci batch TORQUE' is used to control a TORQUE batch job.
'mpcci batch OPENPBS' is used to control a OPENPBS batch job.
'mpcci batch N1GE' is used to control a N1GE batch job.
'mpcci batch LSF' is used to control a LSF batch job.
'mpcci batch PBS' is used to control a PBS batch job.
'mpcci batch PBSPRO' is used to control a PBSPRO batch job.
'mpcci batch GLOBUS' is used to control a GLOBUS batch job.
'mpcci batch LOADLEVELER' is used to control a LOADLEVELER batch job.

Options:

- `chwd <PATH>` Replace the symbolic working directory $(CWD) used inside the projectfile by the absolute pathname specified in the <PATH> argument.
- `help` This screen.
- `globus` GLOBUS batch system control.
- `loadleveler` LOADLEVELER batch system control.
- `lsf` LSF batch system control.
- `n1ge` N1GE batch system control.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>openpbs</td>
<td>OPENPBS batch system control.</td>
</tr>
<tr>
<td>pbs</td>
<td>PBS batch system control.</td>
</tr>
<tr>
<td>pbspro</td>
<td>PBSPRO batch system control.</td>
</tr>
<tr>
<td>torque</td>
<td>TORQUE batch system control.</td>
</tr>
</tbody>
</table>

See §3.4.4 Coupled Analysis in Batch Mode for details.
5.6.3 mpcci batch LSF

Usage:

mpcci batch LSF submit [BATCH-OPTIONS] <projectname>
mpcci batch LSF status <jobid>
mpcci batch LSF kill <jobid>

Synopsis:

'mpcci batch LSF' is used to manage an MpCCI job running under LSF.

Batch commands:

-help                     This screen.
kill <jobid>              Kill a LSF batch job.
status <jobid>            Display a LSF batch job status.
submit [BATCH-OPTIONS] <projectname> Submit a LSF MpCCI job.

See \textit{\textgreater} \textbf{3.4.4 Coupled Analysis in Batch Mode} \textless{} for details.
5.6.4 mpcci batch PBS

The following PBS family queuing system are also available and supported by MpCCI:

- PBS
- OpenPBS
- PBSPro
- Torque

**Usage:**
```
mpcci batch PBS submit [BATCH-OPTIONS] <projectname>
mpcci batch PBS status <jobid>
mpcci batch PBS kill <jobid>
```

**Synopsis:**
'mpcci batch PBS' is used to manage an MpCCI job running under PBS.

**Batch commands:**
- `-help`   This screen.
- `kill <jobid>` Kill a PBS batch job.
- `status <jobid>` Display a PBS batch job status.
- `submit [BATCH-OPTIONS] <projectname>` Submit a PBS MpCCI job.

See §3.4.4 Coupled Analysis in Batch Mode for details.
5.6.5 mpcci batch N1GE

The following Sun Grid Engine family queuing system are also available and supported by MpCCI:

- SGE
- N1GE
- SGEEE

Usage:

mpcci batch N1GE submit [BATCH-OPTIONS] <projectname>
mpcci batch N1GE status <jobid>
mpcci batch N1GE kill <jobid>

Synopsis:

'mpcci batch N1GE' is used to manage an MpCCI job running under N1GE.

Batch commands:

- help This screen.
  kill <jobid> Kill a N1GE batch job.
  status <jobid> Display a N1GE batch job status.
  submit [BATCH-OPTIONS] <projectname> Submit a N1GE MpCCI job.

See &gt;3.4.4 Coupled Analysis in Batch Mode &lt; for details.
5.6.6 mpcci batch LoadLeveler

Usage:

```
mpcci batch LOADLEVELER submit [BATCH-OPTIONS] <projectname>
mpcci batch LOADLEVELER status <jobid>
mpcci batch LOADLEVELER kill <jobid>
```

Synopsis: 'mpcci batch LOADLEVELER' is used to manage an MpCCI job running under LOADLEVELER.

Batch commands:

```
-help

kill <jobid> Kill a LOADLEVELER batch job.
status <jobid> Display a LOADLEVELER batch job status.
submit [BATCH-OPTIONS] <projectname> Submit a LOADLEVELER MpCCI job.
```

See 3.4.4 Coupled Analysis in Batch Mode for details.
5.6.7 mpcci batch GLOBUS

Usage:
- `mpcci batch GLOBUS submit [BATCH-OPTIONS] <projectname>`
- `mpcci batch GLOBUS status <jobid>`
- `mpcci batch GLOBUS kill <jobid>`

Synopsis:
'mpcci batch GLOBUS' is used to manage an MpCCI job running under GLOBUS.

Batch commands:
- `-help` This screen.
- `kill <jobid>` Kill a GLOBUS batch job.
- `status <jobid>` Display a GLOBUS batch job status.
- `submit [BATCH-OPTIONS] <projectname>` Submit a GLOBUS MpCCI job.

See ▷3.4.4 Coupled Analysis in Batch Mode◁ for details.
5.6.8 mpcci clean

Usage:
   mpcci clean

Synopsis:
   'mpcci clean' is used to remove ALL files from the temporary MpCCI directory which is currently

   "<Home>/.mpcci/tmp".

Options:
   -help This screen.
5.6.9 mpcci kill

Usage:

    mpcci kill [OPTIONS] <pattern1> <pattern2> ...

Synopsis:

'mpcci kill' is used to kill processes whose commandline matches a character pattern. The commandline is read via the "ps -ef" command which is also available under Microsoft Windows.

The <patterns> are strings of printable characters, e.g. a piece of the name or an option of a running program. If the <pattern> is a decimal number it will be assumed that this number is a true process id (PID) and not a command line pattern.

The options -i, -v and -q (see below) are mutually exclusive. The last option on the commandline overwrites the others.

'mpcci kill' is self protecting in case of pattern matching: You may kill your whole family (childs, brother, sisters, aunts and uncles), but never yourself and your parents and grandparents ...

'mpcci kill' is also used internally by MpCCI to kill a group of MpCCI processes on the local or remote hosts.

Examples:

    Kill all MpCCI related processes: "mpcci kill mpcci_"
    Kill process with PID 1234:       "mpcci kill 1234"

Options:

- \f <prefix> Special definition of process ids.
  Collects all files with name

    "<prefix>).<PID>"

  where <PID> is a number and is interpreted as a process id. The files "<prefix>).<PID>" are deleted afterwards. Typically, they are only indicator files of size 0 which were
created by 'touch'. Their only purpose is the definition of PID’s via their suffix.

- **-help**  
  This screen.

- **-i**  
  Interactive confirmation (default): Individually confirm each process found.

- **-q**  
  Quiet kill, no confirmation and no printout.

- **-r**  
  Recursively kill all processes specified via PID or pattern AND all their descendants (children, grandchildren, ...).

- **-s <SIG>**  
  Send signal <SIG> to the processes instead of killing them with signal 9 (KILL).

- **-v**  
  Verbose printout, no confirmation: Print a list of process ids before killing them.
5.6.10 mpcci ps

Usage:
  mpcci ps [PATTERNS]

Synopsis:
  'mpcci ps' is a platform independent Unix style

  "ps -ef [|grep PATTERN]"

  'mpcci ps' is also available under MS Windows!

Examples:
  List all processes, run ps -ef: "mpcci ps"
  List all MpCCI related processes: "mpcci ps mpcci"
  List all processes under your account: "mpcci ps <user name>"

Options:
  -help   This screen.
5.6.11 mpcci ptoi

Usage:

mpcci ptoi [OPTIONS] <file> <file> ...

Synopsis:

'mpcci ptoi' is used to convert an MpCCI project file (suffix .csp) into an MpCCI inputfile (suffix .cci). The name of the new MpCCI input file is the same than of the project file plus a .cci suffix.

After a successful conversion you may use the created input file to add further options which are not supported by the MpCCI GUI and to launch the MpCCI server with the modified input file.

Options:

-help This screen.

-overwrite Allow overwrite of an existing MpCCI inputfile.
5.6.12 mpcci server

Usage:

mpcci server [OPTIONS] MpCCI-inputfile|projectfile

Synopsis:

‘mpcci server’ is used to start the MpCCI server and control processes on the local host or as an alternative distributed on multiple hosts in a network.

Options:

-32

   Prefer to run the 32 bit versions of the server and control processes. This option should be used if the default is 64 bit mode but the processor architecture supports 32 bit executables only. If 32 bit executables are not available this option is silently ignored.

-64

   Try to run the 64 bit versions of the server and control processes. This option should be used if the default is 32 bit mode but the 64 bit processor performance is better than 32 bit. If 64 bit executables are not available this option is silently ignored.

-control

   Start the MpCCI control process in parallel to the servers.

   The control process gets automatically started with the -preview option used for debugging (see below).

-display <DISPLAY>

   Redirect the xterm output to the given display <DISPLAY>.

Microsoft Windows with local MpCCI/xterm emulation:
This X11 feature is ignored. A new console window will always pop up on your local machine and the MPICH daemon collects the server output in this single console.

The current setting is DISPLAY=localhost:12.0

-doble
Launch the double precision version of the server.

-help
Prints this screen only and exits.

-hostalias <hostname>
For cases with VPN connections, IPSEC tunneling and NAT translation. <hostname> is the name or dotaddress of the local host seen from the remote hosts.

-hostfile <hostfile>
Distribute the server and control processes on multiple hosts. The <hostfile> contains pairs of hostname and optional username entries

hostname [username]

like the .rhosts file used by the remote shell commands rsh or ssh. Hosts specified via the -hostfile option are merged with hosts specified via the -hostlist option.

-hostlist <hostlist>
Distribute the server and control processes on multiple hosts. The <hostlist> is just a list of hostnames separated by a :

[user@]host1[:[user@]host2] [:[user@]host3]....
Hosts specified via the -hostlist option are merged with hosts specified via the -hostfile option.

-log

Log the output of the server and control processes in files named

<prefix>.stdout.<processrank>

The <prefix> is specified with the option -prefix (see below).

-nocodeid

Relevant in the cases where a code has no code id resp. code name. The server ignores the code name (=code idstring) and assumes the codes are started resp. get a connection to the MpCCI server in the same order they appear in the MpCCI inputfile.

-nolocal

Relevant under UNIX only, ignored under Microsoft Windows. Even if a list of hosts is specified via the -hostlist or -hostfile options per default the first MpCCI server/control process is started on the local host. With this option the first process is started on the first host from the hostlist, whether local or remote.

-port <N>

Specify an alternative TCP/IP port number <N> for the initial communication between the leading MpCCI server and the clients.

The default port no. is 47111.

-prefix <prefix>
Specify a different name prefix for MpCCI output files. The output file names are `<prefix>.something.<processrank>`. The default `<prefix>` is "mpccirun".

```
-preview
```

Skips the MpCCI internal neighborhood search after the geometry was send to the server. The server then writes all geometry data into the given tracefile and exits the coupled computation. The tracefile can be used to visualize the meshes in the coupling regions.

```
-rsh
```

Use the (rsh/rcp) set of commands instead of the secure shell. The current default setting is defined by the environment variable

```
MPCCI_RSHTYPE=rsh
```

```
-sharedfs
```

When launching the MpCCI server processes and distributed domain solver applications on multiple hosts specified via the `-hostxxxx` options certain configuration files need to be accessed from the remote hosts. If there is a common (network) file system and your home directory or the job directory is a shared directory there is no need to distribute any file to the remote hosts with the remote copy command

```
"/usr/bin/rcp"
```

If this options is not specified, MpCCI copies all required files (for MpCCI and for each code) to the remote hosts automatically.

```
-single
```

Launch the single precision version of the server.
-ssh

Use the secure shell set of commands (ssh,scp) instead of the standard settings (rsh,rcp).
The current default setting is defined by the environment variable

MPCCI_RSHTYPE=rsh

-tmpdir <dir>

Dependend on the platform MpCCI creates small temporary files (shell scripts or .BAT files with few lines only) and logging output.

The default -tmpdir is defined by the environment variable

MPCCI_TMPDIR=<Home>/mpcci/tmp

and should be located on your local system.

With -tmpdir you specify an alternative directory for temporary files.

-xterm

All Unix systems:
Run the control and all server processes in separate X11 xterms.

Microsoft Windows with local MpCCI/xterm emulation:
A new console window is opened and the mpich daemon collects the output even from remote systems in the new console window.

-xtermbg <color>

Set xterm background color, e.g. "-xtermbg blue".

-xtermfg <color>

Set xterm foreground(text) color, e.g. "-xtermfg green".
Microsoft Windows with local MpCCI/xterm emulation:
This xterm color is only valid for the master process and the
MPICH daemon sets the text color automatically.

-xtermopt <opt>

Add additional X11 xterm options <opt>, e.g.

"-geometry WxH -132".

Please use only X11 xterm options valid for the local xterm
command.

Microsoft Windows with local MpCCI/xterm emulation:
All X11 xterm options may also be used since the MpCCI/xterm
emulation simply ignores any unsupported option.
5.6.13 mpcci top

Usage:
    mpcci top

Synopsis:
    "mpcci top" displays the process list:
    
    If it is available the Unix 'top' command is launched in a separate xterm.
    On Windows systems the taskmanager may be launched if 'top' is not available.

Options:
    -help This screen.
6 MpCCI Visualizer

The MpCCI Visualizer is suitable for quickly checking whether the coupling process was successful. The coupling region, orphaned nodes and exchanged quantities can be checked to ensure that a coupling has really occurred.

6.1 Using the MpCCI Visualizer

6.1.1 Data Flow

During the MpCCI coupling process, the data exchange between the coupled codes can be observed. The exchanged data can be collected by an additional control process, who writes the data into a tracefile, which can finally be read directly by the MpCCI Visualizer for ".ccv". For the MpCCI Visualizer for ".vtfx" the tracefile has to be converted into the VTFx format.

The tracefile is only written, if the control process is started. This is achieved by selecting the option Start additional control process in the Go Step of the MpCCI GUI. See IV-2.7 Go Step – Starting Servers and Codes for a description of the Go Step and “Getting Started” for a general description of the coupling process.

The name of the tracefile is set in the Edit Step of the MpCCI GUI. The default name is "tracefile.ccv", which is saved in the same directory as the corresponding project (".csp") file.
More and more data is added to the tracefile during the coupling process. The file can already be opened before the process is finished to check data during the process. The visualizer can only read data from the tracefile, not manipulate or write data.

### 6.1.2 Supported Platforms

The MpCCI Visualizer is included in the MpCCI downloads. The MpCCI Visualizer for ".ccv" is available for almost all platforms, MpCCI runs on. The MpCCI Visualizer for ".vtfx" is at present only available on Microsoft Windows and Linux platforms. Please see the Release Notes for a list of platforms.

### 6.1.3 Starting the Visualizer

The MpCCI Visualizer can be started from the command-line with

```
mpcci vis [<tracefile>]
```

where an optional filename can be given as well as from the MpCCI GUI menu by selecting Tools→Visualizer.

If no filename is given, the visualizer starts without opening a file.

If available the MpCCI Visualizer for ".vtfx" will be chosen. Then a program is automatically called which converts the "file.ccv" into the VTFx format ("file.vtfx"). But if the VTFx viewer isn't available on the current platform MpCCI Visualizer for ".ccv" will be chosen.

The command `mpcci vis -h` or `mpcci vis -help` shows a short help text.

### 6.2 MpCCI Visualizer for .ccv

After starting the MpCCI Visualizer for ".ccv", the Control Window pops up. If a file is selected, the Viewer Window will be opened as well.

The Control Window is for selecting the data to be displayed, whereas the Viewer Window offers controls to change the way in which the data is displayed including scaling and moving the meshes, showing vertices, edges or cells etc.

#### 6.2.1 Control Window

The Control Window is shown in Figure 2. At the top of the window, the name of the tracefile is displayed, here "wing.ccv".

There are only three menu entries:

- **File→Open** Opens the file dialog window. Select a tracefile and press the Open button to read the information from the tracefile.
Figure 2: Control Window of the MpCCI Visualizer for ".ccv"

**File→Exit**  Exits the MpCCI Visualizer.

**Help→About**  Shows the version information.

Below the menu, there are several buttons:

**Context**  Not used for standard coupling procedures.

**Step**  The time step of the computation. Information of the exchanged Quantities is saved in every time step. Select the step you want to see, the display in the Viewer Window should change accordingly. The number of steps which were found in the tracefile is shown at the right, in Figure 2, the coupling steps range from 0 to 8 and step 3 is selected.

<table>
<thead>
<tr>
<th>Button</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[↓]</td>
<td>Play backward, i.e. the step is decreased continuously.</td>
</tr>
<tr>
<td></td>
<td>[Space]</td>
<td>Stop, i.e. the play process is stopped at the current step.</td>
</tr>
</tbody>
</table>
### button key description

<table>
<thead>
<tr>
<th>Button</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>✈️</td>
<td>➔</td>
<td>Play forward, i.e. the step is increased continuously.</td>
</tr>
<tr>
<td>🎁</td>
<td>➕</td>
<td>Go to the next step.</td>
</tr>
<tr>
<td>❃</td>
<td>➖</td>
<td>Go to the previous step.</td>
</tr>
<tr>
<td>∞</td>
<td>8</td>
<td>Loop forever, i.e. if the “play forward”-button is pressed, the loop will not stop at the last step, but the step is decreased again (play back) up to step 0, then the step is increased. The MpCCI Visualizer continues playing until the stop button is pressed.</td>
</tr>
<tr>
<td>☀️</td>
<td></td>
<td>Open/close the Viewer Window.</td>
</tr>
</tbody>
</table>

The list of quantities consists of some entries, which are always present and the quantities which are exchanged in the coupling process. (Strictly speaking, not all entries are really quantities). If an item is not bold, no data is available for that quantity. In Figure 2, the tracefile contains no orphaned nodes and three data-sets, `mesh_change`, `rotation1/rotation2` and `force1/force2`.

**Codes** Display the codes in different colors. You can then select the codes with the pick [✓] and unpick [☐] buttons, see below.

**Processes** Display the processes of each code in different colors. This allows to see the partitions of the codes in parallel applications.

**Meshes** Display the meshes in different colors.

**Partitions** Display the partitions in different colors.

**Orphaned Nodes** Displays orphaned nodes. Normal nodes have the value 0, whereas orphaned nodes have the value 1. If no orphaned nodes are present, the entry in the quantity list is not bold and the color bar at the bottom of the Viewer Window only covers values very close to zero.

**Exchanged Quantities** Below the standard entries, the entries for the quantities follow. This depends on the quantities selected in the MpCCI GUI, see ▶ IV-2.5 Coupling Step – Definition of Coupling Regions and Quantities ◀. If different quantity names are used by the codes for the same quantity, both names are displayed together, separated by a “/”.

### 6.2.2 Viewer Window

The Viewer Window is shown in Figure 3. The big display shows the selected quantities on the mesh, here an aircraft wing\(^1\). The color bar at the bottom of the window maps the colors to the numerical values of

---

\(^1\)Special thanks to DLR German Aerospace Center, Institute of Aeroelasticity in Göttingen for providing the aircraft wing data-set
Figure 3: Viewer Window of the MpCCI Visualizer for ".ccv"

the selected quantity. The status bar at the bottom of the Viewer Window shows the displayed quantity at the left, “viewing ’mesh_change’ step 3” in Figure 3 and the sums of the numbers of elements at the right, i.e. both meshes together have 109 264 elements and 54 728 nodes in Figure 3.

The menu entries of the Viewer Window are:

File→Close Closes the Viewer Window. The window can be reopened with the button in the Control Window.

Options→Range... Opens the range control box. In the range control box, minimum and maximum values for the coloring of the models can be selected. All areas with values beyond the selected limits are colored in blue (beyond minimum) or red (beyond maximum). This menu option is disabled for Codes, Processes, Meshes and Partitions.
Help→Keys... Opens a help window, which explains the use of keyboard and mouse (also explained below).

Help→About Shows the MpCCI Visualizer version information.

The mouse buttons can be used directly within the display area of the Viewer Window with the following functions:

**Left mouse button – Rotate** Rotate the objects by moving the mouse while pressing the left mouse button.

**Middle mouse button – Move** Objects move together with the mouse pointer while the middle button is pressed. If you have a wheel mouse, you can also press the wheel to move objects. Wheel rotation has no effect.

**Right mouse button – Resize** If you move the mouse upward while keeping the right mouse button pressed, objects will be enlarged. Moving down scales objects down.

**Double-click – Select** Double-click any mouse button to select a code, mesh or partition. Same as the button. The actual selection is displayed at the top left of the viewing area.

![Surface, Wireframe, Points]

**Figure 4:** Display modes of the MpCCI Visualizer for ".ccv"

In the toolbar, a number of buttons are available. The following list describes the functionality of these buttons and corresponding keys. The keys are also marked at the top left corner of the buttons. Toggle options are marked with “(on/off)”, which means the option can be switched off by pressing the button again. In addition to the keys below, the Control Window keys can be used in the Viewer Window as well.
Figure 5: Data display options of the MpCCI Visualizer for "*.ccv"

<table>
<thead>
<tr>
<th>Button</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat shading</td>
<td>F</td>
<td>Surface representation. Show the surfaces of the meshes.</td>
</tr>
<tr>
<td>Gouraud shading</td>
<td>G</td>
<td>Wireframe representation. Show only the edges of the meshes.</td>
</tr>
<tr>
<td>Centers</td>
<td>C</td>
<td>Point representation. Show only the vertices of the mesh.</td>
</tr>
<tr>
<td>Vectors</td>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

Display modes (Figure 4):

- **S** Surface representation. Show the surfaces of the meshes.
- **W** Wireframe representation. Show only the edges of the meshes.
- **P** Point representation. Show only the vertices of the mesh.

Shading (Figure 5):

- **F** Flat shading. Colors may change suddenly at element borders.
- **G** Gouraud shading. Colors are smoothed at element borders.

Data options (Figure 5):

- **C** Cell centers. Display points at the centers of cells. (on/off)
- **V** Vector data. Display vector data as vectors. (on/off)
- **D** Continuous data. Display continuous data on vertices, edges or surfaces. If this option is switched off, the mesh is not drawn. (on/off)
## Button Key Description

### Relative motion:
The mesh of the second code is moved relatively to that of the first.

<table>
<thead>
<tr>
<th>Button</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X →</td>
<td>X</td>
<td>+x move. Move in positive $x$-direction.</td>
</tr>
<tr>
<td>X ←</td>
<td>Shift+X</td>
<td>−x move. Move in negative $x$-direction.</td>
</tr>
<tr>
<td>Y ↑</td>
<td>Y</td>
<td>+y move. Move in positive $y$-direction.</td>
</tr>
<tr>
<td>Y ↓</td>
<td>Shift+Y</td>
<td>−y move. Move in negative $y$-direction.</td>
</tr>
<tr>
<td>Z /</td>
<td>Z</td>
<td>+z move. Move in positive $z$-direction.</td>
</tr>
<tr>
<td>Z \</td>
<td>Shift+Z</td>
<td>−z move. Move in negative $z$-direction.</td>
</tr>
<tr>
<td>∅</td>
<td>0</td>
<td>Reset motion. Move meshes to their actual positions. The key is the digit “zero”, not the letter “o”!</td>
</tr>
</tbody>
</table>

### Selection:
Pick a code/mesh/partition. This button can be pressed several times: First it is for selecting the code. Pressing again allows you to select the mesh . . . . A selection can also be made by double-clicking any mouse button.

<table>
<thead>
<tr>
<th>Button</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>U</td>
</tr>
</tbody>
</table>

“unpick”. Cancel the last selection, i. e. if a mesh was selected previously, pressing the unpick-button will show both meshes again.

### Miscellaneous:
Toggle background color: white, gray or black.

<table>
<thead>
<tr>
<th>Button</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>R</td>
</tr>
</tbody>
</table>

Reset view. Resizes the meshes to fit into the window and redraws the screen.
6.3 MpCCI Visualizer for VTFx

At present only an online documentation is available for the MpCCI Visualizer for "vtfx."
7 MpCCI Grid Morpher

Please see $\triangleright 5.3.2$ mpcci morpher $\triangleright$ for more information.
The MpCCI Project Manager requires the installation of Perl/Tk, which is not needed for other parts of MpCCI. Perl/Tk is a separate module for Perl, which can be downloaded from [www.cpan.org](http://www.cpan.org). See also III-9 Installing Perl for information on the installation of Perl.
The **MpCCI Project Manager** is a tool to keep an overview of the **MpCCI** projects in your home-directory. It offers possibilities to edit files, create, start and delete projects, etc.

If you start the **MpCCI Project Manager** with the command

```
mpcci pm
```

the **MpCCI Project Manager** scans your home-directory for **MpCCI** files, i.e. "*\.csp"-files, the **MpCCI** project files which contain the definitions of the coupling process (see \[\text{\ref{MpCCI Project Files}}\]) and "*\.cci"-files, the **MpCCI** input files, which are read by the **MpCCI** server (see \[\text{\ref{MpCCI Server Input Files}}\]). These files are listed in the main window, which is shown in Figure 1. The list consists of three columns. In the first, the name of the project file is displayed, the second shows the coupled codes (Figure 1 shows only dummy codes). The location, i.e. the path to the codes, is shown in the last column.

Clicking on a file name selects the file. A selected file is highlighted in the main window as "flap.csp" in Figure 1. Double clicking on a file name will open a separate window, which shows the context of the file in text form.

The **MpCCI Project Manager** offers several functions in the button bar on the right hand side of the main window. If you move the mouse pointer over a button, the button function is shown in the status bar at the bottom of the main window.

- **New** Start the **MpCCI** GUI without any project, i.e. start a new project. Setting up a new project is described in Getting Started.
- **Gui** Start the **MpCCI** GUI with the selected project file.
- **Batch** Run the selected project in batch mode. This is the same as entering `mpcci batch <filename>`, see also \[\text{\ref{Job Control}}\]. This allows to quickly rerun a project without starting the **MpCCI** GUI.
- **Server** Start the **MpCCI** server process with the selected **MpCCI** input file. This only starts the server, thus the coupled codes must be started separately.
- **Observe** Observe the output files of a running simulation. This feature is currently not available.
- **Edit** Edit the selected project file with the editor “vi”. This only works if “vi” is available on your computer, which is the case on most **UNIX/Linux** platforms.
- **Delete** Delete the selected project file and remove it from the projects list. This really removes the file from the hard disk, so be careful using this button!
- **Save** Store the current project list. The list is saved to the file "HOME/\.mpcci/Projects".
- **Rescan** Rebuild the project list by scanning the file system. The **HOME** directory is searched again for project files and the list is updated.
Environment  Display the MpCCI related environment variables in a separate window. This is the same as entering `mpcci env` in the shell, see §2.3 Environment and Environment Variables for a description of the environment.

Processes  Display a list of all processes running on the local machine in a separate window. This is the same as entering `mpcci ps` in the shell.

Exit  Exit the Project Manager.
VI Codes Manual
VI Codes Manual – Contents

1 Overview
  1.1 Common MpCCI Subcommands for Simulation Codes ......................... 10
  1.2 Unit Systems .................................................. 12

2 Abaqus
  2.1 Quick Information ............................................... 13
     2.1.1 Supported Coupling Schemes .................................. 13
     2.1.2 Supported Platforms and Versions ............................ 13
     2.1.3 References .................................................... 14
     2.1.4 Adapter Description ........................................... 14
  2.2 Coupling Process ................................................ 15
     2.2.1 Model Preparation ............................................ 15
     2.2.2 Models Step .................................................. 15
     2.2.3 Coupling Step ................................................ 16
     2.2.4 Go Step ....................................................... 17
     2.2.5 Running the Computation ................................... 18
     2.2.6 Post-Processing ............................................... 20
  2.3 Code-Specific MpCCI Commands ...................................... 21
  2.4 Code Environment ................................................ 22
     2.4.1 Prerequisites for a coupled simulation ...................... 22
  2.5 Code Adapter Reference ........................................... 23
     2.5.1 Patched Input File .......................................... 23
  2.6 Trouble shooting, open issues and known bugs ....................... 24

3 ANSYS
  3.1 Quick Information ................................................. 25
     3.1.1 Supported Coupling Schemes ................................ 25
     3.1.2 Supported Platforms and Versions .......................... 25
     3.1.3 References .................................................... 25
     3.1.4 Adapter Description ........................................... 26
  3.2 Coupling Process ................................................ 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Model Preparation</td>
<td>27</td>
</tr>
<tr>
<td>3.2.2</td>
<td>APDL Script</td>
<td>30</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Models Step</td>
<td>34</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Coupling Step</td>
<td>35</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Go Step</td>
<td>37</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Running the Computation</td>
<td>38</td>
</tr>
<tr>
<td>3.3</td>
<td>Code-Specific MpCCI Commands</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>Code Adapter Reference</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>FINE/Hexa</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>Quick Information</td>
<td>41</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Supported Coupling Schemes</td>
<td>41</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Supported Platforms and Versions</td>
<td>41</td>
</tr>
<tr>
<td>4.1.3</td>
<td>References</td>
<td>41</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Adapter Description</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Coupling Process</td>
<td>42</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Model Preparation</td>
<td>42</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Models Step</td>
<td>44</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Coupling Step</td>
<td>44</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Go Step</td>
<td>45</td>
</tr>
<tr>
<td>4.2.5</td>
<td>Running the Computation</td>
<td>45</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Post-Processing</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>Code-Specific MpCCI Commands</td>
<td>46</td>
</tr>
<tr>
<td>4.4</td>
<td>Code Environment</td>
<td>46</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Prerequisites for a coupled simulation</td>
<td>46</td>
</tr>
<tr>
<td>4.5</td>
<td>Code Adapter Reference</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>FINE/Turbo</td>
<td>48</td>
</tr>
<tr>
<td>5.1</td>
<td>Quick Information</td>
<td>48</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Supported Coupling Schemes</td>
<td>48</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Supported Platforms and Versions</td>
<td>48</td>
</tr>
<tr>
<td>5.1.3</td>
<td>References</td>
<td>48</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Adapter Description</td>
<td>48</td>
</tr>
</tbody>
</table>
## 5.2 Coupling Process
- 5.2.1 Model Preparation .......................................................... 49
- 5.2.2 Models Step ................................................................. 51
- 5.2.3 Coupling Step ............................................................... 51
- 5.2.4 Go Step ........................................................................ 52
- 5.2.5 Running the Computation .................................................. 52
- 5.2.6 Post-Processing .............................................................. 52

## 5.3 Code-Specific MpCCI Commands ............................................. 52

## 5.4 Code Environment ............................................................... 53
- 5.4.1 Prerequisites for a coupled simulation ............................... 53

## 5.5 Code Adapter Reference ....................................................... 53

## 6 Flowmaster

### 6.1 Quick Information ............................................................ 54
- 6.1.1 Supported Coupling Schemes ........................................... 54
- 6.1.2 Supported Platforms and Versions .................................. 54
- 6.1.3 References ..................................................................... 54
- 6.1.4 Adapter Description ...................................................... 55

### 6.2 Coupling Process .............................................................. 55
- 6.2.1 Model Preparation ......................................................... 55
- 6.2.2 Models Step .................................................................. 58
- 6.2.3 Coupling Step ............................................................... 58
- 6.2.4 Go Step ........................................................................ 59

### 6.3 Code-Specific MpCCI Commands ......................................... 60

### 6.4 Code Adapter Description .................................................. 61

## 7 FLUENT

### 7.1 Quick Information ............................................................ 62
- 7.1.1 Supported Coupling Schemes ........................................... 62
- 7.1.2 Supported Platforms and Versions .................................. 62
- 7.1.3 References ..................................................................... 62
- 7.1.4 Adapter Description ...................................................... 63

### 7.2 Coupling Process .............................................................. 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.1 Model Preparation</td>
<td>63</td>
</tr>
<tr>
<td>7.2.2 Models Step</td>
<td>65</td>
</tr>
<tr>
<td>7.2.3 Coupling Step</td>
<td>65</td>
</tr>
<tr>
<td>7.2.4 Go Step</td>
<td>68</td>
</tr>
<tr>
<td>7.2.5 Running the Computation</td>
<td>70</td>
</tr>
<tr>
<td>7.3 Code-Specific MpCCI Commands</td>
<td>75</td>
</tr>
<tr>
<td>7.4 Code Adapter Reference</td>
<td>76</td>
</tr>
<tr>
<td>8.1 Quick Information</td>
<td>80</td>
</tr>
<tr>
<td>8.1.1 Supported Coupling Schemes</td>
<td>80</td>
</tr>
<tr>
<td>8.1.2 Supported Platforms and Versions</td>
<td>80</td>
</tr>
<tr>
<td>8.1.3 References</td>
<td>80</td>
</tr>
<tr>
<td>8.1.4 Adapter Description</td>
<td>80</td>
</tr>
<tr>
<td>8.2 Coupling Process</td>
<td>81</td>
</tr>
<tr>
<td>8.2.1 Model Preparation</td>
<td>81</td>
</tr>
<tr>
<td>8.2.2 pyFlux script</td>
<td>82</td>
</tr>
<tr>
<td>8.2.3 Models Step</td>
<td>86</td>
</tr>
<tr>
<td>8.2.4 Coupling Step</td>
<td>86</td>
</tr>
<tr>
<td>8.2.5 Go Step</td>
<td>88</td>
</tr>
<tr>
<td>8.3 Code-Specific MpCCI Commands</td>
<td>89</td>
</tr>
<tr>
<td>8.4 Code Environment</td>
<td>90</td>
</tr>
<tr>
<td>8.4.1 Prerequisites for a coupled simulation</td>
<td>90</td>
</tr>
<tr>
<td>8.5 Code Adapter Description</td>
<td>90</td>
</tr>
<tr>
<td>9.1 Quick Information</td>
<td>92</td>
</tr>
<tr>
<td>9.1.1 Supported Coupling Schemes</td>
<td>92</td>
</tr>
<tr>
<td>9.1.2 Supported Platforms and Versions</td>
<td>92</td>
</tr>
<tr>
<td>9.1.3 References</td>
<td>92</td>
</tr>
<tr>
<td>9.1.4 Adapter Description</td>
<td>93</td>
</tr>
</tbody>
</table>
9.2 Coupling Process ......................................................... 94
  9.2.1 Model Preparation ................................................. 94
  9.2.2 Models Step ......................................................... 94
  9.2.3 Coupling Step ...................................................... 95
  9.2.4 Go Step ............................................................. 96
  9.2.5 Running the Computation ......................................... 96
  9.2.6 Post-Processing .................................................... 97
9.3 Code-Specific MpCCI Commands .................................... 98
9.4 Code Environment ...................................................... 99
  9.4.1 Prerequisites for a coupled simulation ......................... 99

10 PERMAS ................................................................. 100
10.1 Quick Information ..................................................... 100
  10.1.1 Supported Coupling Schemes ................................... 100
  10.1.2 Supported Platforms and Versions ................................ 100
  10.1.3 References ........................................................ 100
  10.1.4 Adapter Description ............................................... 101
10.2 Coupling Process ...................................................... 102
  10.2.1 Model Preparation ................................................ 102
  10.2.2 Models Step ....................................................... 103
  10.2.3 Coupling Step ..................................................... 103
  10.2.4 Go Step ............................................................. 104
  10.2.5 Running the Computation ....................................... 106
  10.2.6 Post-Processing .................................................. 108
10.3 Code-Specific MpCCI Commands .................................... 109
10.4 Trouble shooting, open issues and known bugs .................... 110

11 RadTherm ............................................................... 111
11.1 Quick Information ..................................................... 111
  11.1.1 Supported Coupling Schemes ................................... 111
  11.1.2 Supported Platforms and Versions ................................ 111
  11.1.3 References ........................................................ 111
  11.1.4 Adapter Description ............................................... 111
11.2 Coupling Process .............................................. 112
  11.2.1 Model Preparation ......................................... 112
  11.2.2 Models Step ............................................... 112
  11.2.3 Coupling Step ............................................. 112
  11.2.4 Go Step ................................................... 113
  11.2.5 Running the Computation ................................ 113
  11.2.6 Post-Processing .......................................... 116
11.3 Code-Specific MpCCI Commands ................................. 116

12 STAR-CD ...................................................... 117
  12.1 Quick Information ............................................ 117
    12.1.1 Supported Coupling Schemes .............................. 117
    12.1.2 Supported Platforms and Versions ....................... 118
    12.1.3 References .............................................. 118
    12.1.4 Adapter Description .................................... 118
  12.2 Coupling Process ............................................ 118
    12.2.1 Model Preparation ....................................... 118
    12.2.2 Models Step ............................................. 121
    12.2.3 Coupling Step .......................................... 121
    12.2.4 Go Step ................................................ 123
    12.2.5 Running the Computation ................................. 126
    12.2.6 Post-Processing ........................................ 127
  12.3 Code-Specific MpCCI Commands ............................... 129
  12.4 Grid Morphing .............................................. 130
    12.4.1 MpCCI Grid Morpher .................................... 130
    12.4.2 Restart with MpCCI Grid Morpher ....................... 132
    12.4.3 pro-STAR Grid Morpher ................................ 133
  12.5 Code Adapter Reference .................................... 138
    12.5.1 STAR-CD 3.26 .......................................... 138
    12.5.2 STAR-CD 4.0x ........................................... 138
    12.5.3 Automatic model preparation for STAR-CD 3.26 and STAR-CD 4.0x ................................ 139
  12.6 Trouble shooting, open issues and known bugs .............. 141
1 Overview

This codes manual contains simulation code-specific information.

There is one chapter for each code, which contains basic information on the code itself and what you should know for preparing a model and running a co-simulation.

⚠️ This Code Manual cannot replace the actual manuals of the simulation codes. References to further information in the code manuals are given as necessary.

The information given for each code is given in the same structure:

Quick Information – Basic properties of the simulation code.

Coupling Process – How to prepare and run a model for co-simulation, including code-specific options in the MpCCI GUI.

Code-Specific MpCCI Commands – Subcommands `mpcci <code name>` which are needed to get information on the simulation code and prepare models for co-simulation. A number of subcommands is available for several codes, these are described in 1.1 Common MpCCI Subcommands for Simulation Codes.

Code Environment – Requirements for installation, licensing and further software needed by the simulation code.

Code Adapter Reference – Short description of the code adapter.
1.1 Common MpCCI Subcommands for Simulation Codes

`mpcci <codename> -releases`

The `-releases` subcommand prints a short list of all available releases of a simulation code.

`mpcci <codename> -info`

The `-info` subcommand prints a more detailed list of the available releases of a simulation code and on the corresponding code adapter. The output looks as follows:

```
> mpcci simulationcode info

SIMULATIONCODE release "3.1":
    HOME: "/opt/simulationcode/sim-3.1"
    EXEC: "/opt/simulationcode/sim-3.1/bin/start-code"
    ARCH: "linux86"

    MpCCI adapter release "3.1": ** NOT INSTALLED **
    HOME: "/home/fritz/mpcci/codes/SIMULATIONCODE/adapters"
    PATH: "/home/fritz/mpcci/codes/SIMULATIONCODE/adapters/3.1/linux86"

SIMULATIONCODE release "2.99":
    HOME: "/opt/simulationcode/sim-2.99"
    EXEC: "/opt/simulationcode/sim-2.99/bin/start-code"
    ARCH: "linux86"

    MpCCI adapter release "2.99":
    HOME: "/home/fritz/mpcci/codes/SIMULATIONCODE/adapters"
    PATH: "/home/fritz/mpcci/codes/SIMULATIONCODE/adapters/2.99/linux86"

Latest SIMULATIONCODE release for MpCCI found: "2.99"
```

Two versions of the code are installed, but a code adapter is only available for version 2.99. Therefore the latest simulation code release which can be used with MpCCI is version 2.99. `HOME` is the main directory of a code installation or adapter, `EXEC` the code executable and `PATH` the path to the code adapter, which does not exist if an adapter is not available. The architecture token given under `ARCH` is the code’s architecture token.
mpcci <codename> -align <ARGS>

The `mpcci <codename> -align` command is used to perform a coordinate transformation. This can be necessary if the coordinate system of the model differs from that of the partner code. The usage is as follows:

Usage:
```
mpcci SIMULATIONCODE align <plane-definition-file> <input-file> <outputfile>
```

Synopsis:
```
'mpcci SIMULATIONCODE align' is used to align/transform the nodal coordinates in the <input-file> so that they match with the coordinate system used by the coupling partner.

The transformed result is written into the <outputfile> - which is identical to the <input-file> except the nodal point coordinates.

The homogenous 4x4 transformation matrix resp. the reference coordinate systems used are indirectly defined via two planes. Each plane is defined via three non colinear points p1, p2 and p3.

The six points that define the two planes are specified in the
```
<plane-definition-file>
```
which contains 18 floating point values in the following order:

```
# SIMULATIONCODE source plane specified via 3 point coordinates
 p1-x  p1-y  p1-z
 p2-x  p2-y  p2-z
 p3-x  p3-y  p3-z

# Target coupling partner code plane specified via 3 point coordinates
 p1-x  p1-y  p1-z
 p2-x  p2-y  p2-z
 p3-x  p3-y  p3-z
```

The transformation matrix determined transforms the SIMULATIONCODE plane into the "target" plane (rotation and scaling, but without shear) so that:

```
SIMULATIONCODE(p1) -> coupling partner(p1)
SIMULATIONCODE(p2) -> coupling partner(p2)
```
To use 'mpcci SIMULATIONCODE align' please just select three characteristic points in your SIMULATIONCODE model and the corresponding points in the partner model and copy the x-y-z values into the <plane-definition-file> in the above order.

```
mpcci <codename> -scan <model file>
```

The `-scan` starts the scanner for a model file, which writes basic information about the model into the scan file "mpcci_<model file>.scan". After creating the file, its content is printed on the screen.

If the scan file "mpcci_<model file>.scan" exists already, no new scanning process is started, instead only the content is printed.

### 1.2 Unit Systems

Many simulation codes do not use a given system of units, but do not consider units at all. This has the advantage that users can select any consistent set of units, define all values in this system and will also receive the values in this system.

For coupling a unit-less code with codes which are based on a specific system of units, **MpCCI** must know which system of units was used to create the model.

**MpCCI** supports the following unit systems, for each system some units are listed. Of course there are more units in each system, please see what is selected in the **MpCCI GUI**.

<table>
<thead>
<tr>
<th>unit system</th>
<th>mass</th>
<th>length</th>
<th>time</th>
<th>forces</th>
<th>el. current</th>
<th>temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI</td>
<td>kg</td>
<td>m</td>
<td>s</td>
<td>N</td>
<td>A</td>
<td>K</td>
</tr>
<tr>
<td>British*</td>
<td>lbm</td>
<td>ft</td>
<td>s</td>
<td>lbf</td>
<td>A</td>
<td>K</td>
</tr>
<tr>
<td>cgs</td>
<td>g</td>
<td>cm</td>
<td>s</td>
<td>dyn</td>
<td>Bi</td>
<td>K</td>
</tr>
<tr>
<td>SI-mm-t-s</td>
<td>t</td>
<td>mm</td>
<td>s</td>
<td>N</td>
<td>A</td>
<td>K</td>
</tr>
<tr>
<td>US-ft-lbf-s</td>
<td>slug</td>
<td>ft</td>
<td>s</td>
<td>lbf</td>
<td>A</td>
<td>K</td>
</tr>
<tr>
<td>US-in-lbf-s</td>
<td>lbf s²/in</td>
<td>in</td>
<td>s</td>
<td>lbf</td>
<td>A</td>
<td>K</td>
</tr>
</tbody>
</table>

Alternatively, you can also select the option **variable**, which allows you to select units individually for each quantity. However, for each quantity only a limited set of units is offered!

⚠️*: The “British” unit system as defined in the **MpCCI GUI** is not a consistent unit system – some of the electromagnetic units are inconsistent! Use at own risk.
2 Abaqus

2.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>SolidStructure, SolidAcoustics, SolidThermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>SIMULIA</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.simulia.com">www.simulia.com</a></td>
</tr>
<tr>
<td>Support</td>
<td>abaqus.custhelp.com</td>
</tr>
</tbody>
</table>

2.1.1 Supported Coupling Schemes

The coupling schemes supported by Abaqus depend on the version:

<table>
<thead>
<tr>
<th>Abaqus Version</th>
<th>Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abaqus 6.6</td>
<td>after solution</td>
</tr>
<tr>
<td>Abaqus 6.7</td>
<td>before solution</td>
</tr>
<tr>
<td>Abaqus 6.8</td>
<td>before solution</td>
</tr>
<tr>
<td>Abaqus 6.9</td>
<td>before solution</td>
</tr>
</tbody>
</table>

Abaqus supports subcycling in both versions, unidirectional and bidirectional transfer is possible. For Abaqus 6.7 and higher after completing the last increment a final exchange, which is inverted regarding the initial exchange, is accomplished.

<table>
<thead>
<tr>
<th>Initial Transfer</th>
<th>Final Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive</td>
<td>send</td>
</tr>
<tr>
<td>send</td>
<td>receive</td>
</tr>
<tr>
<td>skip</td>
<td>-</td>
</tr>
<tr>
<td>exchange</td>
<td>-</td>
</tr>
</tbody>
</table>

2.1.2 Supported Platforms and Versions

MpCCI 3.1 supports Abaqus 6.7, Abaqus 6.8, Abaqus 6.9 and most features of Abaqus 6.6.
Please see also the System Information section of the Support page at www.simulia.com for details on the platforms supported by Abaqus.

### 2.1.3 References

**Abaqus Version 6.7, 6.8 and 6.9 Documentation** The Abaqus documentation is part of your Abaqus installation. Read especially the section “Co-simulation” of the Abaqus Analysis User’s Manual.

**Abaqus Fluid-Structure Interaction User’s Guide** This guide is available via the Abaqus support homepage abaqus.custhelp.com: Log into Abaqus Answers and search for “FSI guide”.

Besides general information, the FSI guide contains several examples of coupled simulations with Abaqus and FLUENT, Abaqus and STAR-CD.

### 2.1.4 Adapter Description

The code adapter for Abaqus is developed by SIMULIA in cooperation with Fraunhofer SCAI. The adapter is distributed as part of the Abaqus software.

<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCI_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>aix52.power</td>
<td>aix52.power</td>
<td>X</td>
</tr>
<tr>
<td>aix53.power</td>
<td>aix53.power</td>
<td>X</td>
</tr>
<tr>
<td>hpux1123_ia64</td>
<td>hpux1123_ia64</td>
<td>X</td>
</tr>
<tr>
<td>hpux11_parisc</td>
<td>hpux11_parisc</td>
<td>X</td>
</tr>
<tr>
<td>irix65_mips4</td>
<td>irix65_mips4</td>
<td>X</td>
</tr>
<tr>
<td>linux_amd64</td>
<td>linux_amd64</td>
<td>X</td>
</tr>
<tr>
<td>linux_em64t</td>
<td>linux_em64t</td>
<td>X</td>
</tr>
<tr>
<td>linux_ia64</td>
<td>linux_ia64</td>
<td>X</td>
</tr>
<tr>
<td>linux_x64</td>
<td>linux_x64</td>
<td>X</td>
</tr>
<tr>
<td>linux_x86</td>
<td>linux_x86</td>
<td>X</td>
</tr>
<tr>
<td>mswin_x86</td>
<td>mswin_x86</td>
<td>X</td>
</tr>
<tr>
<td>xp64_amd64</td>
<td>xp64_amd64</td>
<td>X</td>
</tr>
<tr>
<td>xp64_em64t</td>
<td>xp64_em64t</td>
<td>X</td>
</tr>
<tr>
<td>xp64_x64</td>
<td>xp64_x64</td>
<td>X</td>
</tr>
</tbody>
</table>
2.2 Coupling Process

Please read also ➤ IV-2 Setting up a Coupled Simulation ◄.

2.2.1 Model Preparation

The Abaqus model can be prepared with Abaqus/CAE or as an input file. Please consider the following advice:

- The model can be defined in any of the consistent systems of units supported by MpCCI (see ➤ 1.2 Unit Systems ◄). The basic unit system must be given in the Models Step of the MpCCI GUI. Units of single quantities can be set in the Coupling Step.

- Most Abaqus features can be used in co-simulations. Please see the Abaqus documentation or contact the Abaqus support for further information.

- The Abaqus model must contain a definition of coupling components. This can be either an element-based surface for surface coupling:

  **Abaqus/CAE:** Create a surface using the **Surfaces** tool. See also “13.7.6 Using sets and surfaces in the Assembly module” in the Abaqus/CAE User’s Manual. You can also use surfaces defined in the Part module.

  **Input file:** A surface is created with *SURFACE, NAME=<surface name>, TYPE=ELEMENT, see section “2.3.3 Defining element-based surfaces” of the Abaqus Analysis User’s Manual.

  The surface which serves as coupling component can be selected in the Models Step of the MpCCI GUI.

- Code coupling is only run in one step of a simulation (for coupling in several steps, please use a restart). The co-simulation step is selected in the Go-Step of the MpCCI GUI.

⚠️ It is recommended to create a complete Abaqus model first and test it separately without co-simulation. The quantities which will be transferred by the partner code can be simulated by appropriate loads or boundary conditions.

2.2.2 Models Step

In the models step, the following options must be chosen:

**Please select the Abaqus release**  Select the Abaqus release you wish to use. Only supported releases installed on your system are listed. The selection latest always refers to the latest supported version (default). The release should match the input file. To select a release on a remote machine, please select the remote input file first.
Select scan method  This can be set to:

- Scan for all regions (default) – The input file is scanned for possible coupling regions.
- Scan *CO-SIMULATION option only – This setting is only needed for a restart. In this case the input file only contains the definition of the coupling regions in the \*CO-SIMULATION section and no ordinary region definitions.

Select Abaqus input file  Select the Abaqus input file "*.inp".

Select unit system  Select the unit system which was used in Abaqus. Abaqus has no units built into it, a self-consistent set of units should be used (see chapter “1.2.2 Conventions” of the Abaqus Analysis User’s Manual for more information). In the Go Step you can select from any unit system listed in \>1.2 Unit Systems <.

2.2.3 Coupling Step

Abaqus supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FilmTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>NDisplacement</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>NPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
2.2.4 Go Step

In the Go Step, the following options can be selected:

**Initial quantities transfer**  Select one of receive, send, exchange or skip. The meaning of initial exchange is further described in ▶ V-3.3 Coupling Algorithms ◀.
Enter a job name  This is the name of the Abaqus job, which is also used as base for the Abaqus output files.

Additional command line options  Additional command line options for Abaqus can be given here, they will directly be used when Abaqus is started.

MpCCI use the alternate syntax ”-option value” for the Abaqus command line options.

Enter the co-simulation step number  The co-simulation step is the step of the Abaqus simulation which is coupled. You can e. g. run some initial computations first, followed by a coupled step based on the previous results.

Constant coupling time step and Coupling time step  This button must be checked if the time step size DeltaTime is not selected as a global quantity to be send or received by Abaqus. In this case you must specify the value of the coupling time step size in the field below.

Subcycling  Check this button if you want allow Abaqus to subcycle, i. e. use a smaller time step size than the coupling time step size. The basic principle of subcycling is described in V-3.3.5 Subcycling. If you select this option, you can also choose whether Abaqus has to meet the coupling target times in an exact or loose manner by toggling the Enforce exact target times button.

Optional old job name for a restart  This option is required for a restart computation. It is handed over to Abaqus as oldjob=<> option.

User Subroutine  is the name of a FORTRAN file (source or object file), which is handed to Abaqus as user=<>]. See “User subroutines: overview” of the Abaqus Analysis User’s Manual.

Double precision for Abaqus/Explicit  Use this option to select the precision for Abaqus/Explicit computations. It is ignored for Abaqus/Standard.

Run parallel  Select this to start a parallel run. A panel with additional options appears, which are further described in 2.2.5.1 Parallel Execution.

2.2.5 Running the Computation

When the Start button is pressed, Abaqus is started with the options given in the Go Step. If the Stop button is pressed, a stop-file is created and Abaqus will stop the next time it checks the presence of a stop file.

2.2.5.1 Parallel Execution

Parallel runs of Abaqus are supported by MpCCI.

In the Go Step, a set of additional options can be chosen for a parallel run as shown in Figure 3.
No. of processors  Enter the number of processors to use during the analysis. The corresponding Abaqus command line option is -cpus.

No. of parallel domains (Abaqus/Explicit)  Enter the number of parallel domains in Abaqus/Explicit. The corresponding Abaqus command line option is -domains. As parallel Abaqus is always run with the command line option -parallel=domain (-parallel=loop is not available for coupled runs), the value for No. of processors must be evenly divisible into the value of No. of parallel domains.

Parallelization method  Select the parallelization method either mpi or leave blank which results into the Abaqus default. The corresponding Abaqus command line option is -mp_mode.

Shared file system  Check here if the file system is shared for all hosts and no data copy for local scratch disks is necessary.

Optional 'host host...' to be used  Enter host names for parallel execution of Abaqus.

Optional hostlist file  Specify a hostfile, from which host names are extracted → V-3.4.2 Hostlist files ←.
Use default hostfile  A default hostfile can be configured by setting MPCCI_HOSTLIST_FILE \( \triangleright \) V-3.4.2  Hostlist files \( \triangleleft \).

Please read the Abaqus documentation for a detailed explanations of the possible options.

### 2.2.5.2 Batch Execution

Abaqus is always run as a batch process, therefore no special settings are necessary for batch execution.

### 2.2.6 Post-Processing

Post-processing for the Abaqus part of the results can be performed as in ordinary computations, e.g. with Abaqus/CAE. The "\(<job name>\).odb" file can be found in the same directory as the input file.
2.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for Abaqus are:

Usage:

mpcci Abaqus [-]option

Synopsis:

‘mpcci Abaqus’ is used to get information about Abaqus.

Options:

-align <ARGS> Do a coordinate transformation on all nodes of an input file based on a plane definition file and align the nodal coordinates for the coupling partner.

-help This screen.

-info List verbose information about all Abaqus releases.

-releases List all Abaqus releases which MpCCI can find.

-scan <input-file> Run the scanner and create a scanner output file.

For Abaqus no special MpCCI subcommands are needed. Only the standard subcommands are available. The options -align, -info, -releases and -scan are described in ◄1.1 Common MpCCI Subcommands for Simulation Codes ◄.
2.4 Code Environment

2.4.1 Prerequisites for a coupled simulation

To run a co-simulation you need the following:

- Ordinary Abaqus 6.8, Abaqus 6.7 (or Abaqus 6.6) installation.
- Enough license tokens. Co-simulation requires one additional enabling token.
- In the Abaqus environment file "abaqus_v6.env", the co-simulation feature must be enabled by setting `no_cosimulation=OFF`.
2.5 Code Adapter Reference

2.5.1 Patched Input File

Before a coupled simulation is started, MpCCI patches the Abaqus input file. For a file "*.inp" selected in the Models Step, a new file "mpcci_*.inp" is created containing a new *CO-SIMULATION block. If the original file already contains *CO-SIMULATION keywords, they are removed.

The lines inserted by MpCCI may e.g. look as follows:

```
**
** MpCCI automatically inserted this *CO-SIMULATION keyword.
** Therefore any existing *CO-SIMULATION keyword was removed.
**
*CO-SIMULATION, NAME=COSIMULATION_1, PROGRAM=MPCCI,
         CONTROLS=COSIMULATION_CONTROLS_1
*CO-SIMULATION REGION, TYPE=SURFACE, EXPORT
  ASSEMBLY_BLOCK-1_WALL, COORD
*CO-SIMULATION REGION, TYPE=SURFACE, IMPORT
  ASSEMBLY_BLOCK-1_WALL, CF
*CO-SIMULATION CONTROLS, NAME=COSIMULATION_CONTROLS_1,
       TIME INCREMENTATION=SUBCYCLE, TIME MARKS=YES, STEP SIZE=EXPORT
```

The settings in this block reflect the user’s choices in the MpCCI GUI:

- The selected coupling components are listed below the *CO-SIMULATION REGION keywords. Here it is ASSEMBLY_BLOCK-1_WALL and the exchanged quantities are

  COORD = NPosition, sender is Abaqus
  CF = RelWallForce, sender is partner code

- The time incrementation scheme is selected in the Go Step of the MpCCI GUI. Behind the *CO-SIMULATION CONTROLS keyword you find:

  TIME INCREMENTATION=SUBCYCLE if subcycling was selected or TIME INCREMENTATION=LOCKSTEP, which means Abaqus uses the coupling time step also for the computation of the Abaqus part of the simulation. If SUBCYCLE is selected, the option TIME MARKS determines whether the target times are met in an exact (YES) or loose (NO) manner.

- If the time step size is exchanged, you find either STEP SIZE=EXPORT if Abaqus is the sender or STEP SIZE=IMPORT if Abaqus receives the time step size from the partner code.

The *CO-SIMULATION keywords are described in detail in “Preparing an Abaqus analysis for co-simulation” of the Abaqus Analysis User’s Manual.
2.6 Trouble shooting, open issues and known bugs

<table>
<thead>
<tr>
<th>Feature:</th>
<th>Two-dimensional quadratic elements and modified tetrahedral elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version:</td>
<td>all</td>
</tr>
<tr>
<td>Problem:</td>
<td>When these elements are coupled, MpCCI server will crash with error</td>
</tr>
<tr>
<td></td>
<td>&quot;Assertion failed(pLocalPointsNeighbData == 0)&quot;</td>
</tr>
<tr>
<td>Workaround:</td>
<td>not available</td>
</tr>
<tr>
<td>References:</td>
<td></td>
</tr>
</tbody>
</table>
3 ANSYS

3.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>SolidStructure, SolidThermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>ANSYS, Inc.</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.ansys.com">www.ansys.com</a></td>
</tr>
<tr>
<td>Support</td>
<td>www1.ansys.com/customer</td>
</tr>
</tbody>
</table>

3.1.1 Supported Coupling Schemes

ANSYS is run via an APDL script, therefore all coupling algorithms are possible.

3.1.2 Supported Platforms and Versions

The following versions of ANSYS are supported by MpCCI:

<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCI_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td>mswin_x86</td>
<td>90 100 110 120</td>
</tr>
<tr>
<td>alpha</td>
<td>osf_alpha</td>
<td>X X X X</td>
</tr>
<tr>
<td>hp64</td>
<td>hpux11_parisc</td>
<td>X X X X</td>
</tr>
<tr>
<td>hpia64</td>
<td>hpux1123_ia64</td>
<td>X X X X</td>
</tr>
<tr>
<td>intel</td>
<td>mswin_x86</td>
<td>X X X X</td>
</tr>
<tr>
<td>linem64t</td>
<td>(sles10</td>
<td>linux)_(em64t</td>
</tr>
<tr>
<td>linia32</td>
<td>linux_x86</td>
<td>X X X X</td>
</tr>
<tr>
<td>linia64</td>
<td>linux_ia64</td>
<td>X X X X</td>
</tr>
<tr>
<td>linop64</td>
<td>(sles10</td>
<td>linux)_(amd64</td>
</tr>
<tr>
<td>linx64</td>
<td>(sles10</td>
<td>linux)_(amd64</td>
</tr>
<tr>
<td>sgi64</td>
<td>irix65_mips4</td>
<td>X X X X</td>
</tr>
<tr>
<td>sun64</td>
<td>solaris_sparc</td>
<td>X X X X</td>
</tr>
<tr>
<td>usIII</td>
<td>solaris_sparc</td>
<td>X X</td>
</tr>
<tr>
<td>winx64</td>
<td>(xp64</td>
<td>vista)_(x64</td>
</tr>
</tbody>
</table>

3.1.3 References

The ANSYS installation contains a detailed documentation. We especially refer to:

ANSYS APDL Programmer’s Guide. The ANSYS Parametric Design Language (APDL) is needed to run a co-simulation.
3.1.4 Adapter Description

The ANSYS adapter is a shared library which is loaded by ANSYS. The adapter functions must be called from an APDL script.
3.2 Coupling Process

Please read also ▶ IV-2 Setting up a Coupled Simulation ◀.

3.2.1 Model Preparation

There are some issues which you should consider while creating an ANSYS model for a co-simulation:

Supported element types. Not all ANSYS element types are supported, the supported types are given in Table 1.

Dummy surface elements for surface coupling. For surface coupling, additional surface elements must be used for quantity exchange. For instance SHELL63 elements for a fluid structure interaction or SHELL57 elements for thermal coupling can be used as dummy elements to receive forces from the partner code. The dummy elements must have the corresponding quantities you want to receive or send. Only these elements take part in the coupling process and must be either be deselected when the solution is performed or defined so weak that they do not influence the solution in case of fluid structure interaction. They can be deselected before solution is executed if only nodal quantities are sent or received, because the data transfer will then put the values to the nodes and the nodes of dummy elements are shared by the “real” solid model elements. In case of thermal coupling the dummy elements can not be deselected because the quantities wall heat transfer coefficient, wall temperature and wall heat flux are only supported as element quantities. Therefore the additional layer of shell elements (SHELL57) have to be a part of the solution. To reduce the influence on the solution you should give the shell elements the same material properties as the adjacent solid elements and a small thickness. In case of line coupling on area elements in fluid structure interaction BEAM3 elements can be used. The attached example cases for 2D and 3D fluid structure interaction contains such dummy elements. Such dummy elements are shown in Figure 1.

Put elements for coupling in a component. The elements of the coupling region must be grouped into one or more element components using the cm command.

Exchange of global quantities. For each global quantity a corresponding scalar parameter must be defined. E.g. for the time step size a parameter named “DeltaTime” is needed. Please also assign a dummy value to the parameter. This parameter must be set to the correct value in the APDL script if it is received, or the parameter must be evaluated in the APDL script. Global quantities can thus be regarded as simple variables which can be filled with values by °mpcci, receive or whose values are read by °mpcci, send.

Assign unique material property numbers. Every element of your model must be assigned a unique material property number if you want to send material property values to ANSYS, e.g. the electrical resistivity. In ANSYS you define such numbers with the mp command.

Possible quantities depend on degrees of freedom. Normally the degree of freedom of the elements involved in the coupling process determines which quantities can be transferred. It is laborious to find
out if all degrees of freedom are actually supported by the ANSYS API. As this API is used for the MpCCI ANSYS–adapter, it is not guaranteed that all theoretically supported degrees of freedom are valid.

**Not carefully tested.** Only few of the element type mappings are already validated with certain quantities. The compatibility index in Table 2 shows the validated element-quantity pairs. It is constantly added. Please contact us if you have problems with other combinations or need additional capabilities.

**Binary database file required.** You need to setup your model and store it in a binary database file (db file) because the coupled components will be extracted from this during the MpCCI scanning process.

**APDL script required.** For managing the co-simulation, an APDL script is required. This is described in detail in $\triangleright$3.2.2 APDL Script $\triangleright$.

<table>
<thead>
<tr>
<th>ANSYS Element Types</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAM3, BEAM4, BEAM23</td>
<td>MPCCI_ELEM_LINE</td>
</tr>
<tr>
<td>PLANE13, PLANE42, PLANE55, HYPER56, SHELL57, SHELL63, PLANE67, VISC0106, SHELL131 SHELL143, SHELL157, PLANE181, PLANE182</td>
<td>MPCCI_ELEM_TRIANGLE, MPCCI_ELEM_QUAD</td>
</tr>
<tr>
<td>PLANE2, PLANE35, PLANE53, HYPER74, PLANE77, PLANE82, VISC088, SHELL93, SHELL99, VISC0108, PLANE121, SHELL132, PLANE145, PLANE14, SHELL150, PLANE183</td>
<td>MPCCI_ELEM_TRIANGLE6, MPCCI_ELEM_QUAD8 2D Volume coupling MPCCI_ELEM_TRIANGLE, MPCCI_ELEM_QUAD</td>
</tr>
<tr>
<td>SOLID5, SOLID45, SOLID46, HYPER58, SOLID62, SOLID64, SOLID65, SOLID69, SOLID70, HYPER86, SOLID96, SOLID97, VISCO107, SOLID185</td>
<td>MPCCI_ELEM_PRISM, MPCCI_ELEM_TETRAHEDRON, MPCCI_ELEM_HEXAHEDRON, MPCCI_ELEM_PYRAMID</td>
</tr>
<tr>
<td>SOLID87, VISC089, SOLID90, SOLID92, SOLID95, SOLID98, SOLID122, SOLID123, SOLID127, SOLID128, SOLID147, SOLID148, SOLID168, SOLID186, SOLID187</td>
<td>Tetrahedron and hexahedron solid elements with midpoint nodes can not be handled properly by MpCCI in case of volume coupling.</td>
</tr>
<tr>
<td>SOLID117</td>
<td>Can only be used if all mid nodes lie on the straight line between the corner nodes and only if element quantities are transferred.</td>
</tr>
</tbody>
</table>

Table 1: Supported ANSYS element types
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Joule Heat Density</th>
<th>Lorentz Force Density</th>
<th>Electric Resistivity</th>
<th>Nodal Positions</th>
<th>Wall Forces</th>
<th>Relative Wall Forces</th>
<th>Film Temperature</th>
<th>Wall Heat-transfer Coefficient</th>
<th>Wall Temperature</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAM3</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHELL63</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHELL57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
</tr>
<tr>
<td>PLANE13</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLID5</td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLID45</td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLID69</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLID97</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOLID117</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Quantities supported for different ANSYS elements.

Figure 1: ANSYS Flap and Additional SHELL63 elements for coupling.
3.2.2 APDL Script

In ANSYS, the coupling process is controlled via an APDL script, which calls functions provided by MpCCI.

3.2.2.1 Sample APDL Script

```apdl
/batch
resume, ansys, db ! resume ANSYS database "ansys.db"
~mpcci, init, 2D ! initialize MpCCI for 2D analysis
fini

/solu
antype, transient, new ! new transient analysis
trnopt, full ! full Newton-Rhapson

/com > > >
/com > > > get the initial transfer action defined
/com > > >

/inquire, mpcciaction,ENV,MPCCI_INITIAL_EXCHANGE,1

/com > > >
/com > > > initial data transfer
/com > > >

*if, mpcciaction(1), eq, 'receive', then
   ~mpcci, receive, wait ! receive data via MpCCI
*elseif, mpcciaction(1), eq, 'send'
   ~mpcci, send, wait ! send data via MpCCI
*elseif, mpcciaction(1), eq, 'exchange'
   ~mpcci, exchange, wait ! exchange data via MpCCI
*endif

*do, i, 1, steps+1
 *if, i, ne, 1, then ! first run is dummy
   esel, all $ nsle ! make sure to have all of the model
   cmsel, u, TOP ! deselect the beam elements,
   cmsel, u, BOTTOM ! only used for transfer of values
   nsle
time, PhysicalTime ! time at the end of this loadstep
   solve ! solve this time step
   ~mpcci, exchange, wait ! exchange data via MpCCI
```

30 VI MpCCI 3.1.1-1
MpCCI is initialized by `~mpcci, init, 2D`. This means the model is two dimensional.

First we retrieve the Initial quantities transfer value set in MpCCI GUI. Depending on the value set the script will automatically execute one of the commands:

- `~mpcci, receive, wait`. ANSYS receives data.
- `~mpcci, send, wait`. ANSYS sends data.
- `~mpcci, exchange, wait`. ANSYS exchanges data.

Next the loop for the coupled simulation starts with one dummy run first. ANSYS sometimes has problems without this dummy run. Afterwards the solution `~mpcci, exchange, wait` sends the nodal positions to the partner code. ANSYS will wait until the partner code has received this data. Following it waits for the partner code to finish the solution and receives the results. The loop continues until the final step is reached.

The command `~mpcci, stop` is finishing the MpCCI process regularly and ANSYS could be finished.

⚠️ If element table items should be transferred, generate them before the send or exchange command is executed.

### 3.2.2.2 Available Commands

The command for MpCCI calls within ANSYS is `~mpcci` followed by command line options. The following command line options are valid, where * marked values are default values and options in square brackets [ ] are optional.

```plaintext
~mpcci, WRCPL, filename.cpl
~mpcci, STATUS
~mpcci, *HELP
~mpcci, INIT [, 2D | 3D | *AUTO ]
~mpcci, IPROBE
~mpcci, SEND [, *WAIT | NOWAIT ]
~mpcci, RECEIVE [, *WAIT | NOWAIT ]
~mpcci, EXCHANGE [, *WAIT | NOWAIT ]
~mpcci, WAIT
~mpcci, STOP
```
The available commands have the following functionality:

```
~mpcci, WRCPL, <filename>.cpl
   Writes a component list file with all defined components and variables into "<filename>.cpl". This file is normally generated and used by the MpCCI GUI, but can also be generated with the wrcpl command option. Then the GUI will use this file for the "<filename>.db" file.

~mpcci, STATUS
   Print the actual MpCCI status.

~mpcci, HELP
   Print list of available commands.

~mpcci, INIT [, 2D | 3D | *AUTO ]
   Initializes the MpCCI process.

~mpcci, IPROBE
   Sends an MpCCI IPROBE request to the partner code and get information about the partner code status. If there is data for ANSYS so that a receive transfer could be performed a corresponding message will be given.

~mpcci, SEND [, *WAIT | NOWAIT ]
   Performs a SEND transfer action, sending data from ANSYS to the partner code. If WAIT is given, the data transfer will be forced. Using NOWAIT the transfer will only happen if the partner code is just waiting for new data. If not, ANSYS will continue and no data will be transferred.

~mpcci, RECEIVE [, *WAIT | NOWAIT ]
   Performs a RECEIVE transfer action, receive data from the partner code. If WAIT is given, the data transfer will be forced. Using NOWAIT the transfer will only happen if the partner code is just having new data. If not, ANSYS will continue and no data will be transferred.

~mpcci, EXCHANGE [, *WAIT | NOWAIT ]
   Performs an EXCHANGE transfer operation, first SEND followed by RECEIVE. If WAIT is given, the data transfer will be forced. Using NOWAIT the transfer will only happen if the partner code is waiting for new data and is having new data for ANSYS. If not, ANSYS will continue and no data will be transferred.

~mpcci, WAIT
   Performs a MpCCI ISEND call and completes the pending ISEND request.

~mpcci, STOP
   Performs a MpCCI FINALIZE and stops the MpCCI process within ANSYS.
```
3.2.2.3 Data Access

In the GUI-option receive/send method there are two methods:

**Direct**  direct read or store of data using “UPF” (user programmable feature) subroutines

**ETAB**  only send possible (no receive of values into “ETAB”)

Here the **ETAB** option means that a quantity is read out of an element-table (“ETAB”). It is only valid when sending an element based quantity.

If the quantity is a scalar quantity with dim=1 (e.g. Joule heat density), choose a storage index `sindex`. The user has to generate an element table fulfilling the naming convention:

```
etab, MPCCI_<sindex>, <item>, <component>
```

For example: To get joule heat density from storage index 0 the APDL command should be:

```
etab, MPCCI_00, jheat
```

(see ANSYS documentation of `ETABLE` command for details)

The user has to fill the “ETAB” with sensible values before the quantities are sent to the partner application.

If the quantity has a dimension >1 (vector quantity) then you have to generate element tables following this naming convention:

```
etab, MPCCI_<sindex>, <item>, <component>
etab, MPCCI_<sindex+1>, <item>, <component>
etab, MPCCI_<sindex+2>, <item>, <component>
```

For example: to get define Lorentz force density vector into element tables starting with storage index 5 the APDL command could be:

```
etab, volu, volu  ! element volume
sexp, MPCCI_05, lfx, volu,,-1  ! lorentz force density
sexp, MPCCI_06, lfy, volu,,-1
sexp, MPCCI_07, lfz, volu,,-1
```
3.2.3 Models Step

![Figure 2: ANSYS options in the Models Step](image)

In the models step, the following options must be chosen:

**Select the ANSYS Release**  – Select the ANSYS release you wish to use. Only supported releases are listed. *latest* always refers to the latest supported version (default). The release should match the input file.

**Select ANSYS product to run**  – Choose one of the ANSYS products you have licensed, see also “3.5 Choosing an ANSYS Product” in the ANSYS 100 documentation.

**Name of the ANSYS database file (*)**  – Select the ANSYS database which contains your model definitions.

**Select unit system**  – ANSYS does not use unit definitions, any self-consistent set of units can be used. In the Go Step you can select from:

- **British** – the British unit system, i.e. ft, s, lbm/ft³, etc.
- **SI** – the SI unit system, i.e. m, kg, s, N, Pa, etc.
- **cgs** – the cgs system, i.e. cm, g, s, dyn, Ba, etc.
- **variable** – corresponds to a user-defined system. This requires to set the unit for each quantity separately in the Coupling Step.
The selection of units in the Models panel is only used to determine default values for the units of the quantities. The units are finally selected in the Coupling Step.

### 3.2.4 Coupling Step

**ANSYS** supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>BodyForce</td>
<td>Vector</td>
<td>0.0 N/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CGAngle</td>
<td>Vector</td>
<td>0.0 rad</td>
<td>max/...</td>
<td>Global</td>
<td>Global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>CGOmega</td>
<td>Vector</td>
<td>0.0 rad/s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>CGPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>CGVelocity</td>
<td>Vector</td>
<td>0.0 m/s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>ChargeDensity</td>
<td>Scalar</td>
<td>0.0 C/m³</td>
<td>Field</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Current1</td>
<td>Scalar</td>
<td>0.0 A</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>CurrentDensity</td>
<td>Vector</td>
<td>0.0 A/m²</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>ElectrCond1</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrCond3</td>
<td>Vector</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrCondX</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrCondY</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrCondZ</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectricField</td>
<td>Vector</td>
<td>0.0 V/m</td>
<td>Field</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectricFlux</td>
<td>Vector</td>
<td>0.0 C/m²</td>
<td>Flux dens.</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrRes1</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrRes3</td>
<td>Vector</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Quantity</td>
<td>Dim.</td>
<td>Default Value</td>
<td>Integration Type</td>
<td>Coupling Dimension</td>
<td>Location</td>
<td>Send Option</td>
<td>Receive Option</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>---------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>ElectrResX</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrResY</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrResZ</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Enthalpy</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FilmTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>HeatFlux</td>
<td>Vector</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>HeatSource</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>IntFlag</td>
<td>Scalar</td>
<td>0 max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
<td></td>
</tr>
<tr>
<td>IterationNo</td>
<td>Scalar</td>
<td>0 max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
<td></td>
</tr>
<tr>
<td>JouleHeat</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node, Element</td>
<td>ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>LorentzForce</td>
<td>Vector</td>
<td>0.0 N/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>MagneticField</td>
<td>Vector</td>
<td>0.0 A/m</td>
<td>Field</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>MagneticFlux</td>
<td>Vector</td>
<td>0.0 T</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>NDisplacement</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>NPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>OverPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>RealFlag</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>RefPressure</td>
<td>Scalar</td>
<td>1.12e5 N/m²</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>RelWallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Residual</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>SpecificHeat</td>
<td>Scalar</td>
<td>1.0 J/kg K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Temperature</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Line, Node, Volume</td>
<td>Element</td>
<td>ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>ThermCond1</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
### 3.2.5 Go Step

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThermCond3</td>
<td>Vector</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ThermCondX</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ThermCondY</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ThermCondZ</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>TimeStepNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>udm00</td>
<td>Scalar</td>
<td>0.0</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>ETAB</td>
<td></td>
</tr>
<tr>
<td>uds00</td>
<td>Scalar</td>
<td>0.0</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>ETAB</td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td>Vector</td>
<td>0.0 m/s</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td></td>
<td>Direct</td>
</tr>
<tr>
<td>Voltage1</td>
<td>Scalar</td>
<td>0.0 V</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>WallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Node,</td>
<td></td>
<td>Direct</td>
</tr>
<tr>
<td>WallHeatFlux</td>
<td>Scalar</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>WallHTCoeff</td>
<td>Scalar</td>
<td>0.0 W/m² K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>WallTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Node,</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

Figure 3: ANSYS options in the Go Step

In the Go Step, the following options can be selected:
**Initial quantities transfer** – Select one of receive, send or exchange. The meaning of initial exchange is further described in ▶ V-3.3 Coupling Algorithms ◀.

**Select a gui option** – It is recommended to select -b to start ANSYS in the batch mode. With -g ANSYS is started with a graphical user interface with one of the graphics devices: -d X11 is the standard UNIX device, use -d X11C to activate light-shading on devices with more than 16 colors. -d WIN32 and -d WIN32C are the corresponding devices for Windows. The option -d 3D should be used if you have a 3-D graphics device. Please see “12.3.4 Changing the Number of Contours” in the ANSYS 10.0 documentation.

**Additional command line options** You can specify additional ANSYS command line options here, which are described in “3.1 Starting an ANSYS Session from the Command Level” of the ANSYS documentation.

**Select APDL input script** Select the file containing the APDL script you created for the analysis (see ▶ 3.2.2 APDL Script ◀).

### 3.2.6 Running the Computation

When the [Start] button is pressed, ANSYS is started with the options given in the Go Step. If the [Stop] button is pressed, a stop-file "*.ABT" is created (see also “3.8 Terminating a Running Job” in the ANSYS 10.0 documentation) and ANSYS will stop the next time it checks the presence of a stop file.
## 3.3 Code-Specific MpCCI Commands

The MpCCI commands which are available for ANSYS are:

<table>
<thead>
<tr>
<th>Usage:</th>
<th>mpcci ANSYS [-]option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synopsis:</td>
<td>'mpcci ANSYS’ is used to get information about ANSYS.</td>
</tr>
<tr>
<td>Options:</td>
<td>-align &lt;ARGS&gt; Do a coordinate transformation on all nodes of a .cdb file based on a plane definition file and align the nodal coordinates for the coupling partner.</td>
</tr>
<tr>
<td></td>
<td>-help This screen.</td>
</tr>
<tr>
<td></td>
<td>-info List verbose information about all ANSYS releases.</td>
</tr>
<tr>
<td></td>
<td>-products List all available licensed ANSYS products.</td>
</tr>
<tr>
<td></td>
<td>-releases List all ANSYS releases which MpCCI can find.</td>
</tr>
<tr>
<td></td>
<td>-scan &lt;database file&gt; Run the scanner and create a scanner output file.</td>
</tr>
</tbody>
</table>

The subcommands -align, -info and -releases are described in 1.1 Common MpCCI Subcommands for Simulation Codes.

The subcommand -products lists the available ANSYS products, e.g. those for which licenses are available. The list looks like:

```plaintext
> mpcci ansys -products
ane3fl
ansys
emag
structds
```

See also “3.5 Choosing an ANSYS Product” in the ANSYS 100 documentation.
3.4 Code Adapter Reference

Within the MpCCI distribution the "adapters" directory contains the necessary software to connect the simulation programs to MpCCI depending on your license. The files are located within the subdirectory 
"<MpCCI_home>/codes/ANSYS/adapters".

This subdirectory is further divided by several release subdirectories, e.g. "90" and "100". The version directories are further divided by several architectures (e.g. "linia32"). There you find the library files of the ANSYS adapter (e.g. "libansysmpcci.so"). The connection to MpCCI is established using these shared libraries. The binding to the APDL command `mpcci` is set in the "ansys_ext.tbl" file.

To enable coupling capabilities the `mpcci` command is added to the standard APDL commands. By the command line options you decide which coupling function is called. The basic tasks of the command are to initialize the coupling and to manage the data transfer to MpCCI commands.

⚠️ Ensure that the directory path of your MpCCI-APDL scripts on Windows do not contain any white spaces!
4 FINE/Hexa

4.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>CFD, Fluid, FluidThermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>NUMECA International</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.numeca.com">www.numeca.com</a></td>
</tr>
<tr>
<td>Support</td>
<td><a href="mailto:support@numeca.com">support@numeca.com</a></td>
</tr>
</tbody>
</table>

4.1.1 Supported Coupling Schemes

FINE/Hexa supports exchange after solution. Unidirectional and bidirectional transfer is possible.

4.1.2 Supported Platforms and Versions

<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCI_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>win64</td>
<td>(xp64</td>
<td>vista)</td>
</tr>
<tr>
<td>x86_64</td>
<td>(sles10</td>
<td>linux)</td>
</tr>
</tbody>
</table>

4.1.3 References

FINE/HEXA Documentation is part of the FINE/Hexa distribution. The section “Co-simulation using MpCCI” describes how to use FINE/Hexa with MpCCI.

4.1.4 Adapter Description

The code adapter for FINE/Hexa is developed by NUMECA International in cooperation with Fraunhofer SCAI. The code adapter for FINE/Hexa is based on a dynamic library “libmpcci_cadapt_XXX.so”, which is loaded by FINE/Hexa. It includes the necessary interface functions.

⚠️ Remove the code adapter delivered within the FINE/Hexa installation directory. For example:

- NUMECA_DIR/finehexa210_3/LINUX/_lib_sx86_64/libmpcci_cadapt_64.so and
- NUMECA_DIR/finehexa210_3/LINUX/_lib_sx86_64/libmpcci_64.so
4.2 Coupling Process

4.2.1 Model Preparation

The FINE/Hexa model (*.run" file) has to be prepared with FINE/Hexa GUI. Please consider the following approach for model preparation:

- The FINE/Hexa solver internally operates only in SI units. The geometrical dimensions should best defined in meters. However, it is possible to define a scaling factor in FINE/Hexa GUI.

- The coupling components must be identified during model preparation. Their names correspond to the FINE/Hexa boundaries which can be identified using the page Coupling in FINE/Hexa GUI (see Figure 1). For details, see FINE/Hexa documentation or contact NUMECA International support.

- If you exchange relative pressure (e.g. RelWallForce or Overpressure), you must set the reference pressure to an appropriate value, which usually corresponds to the atmospheric pressure. Moreover, if the relative force is exchanged, the user has to set on the toggle Force and Torque in the boundary conditions panel (for each coupled surface). (See Figure 2)

- Computation with MpCCI is based on the use of a batch file, therefore it has to be saved before by FINE/Hexa GUI.

![Figure 1: Identifying boundaries in FINE/Hexa GUI](image.png)
Figure 2: Setting a reference pressure and Force and Torque in FINE/Hexa GUI
4.2.2 Models Step

The FINE/Hexa version used for the computation is determined by the batch file. The batch file is used to launch the computation by MpCCI. It specifies the version number and solver precision. The model file corresponds to the ".run" file saved by FINE/Hexa GUI as used for a stand-alone FINE/Hexa simulation. As pointed out in the previous section, FINE/Hexa solver internally operates in SI units and therefore no other choices are available.

![Figure 3: FINE/Hexa options in the Models Step](image)

4.2.3 Coupling Step

FINE/Hexa supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>Scalar</td>
<td>1.0 kg/m³</td>
<td>Field</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>IterationNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>NDisplacement</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>NPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>OverPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>RefPressure</td>
<td>Scalar</td>
<td>1.12e5 N/m²</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>RelWallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Quantity</td>
<td>Dim.</td>
<td>Default Value</td>
<td>Integration Type</td>
<td>Coupling Dimension</td>
<td>Location</td>
<td>Send Option</td>
<td>Receive Option</td>
</tr>
<tr>
<td>----------------</td>
<td>-------</td>
<td>---------------</td>
<td>-------------------</td>
<td>--------------------</td>
<td>---------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>TimeStepNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Velocity</td>
<td>Vector</td>
<td>0.0 m/s</td>
<td>Field</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

⚠️ It is not possible for FINE/Hexa solver to receive DeltaTime quantity. The time step can only be sent to the counterpart code.

### 4.2.4 Go Step

In the Go Step, the user has nothing to specify. The Initial quantities transfer is fixed to receive and the computational parameters that define the computation have been set in FINE/Hexa GUI. These parameters control for example if the job has to be performed in parallel or in a serial, in single or double precision, etc.

### 4.2.5 Running the Computation

When the Start button is pressed, MpCCI starts the " .batch" file corresponding to the FINE/Hexa project. The parallel execution setting is done in the FINE/Hexa GUI (Computational Control - Launching Mode) as shown in Figure 4.

If the Stop button is pressed, a stop-file is created in the directory of the computation and FINE/Hexa will stop the next time it checks the presence of a stop file.

![Parallel execution setting in the FINE/Hexa GUI](image)
4.2.5.1 Batch Execution

FINE/Hexa always runs as a batch process.

4.2.6 Post-Processing

Post-processing for the FINE/Hexa part of the results can be performed as in ordinary computations, e.g. with CFVIEW, which is part of FINE/Hexa package.

4.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for FINE/Hexa are:

Usage:
   mpcci FINEHexa [-]option

Synopsis:
   'mpcci FINEHexa' is used to get information about FINEHexa.

Options:
   -help                This screen.
   -info                List verbose information about all FINEHexa releases.
   -releases            List all FINEHexa releases which MpCCI can find on the local system.
   -scan <input-file>   Run the scanner and create an output file.

The subcommands -info and -releases are described in 1.1 Common MpCCI Subcommands for Simulation Codes.

4.4 Code Environment

4.4.1 Prerequisites for a coupled simulation

To run a coupled simulation, you need the following:
• FINE/Hexa 2.10 installation
• License for coupled simulation using MpCCI

4.5 Code Adapter Reference

The code adapter is distributed as a dynamic library, which is located in
"<MpCCI_home>/codes/FINEHexa/adapters/".
A link to the corresponding dynamic library has to be created in FINE/Hexa installation directory as explained in documentation. This dynamic library is used to establish connection between MpCCI and FINE/Hexa.
5 FINE/Turbo

5.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>CFD, Fluid, FluidThermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>NUMECA International</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.numeca.com">www.numeca.com</a></td>
</tr>
<tr>
<td>Support</td>
<td><a href="mailto:support@numeca.com">support@numeca.com</a></td>
</tr>
</tbody>
</table>

5.1.1 Supported Coupling Schemes

FINE/Turbo supports exchange before solution. Unidirectional and bidirectional transfer is possible.

5.1.2 Supported Platforms and Versions

<table>
<thead>
<tr>
<th>platform</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>aix53</td>
<td>X</td>
</tr>
<tr>
<td>ia64</td>
<td>X</td>
</tr>
<tr>
<td>win32</td>
<td>X</td>
</tr>
<tr>
<td>win64</td>
<td>X</td>
</tr>
<tr>
<td>x86</td>
<td>X</td>
</tr>
<tr>
<td>x86_64</td>
<td>X</td>
</tr>
</tbody>
</table>

5.1.3 References

FINE/Turbo V8.8 Documentation is part of the FINE/Turbo distribution. The section “Co-simulation using MpCCI” describes how to use FINE/Turbo with MpCCI.

5.1.4 Adapter Description

The code adapter for FINE/Turbo is developed by NUMECA International in cooperation with Fraunhofer SCAI. The adapter is distributed as part of FINE/Turbo software.
5.2 Coupling Process

5.2.1 Model Preparation

The FINE/Turbo model (".run" file) has to be prepared with FINE/Turbo GUI. Please consider the following approach for model preparation:

- The expert parameter IMPCCI has to be set to 1 (see Figure 1).
- The coupling components must be identified during model preparation. For the moment, the coupling boundaries are identified by grouping solid patches in groups named Moving_Surface or Heated_Surface for mechanical or thermal coupling respectively.
- The component names correspond to the FINE/Turbo block zones.
- When sending relative wall force (RelWallForce) or relative pressure (e.g. Overpressure) a reference pressure must be defined in the Reference Values Panel in Flow Model, which usually corresponds to the atmospheric pressure (see Figure 2).
- If the relative force is exchanged, the user can the toggle Force and Torque in the boundary conditions panel to follow the lift and drag.
- For FSI problems, which involve mesh deformation, some expert parameters have to be set correctly to choose the most appropriate deformer. Please refer to FINE/Turbo documentation or contact NUMECA International support for a complete description. Remeshing is not possible during a computation.
- If nodal position (NPosition) or nodal displacements (NDisplacement) are received, FINE/Turbo offers two methods:
  - the solving of Laplace equation computing the displacement
  - the radial basis function interpolating the displacement on boundary nodes

Please refer to FINE/Turbo documentation for further details.

- Computation with MpCCI is based on the use of a batch file, therefore it has to be saved before by FINE/Hexa GUI.
Figure 1: Setting the expert parameter in FINE/Turbo GUI

Figure 2: Setting the reference pressure in FINE/Turbo GUI
5.2.2 Models Step

The FINE/Turbo version used for the computation is determined by the batch file. The batch file is used to launch the computation by MpCCI. It specifies the version number. Model file corresponds to the "run" file saved by FINE/Turbo GUI. As FINE/Turbo works in SI units, no choice is given for the unit system to the user.

![Figure 3: FINE/Turbo options in the Models Step](image)

5.2.3 Coupling Step

FINE/Turbo supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td>Scalar</td>
<td>1.0 kg/m³</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>NDisplacement</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>NPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>OverPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>RefPressure</td>
<td>Scalar</td>
<td>1.12e5 N/m²</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>RelWallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallHeatFlux</td>
<td>Scalar</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>
5 FINE/Turbo

It is not possible for FINE/Turbo solver to receive DeltaTime or PhysicalTime quantity. The time step can only be sent to the counterpart code.

5.2.4 Go Step

In the Go Step, the user has nothing to specify. The Initial quantities transfer is fixed to receive and the computational parameters that define the computation have been set in FINE/Hexa GUI. These parameters control for example if the job has to be performed in parallel or in a serial, in single or double precision, etc.

5.2.5 Running the Computation

When the Start button is pressed, MpCCI starts the " .batch" file corresponding to the FINE/Turbo project specified in FINE/Turbo GUI.

If the Stop button is pressed, a stop-file is created in the directory of the computation and FINE/Turbo will stop the next time it checks the presence of a stop file.

5.2.5.1 Batch Execution

FINE/Turbo always runs in Batch mode.

5.2.6 Post-Processing

Post-processing for the FINE/Turbo part of the results can be performed as in ordinary computations, e.g. with CFVIEW, which is part of FINE/Turbo package.

5.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for FINE/Turbo are:

Usage:
    mpcci FINETurbo [-]option

Synopsis:
    'mpcci FINETurbo' is used to get information about FINETurbo.

Options:
<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-help</code></td>
<td>This screen.</td>
</tr>
<tr>
<td><code>-info</code></td>
<td>List verbose information about all FINETurbo releases.</td>
</tr>
<tr>
<td><code>-releases</code></td>
<td>List all FINETurbo releases which MpCCI can find on the local system.</td>
</tr>
<tr>
<td><code>-scan &lt;input-file&gt;</code></td>
<td>Run the scanner and create an output file.</td>
</tr>
</tbody>
</table>

The subcommands `-info` and `-releases` are described in 1.1 Common MpCCI Subcommands for Simulation Codes ↓.

## 5.4 Code Environment

### 5.4.1 Prerequisites for a coupled simulation

To run a coupled simulation, you need the following:

- FINE/Turbo 8.8-1 (or higher) installation
- License for coupled simulation using MpCCI and for mesh deformation if needed

## 5.5 Code Adapter Reference

The code adapter is delivered within the FINE/Turbo installation.
6 Flowmaster

6.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>Fluid and Thermal System simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>Flowmaster</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.flowmaster.com">www.flowmaster.com</a></td>
</tr>
<tr>
<td>Support</td>
<td><a href="mailto:UK-support@flowmaster.com">UK-support@flowmaster.com</a></td>
</tr>
<tr>
<td></td>
<td><a href="mailto:US-support@flowmaster.com">US-support@flowmaster.com</a></td>
</tr>
<tr>
<td></td>
<td><a href="mailto:DE-support@flowmaster.com">DE-support@flowmaster.com</a></td>
</tr>
<tr>
<td></td>
<td><a href="mailto:FR-support@flowmaster.com">FR-support@flowmaster.com</a></td>
</tr>
</tbody>
</table>

6.1.1 Supported Coupling Schemes

MpCCI supports coupling with Flowmaster 7.5.1 and higher.
Depending on the Flowmaster analysis type the following scheme is applied:

- Steady-State Analysis
  1. Flowmaster receives the data.
  2. Flowmaster analysis is requested.
  3. Flowmaster sends the data.

- Transient
  1. Flowmaster exchanges the data (send and receive).
  2. Flowmaster analysis is requested.

6.1.2 Supported Platforms and Versions

The following versions of Flowmaster are supported by MpCCI:

<table>
<thead>
<tr>
<th>platform</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>mswin_x86</td>
<td>MPCCI_ARCH 7.5 7.6</td>
</tr>
<tr>
<td>mswin_x86</td>
<td>X X</td>
</tr>
</tbody>
</table>

6.1.3 References

Flowmaster documentation is part of the Flowmaster distribution.
6.1.4 Adapter Description

The Flowmaster adapter is an executable based on the Flowmaster COM interface.

6.2 Coupling Process

You have to create and validate a Flowmaster model.

6.2.1 Model Preparation

Both prescribed flow (inlet/outlet) and pressure boundaries can be incorporated into a MpCCI co-simulation. The boundaries in a CFD model are represented within a Flowmaster network by a combination of flow or pressure sources. Figure 1 and Figure 2 show the components used to represent a prescribed flow boundary and a pressure boundary respectively.

![Figure 1: Prescribes flow boundary](image)

Note that the prescribed flow boundary incorporates a pressure source whereas the pressure boundary uses a flow source. Components within Flowmaster network, which are attached to the boundaries, are connected to the nodes in the normal manner.

- For each boundary component you have to activate the external boundary property of the component.
- For each flow source component you need to specify a flow value to initialize the flow source.

You have to create the.fmlink in order to export the information of your network. (see Figure 3)

1. You have to enter the report dialog and choose the MpCCI ASCII file.
2. Select a boundary component.
3. Click on Add in order to fill the list.
4. Mark the selected component and give a name for this component.
5. Repeat step 1 to 4 for all your boundary components.
6. Select the directory to save your file.
7. Enter the name of your network model with the file suffix ".fmlink".
8. Click on **create**.
Figure 3: Export Flowmaster boundaries dialog
6.2.2 Models Step

In the models step, the following options must be chosen:

**Flowmaster release**  Select the release of Flowmaster you want to use. *latest* (default) will select the latest version which is installed on your system.

**Data file**  Select the Data file of your Flowmaster project.

The MpCCI scanner uses the Flowmaster data file to extract model information. This file has the suffix "*.fmlink".

6.2.3 Coupling Step

The Flowmaster adapter only stores quantities directly ("Direct"). Flowmaster supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoundaryMassFlow</td>
<td>Scalar</td>
<td>0.0 kg/s</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>BoundaryMassFlux</td>
<td>Scalar</td>
<td>0.0 kg/m² s</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>BoundaryTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>BoundaryTotalPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>BoundaryVelocityMagnitude</td>
<td>Scalar</td>
<td>0.0 m/s</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>MassFlowRate</td>
<td>Scalar</td>
<td>0.0 kg/s</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>MassFluxRate</td>
<td>Scalar</td>
<td>0.0 kg/m² s</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Temperature</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
6.2 Coupling Process

### 6.2.4 Go Step

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>VelocityMagnitude</td>
<td>Scalar</td>
<td>0.0 m/s</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**Flowmaster** offers a number of options in the Go panel, see Figure 5.

**User Name** You have to provide the user name to log in the database.

**User Password** The password to authenticate the database server.

**Working Project name** The full project tree to the current project containing the network.

For example, the default root project name is “Flowmaster” and you have two sub-projects “coupling” and “testcases” with the following hierarchy: “Flowmaster\coupling\testcases”.

The project “testcases” contains some network models which one of them has been chosen for the
simulation.
Then you will have to provide the following working project name: “Flowmaster\coupling”.

Please select the Flowmaster analysis type  The analysis type to use for the simulation:

- Incompressible
  
  SS  Steady state simulation.
  SSH  Steady state simulation with heat transfer.
  ST  Transient simulation.
  STH  Transient simulation with heat transfer.

- Compressible
  
  CS  Steady state simulation.
  CT  Transient simulation.

Relaxation factor  Represents the weighting factor for controlling the convergence of the coupled simulation. Let $r_{n-1}$ be the value containing a quantity for the prescribed boundary used by Flowmaster for the (n-1)th coupling and let $r_n$ be the value of the same quantity calculated at the nth coupling. Then

$$r_n = (1 - \mu) \times r_n + \mu \times r_{n-1}$$

where $\mu$ is the weighting factor.

Use maximum number of couplings  Activates the maximum number of couplings to do for the simulation.

Specify the number of coupling  This is the number of coupling that will be done.

6.2.4.1 Running the Computation

By pressing the [Start] button in the Go Step of the MpCCI GUI, MpCCI starts the Flowmaster application. The output of the Flowmaster simulation is logged in a window and a file "mpcci_<model name>.log".

6.2.4.2 Post-Processing

You may use the Flowmaster interface and analyze the results by querying the database or creating a report.

6.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for Flowmaster are:
6.4 Code Adapter Description

Within the MpCCI distribution the "adapters" directory contains the necessary software to connect the simulation programs to MpCCI. The files are located within the subdirectory "<MpCCI.home>/Flowmaster/adapters".

This subdirectory is further divided by several release subdirectories, e.g. "7.5". The version directories are further divided by several architectures, e.g. "mswin.x86". There you find the executable of the Flowmaster adapter, e.g. "MpCCILink.exe". The connection to MpCCI is established using this executable and the implementation is based on the Flowmaster COM API.

For a prescribed flow boundary, the boundary component (component 3 in Figure 1) sets the pressure of the source provided by MpCCI via Flowmaster adapter. After the network analysis is complete the mass flow is read by the boundary component and returned to Flowmaster adapter.
7 FLUENT

7.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>CFD, Fluid, FluidThermal, FluidPlasma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>Fluent Inc. is a wholly owned subsidiary of ANSYS, Inc.</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.fluent.com">www.fluent.com</a></td>
</tr>
<tr>
<td>Support</td>
<td>Fluent Services Center at <a href="http://www.fluentusers.com">www.fluentusers.com</a></td>
</tr>
</tbody>
</table>

7.1.1 Supported Coupling Schemes

FLUENT supports exchange before and after solution. Unidirectional and bidirectional transfer is possible.

7.1.2 Supported Platforms and Versions

<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCI_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>6.3.26  12.0.7  12.0.16</td>
</tr>
<tr>
<td>aix51</td>
<td>aix51_power</td>
<td>X</td>
</tr>
<tr>
<td>aix51.64</td>
<td>aix51_power</td>
<td>X</td>
</tr>
<tr>
<td>hpux11</td>
<td>hpux11_parisc</td>
<td>X</td>
</tr>
<tr>
<td>hpux11.64</td>
<td>hpux11_parisc</td>
<td>X</td>
</tr>
<tr>
<td>hpux11_ia64</td>
<td>hpux1123_ia64</td>
<td>X</td>
</tr>
<tr>
<td>irix65_mips4</td>
<td>irix65_mips4</td>
<td>X</td>
</tr>
<tr>
<td>irix65_mips4_64</td>
<td>irix65_mips4</td>
<td>X</td>
</tr>
<tr>
<td>lnamd64</td>
<td>(sles10</td>
<td>linux</td>
</tr>
<tr>
<td>lnia64</td>
<td>linux_ia64</td>
<td>X X X</td>
</tr>
<tr>
<td>lnx86</td>
<td>linux_x86</td>
<td>X</td>
</tr>
<tr>
<td>ntx86</td>
<td>mswin_x86</td>
<td>X</td>
</tr>
<tr>
<td>ultra</td>
<td>solaris_sparc</td>
<td>X</td>
</tr>
<tr>
<td>ultra_64</td>
<td>solaris_sparc</td>
<td>X X X</td>
</tr>
<tr>
<td>win64</td>
<td>(xp64</td>
<td>vista</td>
</tr>
</tbody>
</table>

7.1.3 References

**FLUENT 6.3 Fluid-Structure Interaction (FSI) Module Manual** This manual describes how to use FLUENT with MpCCI for code coupling. It is available from the FLUENT User Services Center:


**FLUENT 6.3 Documentation** is part of the FLUENT distribution.
**Abaqus Fluid-Structure Interaction User’s Guide** This guide is available for registered Abaqus users via the Abaqus support homepage [abaqus.custhelp.com](http://abaqus.custhelp.com): Log into Abaqus Answers and search for “FSI guide”. The FSI guide contains several examples of coupled simulations with Abaqus and FLUENT.

### 7.1.4 Adapter Description

MpCCI uses the user-defined function (“UDF”) interface, which is provided by FLUENT. A set of such functions is included in the MpCCI distribution as a shared library, which must be loaded by FLUENT.

The user-defined functions must be called at appropriate stages of the simulation. For this, FLUENT offers a number of so-called “function hooks”, where these functions must be “hooked”.

### 7.2 Coupling Process

#### 7.2.1 Model Preparation

Models can be prepared as usually. The coupling components must be defined as separate surfaces or volumes. Figure 1 shows the list of surfaces defined in a FLUENT model, which then appear in the Coupling Step of the MpCCI GUI.

⚠️ If you exchange a relative pressure (e.g. RelWallForce or Overpressure), you must set the reference pressure to an appropriate value, which usually corresponds to the atmospheric pressure. For more information on relative and absolute pressures see the FSI section in ‣ V-3.1.2 Coupling Types ‣.

In FLUENT the reference pressure is set in the Reference Values panel, which can be found under Report ➔ Reference Values.

All further options required for coupling can be set in the MpCCI GUI.

#### 7.2.1.1 Setting UDF-Hooks

The user-defined functions for the MpCCI interface must be hooked to perform initialization of the coupled simulation and data transfer. It is recommended to let MpCCI handle loading and hooking of these functions automatically. Please choose the corresponding options in the Go Step of the MpCCI GUI:

- Auto install/make libudf for installation of the user-defined library,
- Auto load libudf to load it,
- Auto hook functions to hook the library functions, i.e. they will be called at certain stages of the simulation.

See ‣ 7.2.4 Go Step ‣ for a complete description of the Go Step options.
In some cases it is preferable not to use these automatic settings. Instead the necessary functions can also be hooked manually. A description of the functions is given in ▷ 7.4 Code Adapter Reference ◁.

Some functions can also be called while running the computation via the MpCCI Control Panel in the FLUENT GUI, see ▷ 7.2.5.1 The MpCCI Control Panel ◁.

### 7.2.1.2 Using Own UDFs and MpCCI

For FLUENT 6.3.26 the MpCCI functions are no longer stored in the "libudf" directory but in a separate directory "libmpcci". Therefore the MpCCI interface functions do not directly interfere with other interface functions - you can define and hook functions as if you were not using MpCCI.

⚠️ It is not possible to hook your own user-defined functions at places where MpCCI functions need to be hooked for data transfer. It does not make sense to receive data and set the same values by a user-defined function!

### 7.2.1.3 Deforming Meshes

In typical FSI problems, deformations are computed by the solid mechanics code and sent to FLUENT, i.e. FLUENT has to move boundaries and deform the mesh.

The FLUENT settings to achieve this are made automatically if you select Auto hook functions as well as Auto set MDM zones ("MDM" stands for Moving and Deforming Meshes) from the Go Step of the MpCCI GUI. This enables mesh motion, however the exact parameters cannot be chosen by MpCCI and should be set when preparing the model or before starting the iteration.

To setup the mesh deformation without using the MpCCI GUI, the dynamic mesh option must be en-
abled by selecting Define → Dynamic Mesh → Parameters... and selecting Dynamic Mesh in FLUENT. It is recommended to use Smoothing and Remeshing as Mesh Methods.

The coupling components must be defined as dynamic mesh zones which is described in 7.4.2 UDF-Hooks.

⚠️ Currently it is not possible to remesh a coupling component itself during a computation.

### 7.2.2 Models Step

![FLUENT options in the Models Step](image)

In the models step, the following options must be chosen:

**FLUENT version** Select the FLUENT version from 2d, 2ddp, 3d, 3ddp. The version must match your casefile.

**FLUENT release** Select the release of FLUENT you want to use. The version must match your casefile. latest (default) will select the latest version which is installed on your system.

**Run 64 bit version** If a 64 bit version is available on your platform, you can select to run it here. This option is ignored on other platforms.

**casefile** Select the casefile of your FLUENT model.

### 7.2.3 Coupling Step

The FLUENT adapter only stores some quantities directly (“Dir”), most quantities are first written to user-defined memory (“UDM”).
**FLUENT** supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face,</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AcstPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BodyForce</td>
<td>Vector</td>
<td>0.0 N/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>BoundayMassFlux</td>
<td>Scalar</td>
<td>0.0 kg/m² s</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>BoundayStaticPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>BoundayTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>BoundayTotalPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>BoundayVelocityMagnitude</td>
<td>Scalar</td>
<td>0.0 m/s</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>CGAngle</td>
<td>Vector</td>
<td>0.0 rad</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>CGOmega</td>
<td>Vector</td>
<td>0.0 rad/s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>CGPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>CGVelocity</td>
<td>Vector</td>
<td>0.0 m/s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>ChargeDensity</td>
<td>Scalar</td>
<td>0.0 C/m³</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>Current1</td>
<td>Scalar</td>
<td>0.0 A</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>CurrentDensity</td>
<td>Vector</td>
<td>0.0 A/m²</td>
<td>Flux dens.</td>
<td>Face,</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume</td>
<td>UDM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>Density</td>
<td>Scalar</td>
<td>1.0 kg/m³</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>DynPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ElectCond1</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Face,</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume</td>
<td>UDM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectCond3</td>
<td>Vector</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Face,</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume</td>
<td>UDM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectCondX</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ElectCondY</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ElectCondZ</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ElectricField</td>
<td>Vector</td>
<td>0.0 V/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ElectricFlux</td>
<td>Vector</td>
<td>0.0 C/m²</td>
<td>Flux dens.</td>
<td>Face,</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume</td>
<td>UDM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrRes1</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Face,</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume</td>
<td>UDM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrRes3</td>
<td>Vector</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Face,</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Volume</td>
<td>UDM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrResX</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ElectrResY</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ElectrResZ</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>Enthalpy</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>Quantity</td>
<td>Dim.</td>
<td>Default Value</td>
<td>Integration Type</td>
<td>Coupling Dimension</td>
<td>Location</td>
<td>Send Option</td>
<td>Receive Option</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>---------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>------------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>FilmTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>HeatFlux</td>
<td>Vector</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>HeatSource</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>IntFlag</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>IterationNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>JouleHeat</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>JouleHeatLin</td>
<td>Scalar</td>
<td>0.0 W/m³ K</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>LorentzForce</td>
<td>Vector</td>
<td>0.0 N/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>MagneticField</td>
<td>Vector</td>
<td>0.0 A/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>MagneticFlux</td>
<td>Vector</td>
<td>0.0 T</td>
<td>Flux dens.</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MassFlowRate</td>
<td>Scalar</td>
<td>0.0 kg/s</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MassFluxRate</td>
<td>Scalar</td>
<td>0.0 kg/m² s</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>NPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Buf</td>
<td></td>
</tr>
<tr>
<td>OverPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>PorePressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>RealFlag</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>RefPressure</td>
<td>Scalar</td>
<td>1.12e5 N/m²</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>RelWallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>Residual</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>SpecificHeat</td>
<td>Scalar</td>
<td>1.0 J/kg K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>Temperature</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ThermCond1</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ThermCond3</td>
<td>Vector</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ThermCondX</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ThermCondY</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ThermCondZ</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>TimeStepNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>TotalPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>udm00</td>
<td>Scalar</td>
<td>0.0</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>uds00</td>
<td>Scalar</td>
<td>0.0</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDS</td>
</tr>
</tbody>
</table>
7.2.4 Go Step

**FLUENT** offers a number of options in the Go panel, see Figure 3.

**Initial quantities transfer** Select one of receive, send or exchange to determine the coupling algorithm. The construction of coupling algorithms is further described in [V-3.3 Coupling Algorithms](#).

**Run in batch mode** If this option is selected, **FLUENT** will not start the graphical user interface, i.e. no user interaction is possible. For batch mode a journal file, which contains the **FLUENT** commands for analysis execution, is required. See [7.2.5.3 Batch Execution](#) for details.

**Project name** Name of the **FLUENT** project, which is also used as name for the output files. If no name is given here, the name of the case file is used.

**Data file** For the analysis a data file can be read. Same as loading a data file in the **FLUENT** GUI.

**Optional journal files** A semicolon-separated list of journal files can be given, which will be read by **FLUENT** when it is started. A journal file must be given for batch execution.

**Additional command line options** Additional command line options for **FLUENT** can be given here, they will directly be used when **FLUENT** is started.

**Auto install/make libudf** Automatically copy the user-defined library which contains the **MpCCI** interface to the working directory. If this button is not selected, the library must be copied by hand. Therefore it should always be used in a standard analysis.

**Auto read case data** Read the case file which was selected in the Models Step when **FLUENT** is started.

**Auto load libudf** Load the code adapter library. This options should only be unselected for special purposes.

**Auto hook functions** Hooks the user-defined functions of the **MpCCI** code adapter library automatically. If not set, the functions must be hooked manually as described in [7.2.1.1 Setting UDF-Hooks](#).
**Auto set MDM zones**  Sets the interface zones for the Moving Deforming Mesh automatically. If nodal displacements are transferred from a structural code, this option should be selected to allow deformations of the FLUENT mesh.

**Auto set BC’s**  Automatically set the boundary conditions for the coupling regions, which is necessary to enable data transfer on these boundaries.

**Run parallel**  Select this, if you want to run FLUENT in parallel mode. See 7.2.5.2 Parallel Execution.
7.2.5 Running the Computation

Coupled simulations can be run using the FLUENT window. For running a simulation without a graphical interface, see 7.2.5.3 Batch Execution.

Before FLUENT is started by pressing the Start button in the Go Step of the MpCCI GUI, MpCCI creates a journal file "mpcci_{problem name}.jou", which performs the tasks selected in the MpCCI GUI Go Step. These steps are also reflected in FLUENT’s output which partially depend on the settings (e.g. quantities, steady state/transient):

- Load "cosimtools.scm" which contains helper functions needed for a coupled simulation:
  ```
  Loading "/home/user/mpcci/codes/FLUENT/cosimtools.scm"
  Done.
  ```

- Enable coupling with MpCCI,
  ```
  > (rpsetvar 'mpcci/on? #t)mpcci/on?
  ```

- Read the case file if Auto read case data is selected,
  ```
  > /file/read-case "example.cas"
  Reading "example.cas"...
  ```

- Load the MpCCI adapter library "libmpcci" if Auto load libudf is checked,
  ```
  > /define/user-defined/compiled-functions/load libmpcci
  "/home/user/computations/fluent"
  Opening library "libmpcci"...
  Library "libmpcci/lnx86/3d/libudf.so" opened
  UDF_List_globals
  UDF_Write_globals
  (output is followed by a list of all functions)
  ```

- Hook the MpCCI functions if Auto hook functions is set, e.g.
  ```
  > (co-set-udf-hook 'init "UDF_Initialize::libmpcci")
  > (co-set-udf-hook 'adjust "UDF_Adjust::libmpcci")
  > (register-dynamic-function "exchange-mpcci-quantities" #t #f \udf-on-demand
  "UDF_Dynamic_exchange::libmpcci")functions
  ```

- Set dynamic mesh zones if Auto set MDM zones is selected,
> /define/models/dynamic-mesh? yes
> /define/dynamic-zones/create 6 motion type: (stationary rigid-body deforming
user-defined)

user-defined UDF_Grid_position::libmpcci
adjacent cell zone name/id [2]
cell height (air-remesh) (m) [0]

- Set boundary conditions if Auto set BC’s is enabled,

(co-set-zone-profile "wall" "wall-temperature" "UDM00_Profile::libmpcci")Fluent
MpCCI_UDMprofile: Set default zero before MpCCI init.

- Load the MpCCI Control Panel.

load "/home/user/mpcci/codes/FLUENT/mpccipanel.scm")
Loading "/home/user/mpcci/codes/FLUENT/mpccipanel.scm"
Done.

If all coupling functions are hooked - either automatically or as described in \(\uparrow\)7.2.1.1 Setting UDF-Hooks\(\downarrow\) - no special steps need to be carried through to start the simulation:

- Initialize the solution, this will also trigger MpCCI initialization.
- Start the iteration, MpCCI data transfer routines will be called automatically.

### 7.2.5.1 The MpCCI Control Panel

The MpCCI Control Panel (Figure 4) is an extra panel in the FLUENT GUI, which is available if FLUENT is started during a coupled simulation. It allows to perform some coupling actions manually. Select Solve→MpCCI Control... from the FLUENT menu to open the panel.

Select Use MpCCI to enable coupling with MpCCI.

The second box of options hooks or unhooks MpCCI functions. If Initialize MpCCI is checked, the MpCCI initialization will be carried through automatically with FLUENT initialization. If you select the box next to Automated Exchange of Quantities you can decide at which points in the simulation data transfers should take place. Choose one of Before each iteration (recommended for steady-state problems), Before each time step (i.e. exchange before iteration) or After each time step (exchange after iteration). For each selection a specific MpCCI function is hooked.

⚠️ Due to a FLUENT bug, the After each time step option does not work. Please hook the EXECUTE_AT_END function manually instead: Select Define→User Defined →Function Hooks... to open the UDF Hooks panel, press the Edit... button in the Execute at End row, select UDF_At_end::libmpcci in the panel, press Add and OK. See also \(\uparrow\)7.4.2 UDF-Hooks\(\downarrow\).
The further buttons allow to perform coupling tasks on demand. Initialize initialized coupling with MpCCI, Finalize ends the coupling process gracefully, and Abort directly aborts the coupling process.

The transfer buttons trigger a transfer of quantities, Send will yield sending of all quantities which were selected in the Coupling Step of the MpCCI GUI. Receive yields receiving all quantities selected to receive and Exchange will yield a complete transfer of all selected quantities.

Steady Update Mesh yields an update of the mesh, which makes sense after receiving displacements or node positions.

### 7.2.5.2 Parallel Execution

For a parallel run of FLUENT (i.e. FLUENT itself uses several parallel processes) additional options can be selected in the Go Step as shown in Figure 5.

Please read also chapter “31. Parallel Processing” of the FLUENT 6.3 User’s Guide.

- **No. of parallel processes** Select how many FLUENT processes you wish to start.
- **Interconnect** Select the the interconnect type as given in the FLUENT manual, this is passed as
option `-p` to FLUENT.

**Parallel communicator**  Select the parallel communicator as given in the FLUENT manual, this is passed as option `-mpi=` to FLUENT.

**Shared file system (no file copy)**

**Optional 'host host ...' to be used**  Enter host names for parallel execution of FLUENT.

**Optional hostlist file**  Specify a hostfile, from which host names are extracted.

**Use default hostfile**  A default hostfile can be configured by setting `MPCCI_HOSTLIST_FILE`, see V-3.4.2 Hostlist files.

### 7.2.5.3 Batch Execution

FLUENT can be started without the graphical user interface. This is important if you run a simulation on a remote computer with text access only. For batch mode in a coupled simulation you must check the Run in batch mode button in the Go Step and provide a journal file which must be given under Optional journal files:

As no graphical interface is available in batch mode you should perform all steps needed for the simulation in the journal file. A simple journal file for a transient analysis would contain:
FLUENT VI Codes Manual

;; Initialize flow and connect to \mpcci
/solve initialize initialize-flow
;; Perform unsteady iterations for a specified number of time steps
/solve dual-time-iterate 5 20
;; Exit simulation
/exit

If you want to do subcycling (iterations without exchange) the journal file might contain:

;; Define \mpcci functions using cosimtools
(define udf-init (co-get-udf-lib-name "UDF_Init_MpCCI")
(define udf-exit (co-get-udf-lib-name "UDF_Exit_MpCCI")
(define udf-xchg (co-get-udf-lib-name "UDF_Xchg_on_demand")
(define udf-recv (co-get-udf-lib-name "UDF_Recv_on_demand")
(define udf-send (co-get-udf-lib-name "UDF_Send_on_demand")

;; Define number of iterations without coupling
(define IterNum 10)
;; Define number of coupling steps
(define CouplingSteps 10)
;; Define autosave of case and data files
/file/autosave/case-frequency 10
/file/autosave/data-frequency 10
/file/autosave/overwrite-existing-files yes

;; Initialize flow
/solve/initialize initialize
;; Connect to \mpcci
(\%udf-on-demand udf-init)
;; Send data
(\%udf-on-demand udf-send)
;; Do <CouplingSteps> coupling steps with a data exchange
;; and <IterNum> uncoupled iterations between
(do ((i 0 (+ i 1)))
  ((= i CouplingSteps))
  (iterate IterNum)
  (\%udf-on-demand udf-xchg))

;; Write case and data file
wcd file
;; Exit simulation
/exit yes
It is required that the MpCCI libudf is loaded and the necessary functions are hooked. This can be achieved by selecting the required Auto-options in the Go Step (select all if in doubt) or including the corresponding commands in the journal file. The commands for loading and functions hooking can be found in the journal file which is written by MpCCI at start-up of a FLUENT simulation, they are also described in ▷ 7.2.1.1 Setting UDF-Hooks ◁.

7.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for FLUENT are:

**Usage:**
```
mpcci FLUENT [-]option
```

**Synopsis:**
Use mpcci FLUENT to get information about FLUENT and to build/install your private adapter ...

**Options:**
```
-help
This screen.

-info
List verbose information about all FLUENT releases.

-libudf <ARGS>
Install the FLUENT "libudf/ARCH/VERSION/libudf.so"

in your current working directory - where the .cas and .dat files are located - by either just copying the MpCCI libudf’s or remaking the libudf from MpCCI and your own sources located in "libudf/src".
You do NOT need to have a "libudf/Makefile" and/or "libudf/src/makefile" prepared since the makefiles are generated automatically from your FLUENT installation.

ARCH : The FLUENT architecture token
   (automatically determined by MpCCI)
VERSION: The version 2d, 3d_node etc.
Please specify the FLUENT release (e.g. 12.1.4) you
would like to use and a list of versions (2d, 3d_node etc.).

For more information type "mpcci FLUENT libudf".

-releases
List all FLUENT releases which MpCCI can find.

-scan <casfile>
Run the scanner and create a scanner output file.

The subcommands -info and -releases are described in 1.1 Common MpCCI Subcommands for Simulation Codes.

mpcci fluent -libudf <ARGS>
The libudf subcommand installs a version of the MpCCI libudf as described above.

Usage:
mpcci FLUENT libudf <release#|latest> [-64] version version version ...

Examples:
mpcci FLUENT libudf 6.3.26 3d 3ddp 2d 2ddp
mpcci FLUENT libudf latest 3ddp 3ddp_host

7.4 Code Adapter Reference

7.4.1 The MpCCI UDF Library

The MpCCI code adapter for FLUENT uses user-defined functions ("UDF") and user-defined memory ("UDM") to manage the data transfer. The code adapter is distributed as a dynamic library, which is located in "<MpCCI home>/codes/FLUENT/adapters/<FLUENT release>/<platform>/<FLUENT code> ".

Before running the analysis, the code adapter must be copied to a subdirectory "libmpcci" of the working directory. This task is performed automatically by MpCCI if the option Auto install/make libudf is selected in the Go Step. Otherwise the command mpcci FLUENT libudf can be used. The working directory is the directory where the case file "*.cas" is located and where FLUENT is executed.

The adapter functions must be hooked at different places in FLUENT, see 7.2.1.1 Setting UDF-Hooks. A list of all adapter functions is given in Table 2 on page 79.
More information on user-defined functions (UDFs) can be found in the “FLUENT UDF Manual” which is part of the FLUENT documentation.

7.4.2 UDF-Hooks

The data transfer with MpCCI is realized with user-defined functions (“UDF”), which are included in the MpCCI distribution. A list of all UDFs of the MpCCI adapter is given in Table 2 on page 79.

In some cases it is preferable to set or check the necessary functions hooks manually.

Most UDF functions can be hooked using the User-Defined Function Hooks panel, which opens when selecting Define → User-Defined → Function Hooks... from the FLUENT menu.

For a coupled simulation the following steps must be carried through by user-defined functions:

**MpCCI Initialization** is realized in **UDF.Initialize**, which can be hooked in the UDF-hooks panel or by pressing the Initialize button of the MpCCI Control panel.

**Data exchange** for transient problems with dynamic mesh is carried through by **UDF.Dynamic_exchange** (exchange before iteration) or by **UDF.At.end** (exchange after iteration). For steady state problems **UDF.Adjust** should be used to achieve an exchange for each iteration. **UDF.Adjust** and **UDF.At.end** can be hooked over the FLUENT function hook panel, whereas **UDF.Dynamic_exchange** can be hooked over the MpCCI Control panel or by the Auto hook functions in MpCCI Go Step.

**Data storage** For some quantities the data is stored in user defined memory (UDM). To distinguish the quantities, a storage index is set for each quantity in the Coupling Step, as shown in Figure 6. If you do not change it, the index is increased automatically. In addition the corresponding UDM... functions must be hooked at appropriate places.

There are three types of such quantities:

- Profiles for various boundary conditions like inlet or outlet boundaries use the boundary functions **UDM0.0_Profile** to **UDM10.Profile** to set the received quantities on the boundaries. Hook the function in the Boundary Conditions panel of FLUENT (Define → Boundary Conditions... in the menu).
- Source terms are also set in the Boundary Conditions panel but not for inlet or outlet surfaces but for fluid type boundaries: Check Source Terms in the Fluid window and select the UDM...Source function for the appropriate source.
- Material property quantities can be set in the Materials panel (Define → Materials). Select user-defined and the appropriate UDM...Property function in the User-Defined Functions panel.

In addition to this minimal subset of user-function calls, certain situation require additional functions:

**Analysis with Deforming Mesh**: If the quantities **NDisplacement**, **NPosition** or rigid body motions are received by FLUENT, the mesh must be updated by a user-defined function. The appropriate
functions can be hooked by selecting Define→Dynamic Mesh→Zones... from the FLUENT window. Select the coupling components from the list of Zone Names, choose User-Defined as type and select UDF_Grid_motion::libmpcci for NDisplacement and UDF_Grid_position::libmpcci if NPosition is received on the selected component. For the rigid body quantities CGAngle, CGOmega, CGPosition or CGVelocity the type Rigid Body must be selected together with UDF_Grid_position::libmpcci.

**Restart Analysis:** To store global values in the "*.dat" file after the first analysis and reread it from there in a restart you should hook UDF_Read_globals and UDF_Write_globals in the UDF-hooks panel as Read Data and Write Data functions.

**Exchange of time step size:** If the time step size is received by FLUENT it is required to hook a function in the Iterate panel of FLUENT (Solve→Iterate...). Set the Time Stepping Method to Adaptive and select UDF_Deltat::libmpcki as user-defined time step.
<table>
<thead>
<tr>
<th>UDF Hook</th>
<th>MpCCI Function</th>
<th>Purpose / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT</td>
<td>UDF_Initialize</td>
<td>Initialize the connection to MpCCI.</td>
</tr>
<tr>
<td>EXECUTE_AT_END</td>
<td>UDF_At_end</td>
<td>MpCCI data transfer.</td>
</tr>
<tr>
<td>EXECUTE_AT_EXIT</td>
<td>UDF_At_exit</td>
<td>Shutdown the MpCCI connection.</td>
</tr>
<tr>
<td>ADJUST</td>
<td>UDF_Adjust</td>
<td>MpCCI data transfer.</td>
</tr>
<tr>
<td>DELTAT</td>
<td>UDF_Deltat</td>
<td>Set the time step size if it is received by FLUENT.</td>
</tr>
<tr>
<td>ON_DEMAND</td>
<td>UDF_List Globals</td>
<td>List global variables.</td>
</tr>
<tr>
<td></td>
<td>UDF_Dynamic_exchange</td>
<td>MpCCI data transfer.</td>
</tr>
<tr>
<td></td>
<td>UDF_Exit_MpCCI</td>
<td>Shutdown the MpCCI connection.</td>
</tr>
<tr>
<td></td>
<td>UDF_Abort_MpCCI</td>
<td>Abort the MpCCI connection.</td>
</tr>
<tr>
<td></td>
<td>UDF_Init_MpCCI</td>
<td>Initialize the connection to MpCCI.</td>
</tr>
<tr>
<td></td>
<td>UDF_Send_on_demand</td>
<td>MpCCI data transfer: Send only.</td>
</tr>
<tr>
<td></td>
<td>UDF_Recv_on_demand</td>
<td>MpCCI data transfer: Receive only.</td>
</tr>
<tr>
<td></td>
<td>UDF_Xchg_on_demand</td>
<td>MpCCI data transfer.</td>
</tr>
<tr>
<td>RW_FILE</td>
<td>UDF_ReadGlobals</td>
<td>Read global data from &quot;*.dat&quot; file.</td>
</tr>
<tr>
<td></td>
<td>UDF_WriteGlobals</td>
<td>Write global data into &quot;*.dat&quot; file.</td>
</tr>
<tr>
<td>CG_MOTION</td>
<td>UDF_CG_motion</td>
<td>Set the rigid body motion if it is received by FLUENT.</td>
</tr>
<tr>
<td>GRID_MOTION</td>
<td>UDF_Grid_motion</td>
<td>Move nodal coordinates by the received displacement.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function must be hooked if NDisplacement is received.</td>
</tr>
<tr>
<td></td>
<td>UDF_Grid_position</td>
<td>Set nodal coordinates to the received position.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This function must be hooked if NPosition is received.</td>
</tr>
<tr>
<td>PROFILE</td>
<td>UDM00_Profile</td>
<td>Set boundary profile values stored in UDM 0 - 10.</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UDM10_Profile</td>
<td></td>
</tr>
<tr>
<td>SOURCE</td>
<td>UDM00_Source</td>
<td>Set a cell source value stored in UDM 0 - 10.</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UDM10_Source</td>
<td></td>
</tr>
<tr>
<td>PROPERTY</td>
<td>UDM00_Property</td>
<td>Set a cell property value stored in UDM 0 - 10.</td>
</tr>
<tr>
<td></td>
<td>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UDM10_Property</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: User-defined functions (UDFs) of the MpCCI FLUENT adapter
8 FLUX

8.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>Electromagnetism</th>
<th>Thermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>CEDRAT Group</td>
<td></td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.cedrat.com">www.cedrat.com</a></td>
<td></td>
</tr>
<tr>
<td>Support</td>
<td><a href="mailto:support@cedrat.com">support@cedrat.com</a></td>
<td></td>
</tr>
</tbody>
</table>

8.1.1 Supported Coupling Schemes

MpCCI supports coupling with FLUX 3D. FLUX supports exchange after solution. Unidirectional and bidirectional transfer is possible.

⚠️ A coupled simulation involving a remeshing technology on the solid part of the model is not possible for this FLUX version.

8.1.2 Supported Platforms and Versions

The following versions of FLUX are supported by MpCCI:

<table>
<thead>
<tr>
<th>platform</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>mswin_x86</td>
<td>mswin_x86</td>
</tr>
<tr>
<td>x64_x64</td>
<td>x64_x64</td>
</tr>
<tr>
<td>10.2</td>
<td>X</td>
</tr>
</tbody>
</table>

8.1.3 References

FLUX Documentation is part of the FLUX distribution.

8.1.4 Adapter Description

The FLUX multi-physics coupling is enable via the USERs SUBROUTINEs for coupling API. MpCCI has been integrated as pyFlux commands. The user has to provide a PyFLUX script in order to start the co-simulation. (see ▶️ 8.2.5 Go Step ◀️ and ▶️ 8.5 Code Adapter Description ◀️)
8.2 Coupling Process

8.2.1 Model Preparation

There are some issues which you should consider while creating an FLUX model for a co-simulation:

Supported elements types. For volume coupling the region involved in the coupling must be based on first order elements. For surface coupling the region may be based on first and second order elements.

Multi-point support will be automatically created by the MpCCI-FLUX adapter for the region involved in the coupling.

Spatial parameters are used to send or receive the quantity value.

⚠️ There is a constraint by the name in order to used this storage capability. In order to take into account this parameter it is necessary to create a spatial parameter of tabulated type named MPCCI_index (see ▶ 8.2.2.3 Data storage: ◄).

Thermal coupling. There is a software constraint by the name of the spatial parameter that allows the storage of the temperature.

User pyFlux script required. For managing the co-simulation, an pyFlux script is required. This is described in detailed in ▶ 8.2.2 pyFlux script ◄.

Scenario A scenario for the multi physic session has to be created.
8.2.2 pyFlux script

In FLUX, the coupling process is controlled via an pyFlux script, which calls functions provided by MpCCI.

8.2.2.1 Sample pyFlux Script

```python
#! Flux3D 10.2

# Open the scenario 'SCENARIO_1' for a multi physics session
Scenario['SCENARIO_1'].openSessionMultiPhysics(projectName='EXAMPLE.FLU')
# Activate MpCCI pyFlux commands
mpcci = MpCCI()

# Initialize MpCCI
mpcci.initialize()

step=1
while step<=5:
    # Receive data via MpCCI
    mpcci.receive(1)
    # Solve the current step
    solveCurrentStep()
    # Send data via MpCCI
    mpcci.send(1)
    step = step + 1

# Exit the co-simulation
mpcci.exit()

# Close the multi physic session
Scenario['SCENARIO_1'].closeSessionMultiPhysics()
# Close the current project
closeProject()
# Exit FLUX
exit()
```

In this sample pyFlux script, the project file has been loaded by using the option from the MpCCI GUI (see \(\ref{8.2.5 Go Step<}\)). Otherwise you should use this command to open the project:

```python
loadProject('EXAMPLE.FLU')
```
The MpCCI-FLUX adapter has been imported by using the option from the MpCCI GUI (see \( \textbf{8.2.5 Go Step} \)). Otherwise the corresponding command to import the MpCCI module is:

```python
import MpCCI
```

First we open the multi physic session for the scenario \textsc{ScenarIo.1}. MpCCI is initialized by `mpcci.initialize()`. Next the loop for the coupled simulation starts with a receive command `mpcci.receive(1)`. FLUX will wait until the partner code has sent the data. Afterwards the solution is computed for the current step. Following it sends the data with `mpcci.send(1)`. The loop continues until the final step is reached.

The command `mpcci.exit()` is finishing the MpCCI process regularly and FLUX could be finished.

### 8.2.2.2 Available Commands

The command for MpCCI calls within FLUX is available from the pyFlux MpCCI module. The following functions are valid:

```python
mpcci = MpCCI()
mpcci.help()
mpcci.initialize()
mpcci.send(doWait)
mpcci.receive(doWait)
mpcci.exchange(doWait)
mpcci.exit()
```

The available commands have the following functionality:

Function of coupling management:

- **help()**
  
  Provides the list of available commands.

- **MpCCI()**

  Instantiates the MpCCI module.

- **initialize()**

  This function initializes the connection to MpCCI.

- **exit()**

  This function shuts down the MpCCI connection.
Function of data transfer:

- **exchange**(doWait)

  Executes a MpCCI data transfer and waits for data if the value of “doWait” is “1”. With a value of “0” FLUX will receive the available data from the other code if available, otherwise FLUX continues the computation.

- **send**(doWait)

  Sends data to MpCCI. The value of the “doWait” do not influence the data send because sending is always possible, values are buffered.

- **receive**(doWait)

  Receives data and waits for data if the value of “doWait” is “1”. With a value of “0” FLUX will receive the available data from the other code if available, otherwise FLUX continues the computation.

⚠️ The **Initial quantities transfer** value selected in MpCCI GUI will overwrite the first function of data transfer called from the pyFlux script if they do not match.

### 8.2.2.3 Data storage:

Varying parameters are considered as a *Global* quantities. They may be used to send or receive the varying parameter value.

For some quantities the data is stored in a spatial parameter (SPATIAL). To distinguish the quantities, a storage index is set for each quantity in the Coupling Step, as shown in Figure 1. If you do not change it, the index is increased automatically. In addition the corresponding MPCCI(index) spatial parameter must be created and used at the appropriate formula for the boundary.
Figure 1: SPATIAL storage index for quantities received or sent by FLUX.
8.2.3 Models Step

In the models step, the following options must be chosen:

**FLUX release**  Select the release of FLUX you want to use. `latest` (default) will select the latest version which is installed on your system.

**problem file**  Select the problem file of your FLUX project.  
A FLUX project file is represented by a directory having the model name. 

The MpCCI scanner uses the FLUX problem file to extract model information. This file is located inside the directory project and has the name "PROBLEM_FLU.PFL".

8.2.4 Coupling Step

The FLUX adapter only stores some quantities directly ("Direct"), other quantities are first written to user-defined memory ("SPATIAL") which refers to the spatial parameter. FLUX supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current1</td>
<td>Scalar</td>
<td>0.0 A</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CurrentDensity</td>
<td>Vector</td>
<td>0.0 A/m²</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrCond1</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectricField</td>
<td>Vector</td>
<td>0.0 V/m</td>
<td>Field</td>
<td>Line, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ElectrRes1</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Line, Face, Node, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

Figure 2: FLUX options in the Models Step
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location Option</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntFlag</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>IterationNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>JouleHeat</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>JouleHeatLin</td>
<td>Scalar</td>
<td>0.0 W/m³K</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>LorentzForce</td>
<td>Vector</td>
<td>0.0 N/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>MagneticField</td>
<td>Vector</td>
<td>0.0 A/m</td>
<td>Field</td>
<td>Line, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>MagneticFlux</td>
<td>Vector</td>
<td>0.0 T</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>RealFlag</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Residual</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Temperature</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Line, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>ThermCond1</td>
<td>Scalar</td>
<td>0.0 W/mK</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>TimeStepNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>udm00</td>
<td>Scalar</td>
<td>0.0</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Node</td>
<td>SPATIAL</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>Voltage1</td>
<td>Scalar</td>
<td>0.0 V</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

This table represents the quantities mapping used by MpCCI.
<table>
<thead>
<tr>
<th>MpCCI naming</th>
<th>FLUX naming</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentDensity</td>
<td>JS1</td>
</tr>
<tr>
<td>ElectricCond1</td>
<td>SIGMA</td>
</tr>
<tr>
<td>ElectricField</td>
<td>E1</td>
</tr>
<tr>
<td>ElectricRes1</td>
<td>1/SIGMA</td>
</tr>
<tr>
<td>JouleHeat</td>
<td>DVOL_LOSSES</td>
</tr>
<tr>
<td>JouleHeatLin</td>
<td>DRHODT<em>J</em>J or for Harmonic model 0.5<em>Real(DRHODT</em>J*CONJ(J))</td>
</tr>
<tr>
<td>LorentzForce</td>
<td>DFVL</td>
</tr>
<tr>
<td>MagneticField</td>
<td>H MAG1</td>
</tr>
<tr>
<td>MagneticFlux</td>
<td>BMAG1</td>
</tr>
<tr>
<td>Temperature</td>
<td>T1</td>
</tr>
<tr>
<td>ThermCond1</td>
<td>KA</td>
</tr>
</tbody>
</table>

### 8.2.5 Go Step

![FLUX options in the Go Step](image)

Figure 3: FLUX options in the Go Step
FLUX offers a number of options in the Go panel, see Figure 3.

**Initial quantities transfer**  Select one of receive or skip to determine the coupling algorithm. The construction of coupling algorithms is further described in V-3.3 Coupling Algorithms.

**Memory configuration**  If this option is selected, you may adjust the numerical, character memory for FLUX solver and the java memory for FLUX GUI.

**Run in batch mode**  If this option is checked, the simulation job will run in a batch mode.

**Select User pyFlux File**  If this option is selected, MpCCI will load your PyFLUX file and it will be executed at the start of FLUX.

**Auto read problem file**  If this option is selected, the FLUX project will be loaded when FLUX starts.

**Auto load MpCCI lib**  If this option is selected, the MpCCI-FLUX adapter library will be automatically imported.

**Select your user pyFlux library directory**  Select a directory containing your user library. This user library will export your user pyFlux commands into the FLUX environment.

### 8.2.5.1 Running the Computation

When the **Start** button is pressed, FLUX is started with the options given in the Go Step. If the **Stop** button is pressed, a stop-file "STOP" is created and FLUX will stop the next time it checks the presence of a stop file.

### 8.2.5.2 Troubleshooting

On the FLUX application side the file "Flux3D.log" contains the log of the co-simulation. The log file reflect the actions and state of the solver and the MpCCI adapter.

### 8.2.5.3 Post-Processing

After having solved a coupled simulation the results computed on the FLUX side may be visualized by using the FLUX 3D post-processing tool. You may compute and visualize the coupled quantities exchanged.

### 8.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for FLUX are:
Usage:
   mpcci FLUX [-]option

Synopsis:
   'mpcci FLUX' is used to get information about FLUX.

Options:
   -help                   This screen.
   -info                   List verbose information about all FLUX releases.
   -releases               List all FLUX releases which MpCCI can find.
   -scan <problem file>   Run the scanner and create a scanner output file.

The subcommands `-info` and `-releases` are described in \ref{1.1 Common MpCCI Subcommands for Simulation Codes}.

8.4 Code Environment

8.4.1 Prerequisites for a coupled simulation

To run a co-simulation you need the following Flux_FeEx_MultiPhys token on the FLUX license server.

8.5 Code Adapter Description

Within the MpCCI distribution the "adapters" directory contains the necessary software to connect the simulation programs to MpCCI depending on your license. The files are located within the subdirectory 
"<MpCCI_home>/FLUX/adapters".

This subdirectory is further divided by several release subdirectories, e.g. "10.2". The version directories are further divided by several architectures (e.g. "mswin_x86", "xp64_x64"). There you find the library files of the FLUX adapter (e.g. "fluxmpcci.dll"). The connection to MpCCI is established using these shared libraries. The binding to the pyFlux command is set by using the available user environment from FLUX. This process is done if FLUX is started by MpCCI.

To enable coupling capabilities the mpcci command is added to the standard pyFlux commands. By the command line options you decide which coupling function is called. The basic tasks of the command are
to initialize the coupling and to manage the data transfer to MpCCI commands.
9 MSC.Marc

9.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>SolidStructure, SolidThermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>MSC Software Corporation</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.mscsoftware.com">www.mscsoftware.com</a></td>
</tr>
<tr>
<td>Support</td>
<td><a href="http://www.mscsoftware.com/products/marc_support.cfm">www.mscsoftware.com/products/marc_support.cfm</a></td>
</tr>
</tbody>
</table>

9.1.1 Supported Coupling Schemes

MSC.Marc currently uses exchange after solution.

9.1.2 Supported Platforms and Versions

MSC.Marc is supported on the following platforms:

<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCI_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2005r3</td>
</tr>
<tr>
<td>hpux1123_i.a64</td>
<td>hpux1123_i.a64</td>
<td>X</td>
</tr>
<tr>
<td>irix65_mips4</td>
<td>irix65_mips4</td>
<td>X</td>
</tr>
<tr>
<td>linux_amd64</td>
<td>linux_amd64</td>
<td>X</td>
</tr>
<tr>
<td>linux_em64t</td>
<td>linux_em64t</td>
<td>X</td>
</tr>
<tr>
<td>linux_i.a64</td>
<td>linux_i.a64</td>
<td>X</td>
</tr>
<tr>
<td>linux_x86</td>
<td>linux_x86</td>
<td>X</td>
</tr>
<tr>
<td>mswin_x86</td>
<td>mswin_x86</td>
<td>X</td>
</tr>
<tr>
<td>osf_alpha</td>
<td>osf_alpha</td>
<td>X</td>
</tr>
<tr>
<td>sles10_amd64</td>
<td>sles10_amd64</td>
<td>X</td>
</tr>
<tr>
<td>solaris_sparc</td>
<td>solaris_sparc</td>
<td>X</td>
</tr>
</tbody>
</table>

⚠️ For 64 Bit systems, code coupling is only supported for the 32-bit integer version (i4-version) of MSC.Marc.

For more information on supported platforms and compilers see also the MSC.Marc Release Guide.

9.1.3 References

MSC.Marc, Volume A: Theory and User Information by MSC.Software Corporation. This part of the MSC.Marc documentation contains general information on performing a simulation with MSC.Marc.

MSC.Marc, Volume D: User Subroutines and Special Routines This part of the MSC.Marc documentation contains information on the user subroutines which are also used by the code adapter. See especially chapter 12.
MSC.Marc, Volume C: Program Input for information on the COUPLING REGION option, which is used by MpCCI to define the coupling region.

9.1.4 Adapter Description

The code adapter is provided in an object file. Similar to object files of user-defined functions these files are linked with MSC.Marc before starting the computation. Therefore the appropriate compiler is needed to run a coupled simulation (see ▶9.4.1 Prerequisites for a coupled simulation ◄).
9.2 Coupling Process

Please read also the corresponding ▶ IV-2 Setting up a Coupled Simulation ◀.

9.2.1 Model Preparation

MSC.Marc does not use a fixed unit system, so any consistent unit system may be used to define the MSC.Marc model.

The MSC.Marc model can be defined like a normal model. For the coupled analysis, the surfaces which form the coupling region must be defined as user sets:

- sets of finite element edges or geometric curves, for 2-D surface coupling;
- sets of finite element faces or geometric surfaces, for 3-D surface coupling;
- sets of finite elements or contact bodies, for volume-based coupling in any dimension.

The sets will then be listed in the Coupling Step of the MpCCI GUI.

⚠ Coupling is only supported for the new input file format, so the option NEW-STYLE TABLE must be selected in the RUN JOB menu of MSC.Marc Mentat (option tables in the input file). MpCCI has to insert boundary conditions into the input file and only supports the new format.

Instead of directly submitting the file from MSC.Marc Mentat, simply write an input file. The analysis is started from the MpCCI GUI.

It is recommended to first start a stand-alone MSC.Marc computation to check if the model is set up correctly.

9.2.2 Models Step

In the models step the following options must be chosen:

Select MSC.Marc version Select the MSC.Marc version you want to use. latest (default) selects the latest version which is installed on your system.

Multiple input file DDM job This option is needed for parallel runs of MSC.Marc in which the domain decomposition is done in the GUI and multiple input files have been written. For single file DDM, the button must not be selected.

Select MSC.Marc input file Select the MSC.Marc input file for the analysis.
Select unit system Select the unit system which was used to define the MSC.Marc model. MSC.Marc has no units built into it, a self-consistent set of units should be used. In the Go Step you can select from:

- British – the British unit system, i.e. ft, s, lbm/ft³, etc.
- SI – the SI unit system, i.e. m, kg, s, N, Pa, etc.
- cgs – the cgs system, i.e. cm, g, s, dyn, Ba, etc.
- variable – corresponds to a user-defined system. This requires to set the unit for each quantity separately in the Coupling Step.

The selection of units in the Models panel is only used to determine default values for the units of the quantities. The units are finally selected in the Coupling Step.

### 9.2.3 Coupling Step

The following quantities are supported:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FilmTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Line, Face, Element</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDisplacement</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>NPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>OverPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>RelWallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Line, Face, Node</td>
<td></td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Line, Face, Node</td>
<td></td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallHeatFlux</td>
<td>Scalar</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Line, Face, Element</td>
<td></td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallHTCoeff</td>
<td>Scalar</td>
<td>0.0 W/m² K</td>
<td>Field</td>
<td>Line, Face, Element</td>
<td></td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Line, Face, Node</td>
<td></td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

⚠️ If the time step size is received by MSC.Marc, the option AUTO STEP must be selected in MSC.Marc Mentat to enable time step adaptation in MSC.Marc.
9.2.4 Go Step

In the Go Step the following options can be chosen:

**Initial quantities transfer**

**Enter a job name**

**Optional restart file**  If you want to restart a previous computation, select the restart file here.

**Optional post file**  If you want to restart a previous computation from a post file, select it here.

**Optional view factor file**  Select a view factor file.

**User Subroutine**  Select a file which the user subroutines.

**Additional command line options**  If you need any further command line options for starting MSC.Marc, enter them here. See also Volume C, Appendix B of the MSC.Marc manual. Do not set command line options which are already set by the MpCCI GUI (e.g. `-jid` for the job name).

**Run parallel**  Select this option for a parallel run. See 9.2.5.1 Parallel Execution below.

9.2.5 Running the Computation

Instead of directly submitting the file from MSC.Marc Mentat, simply write an input file. The analysis is started from the MpCCI GUI.

9.2.5.1 Parallel Execution

There is no limitation regarding parallel execution of MSC.Marc in a coupled simulation. Both, automatic domain decomposition and manual decomposition are supported. The coupling region information is automatically passed to all subprocesses.

⚠️ If you use multiple input files, the option Multiple input file DDM job must be selected in the Models Step.

If Run parallel is selected in the Go Step of the MpCCI GUI, the following options can be selected for a parallel run of MSC.Marc:

**No. of parallel domains**  Enter the number of parallel domains here. The number must correspond to the number of domain input files if multiple input files are used.

**Shared file system (no file copy)**  If you work on a shared file system, ...

**Optional 'host host ...' to be used**  Enter a list of host names for a simulation distributed on different computers.
Optional hostlist file

Use default hostfile

9.2.6 Post-Processing

Pressures or forces received by MSC.Marc in a coupled simulation can be visualized in MSC.Marc Mentat: Select External Force in the post-processing menu.
9.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for MSC.Marc are:

Usage:
```
mpcci MSC.Marc [-]option
```

Synopsis:

'mpcci MSC.Marc’ is used to get information about MSC.Marc.

Options:

- **-align <ARGS>**  Do a coordinate transformation on all nodes of a .dat file based on a plane definition file and align the nodal coordinates for the coupling partner.
- **-help**  This screen.
- **-info**  List verbose information about all MSC.Marc releases.
- **-releases**  List all MSC.Marc releases which MpCCI can find.

The subcommands `-align`, `-info` and `-releases` are described in 1.1 Common MpCCI Subcommands for Simulation Codes.
9.4 Code Environment

9.4.1 Prerequisites for a coupled simulation

As the MpCCI code adapter is based on user-defined methods which must be linked statically to the MSC.Marc executable, the proper compiler is required for coupled simulations. The path to the compiler must be given in an appropriate shell variable, e.g. INTEL for the Intel compiler. MpCCI will complain if the variable is not set correctly.

It is recommended to first try to run an example which uses user subroutines to check whether the compiler is installed correctly.

For more information on compiler requirements see the list of supported platforms in the MSC.Marc release guide, chapter 6.
10 PERMAS

10.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>SolidStructure, SolidAcoustics, SolidThermal, Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>INTES, Ingenieurgesellschaft für technische Software mbH</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.intes.de">www.intes.de</a></td>
</tr>
<tr>
<td>Support</td>
<td><a href="http://www.intes.de/support/">www.intes.de/support/</a></td>
</tr>
<tr>
<td>Tutorials</td>
<td>⊳ VII-4 Exhaust Manifold ⊲</td>
</tr>
</tbody>
</table>

10.1.1 Supported Coupling Schemes

PERMAS supports exchange before and after solution. Unidirectional and bidirectional transfer is possible.

10.1.2 Supported Platforms and Versions

<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCI_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>aix51_power</td>
<td>aix51_power</td>
<td>11 12</td>
</tr>
<tr>
<td>hpux1123_ia64</td>
<td>hpux1123_ia64</td>
<td>X</td>
</tr>
<tr>
<td>hpux11_parisc</td>
<td>hpux11_parisc</td>
<td>X</td>
</tr>
<tr>
<td>linux_em64t</td>
<td>linux_em64t</td>
<td>X X</td>
</tr>
<tr>
<td>linux_x86</td>
<td>linux_x86</td>
<td>X</td>
</tr>
<tr>
<td>xp64_x64</td>
<td>xp64_x64</td>
<td>X</td>
</tr>
</tbody>
</table>

10.1.3 References

The PERMAS documentation is part of the PERMAS distribution, which can be invoked after the PERMAS installation by the command `permasdoc`.

**Installation** Installation of PERMAS.

**Release** Release notes and bug fixes.


**Interfaces** Manuals for interfaces with MEDINA, IDEAS, CATIA, PATRAN.

**PERMAS with MpCCI** Manual describing the necessary steps for an interconnection with MpCCI.
10.1.4 Adapter Description

The code adapter for PERMAS is developed by INTES in cooperation with Fraunhofer SCAI. The adapter is distributed as part of the PERMAS software.
10.2 Coupling Process

Please read also ▷ IV-2 Setting up a Coupled Simulation ◁.

10.2.1 Model Preparation

The PERMAS model can be prepared with several pre-processing tools (e.g. MEDINA, CATIA, IDEAS).

Please consider the following approach for model preparation:

- Define the coupling surfaces in the pre-processor or in PERMAS. The naming rule for coupling surfaces is “CCI_<sid>”, where CCI indicates a MpCCI surface and <sid> is the surface identifier of the CCI surface.
- Make your PERMAS definitions in the pre-processor, in the input file "<project>.uci" and if needed in "<project>.dat".
- Create dummy loads of zero magnitude in PERMAS which are replaced by received data during coupled simulation:

<table>
<thead>
<tr>
<th>Received Quantity</th>
<th>Dummy Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>FilmTemp,WallHTCoeff</td>
<td>$DISLOAD TEMPFILM</td>
</tr>
<tr>
<td>WallHeatFlux</td>
<td>$DISLOAD SURFHS / SURFHSS</td>
</tr>
<tr>
<td>OverPressure</td>
<td>$DISLOAD SURF3D</td>
</tr>
<tr>
<td>RelWallForce, WallForce</td>
<td>$CONLOAD</td>
</tr>
</tbody>
</table>

Examples:

$LOADING name = NLV
$DISLOAD TEMPFILM LPAT = 1 NODES=ALL DOFTYPE = TEMP
PRP_101 0.000000
PRP_102 0.000000
$END LOADING

$LOADING NAME = LOADVAR_1
$CONLOAD LPAT = 1 RSYS = 0 DOFTYPE = DISP
704343 0.0
704344 0.0
$END LOADING

$LOADING NAME = LOADVAR_2
$DISLOAD SURF3D LPAT = 2 IDS = ELGEO DOFTYPE = DISP
13943 3: 1.000000E-05 1.0000000E-05 1.0000000E-05
$END LOADING
10.2.2 Models Step

In the Models Step, the following options must be chosen (Figure 1):

**PERMAS release** Select the release of PERMAS you want to use, latest (default) will select the latest version which is installed on your system.

**model file** Select the PERMAS project file "*.uci" or "*.dat".

**unit system** – select the unit system which was used in PERMAS. PERMAS has no built in units. A self-consistent set of units should be used (see chapter “1.5.7 System of Units” of the PERMAS User’s Reference Manual for more information). In the Models Step you can select from:

- British – the British unit system, i.e. ft, s, lbm/ft$^3$, etc.
- SI – the SI unit system, i.e. m, kg, s, N, Pa, etc.
- cgs – the cgs system, i.e. cm, g, s, dyn, Ba, etc.
- variable – corresponds to a user-defined system. This requires to set the unit for each quantity separately in the Coupling Step.

10.2.3 Coupling Step

In PERMAS the exchanged quantities are directly read or written.

PERMAS supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Film2Temp</td>
<td>Biscalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
### 10.2.4 Go Step

![PERMAS options in the Go Step](image)

PERMAS offers a number of options in the Go Step, see Figure 2.

- **Run parallel**  Run PERMAS in parallel mode, see §10.2.5.1 Parallel Execution.

- **Core memory size in MB**  Explicit definition of the memory size for PERMAS (see chapter “1.4 The PERMAS execution” of the PERMAS User’s Reference Manual for more information).

- **Configure coupling algorithm**  Configure parameters for coupling algorithm, see §10.2.4.1 Configure Coupling Algorithm.

#### 10.2.4.1 Configure Coupling Algorithm

- **Select the coupling algorithm**  The choice of the coupling algorithm, the initial transfer of the partner code and in case of a Gauss-Seidel procedure the numeration of the codes defined by its positions in the MpCCI GUI will define the coupling procedure. Possible coupling schemes are depicted in Figure 4 and Figure 5. If a possible algorithm is advisable, might depend on the particular problem description.
PERMAS will send and receive data before each solution step (Figure 4).

Jacobi  PERMAS will send and receive data before each solution step (Figure 4).

Gauss-Seidel  Before the solution run starts, PERMAS sends and receives data from/to all codes with a code id which is lower than its own code id and receives data from codes with a code id which is higher than its own code id. After the solution run PERMAS sends data to all codes with a higher code id (Figure 5).

No. of exchange iterations  Total number of exchange iterations of PERMAS.

Use additional printout steps  Additional printout of the result data (e.g. "*.bof") after each coupling step. Data in "*.bof" is accumulated.

Use several meshes with one part  Activate this button, when the coupling region has several faces.

Residual limit for convergence test  Define the residual limit for the convergence test of the complete coupled simulation run.

Relaxation factor  Specify a relaxation factor which will be applied to the defined iterations.
10.2.5 Running the Computation

When the **Start** button is pressed, PERMAS is started with the options given in the Go Step. If the **Stop** button is pressed, a stop-file is created and PERMAS will stop the next time it checks the presence of a stop file.

10.2.5.1 Parallel Execution

If the Run parallel check button is activated the following options are available (Figure 6):

**No. of parallel processes**  Select the number of parallel PERMAS processes to be launched.
10.2.5.2 Batch Execution

PERMAS always runs as a batch process. Therefore no special settings are necessary for batch execution.
10.2.6 Post-Processing

Post-processing of PERMAS results can be performed e.g. with MEDINA. The results "<job name>.bof" or any other result file supported by PERMAS are stored in the same directory as the input file.
10.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for PERMAS are:

Usage:

```
mpcci PERMAS [-]option
```

Synopsis:

‘mpcci PERMAS’ is used to get information about PERMAS.

Options:

- `-help` This screen.
- `-info` List verbose information about all PERMAS releases.
- `-releases` List all PERMAS releases which MpCCI can find.
- `-scan <UCI or data file>` Run the scanner and create a scanner output file.

The subcommands `-info` and `-releases` are described in 1.1 Common MpCCI Subcommands for Simulation Codes.
10.4 Trouble shooting, open issues and known bugs

| Feature: | Fluid-Structure-Interaction |
| Version: | 11,12 |
| Problem: | Coupling OverPressure results in unphysical behavior |
| Workaround: | Use WallForce or RelWallForce |
| References: | |

110 VI MpCCI 3.1.1-1
11 RadTherm

11.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>Radiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>ThermoAnalytics</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.thermoanalytics.com">www.thermoanalytics.com</a></td>
</tr>
<tr>
<td>Support</td>
<td><a href="mailto:support@thermoanalytics.com">support@thermoanalytics.com</a></td>
</tr>
</tbody>
</table>

11.1.1 Supported Coupling Schemes

RadTherm may only receive data before a solution and send data after a solution.

RadTherm supports the following Coupling Type V-4.4 Coupling Step ◀:

- Steady state radiative heat transfer
- Transient radiative heat transfer

11.1.2 Supported Platforms and Versions

The following versions of RadTherm are supported by MpCCI:

<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCI_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>9.0.1 9.1.0 9.1.2 9.2.0</td>
</tr>
<tr>
<td>hp hpux11_parisc</td>
<td></td>
<td>X X</td>
</tr>
<tr>
<td>i686 linux_x86</td>
<td></td>
<td>X X X</td>
</tr>
<tr>
<td>ia64 linux_ia64</td>
<td></td>
<td>X X X X</td>
</tr>
<tr>
<td>lnx9 linux_x86</td>
<td></td>
<td>X X X X</td>
</tr>
<tr>
<td>mswin_x86 mswin_x86</td>
<td></td>
<td>X X X X</td>
</tr>
<tr>
<td>sun solaris_sparc</td>
<td></td>
<td>X X X X</td>
</tr>
<tr>
<td>x86_64 (sles10</td>
<td>linux)(amd64</td>
<td>em64t</td>
</tr>
</tbody>
</table>

11.1.3 References

RadTherm Documentation is part of the RadTherm distribution.

11.1.4 Adapter Description

The code adapter for RadTherm is based on a shared library "libradthermmpcci.so|sl|dll" which is loaded by RadTherm and it includes the necessary interface functions.
11.2 Coupling Process

11.2.1 Model Preparation

Models can be prepared as usually. The coupling components must be defined as separate parts.

11.2.2 Models Step

![Figure 1: RadTherm options in the Models Step](image.png)

In the models step, the following options must be chosen:

**RadTherm program family**  Select the RadTherm program family from muses, musespro, radtherm, radthermir, radthermrt.

**RadTherm release**  Select the release of RadTherm you want to use. latest (default) will select the latest version which is installed on your system.

**.tdf file**  Select the "*.tdf" file of your RadTherm model.

11.2.3 Coupling Step

The RadTherm adapter only stores quantities directly ("Direct").

RadTherm supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FilmTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>IterationNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>TimeStepNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>WallHeatFlux</td>
<td>Scalar</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>
11.2.4 Go Step

![RadTherm Go Step](image)

Figure 2: RadTherm Go Step.

In the Go Step the following options can be chosen:

**Run in batch mode**  If you want to run RadTherm in batch mode, select the option.

**No. of parallel processes**  Specify the number of processes for a parallel run. See "11.2.5.2 Parallel Execution" below.

11.2.5 Running the Computation

To run the RadTherm computation you have to go in the section Analyse and click on the Run button.

11.2.5.1 Hooks Functions

Before starting the RadTherm computation the model has to be correctly setup to use the MpCCI Adapter by using the RadTherm GUI.

You have to select the Analyse → Params section in the RadTherm GUI in order to access the hooks functions setup Figure 3. By clicking on the button HookFunctions... the setup windows will appear. You have to setup these functions for the following available RadTherm hooks:

**Solution Start**  Select the routine radthermmppcci::TaiMpCCI_solution_start(). This is the function to initialize RadTherm with MpCCI.
**Time Step Start**  Select the routine `radthermpcci::TaiMpCCI_timestep_start()`. This is the function to receive data before a new solution computation.

⚠️ This function has only to be hooked for transient simulation.

**Iteration Start**  Select the routine `radthermpcci::TaiMpCCI_iteration_start()`. This is the function to receive data before a new solution computation.

⚠️ This function has only to be hooked for steady state simulation.

**Iteration End**  Select the routine `radthermpcci::TaiMpCCI_iteration_end()`. This is the function to send data after a solution computation.

⚠️ This function has only to be hooked for steady state simulation.

**Time Step End**  Select the routine `radthermpcci::TaiMpCCI_timestep_end()`. This is the function to receive data after a solution computation.

⚠️ This function has only to be hooked for transient simulation.

**Solution End**  Select the routine `radthermpcci::TaiMpCCI_solution_end()`. This is the function to terminate the coupled simulation.

![Figure 3: RadTherm Hooks Functions Setup.](image-url)
This setting has to be saved in the RadTherm model.

⚠️ RadTherm saves the library information in the model. You may encounter some problems if you change your RadTherm platform computation. You will have to re-setup the hooks functions and delete correctly the old reference to the library.

11.2.5.2 Parallel Execution

There are two possibilities to run RadTherm in parallel:

**In batch mode** By activating the batch mode you are able to specify the number of parallel processes to use.

⚠️ To run a RadTherm job in batch the the function hooks has to be already set in the model.

**In RadTherm GUI** Before starting RadTherm computation you have to specify the number of processors to use in the Edit → Preferences → Application Figure 4.

![Figure 4: Number of processors setting in RadTherm GUI.](image)
11.2.6 Post-Processing

After having solved a coupled simulation the results computed on the RadTherm side may be visualized by using the RadTherm post-processing tool.

11.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for RadTherm are:

Usage:
    mpcci RadTherm [-]option

Synopsis:
    'mpcci RadTherm' is used to get information about RadTherm.

Options:
    -help                      This screen.
    -info                      List verbose information about all RadTherm releases.
    -releases                  List all RadTherm releases which MpCCI can find.
    -scan <tdf file>           Run the scanner and create a scanner output file.
12 STAR-CD

12.1 Quick Information

<table>
<thead>
<tr>
<th>Physical Domains</th>
<th>CFD, Fluid, FluidThermal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company Name</td>
<td>CD-adapco</td>
</tr>
<tr>
<td>Company Homepage</td>
<td><a href="http://www.cd-adapco.com">www.cd-adapco.com</a></td>
</tr>
<tr>
<td>Tutorials</td>
<td>⊳ VII-4 Exhaust Manifold ⊲</td>
</tr>
<tr>
<td></td>
<td>⊳ VII-3 Elastic Flap in a Duct ⊲</td>
</tr>
<tr>
<td></td>
<td>⊳ VII-5 Busbar System ⊲</td>
</tr>
</tbody>
</table>

⚠ The STAR-CD adapters for STAR-CD 3.26 and STAR-CD 4.0x are based on different philosophies. For STAR-CD 3.26 the standard user coding is required. Templates for some of the user subroutines are part of the adapter for STAR-CD 3.26. With STAR-CD 4.0x the STAR-CD plugin concept is used. Open issues and known bugs are documented in ⊳ 12.6 Trouble shooting, open issues and known bugs ⊲.

12.1.1 Supported Coupling Schemes

STAR-CD supports exchange before solution. The initial transfer settings determine the coupling algorithm ⊳ V-3.3 Coupling Algorithms ⊲. Unidirectional and bidirectional transfer is possible.

STAR-CD 3.26

A modification of the transfer procedure is possible by modifying "posdat.f" and for node movements in "newxyz.f" (see also ⊳ 12.5.3 Automatic model preparation for STAR-CD 3.26 and STAR-CD 4.0x ⊲).

STAR-CD 4.0x

In the plugin version of the adapter at the end of a simulation run a final transfer will be accomplished which is inverted to the initial exchange setting.

<table>
<thead>
<tr>
<th>Initial Transfer</th>
<th>Final Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>receive</td>
<td>send</td>
</tr>
<tr>
<td>send</td>
<td>receive</td>
</tr>
<tr>
<td>skip</td>
<td>-</td>
</tr>
<tr>
<td>exchange</td>
<td>-</td>
</tr>
</tbody>
</table>

⚠ In a simulation where the solution algorithm was set to transient SIMPLE or PISO, after the last iteration step another transfer of data is carried out. Unfortunately at this last exchange received node movements are not accounted for in STAR-CD. This might be important for a restart as the partner code might start from a different state of nodal positions.
12.1.2 Supported Platforms and Versions

The list of supported platforms and versions is given in Table 1.

12.1.3 References

STAR-CD Documentation is part of the STAR-CD distribution (e.g. STAR-CD User Guide).

12.1.4 Adapter Description

The code adapter is a DSO (Dynamic Shared Object) library.

STAR-CD 3.26

For STAR-CD 3.26 the library "libstarcdmpcci-<precision>.a" is linked as part of the user subroutine library "libstarusr.so" in the ufile directory. Data transfer is accomplished over the "posdat.f" user subroutine. Several other user subroutines are used to handle data management in the adapter (see ▶ 12.5 Code Adapter Reference ▷).

STAR-CD 4.0x

For STAR-CD 4.0x the MpCCI plugin library "libstarcdmpcci.so" of the desired precision is retrieved from MpCCI installation and dynamically loaded by STAR-CD at runtime and all data transfer and management is carried out over plug-in routines.

12.2 Coupling Process

Please read also ▶ IV-2 Setting up a Coupled Simulation ▷.

12.2.1 Model Preparation

The STAR-CD model can be prepared with pro-STAR. Please consider the following approach for model preparation:

- The STAR-CD solver internally operates only in SI units. The geometrical dimensions should best be defined in meters. However, it is possible to scale the mesh dimensions by a scaling factor during the preparation of the geometry files (".geom").

- The STAR-CD model must contain a definition of the coupling domains and boundary regions (e.g. couple-wall). Boundary faces of coupled boundaries may not lie on trimmed cells. Those boundary
faces must be removed from coupling region. Polyhedral grids must be triangulated on the boundary surface using pro-STAR command `TRISURF` (Figure 1).

- When receiving wall temperature the particular boundary’s Wall Heat option must be set to Fixed, receiving heat flux requires option Flux.

- To enable a coupled simulation with an unprepared model several modifications of the model are required. For popular coupling types (▷ V-3.1.2 Coupling Types ◁) the modifications can be carried out by MpCCI automatically. For special configurations one might be obliged to accomplish manual rework, which is described in ◷ 12.5 Code Adapter Reference ◁.

- Restarting a STAR-CD case will work as usual by definition in pro-STAR (see command `RDATA`). In the MpCCI Go Step the restart options of the problem file may be overwitten by selecting option Restart prepared case (▷ 12.2.4 Go Step ◁), which will invoke STAR-CD with option `-restart`. This option will cause a Standard Restart.

![Figure 1: Triangulation of the boundaries of a polyhedral mesh with command TRISURF](image)

⚠️ It is recommended to create a complete STAR-CD model first and test it separately without cosimulation. The quantities, which will be later on transferred by the partner code, can be simulated by appropriate loads or boundary conditions if desired.
<table>
<thead>
<tr>
<th>platform</th>
<th>MPCCl_ARCH</th>
<th>supported versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>aix64_5.1-pwr3-xlf_7.1-dso</td>
<td>aix51_power</td>
<td>X</td>
</tr>
<tr>
<td>aix64_5.2-pwr3-xlf90_9.1-dso</td>
<td>aix5(1</td>
<td>2).power</td>
</tr>
<tr>
<td>hpux64_11.00-pa8000-f90-dso</td>
<td>hpux11_parisc</td>
<td>X</td>
</tr>
<tr>
<td>hpux64_11.22-itanium2-f90-dso</td>
<td>hpux1123_ia64</td>
<td>X</td>
</tr>
<tr>
<td>irix64_6.5-mips4-f77_7.3-dso</td>
<td>irix65_mips4</td>
<td>X</td>
</tr>
<tr>
<td>linux64_2.4-absoft_9.0a-gcc_4.1.1-glibc_2.2.5-dso</td>
<td>(sles10</td>
<td>linux).(amd64</td>
</tr>
<tr>
<td>linux64_2.4-absoft_9.0a-glibc_2.2.5-dso</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>linux64_2.4-itanium-efc_7.1-glibc_2.2.4-dso</td>
<td>linux_ia64</td>
<td>X</td>
</tr>
<tr>
<td>linux64_2.4-itanium-ifort_9.0-gcc_4.1.1-glibc_2.2.4-dso</td>
<td>linux_ia64</td>
<td>X</td>
</tr>
<tr>
<td>linux64_2.4-pgf90_6.2-glibc_2.2.5-dso</td>
<td>(sles10</td>
<td>linux).(amd64</td>
</tr>
<tr>
<td>linux64_2.4-pgf90_6.2-glibc_2.2.5-dso</td>
<td>(sles10</td>
<td>linux).(amd64</td>
</tr>
<tr>
<td>linux_2.4-absoft_8.2ap1-glibc_2.2.2-dso</td>
<td>linux_x86</td>
<td>X</td>
</tr>
<tr>
<td>linux_2.4-absoft_9.0a-gcc_4.1.1-glibc_2.2.2-dso</td>
<td>linux_x86</td>
<td>X</td>
</tr>
<tr>
<td>osf_5.1-f77.5.4.1-dso</td>
<td>osf_alpha</td>
<td>X</td>
</tr>
<tr>
<td>sunos64_5.10-ultra3-f90_8.2-dso</td>
<td>solaris_sparc</td>
<td>X</td>
</tr>
<tr>
<td>sunos64_5.8-ultra3-f90_7.0-dso</td>
<td>solaris_sparc</td>
<td>X</td>
</tr>
</tbody>
</table>
12.2.2 Models Step

In the Models Step, the following options must be selected: (Figure 2):

**STAR-CD release** Select the release of STAR-CD you would like to use, latest (default) will select the latest installed STAR-CD version for which an MpCCI adapter is also installed. Please make sure that the model file was not created with a STAR-CD version newer than the STAR-CD version used with MpCCI.

**Name of the Star-CD modelfile** Select the model file (".mdl") of your STAR-CD model.

12.2.3 Coupling Step

In STAR-CD, the exchange of all quantities is handled by user subroutines or plug-ins, which depends on STAR-CD Version (STAR-CD 3.26, STAR-CD 4.0x). Transfer data is read and written directly or over user scalars (see STAR-CD documentation “Additional Scalars”). The scalar array is solely used for storing the received or to be sent data. STAR-CD offers scalar arrays and the user needs to decide on which scalar array the data will be stored. The array index is defined in the Quantities section of the corresponding quantity to be exchanged as depicted in Figure 3. The storage index is automatically increased in the MpCCI GUI whenever new quantities are added. However, the user is responsible to choose a free storage index if some scalar arrays are already reserved for settings going beyond MpCCI (e.g. multi-component models).

**STAR-CD** supports the following quantities for coupling:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

Figure 2: STAR-CD options in the Models Step
<table>
<thead>
<tr>
<th>Quantity</th>
<th>Dim.</th>
<th>Default Value</th>
<th>Integration Type</th>
<th>Coupling Dimension</th>
<th>Location</th>
<th>Send Option</th>
<th>Receive Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcstPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>BodyForce</td>
<td>Vector</td>
<td>0.0 N/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>BoundaryMassFlux</td>
<td>Scalar</td>
<td>0.0 kg/m² s</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>BoundaryTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>BoundaryTotalPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>CGAngle</td>
<td>Vector</td>
<td>0.0 rad</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CGOmega</td>
<td>Vector</td>
<td>0.0 rad/s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CGPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CGVelocity</td>
<td>Vector</td>
<td>0.0 m/s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ChargeDensity</td>
<td>Scalar</td>
<td>0.0 C/m³</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>Current1</td>
<td>Scalar</td>
<td>0.0 A</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CurrentDensity</td>
<td>Vector</td>
<td>0.0 A/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>DeltaTime</td>
<td>Scalar</td>
<td>1.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Density</td>
<td>Scalar</td>
<td>1.0 kg/m³</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>DynPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrCond1</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrCond3</td>
<td>Vector</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrCondX</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrCondY</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrCondZ</td>
<td>Scalar</td>
<td>0.0 S/m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectricFlux</td>
<td>Vector</td>
<td>0.0 C/m²</td>
<td>Flux dens.</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrRes1</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrRes3</td>
<td>Vector</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrResX</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrResY</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ElectrResZ</td>
<td>Scalar</td>
<td>0.0 ohm m</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>Enthalpy</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>FilmTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>HeatFlux</td>
<td>Vector</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>HeatSource</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>IntFlag</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>IterationNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>JouleHeat</td>
<td>Scalar</td>
<td>0.0 W/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>LorentzForce</td>
<td>Vector</td>
<td>0.0 N/m³</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>Quantity</td>
<td>Dim.</td>
<td>Default Value</td>
<td>Integration Type</td>
<td>Coupling Dimension</td>
<td>Location</td>
<td>Send Option</td>
<td>Receive Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------</td>
<td>---------------</td>
<td>------------------</td>
<td>--------------------</td>
<td>----------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>MagneticFlux</td>
<td>Vector</td>
<td>0.0 T</td>
<td>Flux dens.</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
<tr>
<td>MassFlowRate</td>
<td>Scalar</td>
<td>0.0 kg/s</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MassFluxRate</td>
<td>Scalar</td>
<td>0.0 kg/m² s</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>NDisplacement</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Buffered</td>
<td></td>
</tr>
<tr>
<td>NPosition</td>
<td>Vector</td>
<td>0.0 m</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Buffered</td>
<td></td>
</tr>
<tr>
<td>OverPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>Scalar</td>
<td>0.0 s</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>PorePressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>RealFlag</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>RefPressure</td>
<td>Scalar</td>
<td>1.12e5 N/m²</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>RelWallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>Residual</td>
<td>Scalar</td>
<td>0.0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>SpecificHeat</td>
<td>Scalar</td>
<td>1.0 J/kg K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>Temperature</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ThermCond1</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ThermCond3</td>
<td>Vector</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ThermCondX</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ThermCondY</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>ThermCondZ</td>
<td>Scalar</td>
<td>0.0 W/m K</td>
<td>Field</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>TimeStepNo</td>
<td>Scalar</td>
<td>0</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>TotalPressure</td>
<td>Scalar</td>
<td>0.0 N/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td>Vector</td>
<td>0.0 m/s</td>
<td>Field</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>Voltage1</td>
<td>Scalar</td>
<td>0.0 V</td>
<td>max/...</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Wall2Temp</td>
<td>Biscalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>WallForce</td>
<td>Vector</td>
<td>0.0 N</td>
<td>Flux</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>WallHeatFlux</td>
<td>Scalar</td>
<td>0.0 W/m²</td>
<td>Flux dens.</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>WallHTCoeff</td>
<td>Scalar</td>
<td>0.0 W/m² K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
<tr>
<td>WallTemp</td>
<td>Scalar</td>
<td>300.0 K</td>
<td>Field</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

### 12.2.4 Go Step

STAR-CD offers a number of options in the Go Step, see Figure 4.
Figure 3: Coupling Step: Specification of the storage index for additional scalar arrays in STAR-CD

**Initial quantities transfer** In STAR-CD the first exchange is invoked prior to the first iteration step (e.g. a time step in a transient simulation). The first exchange configuration of STAR-CD and the partner code determines the coupling algorithm. Select one of receive, send, exchange or skip to configure this first exchange. The meaning of the options are further described in \( \triangleright \) V-3.3 Coupling Algorithms \( \triangleright \).

**Please enter the startup procedure** Define what startup procedure is desired:

- **Prepare case and start** MpCCI modifies your model file ("<casename>.mdl") and generates all STAR-CD input files (e.g. "<casename>.prob"). A geometry file "<casename>.geom" is written if nonexistent. For STAR-CD 3.26 several necessary user subroutines will be placed in the ufile directory if not existent and these user subroutines will be hooked into the STAR-CD model. For STAR-CD 4.0x necessary hooks are activated and if Use MpCCI plug-ins (Star V4 only) is selected the plugin is activated in the STAR-CD model file.

- **Start prepared case** MpCCI assumes that all input files (e.g. "casename.prob") for the simulation run with the necessary modifications (hooked user subroutines etc.) are already existent. Modifications could have been accomplished by MpCCI through Prepare case and start or manually in pro-STAR.

- **Restart prepared case** This will lead to the same assumption as for Start prepared case, but will also start the STAR-CD simulation with the option -restart, which will overwrite restart settings
Figure 4: STAR-CD options in the Go Step

in problem file to launch a Standard Restart. An Initial Field Restart must be configured over pro-STAR (see command RDATA).

A detailed description of the preparation procedure for advanced users is given in 12.5 Code Adapter Reference.

**Backup model file**  In the start-up procedure, depending on the settings above, the model file ("<casename>.md1") might be modified by MpCCI. Pushing this button will create a backup of your model file.

**Use the MpCCI plugin library (Star V4 only)**  Enabling this button will cause MpCCI to use the plug-in library in STAR-CD. It can only be used with STAR-CD 4.04 or a higher version. For an activation of the plug-in in the STAR-CD model the option Prepare case and start must be selected. See 12.5.3 Automatic model preparation for STAR-CD 3.26 and STAR-CD 4.0x for a manual activation of the plugin in pro-STAR.

**Use a data filter subroutine (Star V4 only)**  A user subroutine "starmpcci.filterdata(level)" must be placed in the ufile directory. The subroutine has an integer value argument level, which is “0” before a transfer and “1” after a transfer. In this subroutine the quantities can be manipulated at level 0 before they are sent and restored at level 1 after they were sent to the partner code. It can only be used with STAR-CD 4.04 or higher.
Rebuild shared library  If this option is selected, STAR-CD will rebuild the DSO library (DSO - dynamic shared objects) with the user subroutines in "<starcdworkdir>/ufile/". An alternative procedure would be to rebuild those libraries with mpcci starcd -dsomake before the simulation is started (▷12.3 Code-Specific MpCCI Commands◁ and ▷12.5 Code Adapter Reference◁).

Overwrite ufile sources  MpCCI will copy the necessary ufile sources ("posdat.f", "newxyz.f", "dtstep.f", etc.) needed for STAR-CD 3.26 from the MpCCI directory to ufile directory. Existing sources are automatically copied into backup files. Additional activation of Rebuild shared library is necessary. This procedure corresponds to calling mpcci starcd -ufile prior to the simulation run (▷12.3 Code-Specific MpCCI Commands◁ and ▷12.5 Code Adapter Reference◁).

Double precision mode  Use this option to select the double precision mode for STAR-CD computations.

Additional command line options  It is possible to invoke STAR-CD with additional command line options (Star Run Options). Please refer to the STAR-CD User Guide for possible options.

Steady state: No. of iterations without coupling  For a steady state simulation (e.g. thermal simulation) it might be necessary to let STAR-CD do subcycling in order to reach a stable solution. After each coupling step STAR-CD will carry out the specified number of iterations before the next coupling step is accomplished.

Run parallel  Run STAR-CD in parallel mode, see ▷12.2.5.1 Parallel Execution◁

Use grid morpher  Activate the MpCCI grid morpher to adapt the mesh to moved or deformed boundaries which are caused e.g. by structural deformations. Depending on the degree of deformation, deactivation of the grid morpher will probably result in corrupted grids. The morpher is described in detail in ▷12.4.1 MpCCI Grid Morpher◁. Alternatively the pro-STAR grid morpher might be applied ▷12.4.3 pro-STAR Grid Morpher◁.

12.2.5 Running the Computation

When the [Start] button is pressed, STAR-CD is started with the options given in the Go Step. If the [Stop] button is pressed, a stop-file is created and STAR-CD will stop the next time it checks the presence of a stop file.

12.2.5.1 Parallel Execution

If the Run parallel check button is activated the following options are raised (Figure 5):

No. of parallel processes  Select the number of parallel STAR-CD processes to be launched.

Parallel MPI communicator  Select the MPI implementation. Further details may be taken from the STAR-CD User Guide under “star run options”.

126   VI   MpCCI 3.1.1-1
12.2 Coupling Process

**Shared file system**  Check here if the file system is shared for all hosts and no data copy for local scratch disks is necessary (see STAR-CD User Guide “Star Run Options” -distribute).

**Optional 'host host...' to be used**  Enter host names for parallel execution of STAR-CD.

**Optional hostlist file**  Specify a hostfile, from which host names are extracted.

**Use default hostfile**  A default hostfile can be configured by setting MPCCI_HOSTLIST_FILE "V-3.4.2 Hostlist files".

In case that a hostlist and a hostfile are defined, MpCCI adds up the hosts and passes as much hosts as defined processes to STAR-CD. The order of the host definitions in the MpCCI GUI defines the priority of the employed hosts (1-hostlist, 2-hostfile 3-default hostfile).

### 12.2.5.2 Batch Execution

STAR-CD is always run as a batch process, therefore no special settings are necessary for batch execution.

### 12.2.6 Post-Processing

Post-processing for the STAR-CD part of the results can be performed as in ordinary computations, e.g. with pro-STAR. The results "<job name>.pst/ccmp" or "<job name>.pstrt/ccmt" file can be found in the same directory as the input file.
To visualize the data from user scalar you should select the data type **Cell & Wall/Bound (Smooth)** option in pro-STAR.
12.3 Code-Specific MpCCI Commands

The MpCCI subcommands available for STAR-CD are:

Usage:
    mpcci StarCD [-]option

Synopsis:
    Use mpcci StarCD to get information about StarCD, to build/install your private PNP DSO adapter and more...

    Most of the required steps are automatically done within the MpCCI GUI. The commandline version may be helpful in case of problems and for debugging.

Options:
    -dsomake [-dp] [release]
        Install the StarCD PNP DSO library for MpCCI in your local "ufile" directory.

        If you have your own sources the new PNP DSO will contain both, MpCCI and your own code.
        If you wish to modify the sources then please first copy the MpCCI versions of POSDAT and NEWXYZ using the option -ufile.

        [-dp] is for linking DOUBLE PRECISION code.

    -gmdmake <case>
        Make an MpCCI grid morpher data file from a StarCD case file.

    -help
        This screen.

    -info
        List verbose information about all StarCD releases.

    -magic [case]
        Print the magic tokens of the MpCCI grid morpher data file
<case>.gmd or all gmd files in the current directory.

-probcheck [case]
    Print the MpCCI relevant switches from the problem file
    <case>.prob or all problem files in the current directory.

-releases
    List all StarCD releases which MpCCI can find.

-scan <case>
    Run the scanner and create a scanner output file.

-ufile [release]
    Copy the MpCCI versions of POSDAT and NEWXYZ into your local "ufile" directory and do some checks.
    Existing sources are automatically copied into backup files before.

    You may then edit the sources, add your own code to POSDAT and NEWXYZ and make the PNP DSO (see -dsomake).

The subcommands -info and -releases are described in ▷ 1.1 Common MpCCI Subcommands for Simulation Codes ◁.

12.4 Grid Morphing

The grid morpher will smooth a grid based on the displacements of some boundary or interior vertices. A moving or morphing grid capability is always required with fluid-structure interactions. The user has the choice between the MpCCI Grid Morpher (▷ 12.4.1 MpCCI Grid Morpher ◁) and the pro-STAR Grid Morpher (▷ 12.4.3 pro-STAR Grid Morpher ◁). Restarting a previously morphed analysis with MpCCI Grid Morpher is described in ▷ 12.4.2 Restart with MpCCI Grid Morpher ◁

12.4.1 MpCCI Grid Morpher

The morpher controls the results of the morphing step, it may control the length of the edges, the shape and size of faces and also checks the quality of cells. There are options available to limit the compression and elongation of edges.

The morpher allows vertices to float along semi-planar boundaries, e.g. symmetry planes.

The following options (Figure 6) may be adjusted, whereas the default values are in many cases appropriate.
Optional morpher hostname Enter an optional hostname for a remote execution of the MpCCI Grid Morpher.

Morpher port no. specifies the port address on which the MpCCI Grid Morpher listens for the client connection. The default port is in most cases acceptable, modifications are only required if firewalls block connections or multiple morphers are running in parallel on the same host.

Optional morphing script Invoke a shell script where all the options described below are specified. In the case where no script is defined, a script with the current options is stored in the working directory of the STAR-CD model. The shell script file will be named "mpcci_morpher_<model name>.sh".

Please select the output level Activate output level of the MpCCI Grid Morpher.

quiet no output
default medium level output
verbose high level output
full highest level output (verbose and received node displacements)

Check edges Activate to invoke quality checks for edges during morpher run.

Check faces Activate to invoke quality checks for faces during morpher run.

Check cells Activate to invoke quality checks for cells during morpher run.

List of cell type ids (Integers or ALL) Enter the cell table number of the fluid domains where the grid morpher should be applied. If you want to involve all fluid domains please enter “ALL”.

Min. relative edge length change Enter a lower limit for the ratio of the edge length, calculated by the morpher, to the edge length of the input grid provided to the morpher. A modified value might be interesting if the morpher generates too small edge lengths.

Max. relative edge length change Enter an upper limit for the ratio of the edge length, calculated by the morpher, to the edge length of the input grid provided to the morpher. This might be desired if the morpher generates too large edge lengths.

Min. change of face area Enter a lower limit for the ratio of the face area, calculated by the morpher, to the face area of the input grid provided to the morpher. This might be desired if the morpher generates too small face areas.

Max. change of face area Enter an upper limit for the ratio of the face area, calculated by the morpher, to the face area of the input grid provided to the morpher. This might be desired if the morpher generates too small face areas.

Max. face aspect ratio Enter an upper limit for the cells aspect ratio. For triangular faces the aspect ratio is build by the height to the corresponding maximum edge length.
**Min. angle allowed in faces and cells** Enter a lower limit for angles in faces and cells to prevent distortion.

**Fixed nodes on fixed boundaries** Enter the amount of cell levels to be fixed seen from boundaries with fixed nodes.

**Fixed nodes on deformed boundaries** Enter the amount of cell levels to be fixed seen from boundaries with displaced nodes. The specified number of node levels will be moved with the boundary in a rigid manner.

**Optional list of floating boundary regions** Enter the boundary region numbers corresponding to the STAR-CD model, where a sliding of vertices is permitted. By default floating is permitted only on symmetry and cyclic boundaries. Vertices on inlet, outlet and wall regions are fixed by default.

**Optional list of fixed boundary regions** Enter the boundary region numbers corresponding to the STAR-CD model, where a floating of vertices is explicitly not permitted (e.g. in the case where you might want to fix vertices on a symmetry plane).

**Factor for level based damping** Include a level based damping (default value is recommended).

**Morphing relaxation factor** Enter a relaxation factor for solving the equation system. Larger values will speed up convergence. In case of instability it might be helpful to reduce the value.

**Max. no. of iterations** Number of iterations for solving the equation system.

**Convergence tolerance** The convergence tolerance is the ratio of the maximum node displacement at the beginning of the morphing iteration to the node displacement of the current morphing iteration step. The iterative process for solving the equation system will stop, when the convergence tolerance is reached.

**Smoothing steps** Specify the amount of laplacian smoothing steps. Smoothing should be handled with care.

The morpher process may be monitored within an additional output terminal and a morpher log file in the STAR-CD working directory ("mpcci_morpher.<model name>.log").

### 12.4.2 Restart with MpCCI Grid Morpher

There are two ways to carry out a restart with previously morphed data:

- Take the original model file and read the physical quantities including moved grid coordinates from restart file ("<casename>.pst/<casename>.ccmp"). The MpCCI Grid Morpher reads the grid coordinates from original model file and then receives the restart coordinates in one step. For large displacements the morpher might collapse.
For a restart run one can build up a new model file with the already moved coordinates:

1. Start pro-STAR
2. Resume original model "<case_org>.mdl"
3. Read previous result "<case_org>.ccmp/.pst" which contains moved coordinates
4. Get the moved coordinates with `GETV_COORD`, pro-STAR will confirm `VERTEX COORDINATES STORED`
5. Define restart from "<case_org>.ccmp/.pst"
6. Write problem file "<case_restart>.prob"
7. Write geom file "<case_restart>.geom"
8. Save model as "<case_restart>.mdl", confirm
   
   **WARNING - THE COORDINATES CURRENTLY STORED WERE READ FROM POST DATA. IF YOU CONTINUE YOU WILL OVERWRITE YOUR ORIGINAL GEOMETRY. CONTINUE (N/Y)?
   
   with Y
9. You have now stored a model with moved coordinates which can be retrieved by the morpher. Use <case_restart> for further simulation. Be careful not overwrite your original model.

### 12.4.3 pro-STAR Grid Morpher

The pro-STAR Grid Morpher is available since STAR-CD 4.04 and supplied and supported by CD-adapco. There is a pro-STAR Grid Morpher documentation in the STAR-CD supplementary notes. Further documentation can be retrieved in pro-STAR help regarding the command `pro-STAR morp`. The pro-STAR Grid Morpher is applied by calling pro-STAR in batch mode during the simulation. The nodal deformations are received through MpCCI Code Adapter and handed over to pro-STAR.

In order to use the pro-STAR grid Morpher the following steps must be accomplished. As the procedure might change in future it is advisable to contact CD-adapco for recent definitions.

1. Open model file "<casename>.mdl" in pro-STAR.
2. Add the following lines to the extended data block:

   ```
   BEGIN MOVINGMESH
   IGNORE_LAHEAD_REQUESTS
   NXYZ_BEFORE
   NO_NXYZ_AFTER
   MOVE_ALL_PROCS
   END MOVINGMESH
   ```
3. Create a file "morph.cgrd" containing the line:

```
IFILE starmorph.MAC
```

4. Create an event file "<casename>.evn" in pro-STAR:

```
MEMO MAXEVE 10
MEMO WRITE
MVGR ON EVENT PROSTAR
EVFI INITIAL <casename>.evn
EVSTEP 1 TIME 0.0
EGRID READ morph.cgrd
EVSAVE 1
```

5. Provide a macro "starmorph.MAC" which is executed by pro-STAR. The following commands are exemplary:

```
! scale of boundary point length scale of influence (m)
*set scl1 0.0
*set scl2 0.01
*set scl3 0.1
!
! set increment frequency for updating reference mesh
!- incrm = 0 reference grid is always initial grid
!- incrm = 1 reference grid is always previous step
!- incrm = n reference grid is updated every nth step
*set incrm 10
!
! copy the moved and all other vertices coordinates into
! vertex register 1
vreg copy,, 0,1
!
! write out the reference grid at beginning
*if time eq 0.
  v writ PROSTAR_VERT1.vrt,,,,,bloc
*else
  ! read in the reference grid during run
  v read PROSTAR_VERT1.vrt,,,,,bloc
*endif
!
! reset the morpher
morp clear
!
```
!grab the vertices on the fsi interface
!(here e.g. boundary region 5)
bset news rlist 5
vset news bset
!
!If morphing in 2d in xy-plane is desired
morp dime xy
!
!limit number of control points on the fsi surface
!(here e.g. to 200)
morp tout 200
!
!set the control points on the fsi surface for morphing
morp vreg 1,vset,,, scl1
!
!fix the vertices on boundary planes
!(here e.g. region 1
morp gpla regi 1,, scl2
!
!allow sliding on boundary planes
!(here e.g. region 2)
morp gsym regi 2,, scl3
!
!grab the vertices on other boundaries to fix vertices
!(here e.g. region 8,9 and 10)
bset news rlist 8,9,10
vset news bset
!maybe thin out to get less control points on other boundaries
!(here e.g. to 100)
morp tout 100
!
!set control points of other boundaries for morphing
morp vreg 1,vset,,, scl2
!
!execute the morpher
morp exec
!
!update the reference grid at requested increments
bset news rlist 5
vset news bset
vreg copy vset 1 0
!
*if incrm gt 0
  *calc modu,,mod,ITER,,incrm
  *if modu eq 0
    vwr it PROSTAR_VERT1.vrt,,,bloc
  *endif
*endif

6. Activate necessary hooks as described in \texttt{12.5.3 Automatic model preparation for STAR-CD 3.26 and STAR-CD 4.0x <i>}.\texttt{

7. Use option \texttt{Start prepared case} in Go Step of MpCCI GUI.

When post-processing the results in \texttt{pro-STAR} carry out the following commands:

1. Connect to event file \texttt{EVFI,CONN,<casename>.evn}
2. Load result file \texttt{TRLO,<casename>.ccmt,MVGRID,, C}
Figure 6: STAR-CD Options for MpCCI grid morpher

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use grid morpher?</td>
<td></td>
</tr>
<tr>
<td>Optional morpher hostname</td>
<td></td>
</tr>
<tr>
<td>Morpher port no.</td>
<td>47010</td>
</tr>
<tr>
<td>Optional morphing script</td>
<td></td>
</tr>
<tr>
<td>Please select the output level</td>
<td></td>
</tr>
<tr>
<td>Min. relative edge length change</td>
<td>0.1</td>
</tr>
<tr>
<td>Max. relative edge length change</td>
<td></td>
</tr>
<tr>
<td>Min. change of face area</td>
<td>10.0</td>
</tr>
<tr>
<td>Max. change of face area</td>
<td>0.01</td>
</tr>
<tr>
<td>Max. face aspect ratio</td>
<td>10.0</td>
</tr>
<tr>
<td>Min. angle allowed in faces and cells</td>
<td>10.0</td>
</tr>
<tr>
<td>Fixed nodes on fixed boundaries</td>
<td>0</td>
</tr>
<tr>
<td>Fixed nodes on deformed boundaries</td>
<td>1</td>
</tr>
<tr>
<td>Optional list of floating boundary regions</td>
<td>14 15 16</td>
</tr>
<tr>
<td>Optional list of fixed boundary regions</td>
<td></td>
</tr>
<tr>
<td>Factor for level based damping</td>
<td>0.02</td>
</tr>
<tr>
<td>Morphing relaxation factor</td>
<td>1</td>
</tr>
<tr>
<td>Max. no. of iterations</td>
<td>30</td>
</tr>
<tr>
<td>Convergence tolerance</td>
<td>1.0E-3</td>
</tr>
<tr>
<td>Smoothing steps</td>
<td>0</td>
</tr>
</tbody>
</table>
12.5 Code Adapter Reference

The MpCCI code adapter for STAR-CD 3.26 and STAR-CD 4.0x is handled by different procedures. Whereas for STAR-CD 3.26 the data transfer and management is accomplished by user subroutines in the ufile directory, transfer is handled by STAR-CD 4.0x over plug-ins, which do not require user subroutines. As a result of the plug-in library, it is not necessary to compile user subroutines at runtime, unless the user has own user subroutines in the ufile directory. The necessary libraries or plug-ins can be found in "<MpCCI_home>/codes/StarCD/adapters/<StarCDrelease>/"

12.5.1 STAR-CD 3.26

The adapter involves user subroutines and user-defined scalar memory to manage the data transfer. The required user subroutines are located in the "ufile" subdirectory of the MpCCI installation. Additionally in this subdirectory subroutine examples for further coupling types "tut_<subroutine name> " are supplied. The necessity of certain user subroutines for a coupled simulation is summarized in Table 3.

Before the simulation run the user subroutines including the code adapter calls must be copied manually or automatically by MpCCI from "<MpCCI_home>/codes/StarCD/adapters/<StarCDrelease>/<ufile>" to the subdirectory "ufile" of the STAR-CD working directory. The procedure could be carried out manually by using MpCCI commands (see also ▶ 12.3 Code-Specific MpCCI Commands ▶)

- mpcci starcd -ufile [release] Copy sources to ufile directory.

- mpcci starcd -dsomake [release] Create DSO library.

or automatically by MpCCI if the option Rebuild shared libraries is selected in the Go Panel. If there are no changes of the platform or of the user subroutines, there is no need to rebuild the STAR-CD DSO library.

12.5.2 STAR-CD 4.0x

In comparison to the adapter of STAR-CD 3.26 the data transfer and management is now handled over plug-ins, which are implemented in the STAR-CD solver and no user subroutines are needed anymore. The activation of the plug-ins are invoked either in pro-STAR (Table 3) or automatically by MpCCI, when the button Use MpCCI plug-ins for V4 and Prepare case and start is activated (Figure 4). STAR-CD will verify the adequate (platform, precision) plug-in library "libstarcdmpcci.so" in the MpCCI installation.

As the data transfer is handled over the plug-in, "posdat.f" and the activation of a user defined posdat in pro-STAR is not needed anymore. Nevertheless STAR-CD must know which other plug-ins for data management are needed. Therefore like in STAR-CD 3.26 user defined subroutines must be activated automatically or manually (e.g. user bcdefw, see ▶ 12.5.3 Automatic model preparation for STAR-CD 3.26 and STAR-CD 4.0x ▶). If the MpCCI plug-in and a user subroutine is activated in the model (e.g. bcdefw), STAR-CD will first call the plug-in bcdefw which is included in the plug-in library and afterward call, if existing, the user bcdefw in the ufile directory.
12.5.3 Automatic model preparation for STAR-CD 3.26 and STAR-CD 4.0x

MpCCI has a limited ability to analyze the input in the MpCCI GUI and then decide what modifications have to be done to enable the model for a coupled simulation. This will be carried out when starting the simulation in the Go Step (\ref{12.2.4 Go Step}) when option \textit{Prepare case and start} is selected. One the other hand a user can do the model preparations. The commands which have to be carried out in \texttt{pro-STAR} are depicted in Table 3.

In Table 3 the variables stand for:

- $<\text{name}> = \text{arbitrary name}$,
- $<\text{region id}> = \text{boundary region id in STAR-CD of the coupled surface}$,
- $<\text{case}> = \text{STAR-CD casename}$,
- $<\text{scid}> = \text{scalar index for data storage}$,
- $<\text{ctype}> = \text{cell table number to apply source terms, defined in the Coupling Step}$,
- $\$ = \text{settings which are present in the original model}$,
- $n \times \$ = \text{depending on solver settings \texttt{pro-STAR} will prompt for your present settings (e.g. Roughness options)}$. 

<table>
<thead>
<tr>
<th>Feature</th>
<th>STAR-CD assignment</th>
<th>pro-STAR command</th>
<th>Quantity example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data transfer</td>
<td>user posdat</td>
<td>PRFIELD, $, $, $, USER</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>(only 3.26)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data transfer</td>
<td>activate plugin</td>
<td>PLUGIN, mpcci, ON</td>
<td>all</td>
</tr>
<tr>
<td></td>
<td>(only 4.0x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MpCCI Morpher</td>
<td>user newxyz</td>
<td>MVGRID, ON, $</td>
<td>NPosition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDATA, NEW, 1 $ KEEP_VERTDAT</td>
<td>(only 3.26)</td>
</tr>
<tr>
<td>pro-STAR Morpher</td>
<td>event handling</td>
<td>MVGRID, ON, EVENT, PROSTAR</td>
<td>NPosition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>see &gt; 12.4.3 pro-STAR Grid Morpher &lt;</td>
<td></td>
</tr>
<tr>
<td>Receiving time step</td>
<td>user dtstep</td>
<td>DELTIME, USER</td>
<td>DeltaTime</td>
</tr>
<tr>
<td>STAR-CD parallel</td>
<td>switch 110</td>
<td>SWITCH, 110, ON</td>
<td></td>
</tr>
<tr>
<td>Receive/send</td>
<td>scalar array</td>
<td>SC, &lt;scid&gt;, DEFINE, PASSIVE, UNDEF, 1 &lt;name&gt;</td>
<td>WallTemp</td>
</tr>
<tr>
<td></td>
<td>scalar post</td>
<td>SC, &lt;scid&gt;, OFF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>POWA, CW &lt;scid&gt;, Y</td>
<td></td>
</tr>
<tr>
<td>Receive/send</td>
<td>user bcdefw</td>
<td>RDEF, &lt;region id&gt;, WALL, USER</td>
<td>WallTemp</td>
</tr>
<tr>
<td>Face(2D)</td>
<td></td>
<td>n x $</td>
<td></td>
</tr>
<tr>
<td>Receive/send</td>
<td>user bcdefi</td>
<td>RDEF, &lt;region id&gt;, INLET, USER</td>
<td>AbsPressure</td>
</tr>
<tr>
<td>Face(2D)</td>
<td></td>
<td>n x $</td>
<td></td>
</tr>
<tr>
<td>Receive/send</td>
<td>user fluinj</td>
<td>RSOURCE, MASS, CTAB, &lt;ctype&gt;, USER</td>
<td>Density</td>
</tr>
<tr>
<td>Volume(3D)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive/send</td>
<td>user sourmom</td>
<td>RSOURCE, MOME, CTAB, &lt;ctype&gt;, USER</td>
<td>BodyForce</td>
</tr>
<tr>
<td>Volume(3D)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive/send</td>
<td>user sourrent</td>
<td>RSOURCE, ENTH, CTAB, &lt;ctype&gt;, USER</td>
<td>JouleHeat</td>
</tr>
<tr>
<td>Volume(3D)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Definitions in pro-STAR to enable the model for a coupled simulation
### 12.6 Trouble shooting, open issues and known bugs

<table>
<thead>
<tr>
<th>Feature: Volume Coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 3.26, 4.04</td>
</tr>
<tr>
<td>Problem: User scalars can not be retrieved from STAR-CD result file. In a restart run STAR-CD retrieves zeros for the first iteration.</td>
</tr>
<tr>
<td>Workaround: Possible workaround for fluids is to hook the user subroutine &quot;scalfn.f&quot; inserting PHI=SCALAR(&lt;scid&gt;). There is no workaround for coupled solid volumes.</td>
</tr>
<tr>
<td>References: &gt; VII-5 Busbar System &lt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature: Receiving time step from partner code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 3.26</td>
</tr>
<tr>
<td>Problem: Time step is set before reception. First time step is missing.</td>
</tr>
<tr>
<td>Workaround: Define a constant or variable time step in STAR-CD model which is retrieved by MpCCI or set the time step in &quot;dtstep.f&quot; DTSTEP=⟨dt⟩ before call of STARMPCCI_DTSTEP.</td>
</tr>
<tr>
<td>References: &gt; V-3.3.4 Exchange of Time Step Size &lt;, &gt; VII-3 Elastic Flap in a Duct &lt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature: Receiving time step from partner code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 4.04</td>
</tr>
<tr>
<td>Problem: Initial transfer is set to skip or send. Time step is unknown and not retrieved from model file.</td>
</tr>
<tr>
<td>Workaround: Change your coupling algorithm to have an initial reception of time step. &gt; V-3.3.4 Exchange of Time Step Size &lt;, &gt; VII-3 Elastic Flap in a Duct &lt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature: Polyhedral Meshes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 4.04</td>
</tr>
<tr>
<td>Problem: Coupled faces must be triangulated.</td>
</tr>
<tr>
<td>Workaround: Use the command</td>
</tr>
<tr>
<td>Feature</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>Face Coupling</td>
</tr>
<tr>
<td>Transfer after last iteration</td>
</tr>
<tr>
<td>Receiving Wall Temperature</td>
</tr>
<tr>
<td>Polyhedral Meshes</td>
</tr>
<tr>
<td>Restarting a previously morphed (MpCCI morpher) simulation by reading moved geometry from result file (.pst/.ccmp).</td>
</tr>
</tbody>
</table>
### Feature:
Receiving nodal positions/displacements in STAR-CD

#### Version:
4.04

#### Problem:
MpCCI Grid Morpher is not activated, STAR-CD does not get nodal displacements. The plugin for receiving nodal positions/displacements is not called in STAR-CD

#### Workaround:
Add lines to Extended Data Block in pro-STAR:
```
BEGIN MOVINGMESH
NXYZ,BEFORE
NO_NXYZ_AFTER
MOVE_ALL_PROCS
END MOVINGMESH
```

### Feature:
Restart with deformed Coupling Regions

#### Version:
3.26, 4.04

#### Problem:
At the end of the previous run the coupling regions to not match due to time difference. There are orphaned nodes when restarting.

#### Solution:
Doing a neighborhood search in MpCCI fails due to mesh differences. Use MpCCI restart files as described in ▷ V-3.3.6 Restarting a Coupled Simulation ◄.

### Feature:
Face coupling

#### Version:
3.26

#### Problem:
The coupling regions are incomplete with orphaned nodes

#### Solution:
Delete doubly defined boundaries for coupling region.

### Feature:
Parallel run on remote host

#### Version:
4.04, 4.06, 4.08

#### Problem:
STAR-CD exits with error message:

```
PNP: ***ERROR*** The required "mpcci" plug-in is not available in the STAR installation.
PNP: ==> Please check that the plug-in has been installed properly.
```

#### Solution:
The hosts configured for parallel run of STAR-CD do not include the host where MpCCI starts STAR-CD. Either include the host on which MpCCI starts STAR-CD in hostlist or provide the environment variable `STARPLUGIN_MPCCI=<MpCCI_home>/codes/StarCD` to all hosts intended for parallel run.
VII Tutorial
# VII Tutorial – Contents

## 1 Introduction

## 2 Vortex-Induced Vibration of a Thin-Walled Structure

2.1 Problem Description .......................................................... 8
2.2 Model Preparation .............................................................. 9
   2.2.1 Fluid Model ............................................................... 9
   2.2.2 Solid Model .............................................................. 13
2.3 Models Step ..................................................................... 14
2.4 Coupling Step .................................................................. 18
2.5 Edit Step ........................................................................... 19
2.6 Go Step ............................................................................. 20
2.7 Running the Computation .................................................... 23
2.8 Discussion of Results .......................................................... 25

## 3 Elastic Flap in a Duct

3.1 Problem Description .......................................................... 26
3.2 Model Preparation .............................................................. 26
   3.2.1 Solid Model ............................................................... 27
   3.2.2 Fluid Model .............................................................. 28
3.3 Models Step ..................................................................... 30
3.4 Coupling Step .................................................................. 34
3.5 Edit Step ........................................................................... 38
3.6 Go Step ............................................................................. 38
3.7 Running the Computation .................................................... 43
   3.7.1 Starting the Simulation ................................................. 43
   3.7.2 End of the Simulation .................................................. 45
3.8 Discussion of Results .......................................................... 45
## Exhaust Manifold 47

4.1 Problem Description ................................................................. 47
4.2 Model Preparation ................................................................. 48
  4.2.1 Solid Model ................................................................. 48
  4.2.2 Fluid Model ................................................................. 49
  4.2.3 Uncoupled Flow Simulation ............................................. 50
  4.2.4 Prepare Models for Coupled Simulation ............................... 51
4.3 Models Step ................................................................. 52
4.4 Coupling Step ................................................................. 53
4.5 Edit Step ................................................................. 54
4.6 Go Step ................................................................. 57
4.7 Running the Computation ..................................................... 61
4.8 Post-processing ................................................................. 62

## Busbar System 64

5.1 Problem Description ................................................................. 64
5.2 Model Preparation ................................................................. 65
  5.2.1 Fluid Model ................................................................. 65
  5.2.2 Electromagnetic Model .................................................. 66
5.3 Models Step ................................................................. 67
5.4 Coupling Step ................................................................. 70
5.5 Edit Step ................................................................. 74
5.6 Go Step ................................................................. 75
5.7 Running the Computation ..................................................... 78
5.8 Discussion of Results ............................................................. 79

## Pipe Nozzle 80

6.1 Problem Description ................................................................. 80
6.2 Model Preparation ................................................................. 80
  6.2.1 Fluid Model ................................................................. 81
  6.2.2 Solid Model ................................................................. 83
6.3 Models Step ................................................................. 83
6.4 Coupling Step ................................................................. 85
## VII Tutorial Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5</td>
<td>Edit Step</td>
<td>86</td>
</tr>
<tr>
<td>6.6</td>
<td>Go Step</td>
<td>86</td>
</tr>
<tr>
<td>6.7</td>
<td>Running the Computation</td>
<td>88</td>
</tr>
<tr>
<td>6.8</td>
<td>Discussion of Results</td>
<td>89</td>
</tr>
</tbody>
</table>

### 7 Cube in a Duct Heater

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Problem Description</td>
<td>90</td>
</tr>
<tr>
<td>7.2</td>
<td>Model Preparation</td>
<td>91</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Radiation Model</td>
<td>91</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Fluid Model</td>
<td>91</td>
</tr>
<tr>
<td>7.3</td>
<td>Models Step</td>
<td>93</td>
</tr>
<tr>
<td>7.4</td>
<td>Coupling Step</td>
<td>94</td>
</tr>
<tr>
<td>7.5</td>
<td>Edit Step</td>
<td>96</td>
</tr>
<tr>
<td>7.6</td>
<td>Go Step</td>
<td>96</td>
</tr>
<tr>
<td>7.7</td>
<td>Running the Computation</td>
<td>100</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Starting the Simulation</td>
<td>100</td>
</tr>
<tr>
<td>7.7.2</td>
<td>End of the Simulation</td>
<td>102</td>
</tr>
<tr>
<td>7.8</td>
<td>Discussion of Results</td>
<td>102</td>
</tr>
</tbody>
</table>

### 8 Y-Junction

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>Problem Description</td>
<td>106</td>
</tr>
<tr>
<td>8.2</td>
<td>Model Preparation</td>
<td>106</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Network Model</td>
<td>107</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Fluid Model</td>
<td>109</td>
</tr>
<tr>
<td>8.3</td>
<td>Models Step</td>
<td>111</td>
</tr>
<tr>
<td>8.4</td>
<td>Coupling Step</td>
<td>113</td>
</tr>
<tr>
<td>8.5</td>
<td>Edit Step</td>
<td>115</td>
</tr>
<tr>
<td>8.6</td>
<td>Go Step</td>
<td>116</td>
</tr>
<tr>
<td>8.7</td>
<td>Running the Computation</td>
<td>119</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Starting the Simulation</td>
<td>119</td>
</tr>
<tr>
<td>8.8</td>
<td>Discussion of Results</td>
<td>120</td>
</tr>
</tbody>
</table>
1 Introduction

The tutorial is a collection of examples. The files needed to run these examples are included in the MpCCI distribution in the directory "<MpCCI_home>/tutorial". All analysis steps are explained in detail. Please keep in mind that the descriptions are written to demonstrate the usage of MpCCI, they are not very useful if you want to learn how to use the simulation codes.

Overview of Tutorials

▷ 2 Vortex-Induced Vibration of a Thin-Walled Structure

- Physical domains: Fluid-Structure Interaction
- Codes: Abaqus, ANSYS, FINE/Hexa, FINE/Turbo, FLUENT, MSC.Marc
- Quantities: NPosition, RelWallForce
- Analysis type: Transient with pre-computed flow field
- Special topics: Different unit systems

▷ 3 Elastic Flap in a Duct

- Physical domains: Fluid-Structure Interaction
- Codes: Abaqus, ANSYS, FLUENT, MSC.Marc, STAR-CD
- Quantities: DeltaTime, NPosition, OverPressure
- Analysis type: Transient
- Special topics: Exchange of time step size, grid morphing

▷ 4 Exhaust Manifold

- Physical domains: Thermal Coupling Fluid-Solid
- Codes: Abaqus, PERMAS, FLUENT, STAR-CD
- Quantities: WallTemp, FilmTemp, WallHTCoeff
- Analysis type: Steady-state with pre-computed flow field
- Special topics: Subcycling of fluid solver

▷ 5 Busbar System

- Physical domains: Electrothermal Analysis
- Codes: ANSYS, FLUX, FLUENT, STAR-CD
- Quantities: JouleHeat, Temperature
- Analysis type: Steady-state flow field calculation and magnetic calculation in frequency domain

▷ 6 Pipe Nozzle

- Physical domains: Fluid-Structure Interaction
- Codes: Abaqus, FLUENT
- Quantities: RelWallForce
- Analysis type: Steady-state unidirectional exchange
- Special topics: Axisymmetry, different unit systems
7 Cube in a Duct Heater

Physical domains: Thermal Coupling Fluid
Codes: RadTherm, FLUENT, STAR-CD
Quantities: FilmTemp, WallHTCoeff, WallTemp
Analysis type: Steady-state radiative heat transfer

8 Y-Junction

Physical domains: Fluid Mechanics and Fluid Pipe System
Codes: Flowmaster FLUENT, STAR-CD
Quantities: BoundaryMassFlux, BoundaryTotalPressure, MassFlowRate, TotalPressure
Analysis type: Steady-state
2 Vortex-Induced Vibration of a Thin-Walled Structure

2.1 Problem Description

Figure 1: Vortex-induced vibration: Geometry and boundary conditions, flow field around the deformed structure. The dimensions are given in cm.

Topics of this Tutorial

- Fluid-Structure Interaction (FSI)
- Models with different unit systems: The solid model uses cgs units, the fluid model SI units.
- 2D-model

Simulation Codes

- Fluid Mechanics: FINE/Hexa 2.10, FINE/Turbo 8.8-1, FLUENT 6.3.26
- Solid Mechanics: Abaqus 6.7, ANSYS 110, MSC.Marc 2005r3

Description

The example is taken from an article by Walhorn et al. [2002] and is originally based on a computation by Wall [1999].

The computation consists of two parts. First, only the flow field is computed until a Von Kármán vortex street develops behind the square. The first step is computed without coupling, the flexible structure
remains rigid. After this initial calculation, a coupled analysis is carried through, the structure is now flexible. This yields vibration of the structure with increasing amplitude.

### 2.2 Model Preparation

The simulation couples a solid mechanics model with a fluid mechanics model. The files which you need for the simulation are included in the MpCCI distribution. Create a new directory and copy the particular subdirectories from "&lt;MpCCI_home&gt;/tutorial/VortexVibration" which correspond to the simulation codes you want to use.

#### 2.2.1 Fluid Model

The fluid domain comprises the whole rectangular area in Figure 1 except for the rigid square and the flexible structure.

The fluid has a constant density of \( \rho = 1.18 \times 10^{-3} \text{ g cm}^{-3} \) and a viscosity \( \mu = 1.82 \times 10^{-4} \text{ g cm s}^{-1} \).

**FINE/Hexa**

The mesh is provided with this tutorial. It is a ".igg" file. First you have to set up a pre-computation project for the first step without coupling. After that the parameters for the coupled simulation settings have to be defined.

### Pre-Computation

For the pre-computation following Project Parameters have to be set:

**Physical configuration** Set parameters for:

- **General properties** Select Time Configuration and the Block properties. The default values Steady and Fluid block are correct.

- **Fluid Model** Select Incompressible air and edit following values:
  - **Specific Heat Law** Constant \( C_p = 1006 \frac{\text{J}}{\text{kg K}} \)
  - **Heat Conduction Law** Constant \( K = 0.1 \frac{\text{W}}{\text{m K}} \)
  - **Viscosity Law** Constant Dynamic Viscosity \( \mu = 1.82e - 5 \text{ m}^2 \text{s}^{-1} \)
  - **Density Law** Boussineq law with density \( \rho = 1.18 \frac{\text{kg}}{\text{m}^3} \), compressibility \( 1e - 11 \text{Pa}^{-1} \) and dilatation \( 6.4e - 5 \text{K}^{-1} \)

- **Flow Model** Select Laminar Navier-Stokes as flow model.
  The Reference length and Reference velocity are just used to compute the Reynolds number, which is not used by the solver. As a consequence, the user can avoid to put values for these parameters.
Boundary conditions \rightarrow Boundary conditions Set conditions for following boundaries:

**INLET** Select Subsonic and Cartesian boundary condition,
Static Quantities Imposed,
Static Pressure Extrapolated
Set values for velocity: \( V_x = 0.315, V_y = V_z = 0 \)
and temperature: \( Static \ Temperature = 293 \).

**OUTLET** Select Pressure Imposed and Static Pressure Imposed.
Set value \( Static \ Pressure = 101325 \).

**SOLID** For the two first patches named VortexVibration\_group\_0 and VortexVibration\_group\_2, Euler and Euler Solid Wall have to be selected.
For the three following patches (groups 6 to 8), Navier-Stokes, Cartesian and Adiabatic have to be selected. In addition, select Compute force and torque because we want to exchange the relative wall forces.
For the last five patches (groups 9 to 13) set the Navier-Stokes Cartesian adiabatic type as done for the previous patches but not computing force and torque.

**Initial Solution** \rightarrow Initial Solution Set Coordinate System to Cartesian,
\( Pressure = 101325, Temperature = 293 \)
and values for velocity: \( V_x = 0.315, V_y = V_z = 0 \).

**Numerical Parameters** Set parameters for:

**General parameters** Set Number of grid(s) to 4 and
select Correction damping and Coarse grid initialization.

**Numerical schemes** For Discretization scheme select Central and set CFL number to 3.0.
The parameters for the preconditioning are:
\( \beta^* \ coefficient = 3.0, \)
\( Characteristicvelocity = 0.315 \) and
Local velocity scaling selected.

**Computational control** Following settings have to be done:

**Outputs** Select External flow.

**Control Variables** \rightarrow General Set Number of iterations = 1000,
\( Convergence \ criteria = -5.0 \) and
\( Save \ solution \ every = 100. \)

**Launching Mode** Select Serial.

Save the project and also the run files and launch the computation by starting the solver.
Setting up the Coupled Simulation

Now the parameters for the coupled simulation can be set. Therefore create a new computation from the existing pre-computation and name it e.g. serial.

Following Project parameters changed compared to the pre-computation parameters:

**Physical configuration → General properties** Select Time Configuration and set it Unsteady.

**Initial Solution → Initial Solution** Because the initial solution is started from the pre-computation, select From file and select as File name the pre-computation ".run" file (e.g. "preComputation.run" provided with this tutorial).

**Optional Models → Coupling** As Coupling method select Coupling with MpCCI.
   Set Reference pressure = 101325.0.
   Select the three surfaces that have to be coupled (groups 6 to 8), select Coupling activation and choose the Deformation type (moving or not moving grid points).

**Numerical Parameters → General parameters** Deactivate Coarse grid initialization.

**Computational control → Control Variables** Following settings have to be done:

- **General** Set Physical time step = 0.001 and Number of time steps = 5001.

- **Advanced** Since the FINE/Hexa package is without an FSI interface, the coupling has to be set through expert parameters:

  **IMpCCIzSize** has to be set to 0.002. This parameter corresponds to the size of the mesh in z direction. It is pointed out that the vortex vibration is a 2D case. FINE/Hexa 2D meshes contain always one cell in the z-direction. Therefore when computing forces, they have to be scaled.

Finally the run and batch files must be saved in order to use them with MpCCI.

**FINE/Turbo**

For FINE/Turbo following configuration settings have to be done:

**Configuration** Set parameters for:

**Fluid Model** New incompressible fluid has to be added. Its properties are:

- **Specific Heat Law** Constant \( C_p = 1004.5 \frac{J}{kg \, K} \)
- **Heat Conduction Law** Constant \( K = 0.1 \frac{W}{m \, K} \)
- **Viscosity Law** Constant Kinematic Viscosity \( \nu = 1.5424e - 5 \frac{m^2}{s} \)
Density Law Boussineq law with density \(1.18 \frac{kg}{m^3}\), compressibility \(0 Pa^{-1}\) and dilatation \(0 K^{-1}\).

Flow Model The time configuration is set to unsteady.
Select Laminar Navier-Stokes as Mathematical model.
No gravity forces and passive tracers are used.
The characteristic and reference quantities are set as follows:
Length: 0.01 m
Velocity: 0.315 \(\frac{m}{s}\)
Density: \(1.18 \frac{kg}{m^3}\)
Temperature: 293 K
Pressure: 101325 Pa

Boundary conditions Set conditions for following boundaries:

Inlet All patches are grouped. Select Subsonic Cartesian boundary condition,
Static Quantities Imposed and Static Pressure Extrapolated
Set imposed values for velocity: \(V_x = 0.315 \frac{m}{s}\), \(V_y = 0 \frac{m}{s}\), \(V_z = 0 \frac{m}{s}\)
and temperature: Static Temperature = 293 K.

Outlet Subsonic outlet is selected and static pressure is imposed. Select Pressure Imposed and Static Pressure Imposed at 101325 Pa.
Neither Backflow Control nor Rotating Boundary Condition are selected.

Solid All patches have Navier-Stokes Cartesian boundary condition set as Adiabatic with default value.
Patches corresponding to the flexible structure are grouped under Moving_Surface name. For this group, Compute force and torque can be activated to follow evolution of lift and drag during computation.

Numerical Model CFL value is kept to 3.
Spatial and temporal discretization settings (accessible through expert mode) are kept as default.

Multigrid parameters Current grid level is set to 1 1 0. 3 grids are used without coarse grid initialization.
Sweeps scheme (accessible through expert mode) is Increase on higher level.

Preconditioning parameters Hakimi method is selected with \(\beta^* coefficient = 3\)
and Characteristic velocity = 0.315.
Local velocity scaling is not activated.

Initial Solution A restart is performed from a converged steady computation on same grid level. The history of convergence is reset.

Outputs User is free to select desired output among those proposed. These outputs will be available for the post-processing with CFView.
**Computational steering** Following settings have to be done:

- **Control Variables**  
  **Physical Time Step** is set to 0.001s.

To reduce the computation time, the **Number Of Physical Time Step** is set to 5000 to get a solution on 5s.

No steady initialization is performed.

Solution is saved every 50 time steps with **Multiple Files** to have the possibility to post-process all the unsteady results and not just only the last solution at $t = 5s$.

Some of the expert parameters have to be modified to allow coupling through **MpCCI**. Be sure that expert mode is activated on this page.

For Integer:

- **IMPCCI** is set to 1 to activate MpCCI exchanges.
- **IMOVGR** is set to 1 to activate mesh deformation module.
- **IMOVTY** is set to 3 to use radial basis function interpolation for the mesh deformation.
- **IMVTPY** is set to 2 to define deformation as elastic.
- **ISVMAT** is set to 1 (default value) to use inverse matrix for radial basis function interpolation.
- **MOVRBF** is set to 2 (default value) to use TPS radial basis function.
- **ICOARS** is set to 0 1 (default value) to define coarsening level. All nodes on flexible structure are taken into account for mesh deformation. On other boundaries only one node in two is used.
- **IINT** is set to 0 to define the project as an external (allow to follow lift and drag evolution).

For Float:

- **MOVRAD** is set to 1 (default value). No need to scale the distance between nodes for this computation.
- **IDCDP** is set to 1 0 0 to set drag direction along $x$.
- **IDCLP** is set to 0 1 0 to set lift direction along $y$.

- **Convergence History**  
  As the computation is launched in batch mode by **MpCCI** no convergence history can be followed in **FINE/Turbo** GUI during the computation.

Finally the run and batch files must be saved in order to use them with **MpCCI**.

**FLUENT**

The mesh was generated with **Gambit**, around the flexible structure, the elements are smaller than those at the outer boundaries. The surface of the flexible structure is defined as a separate boundary named “flexible-struct”.

... 

### 2.2.2 Solid Model

The solid part corresponds to a rectangle:
The material is linear elastic with density $\rho = 2.0 \text{ g/cm}^3$, elastic modulus $E = 2.0 \cdot 10^6 \text{ g cm}^{-2} \text{ s}^2$ and Poisson’s ratio $\nu = 0.35$.

The mesh consists of $128 \times 6$ linear 4-node plane-strain elements. All nodes at the left end are fixed, i.e. they cannot move in $x$ and $y$-direction. The whole surface except for the fixed end is defined as an element set, which can be selected as coupling component.

All solid models are created using the cgs unit system, i.e. the units given above.

The solid model receives forces from the fluid model, which are computed with an out-of-plane thickness $t = 1 \text{ m}$. Therefore the solid models must have the same thickness $t = 100 \text{ cm}$.

### Abaqus

In Abaqus, CPE4 elements are used.

### ANSYS

In ANSYS, element PLANE42 is used, the coupling surface is covered with BEAM3 elements, which are deselected for the solution.

ANSYS does not provide the possibility to use a plane strain model with a given thickness. Therefore a plane stress model is used instead, which yields a softer structure.

### MSC.Marc

In MSC.Marc element type 11 is used, i.e. plane-strain full integration quadrilateral elements.

---

### 2.3 Models Step

Start the MpCCI GUI by running the command `mpcci gui`. In the first step, the Models Step, the input files of the coupled codes are defined and scanned.
Select FINE/Hexa as the first code to couple. The FINE/Hexa version is based on a batch file which is also used as input file for scanning. Select "vortexVibration_serial.run" in the subdirectory "FINEHexa/vortexVibration_serial/" as input file. The FINE/Hexa solver internally only operates in SI units. Press the **Start Scanner** button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.

**FINE/Turbo**

Select FINE/Turbo as the first code to couple. The FINE/Turbo version is based on a batch file which is also used as input file for scanning. Select "VortexVib_220_cpl_ustd.run" in the subdirectory "FINETurbo/VortexVib_220_init_ustd/" as input file. The FINE/Turbo solver internally only operates in SI units. Press the **Start Scanner** button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.

**FLUENT**

Select FLUENT as the first code to couple. The model is two-dimensional, thus select the FLUENT version 2d. The FLUENT release should be 6.3.26 or higher. Select the input file "FLUENT/plate.cas" and press the **Start Scanner** button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.
Select Abaqus as the second code to couple. Select any Abaqus 6.7 release.
Choose the input file "Abaqus/plate.inp".
The Abaqus model was created using the cgs unit system, i.e. the values given in Figure 1 were directly entered in Abaqus/CAE and the geometry was created in cm. It is important to select cgs in the Models Step to ensure that dimensions and quantities are transferred correctly.
Please press the Start Scanner button and a green check mark should appear at the top as shown on the left.

Select ANSYS as the second code to couple and ANSYS 110. You can select any ANSYS product which can be used for structural analysis, e.g. ansys.
Choose the database file "ANSYS/plate.db".
The ANSYS model was created using the cgs unit system, i.e. the values given in Figure 1 were directly entered in ANSYS and the geometry was created in cm. It is important to select cgs in the Models Step to ensure that dimensions and quantities are transferred correctly.
Please press the Start Scanner button and a green check mark should appear at the top as shown on the left.
Select **MSC.Marc** as the second code to couple. Choose the input file "**MSC.Marc/plate.dat**". The **MSC.Marc** model was created using the cgs unit system, i.e. the values given in Figure 1 were directly entered in **MSC.Marc Mentat** and the geometry was created in cm. It is important to select **cgs** in the Models Step to ensure that dimensions and quantities are transferred correctly. Please press the **Start Scanner** button and a green check mark should appear at the top as shown on the left.
2.4 Coupling Step

Press the **Next >** button at the bottom of the Models Step to get to the Coupling Step.

![Coupling Step Diagram]

In this example the coupling region corresponds to the surface of the flexible structure. **MpCCI** treats this region as a “Face” because it represents a 2D structure.

1. Select the **Face 2D** tab.

2. Select the coupling components by dragging them into the **Coupled** boxes.

### FINE/Hexa
Select all three coupling components.

### FINE/Turbo
Select all coupling components.

### FLUENT
Select the coupling component “flexible-struct”.

---

18  VII  MpCCI 3.1.1-1
Abaqus

Select “ASSEMBLY_OUTSIDE”.

ANSYS

Select “OUTSIDE”.

MSC.Marc

Select “outside”.

3. Select the quantities NPosition and RelWallForce.

Proceed to the Edit Step by pressing [Next >].

2.5 Edit Step

Some codes require changes in the Edit Step.

FINE/Turbo

Set Contact → SearchAlgorithm to Intersection.
All other parameters including Contact → Intersection are left to their default values.

Proceed to the Go Step by pressing [Next >].
2.6 Go Step

In the server panel, no changes are necessary. The code panels are described for each code.

Set Initial exchange to exchange. Nothing else has to be done because FINE/Hexa always runs in batch mode. So the batch file generated by FINE/Hexa GUI in §2.2 Model Preparation defines all computational parameters.
No options need to be set. **Initial exchange** is set to exchange by default and FINE/Turbo always runs in batch mode. So the batch file generated by FINE/Turbo GUI in ▶2.2 Model Preparation◁ defines all computational parameters.

Set the **Initial quantities transfer** to **exchange**, choose the data file "FLUENT/plate-2160.dat" and keep the default settings for all other options.
Set Initial quantities transfer to receive.
Select Constant coupling time step to use a constant time step.
Set the coupling time step to 0.001, which equals the time step size in the Abaqus input file.
Deactivate Subcycling.

Set Initial quantities transfer to receive.
Keep -b as gui option to start ANSYS in batch mode.
Additional command line options are not required.
Select the APDL input script "runplate.ans" by pressing the [Browse] button.
2.7 Running the Computation

Save the MpCCI project file with name "plate.csp" over the MpCCI GUI menu File→Save Project As.
Press the three Start buttons in the Go Step and the simulation codes should start. Some codes require additional actions:

**FLUENT**

The computation needs not to be initialized as a flow field is loaded from "FLUENT/plate-2160.dat". Select Solve→Iterate... to open the Iterate panel. Set the Number of Time Steps to 10000 and press Iterate to start the simulation.

During the simulation, FLUENT will display the residual, the resulting moment of the forces on the coupling surface and the pressure distribution in the fluid domain. You can also see the deformation of the flexible structure in the pressure distribution window.
Figure 2: Vortex-induced vibration: Part of the flow field in FLUENT with velocity vectors and pressure distribution (red=high pressure, blue=low pressure).

Figure 3: Vortex-induced vibration: Vertical deflection of the tip.
2.8 Discussion of Results

The vortex street yields pressure changes at the surface of the flexible structure, a typical pressure distribution is shown in Figure 2. The resulting forces accelerate the structure and cause a vibration with increasing amplitude.

However, as the structural models are not perfectly identical – the Abaqus model is a little stiffer than the MSC.Marc model. The ANSYS model is much weaker as the other two, because the ANSYS model is based on a plane stress assumption while the others use plane strain. Therefore the natural frequencies of the models differ, which yields different responses as shown in Figure 3.
3 Elastic Flap in a Duct

3.1 Problem Description

Figure 1: Elastic flap: Geometry [m] and boundary conditions

Topics of this Tutorial

- Fluid-Structure Interaction (FSI)
- Coupling of time step (sent by solid mechanics code)
- Usage of the MpCCI grid morpher for STAR-CD
- 3D-model

Simulation Codes

- Solid Mechanics: Abaqus 6.6 or 6.7, ANSYS 110, MSC.Marc 2005r3
- Fluid Mechanics: FINE/Hexa 2.10, FINE/Turbo 8.8-1, FLUENT 6.3.26, STAR-CD 3.26 or 4.0x

3.2 Model Preparation

The simulation couples a solid mechanics model with a fluid mechanics model. The files which you need for the simulation are included in the MpCCI distribution. Create a new directory and copy the subdirectories from "<MpCCI_home>/tutorial/ElasticFlap" which correspond to the simulation codes you want to use.
3.2.1 Solid Model

The solid part corresponds to a rectangle (Figure 3):

![Figure 2: Elastic flap: Mesh of the structural domain](image)

The material is linear elastic with density $\rho = 1000 \text{ kg/m}^3$, elastic modulus $E = 1.0 \times 10^8 \text{ Pa}$ and Poisson’s ratio $\nu = 0.49$.

The mesh consists of $20 \times 20 \times 2$ brick elements with 20 nodes each. All nodes at the top are fixed, i.e. they cannot move in any direction.

The solid mechanics code shall determine the time step size $\Delta t$. Initially it is set to $\Delta t = 0.25 \text{ ms}$, the automatic size adaptation is limited by $0.1 \text{ ms} \leq \Delta t \leq 2 \text{ ms}$.

A control-point/node is selected in each solid model for evaluation of the results, see ▷3.8 Discussion of Results ◁.

**Abaqus**

C3D20R elements are used. The whole surface except for the fixed end is defined as an element set, which can be selected as coupling component.

**ANSYS**

The standard 20-node brick element type SOLID95 - 3D 20-Node Structural Solid is used. In ANSYS, no automatic time stepping scheme can be used for a coupled simulation. Instead only two different time step sizes are used for the computation. In the APDL script the step size is set by:
*if, i, le, 400, then  ! compute physical time from step  
time, (i-1)*0.00025  ! small step for t <= 0.1  
*else  
time, (i-401)*0.001 + 0.1 ! larger step for t>0.1

I.e. a time step of $\Delta t = 0.00025\,\text{s}$ is used up to $t = 0.1\,\text{s}$, after that a time step size of $\Delta t = 0.001\,\text{s}$ is used. The time step size is sent to the CFD code.

**MSC.Marc**

The fully integrated 20-node brick elements 21 are used.

...  

### 3.2.2 Fluid Model

The fluid domain comprises the whole rectangular area in Figure 3 except for the flexible elastic flap structure.

![Figure 3: Elastic flap: Partial mesh of the fluid domain](image)

**FINE/Hexa**

- Open the FINE/Hexa project from "FINEHexa/elasticFlap.iec".

---

28 VII MpCCI 3.1.1-1
• Open the **Coupling Parameters** options dialog and check for the following boundaries “Elastic_flap_group_0”, “Elastic_flap_group_1”, “Elastic_flap_group_2”, “Elastic_flap_group_3”, “Elastic_flap_group_4” Figure 4 that

  – coupling method is set to Coupling with MpCCI
  – Reference pressure is equal to 101325
  – Coupling activation is checked
  – Deformation type is set on Moving grid points

• Open the **Boundary Conditions** options dialog Figure 5 and activate the **Compute force and torque** for the same boundaries previously activated for the coupled. This is required when RelWallForce or OverPressure quantity is exchanged.

• Save the run file for the selected computation Serial in that example and save the batch file under the **Solver**.

![Figure 4: Elastic flap: Coupling activation](image-url)
The mesh was generated with Gambit and consists of tetrahedral elements. The surface of the flexible structure is defined as a separate boundary named “couple-flap”.

The mesh corresponds to the mesh which is used in FLUENT. The boundary regions in the middle are subdivided, because they will later be addressed by the MpCCI grid morpher as sliding boundaries.

### 3.3 Models Step

Start the MpCCI GUI by running the command `mpcci gui`. In the first step, the Models Step, the input files of the coupled codes are defined and scanned.
- Select **Abaqus** as the first code to couple.
- Select any **Abaqus 6.6** or **Abaqus 6.7** release or higher.
- Choose the input file "**Abaqus/elastic_flap.inp**" and the SI unit system (which is the standard).
- Please press the **Start Scanner** button and a green check mark should appear at the top as shown on the left.

- Select **ANSYS** as the first code to couple. Select the release 110.
- You can select any **ANSYS** product which can be used for structural analysis, e.g. **ansys**.
- Choose the input file "**ANSYS/elastic_flap.db**"
- The SI unit system is used here (which is the standard).
- Please press the **Start Scanner** button and a green check mark should appear at the top as shown on the left.
Select MSC.Marc as the first code to couple.

Select release 2005r3 or higher.

Choose the input file "MSC.MARC/elastic_flap_job1.dat" and the SI unit system (which is the standard).

Please press the Start Scanner button and a green check mark should appear at the top as shown on the left.

Select FINE/Hexa as the first code to couple. The FINE/Hexa version is based on a batch file which is also used as input file for scanning.

Select "elasticFlap_serial.run" in the subdirectory "FINEHexa/elasticFlap_serial/" as input file.

The FINE/Hexa solver internally only operates in SI units.

Press the Start Scanner button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.
• Select FINE/Turbo as the first code to couple. The FINE/Turbo version is based on a batch file which is also used as input file for scanning.

• Select "ElasticFlap_mpcci_111.run" in the subdirectory "FINETurbo/ElasticFlap_mpcci_111/" as input file.

• The FINE/Turbo solver internally only operates in SI units.

• Press the [Start Scanner] button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.

• Select FLUENT as the second code to couple.

• The model is three-dimensional, thus select the FLUENT version 3d.

• The FLUENT release should be set to 6.3.26 or higher.

• Select the input file "FLUENT/elastic_flap.cas" and press the [Start Scanner] button.

• Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.
Press the Next button at the bottom of the Models Step to get to the Coupling Step.

3.4 Coupling Step

In this example the time steps of the two solvers are coupled, whereas the CSM-code (CSM = Computational Structural Mechanics) will send the time step to the CFD-code (CFD = Computational Fluid Dynamics).

Select Global (0D) tab, and then configure the corresponding coupling components as depicted in Figure 6 and described in the following.

**Abaqus**

Double-click or drag down the coupling component “Time-Step-Size”. The component will appear in the coupled field of the solver.

**ANSYS**

Double-click or drag down the coupling component “DELTATIME”. The component will appear in the coupled field of the solver.
Figure 6: Elastic flap: Coupling of global variable “time step”, CSM-Code corresponds to Abaqus or ANSYS and CFD-Code corresponds to FINE/Hexa, FINE/Turbo, FLUENT or STAR-CD

**MSC.Marc**

Double-click or drag down the coupling component “Time-Step-Size”. The component will appear in the coupled field of the solver.

**FINE/Hexa**

Double-click or drag down the coupling component “Time-Step-Size”. The component will appear in the coupled field of the solver. Select FINE/Hexa as **Sender** for **DeltaTime** in the right column.

⚠️ In this example the CFD-Code and not the CSM-Code will be the sender!

**FINE/Turbo**

Double-click or drag down the coupling component “Time-Step-Size”. The component will appear in the coupled field of the solver. Select FINE/Turbo as **Sender** for **DeltaTime** in the right column.
In this example the CFD-Code and not the CSM-Code will be the sender!

**FLUENT**

Double-click or drag down the coupling component “Time-Step-Size”. The component will appear in the coupled field of the solver. Select the CSM-Code as **Sender** for **DeltaTime** in the right column.

**STAR-CD**

Double-click or drag down the coupling component “Time-Step-Size”. The component will appear in the coupled field of the solver. Select the CSM-Code as **Sender** for **DeltaTime** in the right column.

In this example the coupling region corresponds to the surface of the flexible structure. **MpCCI** treats this region as a “Face” because it represents a 2D structure.

![Diagram image]

Figure 7: Elastic flap: Coupling of “NPosition” and “OverPressure”, CSM-Code corresponds to Abaqus or MSC.Marc and CFD-Code corresponds to FINE/Hexa, FINE/Turbo, FLUENT or STAR-CD.
Select **Face (2D)** tab, and then configure the corresponding coupling components as depicted in Figure 7 and described in the following.

**Abaqus**
Double-click or drag down the coupling component “ASSEMBLY_WALL_WALL-SURFACE”, which characterizes the surface of the flap. The component will appear in the coupled field of the solver.

**ANSYS**
Double-click or drag down the coupling component “WALLSURFACE”, which characterizes the surface of the flap. The component will appear in the coupled field of the solver.

**MSC.Marc**
Double-click or drag down the coupling component “wallsurface”, which characterizes the surface of the flap. The component will appear in the coupled field of the solver.

**FINE/Hexa**
Double-click or drag down all coupling components, which characterize the surface of the flap. The component will appear in the coupled field of the solver.
Select “elem” for the quantity location.

**FINE/Turbo**
Double-click or drag down all coupling components, which characterize the surface of the flap. The component will appear in the coupled field of the solver.

**FLUENT**
Double-click or drag down the coupling component “couple-flap”, which characterizes the surface of the flap. The component will appear in the coupled field of the solver.

**STAR-CD**
Double-click or drag down the coupling component “couple-flap”, which characterizes the surface of the flap. The component will appear in the coupled field of the solver.

... In the quantities field activate the NPosition and OverPressure check buttons. By clicking on the quantity name a configuration area for this quantity is opened on the right side. Choose the CSM-code as sender of NPosition and the CFD-code (FINE/Hexa, FINE/Turbo, FLUENT or STAR-CD) as sender of OverPressure. OverPressure corresponds the relative pressure (▷ V-3.1.2 Coupling Types ◄) and will not account for frictional forces, which are included in the relative wall forces (RelWallForce).

Except for selecting the coupling regions, no further settings are required, proceed to the Edit Step by pressing [Next >].
3.5 Edit Step

No changes are required in the Edit Step, proceed to the Go Step by pressing Next >.

3.6 Go Step

In the server panel, no changes are necessary. The code panels are described for each code.
• Set Initial quantities transfer to receive. All other options should be changed.

• It is important to deselect the option Constant coupling time step (which is the default) because the time step size is flexible: It is determined by Abaqus and sent to the CFD code.

• Set Initial quantities transfer to receive.

• Keep the gui option set to -b.

• Select the APDL input script "runflap.ans".
MSC.Marc

- Set Initial quantities transfer to receive.
- All other options should not be changed.

FINE/Hexa

- Set Initial quantities transfer to exchange.
- Nothing else has to be done because FINE/Hexa always runs in batch mode. So the batch file defines all computational parameters.

FINE/Turbo

- No options need to be set. Initial quantities transfer is set to exchange by default and FINE/Turbo always runs in batch mode. So the batch file defines all computational parameters.
FLUENT

- Set the Initial quantities transfer to exchange.
- Keep the default settings for all other options.
- Set the Initial quantities transfer to exchange.

- To prepare the model for a coupled simulation select as start-up procedure Prepare case and start.

- STAR-CD 3.26: To obtain all necessary user subroutines check button for Rebuild shared library. Deactivate Use the MpCCI plugin library (Star V4 only). Deactivate Use a data filter subroutine (Star V4 only).

- STAR-CD 4.0x: Activate Use the MpCCI plugin library (Star V4 only). Deactivate Use a data filter subroutine (Star V4 only).
3.7 Running the Computation

3.7.1 Starting the Simulation

Save the MpCCI project file with name "elastic_flap.csp" over the MpCCI GUI menu File→Save Project As.

Press the three [Start] buttons in the Go Step and the simulation codes should start. Some codes require additional actions:

- Select the option Use grid morpher?. Further options for the grid morpher will become visible.
- Enter 2 in the entry for List of cell type ids to activate the morpher only for the fluid part around the flap of cell type 2.
- Enter the boundary regions with number 14, 15, and 16 in the entry of Optional list of floating boundary regions to allow nodes to slide on boundary regions close to the flap.

Refer to ▷ VI-12.4 Grid Morphing for more information on the MpCCI grid morpher options.

The numbers for cell types and boundary regions correspond to the numbering in the STAR-CD model file.
Initialize the flow field by selecting **Solve → Initialize → Initialize** to open the initialization panel. And press the button **Init**.

Select **Solve → Iterate...** to open the Iterate panel. As FLUENT receives the time step size from the solid mechanics code, the **Time Stepping Method** must be set to **Adaptive**. If you select this option, the panel is extended by the new part **Adaptive Time Step Parameters** at the right. Please select *UDF_Deltat::libmpcci* as **User-Defined Time Step** at the bottom. This user-defined function receives the time step sizes from MpCCI. When the simulation is started, FLUENT needs an initial time step, which will be retrieved from the the FLUENT entry field **Time Step Size(s)**. The initial time step of 0.00025 matches the initial time step of the CSM-code to keep the codes simultaneous in time (**V-3.3.4 Exchange of Time Step Size**).

As the further time step sizes are variable, you should select a large **Number of Time Steps**, e.g. 1000 which exceeds the number of time steps of the solid mechanics code.

Finally press **Iterate** to start the simulation.

During the simulation, FLUENT will display the residuals and a section through the flap, where you can see its deformation.
If time step is received, STAR-CD requires like FLUENT an initial time step. This is always necessary even if the partner code is adjusted to send or exchange. This is due to the fact that the user subroutine to retrieve the received time step, namely "dtstep.f", is called in advance of the exchange procedure in "posdat.f".

MpCCI has a built-in procedure to retrieve the initial time step from "exhaust_manifold.mdl" and insert it into the user subroutine "dtstep.f". After the first reception of the partner’s code time step, the initial time step will be overwritten. In this tutorial the initial time step is set to 0.00025 as a constant time step in "exhaust_manifold.mdl", which equals the initial time step of the CSM-code (▷ V-3.3.4 Exchange of Time Step Size◁).

The MpCCI plug-in to receive data is called prior to the time step setting. Therefore the initial time step is already received from Abaqus if the initial transfer is set to receive or exchange like in this tutorial.

3.7.2 End of the Simulation

The CSM codes determine the time step size. Therefore only they have the information when to finish the simulation. In this case it is almost impossible to obtain a “clean exit” of both codes.

- Abaqus simply stops the simulation and exits once it has reached the end time, which causes the CFD code to show an error message as the connection is lost.

- ANSYS simply stops the simulation and exits, which causes the CFD code to show an error message as the connection is lost.

- MSC.Marc finally sends a time step size of zero seconds, which causes errors in the CFD codes. Please stop both codes at this point and do neglect the result of the final time step.

If the simulation has reached the end time of $t = 0.2\text{s}$ you can kill the remaining codes with the Kill button. All of the codes used in this example still write complete result files, therefore no data loss should be expected and it is possible to obtain the results described below.

3.8 Discussion of Results

To compare the results of the different simulations, a control point was selected in the solid mechanics codes, see Figure 2. To display the motion of this point in x-direction similar to Figure 8 proceed as follows:
**Abaqus**

Open the output database "abaqus.run.odb" in Abaqus/CAE to plot the displacement of the control point:

Select Result→History Output from the menu bar to open the History Output panel.
Select U1 at Node 1 in NSET CONTROL-NODE and press Plot to plot the displacement in x-direction.

**ANSYS**

Start ANSYS and select TimeHist PostPro, open the file "file.rst". In the Time History Variables panel, press the + button and select Nodal Solution → DOF Solution → X-Component of displacement and select node 366. Press the Graph Data button to plot the graph.

**MSC.Marc**

In MSC.Marc Mentat select RESULTS and open the "mpcci_elastic_flap_job1.t16" file. Switch to HISTORY PLOT, press SET NODES and select the set “control-node”. Press COLLECT DATA and collect data from step 0 to 1000 with increment size 1 (MSC.Marc Mentat will stop at last step). Go to NODES/VARIABLES and press ADD VARIABLE. Select “Time” and “Displacement X” to plot the curve. Finally press FIT to fit the plot limits, otherwise the curve is displayed very small.

![Graph](image.png)

Figure 8: Elastic flap: Displacement of the control point for Abaqus, ANSYS and MSC.Marc coupled with FLUENT.
4 Exhaust Manifold

4.1 Problem Description

Figure 1: Manifold: Geometry [m] and boundary conditions

Topics of this Tutorial

- Thermal coupling fluid and solid
- Steady state
- Incompressible fluid
- Fluid solver iterations without coupling
- Fluid solver subcycling
- 3D-model
Simulation Codes

- Solid Mechanics: Abaqus 6.7, PERMAS 11
- Fluid Mechanics: STAR-CD 3.26, 4.04 or higher, FLUENT 6.3.26

4.2 Model Preparation

The simulation couples a solid mechanics model with a fluid mechanics model. The files which you need for the simulation are included in the MpCCI distribution. Create a new directory "ExhaustManifold" and copy the subdirectories from "<<MpCCI_home>/tutorial/ExhaustManifold" which correspond to the simulation codes you want to use.

4.2.1 Solid Model

The solid part of the manifold has the following properties (Figure 2):

![Figure 2: Manifold: Mesh of the solid domain](image)

The mesh consists of 15244 8-node brick elements.
<table>
<thead>
<tr>
<th>Conductivity</th>
<th>55.0 $\frac{W}{m\cdot K}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Film temperature (outside)</td>
<td>300.0 $K$</td>
</tr>
<tr>
<td>Heat coefficient (outside)</td>
<td>50.0 $\frac{W}{m^2\cdot K}$</td>
</tr>
</tbody>
</table>

Table 1: Manifold: Boundary conditions for solid model

**Abaqus**

In Abaqus DC3D8 elements are used.

**PERMAS**

In PERMAS HEXE8 solid elements and CONA4 convection elements are used.

...  

### 4.2.2 Fluid Model

The fluid domain comprises the inner part of the pipe (Figure 3).

![Figure 3: Manifold: Mesh of the fluid domain](image)
<table>
<thead>
<tr>
<th>Density</th>
<th>$1.225 \frac{kg}{m^3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viscosity</td>
<td>$1.7894E - 05 \frac{kg}{m \cdot s}$</td>
</tr>
<tr>
<td>Conductivity</td>
<td>$0.0242 \frac{W}{m \cdot K}$</td>
</tr>
<tr>
<td>Specific heat</td>
<td>$1006.43 \frac{J}{kg \cdot K}$</td>
</tr>
<tr>
<td>Inlet Velocity</td>
<td>$6.0 \frac{m}{s}$</td>
</tr>
<tr>
<td>Temperature</td>
<td>$600.0 K$</td>
</tr>
<tr>
<td>Turbulent Intensity</td>
<td>$0.03$</td>
</tr>
<tr>
<td>Turbulent Length</td>
<td>$0.003 m$</td>
</tr>
<tr>
<td>Outlet Relative pressure</td>
<td>$0 Pa$</td>
</tr>
<tr>
<td>Turbulent Intensity</td>
<td>$0.03$</td>
</tr>
<tr>
<td>Turbulent Length</td>
<td>$0.003 m$</td>
</tr>
</tbody>
</table>

Table 2: Manifold: Boundary conditions for fluid model

The fluid mesh consists of 102236 hexahedral elements.

**FLUENT**
A standard $k - \epsilon$ model with enhanced wall treatment is applied.

**STAR-CD**
A low-Reynolds $k - \epsilon$ model with hybrid wall is applied.

### 4.2.3 Uncoupled Flow Simulation

Prior to a coupled simulation, it is advisable to carry out an uncoupled flow simulation with simplified boundary conditions. The temperature of the later coupled interface of fluid and solid is set to constant $600 K$.

**FLUENT**
Start FLUENT and read the case file "FLUENT/exhaust_manifold.cas". Carry out a flow simulation with 200 iterations and save the result in "FLUENT/exhaust_manifold.dat".
4.3 Models Step

**STAR-CD**

Start STAR-CD and read the model file "STARCD/exhaust_manifold.mdl". Carry out a flow simulation with 200 iterations. The result is saved in "STARCD/exhaust_manifold.pst/.ccmp" depending on the STAR-CD version.

...  

### 4.2.4 Prepare Models for Coupled Simulation

After the uncoupled simulations have been completed, the input files need modifications to accomplish the coupled simulation using the uncoupled results for a restart.

**FLUENT**

No modifications have to be done, as we later carry out a GUI based coupled simulation.

**STAR-CD**

1. Start pro-STAR and resume the model file "exhaust_manifold.mdl".

2. In the STAR GUIde go to Analysis Preparation/Running→Set Run Time Controls.
   Set Number of iterations to “100”.

3. Change to Analysis Preparation/Running→Analysis (Re)Start.
   Set Restart File Option to Standard Restart for STAR-CD 3.26 or Initial Field Restart with options Restart (New Boundary Values) and Continue for STAR-CD 4.0x.

4. Choose the uncoupled result file "exhaust_manifold.pst" or "exhaust_manifold.ccmp" as Restart File, depending on the STAR-CD version.

5. Save the model "exhaust_manifold.mdl" and exit pro-STAR.

6. (Only STAR-CD 3.26) Create a subdirectory "STARCD/ufile" by renaming the provided ufile directory "ufile_326".

7. (Only STAR-CD 3.26) Check if the user subroutine "bcdefw.f" is located in "STARCD/ufile". Otherwise copy subroutine "bcdefw.f" from 
   "<MpCCI home>/tutorial/ExhaustManifold/STARCD/ufile_326".

*(The routine "bcdefw.f" is needed for retrieving the received wall temperature from the scalar array for STAR-CD 3.26. All other subroutines which are needed for a coupled simulation will be copied automatically by MpCCI. For STAR-CD 4.0x the data transfer and management is handled via plug-ins and no user subroutines are needed for a coupled simulation.)*

...
4.3 Models Step

Start the MpCCI GUI by running the command `mpcci gui` in the directory "ExhaustManifold". In the first step, the Models Step, the input files of the coupled codes are defined and scanned.

**Abaqus**

- Select *Abaqus* as the first code to couple. Select any *Abaqus 6.7* release.
- Choose the input file "Abaqus/exhaust_manifold.inp" and the SI unit system (which is default).
- Please press the **Start Scanner** button and a green check mark should appear at the top as shown on the left.

**PERMAS**

- Select *PERMAS* as the first code to couple. Select any *PERMAS 11* release.
- Choose the input file "PERMAS/exhaust_manifold.dat" and the SI unit system (which is default).
- Please press the **Start Scanner** button and a green check mark should appear at the top as shown on the left.
4.4 Coupling Step

Press the Next button at the bottom of the Models Step to get to the Coupling Step.

In this example the coupling region corresponds to the inner surface of the pipe. MpCCI treats this region as a “Face” because it represents a 2D structure.

Select Face (2D) tab, and then configure the corresponding coupling components as depicted in Figure 4, using exemplary PERMAS and STAR-CD, and as described in the following.
Abaqus

Double-click or drag down the coupling component “ASSEMBLY_MANIFOLD.Inner_SURFACE”, which characterizes the inner surface of the pipe. The component will appear in the coupled field of the solver.

PERMAS

Double-click or drag down the coupling component “CCI.1”, which characterizes the inner surface of the pipe. The component will appear in the coupled field of the solver.

FLUENT

Double-click or drag down the coupling component “wall”, which characterizes the inner surface of the pipe. The component will appear in the coupled field of the solver.

STAR-CD

Double-click or drag down the coupling component “wall”, which characterizes the inner surface of the pipe. The component will appear in the coupled field of the solver.

In the quantities field choose as coupling type “steady state surface heat transfer” and activate the quantities by pressing Set as marked red in Figure 4. The quantities “FilmTemp,” “WallHTCoeff” and “WallTemp” are activated. MpCCI will set the sender of the corresponding quantities.

No further settings are required, proceed to the Edit Step by pressing Next>.

4.5 Edit Step

Select TraceFile and replace "tracefile.ccv" by "exhaust_manifold.ccv" as depicted in Figure 5. Proceed to the Go Step by pressing Next>.
Figure 4: Manifold: Coupling of “FilmTemp”, “WallHTCoeff” and “WallTemp”, here as an example for PERMAS and STAR-CD
Figure 5: Manifold: Rename tracefile written by MpCCI during simulation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>exhaust_manifold.cou</td>
<td>Specifies the name of the MpCCI tracefile to be created. If no name or an empty name is specified, no tracefile will be written. Note: The MpCCI control process is needed for writing a tracefile (see server settings in Go-Step). (Default: &quot;tracefile.cou&quot;)</td>
</tr>
<tr>
<td>CloseAfterWriting</td>
<td></td>
<td>MpCCI will close the tracefile each time after writing data into it. (Default: on)</td>
</tr>
<tr>
<td>ImplicitCouplingSteps</td>
<td></td>
<td>Tells MpCCI whether or not to generate a new coupling step in the tracefile during a call of &quot;CCL::CheckConvergence&quot;. (Default: on)</td>
</tr>
<tr>
<td>TraceCommValues</td>
<td></td>
<td>In addition to the geometry data, MpCCI will also write communication values into the tracefile. (Default: on)</td>
</tr>
<tr>
<td>TraceMeshValues</td>
<td></td>
<td>MpCCI will write communication values of coupling type &quot;mesh&quot; into the tracefile (unless &quot;trace_comm_values&quot; is switched off). (Default: on)</td>
</tr>
<tr>
<td>TraceMeshChangeValues</td>
<td></td>
<td>MpCCI will write communication values of coupling type &quot;mesh_change&quot; into the tracefile (unless &quot;trace_comm_values&quot; is switched off). (Default: on)</td>
</tr>
</tbody>
</table>
In the Go Step the MpCCI server and the coupled codes for the start-up are configured. The choice of start-up options depends on the selected codes to couple. First start by configuring the server settings:

- Accept default options as depicted.
- The MpCCI server will write the tracefile which was defined in the Edit Step by activating Start additional control process.
- All output files of the server will have the prefix "mpccirun".
Deactivate Subcycling as there is no subcycling of Abaqus included.

MpCCI will run the simulation not on the original model, but rather will use a copy with the prefix “abaqus.run”.

When starting the simulation Abaqus will receive data from the CFD code and accomplish a steady state heat transfer analysis. After the analysis data is exchanged with the partner code and another analysis is started.

Activate Configure coupling algorithm

Select Gauss-Seidel as coupling algorithm.

The choice of Gauss-Seidel and the Initial quantities transfer of the partner code will determine the coupling scheme as depicted in Figure 5 (part VI) of › VI-10 PERMAS ◄.

Set No. of exchange iterations to 10.
• Set Initial quantities transfer to send.
• Deactivate Auto hook functions
• Deactivate Auto set MDM zones.
• The option Auto set BC's will hook the UDM for the coupling boundary “wall”.
• Activating Auto read case data will automatically cause FLUENT to read the case and data file of the uncoupled simulation.
STAR-CD 3.26

- Set the Initial quantities transfer to send.
- As startup procedure select Prepare case and start to accomplish the necessary modifications in the STAR-CD model for a coupled simulation.
- The activated option Rebuild shared library will cause a compilation of the user subroutines, which are copied to the ufile directory by MpCCI.
- Disable Use the MpCCI plugin library (Star V4 only) to deactivate MpCCI plug-in, which is not available for STAR-CD 3.26.
- Disable Use a data filter subroutine (Star V4 only) which is not available for STAR-CD 3.26.
- As MpCCI will build an new model file, save the original model file by pushing the Backup model file button.
- Set Steady state: No. of iterations without coupling to “10”. Between exchanges of the data, STAR-CD will carry out ten iteration steps (subcycling).

⚠️ Make sure, that you have already copied "bcdefw.f" to the ufile directory in step 4.2.4 Prepare Models for Coupled Simulation ⬤
4.7 Running the Computation

Save the MpCCI project file with name "exhaust_manifold.csp" via the MpCCI GUI menu 
File→Save Project As.
Press the three Start buttons in the Go Step and the simulation codes should start. Some codes require 
additional actions:

**FLUENT**

1. Open the MpCCI panel over Solve→MpCCI Controls....
2. Press Initialize to start initial handshaking.
3. Press Send to send data to partner code.
4. Open Solve→Iterate and start ten iterations. As we have not auto-hooked functions for coupling 
FLUENT will iterate without sending or receiving any data. The functions On Demand... on the 
MpCCI panel will hook the necessary functions for an exchange.
5. Press **Exchange** to send data to partner code.

6. Accomplish another ten iterations.

7. Repeat step 5. and 6. until you have reached ten exchanges.

In order to survey the convergence of the simulation, **FLUENT** will display the residual and the integral total surface heat flux on the coupling surface. To avoid a repeated manual execution of the iterative process, a journal script might be recorded and executed.

### STAR-CD

In order to survey the convergence of the simulation, **STAR-CD** will write the integral total surface heat flux through the coupling surface on "exhaust_manifold.erd". **STAR-CD** will run for 100 iterations.

... 

### 4.8 Post-processing

Use the following files for the evaluation of the results.

#### MpCCI

**MpCCI** tracefile "exhaust_manifold.ccv", which can be opened by command `mpcci vis exhaust_manifold.ccv`, with which one can trace the exchange process of the coupled surface.

#### Abaqus

**Abaqus** result file "exhaust_manifold.odb" and temperatures for node set nodes_evaluate on "exhaust_manifold.dat".

#### MEDINA

**MEDINA** result file "exhaust_manifold.bof" and temperatures for node set nodes_evaluate on "exhaust_manifold.post".

#### FLUENT

**FLUENT** result file "exhaust_manifold.dat".

#### STAR-CD

**STAR-CD** result file "exhaust_manifold.pst" or "exhaust_manifold.ccmp".

...
Figure 6: Manifold: Temperature distribution on coupled surface
5 Busbar System

5.1 Problem Description

Figure 1: Busbar: Geometry [m].

Topics of this Tutorial
- Magneto-Thermal
- 3D-model
- Volume coupling
- Several coupling regions
- Fluid solver iterations without coupling

Simulation Codes
- Fluid Mechanics: FLUENT 6.3.26 / STAR-CD 4.04 or higher
- Electromagnetism: FLUX 10.2 / ANSYS 100 / ANSYS 110
Description

The example is taken from an article by Lyttle et al. [2006]. The task is to solve the electrothermal coupled problem of a busbar system carrying a three-phase current where the current leads to power losses due to the finite resistivity of the conductor. The busbars are made of copper, the RMS current level is 1600 A at a frequency of 60 Hz. Because of the power losses the conductor is heated up as well as the surrounding air leading to free convection around the busbars. To model the convection process the Boussinesq approximation for the density of air is used. A linear rise of resistivity dependent on the temperature of the conductor is also modeled.

The electromagnetic calculation of power-losses is considering:

- eddy currents,
- skin effect,
- proximity effect,
- temperature dependent rise of resistivity.

The thermal calculation is considering:

- heat conduction (fluid and solid),
- heat convection (including buoyancy effects),
- laminar flow.

5.2 Model Preparation

The simulation couples a electromagnetic model with a fluid model. The files you need for the simulation are included in the MpCCI distribution. Create a new directory and copy the subdirectories from "<MpCCI_home>/tutorial/Busbar" corresponding to the simulation codes you want to use.

5.2.1 Fluid Model

Both the STAR-CD and FLUENT models are based on the same grid with boundary conditions as similar as possible. On the front and back side of the fluid cabinet the conductors enter the fluid domain. Here are adiabatic wall boundary conditions set. On the left and right side of the cabinet a wall boundary condition with fixed temperature of 293.15 K is defined. On the bottom and top of the cabinet air can enter and leave the domain on pressure boundary conditions.

FLUENT

No modifications have to be done, as we later carry out a GUI based coupled simulation.
STAR-CD

No modifications have to be done, as in STAR-CD 4.04 and higher MpCCI is treated as plug-in and no user subroutines are needed.

5.2.2 Electromagnetic Model

A busbar conductor corresponds to three rectangles each with a dimension of $12.7 \text{ mm} \times 76.2 \text{ mm} \times 250 \text{ mm}$. The magnetic property of copper is linear isotropic with a relative permeability of 1. The temperature dependent electrical resistivity is calculated in this way:

$$\varrho(T) = \varrho_0 (1 + \alpha(T - T_{\text{ref}}))$$

At reference temperature $T_{\text{ref}} = 300 \text{ K}$ the value of resistivity is $\varrho_0 = 1.7241 \times 10^{-8} \ \Omega\text{m}$. The temperature coefficient is $\alpha = 0.004 \ \text{K}^{-1}$. The calculation of the current flow and magnetic field is done in frequency domain.

FLUX

All necessary model information is prepared, no further action is needed. An electric circuit is defined to set the current loads for the finite element solution. Beside the model file an pyFlux script file "busbar.py" is used to run the co-simulation. For coupling these commands are used additionally to the pyFlux commands controlling the solver:

- `mpcci.initialize()`: Initialize the connection to MpCCI.
- `mpcci.receive(1)`: Receive data from the partner code and wait until the data exchange is done.
- `mpcci.send(1)`: Send data to the partner code.
- `mpcci.exit()`: Exit MpCCI.

ANSYS

The "ANSYS/busbar.db" contains all finite element information. Beside the model file an ANSYS input file "ANSYS/startjob.ans" is used to define all boundary conditions, loads and to control the coupling process. At the front and backside of the conductors the voltage degree of freedom will be coupled and on one side for every conductor the voltage degree of freedom is set to 0. At the front side of the conductors a nodal load amps is defined separately for every conductor with real and imaginary part depending on the phasing. The elements are grouped into element components named ”phase-a”, ”phase-b” and ”phase-c” to make it possible for the MpCCI ANSYS adapter to detect the elements. For coupling these commands are used:

- `~mpcci, init, 3D`: Initialize the connection to MpCCI.
• `~mpcci, send, wait`: Send data to the partner code and wait until the data exchange is done.

• `~mpcci, receive, wait`: Receive data from the partner code and wait until the data exchange is done.

Furthermore after the frequency domain solution, the effective power loss density is provided via ANSYS command `powerh` which generates an element table named `plossd`. This element table is just copied to an element table named `mpcci_00` using this command: `[smult, mpcci_00, plossd, , 1]`. Because the MpCCI adapter can read out of element tables with fulfilling the naming convention `mpcci_<index>` . The `<index>` has to be set in the MpCCI GUI.

## 5.3 Models Step

Start the MpCCI GUI by running the command `mpcci gui` in the directory "Busbar".

Select FLUX as the first code to couple and FLUX release 10.2.
Select the input file "FLUX/Busbar.FLU/PROBLEM.FLU.PFL" and press the [Start Scanner] button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.
Select ANSYS as the first code to couple and ANSYS release 100 or 110. You can select any ANSYS product which can be used for electromagnetic analysis, e.g. emag. Choose the database file "ANSYS/busbar.db". Please press the [Start Scanner] button and a green check mark should appear at the top as shown on the left.

Select FLUENT as the second code to couple. The model is three-dimensional, thus select the FLUENT version 3ddp. The FLUENT release should be 6.3.26. Select the input file "FLUENT/busbar.cas" and press the [Start Scanner] button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.
Select STAR-CD as the second code to couple. The STAR-CD release should be 4.04 or higher. Select the input file "STARCD/busbar.mdl" and press the [Start Scanner] button. Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.
5.4 Coupling Step

Press the Next button at the bottom of the Models Step to get to the Coupling Step.

In this example the coupling region corresponds to the solid conductors. MpCCI treats this region as a “Volume” because it represents a 3D structure. There are 3 conductors to couple and for each conductor from the EMAG and CFD code a couple region will be created. This will increase the performance of the neighborhood search.

Overview of the components to couple for each code:

1. **Volume (3D)**
2. **Coupled**
3. **Temperature (FLUENT)**
1. Select the Volume 3D tab.

2. Select the coupling components for the first coupling region Volume_1 by double-clicking them or dragging them into the Coupled boxes.

3. Select the quantity JouleHeat and the EMAG Code as sender. Select the quantity Temperature and the CFD Code as sender.

For the quantity Temperature:
- Select the receive method Direct.

For the quantity JouleHeat:
• Select the location elem.

• Select the method ETAB.

• Select the index 0.

For the quantity Temperature:

• Select the location node.

---

STA-R-CD

---

For the quantity JouleHeat:

• Select index 1

• ...
1. For the next coupled region click on the **Add** button in the **Regions** part. This will insert a new region **Volume_2** to couple.

2. Select the coupling components by double-clicking them or dragging them into the **Coupled** boxes.

   **FLUX**
   Select the coupling component “PHASE-B”.

   **ANSYS**
   Select the coupling component “phase-b”.

   **FLUENT**
   Select the coupling component “phase_b-solid”.


Select the coupling component “phase-b”.

3. The same quantities setting as made for Volume_1 will be used. Therefore use Copy from region from the Predefined Sets and click on the Set button to apply the setting from the selected Volume_1.

4. Repeat the step 1 to 3 for the last conductor.

Proceed to the Edit Step by pressing Next >.

5.5 Edit Step

No changes are required in the Edit Step.
5.6 Go Step

In the server panel, no changes are necessary. The code panels are described for each code below.
1. Set Initial quantities transfer to receive.

2. Select the Memory configuration and increase the Numerical memory MB to 1600.

3. Select the User pyFlux script "busbar.py" by pressing the **Browse** button. The "busbar.py" file is located under the same root directory as the "BUSBAR.FLU" model file.

4. Activate the Auto read problem file option which is the default.

5. Activate the Auto load MpCCI lib option which is the default.

1. Set Initial quantities transfer to receive.

2. Keep -b as gui option to start ANSYS in batch mode. Additional command line options are not required.

3. Select the APDL input script "startjob.ans" by pressing the **Browse** button.
1. Set the Initial quantities transfer to exchange.

2. Select the Optional journal Files "busbar.jou" by pressing the Browse button.

3. Deselect the Auto hook functions.

4. Deselect the Auto set MDM zones.
1. Set the Initial quantities transfer to exchange.

2. As startup procedure select Prepare case and start to accomplish the necessary modifications in the STAR-CD model for a coupled simulation.

3. Activate Use the MpCCI plugin library (Star V4 only) to enable the MpCCI plug-in. Deactivate Use a data filter subroutine (Star V4 only).

4. As MpCCI will build a new model file, save the original model file by pushing the Backup model file button.

5. Activate subcycling for STAR-CD by defining 20 iterations without coupling.

5.7 Running the Computation

Press the Start button of the server and save the project as "busbar.csp". Now start the codes by pressing their Start buttons and the simulation codes should start. Some codes require additional actions:

**FLUX**

The FLUX reads the pyFlux script file "busbar.py" and starts the computation.

**ANSYS**

ANSYS reads the APDL script file "startjob.ans" and starts the computation.

**FLUENT**

FLUENT reads the journal file "busbar.jou" and automatically starts the computation by calling the function "MpCCI-solve nbExchange nbIterations" respectively with the values ten and twenty. The coupling scheme used for FLUENT is:

- a data exchange before each solving cycle.
- "nbIterations" iterations for a solving cycle.
The \texttt{STAR-CD} job will be prepared and started. The data transfer is handled by the \texttt{MpCCI} plug-in calls in \texttt{STAR-CD}.

### 5.8 Discussion of Results

In this busbar simulation it is seen that the coupling takes ten steps to reach a maximum temperature. Exemplarily the temperature distribution in the middle plane of the cabinet from the \texttt{FLUENT} solution is given in the next picture.

![Figure 2: FLUENT results: temperature distribution [K]](image)
6 Pipe Nozzle

6.1 Problem Description

Topics of this Tutorial

- Fluid-Structure Interaction (FSI)
- Axisymmetric model
- Steady state one-way force mapping
- Use of UDF-functions in FLUENT
- Models with different unit systems.

Simulation Codes

- Fluid Mechanics: FLUENT 6.3.26
- Solid Mechanics: Abaqus 6.6 or 6.7

6.2 Model Preparation

The model is axisymmetric, thus only a section of the model is created and meshed. The axis of revolution is different for different simulation codes. MpCCI uses the vertical $y$-axis. For simulation codes which use different axes, the data is converted in the code adapter.

The solid material is a simple linear elastic material with elastic modulus $E = 2000$ MPa and $\nu = 0.3$. The fluid is modeled as ideal gas with properties of air.

The fluid is streaming into the nozzle at the pressure inlet as shown in Figure 1. During the simulation the pressure is slowly increased with a user-defined function up to a value of $p_i = 1$ MPa.
The simulation is a steady state simulation, the surface forces are only transferred once from the fluid model to the solid model, where they are applied as boundary conditions to compute the stress distribution.

### 6.2.1 Fluid Model

**FLUENT**

Fluent uses the horizontal $x$-axis as axis of revolution, thus the model is created as shown in Figure 1. A user-defined function is used to slowly increase the inlet pressure in order to ensure convergence of the FLUENT solver.

Before starting the computation, the "libudf" library for FLUENT must be built:
• Change to the "FLUENT" directory, start fluent 2d and read the casefile "nozzle.cas". FLUENT will print an error message `open_udf_library: No such file or directory`, which indicates that the UDF library is still missing. The error message should not appear if the library was built and is present in the corresponding "libudf" directory.

• Select `Define→User-Defined→Functions→Compiled...` to open the Compiled UDFs panel shown on the left.

• Press the left `Add...` button and select the file "inpressure.c".

• Press `Build` to compile the "libudf" library.

• Finally exit the panel with `Load` to load the library.

• Quit FLUENT, you do not need to save changes.
6.2.2 Solid Model

Abaqus uses the vertical $y$-axis as axis of revolution in axisymmetric models. Therefore the geometry must be flipped by swapping $x$- and $y$-coordinates, which yields the configuration shown on the left.
Only one iteration step is carried through.

6.3 Models Step

⚠️ Before starting the coupled simulation, ensure that you have built the udf library for FLUENT as described above.

Start the MpCCI GUI by entering the command `mpcci gui`. In the first step, the Models Step, the input files of the coupled codes are defined and scanned.
- Select FLUENT as the first code to couple.
- The model is axisymmetric, i.e. a 2d model. Select the FLUENT version 2d.
- The FLUENT release should be set to 6.3.26 or higher.
- Select the input file "FLUENT/nozzle.cas" and press the [Start Scanner] button.
- Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.

- Select Abaqus as the second code to couple.
- Select any Abaqus 6.6 or Abaqus 6.7 release.
- Choose the input file "Abaqus/nozzle.inp".
- Set the unit system to SI-mm-t-s, because the Abaqus model was created using these units.
- Please press the [Start Scanner] button and a green check mark should appear at the top as shown on the left.
- Press Next > to proceed.
In the Coupling Step, select the Face (2D) tab.

**FLUENT**

Select the coupling component “pipesurface” by dragging it to the list of Coupled components.

**Abaqus**

Select the coupling component “ASSEMBLY_NOZZLE-1_OUTSIDE”.

... 

Choose the quantity **RelWallForce** with FLUENT as sender. Press [Next >] to proceed.
6.5 Edit Step

No changes are required in the Edit Step, proceed to the Go Step by pressing Next.

6.6 Go Step

In the server panel, no changes are necessary. The code panels are described for each code below.
- Set the Initial quantities transfer to send.
- Deselect Auto hook functions. This is necessary to keep FLUENT from sending data in each iteration.
6.7 Running the Computation

- Start the MpCCI server by pressing the left Start button and save the project as "nozzle.csp". Then start FLUENT and Abaqus by pressing the other Start buttons.

- In the FLUENT window, select Solve→Initialize→Initialize... and press the Init button to initialize the FLUENT solution. Then close the initialization panel by pressing Close.

- Open the FLUENT Iterate panel by selecting Solve→Iterate... Set the number of iterations to 120 and press the Iterate button. You should see a plot of the FLUENT results during the iterations.

- Close the Iterate panel and open the MpCCI panel which you find under Solve→MpCCI Control....

- In the MpCCI panel, press the Initialize button in the middle of the panel, which will start the neighborhood search in both codes – you should notice according output in the code windows.

- To send the surface forces from FLUENT to the partner code, press the Send button once. The partner code will now compute the stresses and quit. Close any appearing message windows.

- Press the Finalize button in the MpCCI panel of FLUENT, which disconnects FLUENT from MpCCI and exit FLUENT.

- Quit MpCCI, the simulation is finished. Have a look at the stress results in the structural code and compare them to those shown below.
6.8 Discussion of Results

The stress distribution in the Abaqus model is shown on the left. To verify the results, the Abaqus model was alternatively loaded with the inlet pressure at the interior surface, while no load was applied at the outside surface. This yielded almost identical results to the coupled simulation.
7 Cube in a Duct Heater

7.1 Problem Description

The example case presented in this tutorial describes the forced convection. For this purpose a cube is inserted in an rectangular duct, which is heated up by a heater at the bottom of the duct. Additionally the cube is cooled by air passing through the rectangular duct.

Figure 1: Duct Heater: Geometry [m] and boundary conditions section view

Topics of this Tutorial
- Radiative Heat Transfer
- Steady State
- Forced convection
- 3D-model

Simulation Codes
- Radiation: RadTherm 9.0.1 or higher
- Fluid Mechanics: FLUENT 6.3.26, STAR-CD 3.26 or 4.0x

Description
The example case presented in this tutorial describes the forced convection. For this purpose a cube is inserted in an rectangular duct, which is heated up by a heater at the bottom of the duct. Additionally the cube is cooled by air passing through the rectangular duct.
7.2 Model Preparation

The simulation couples a radiation model with a fluid mechanics model. The files which you need for the simulation are included in the MpCCI distribution. Create a new directory "DuctHeater" and copy the subdirectories from "<MpCCI_home>/tutorial/DuctHeater" which correspond to the simulation codes you want to use.

7.2.1 Radiation Model

RadTherm

All values at standards settings use temperature calculated except for the heater. The heater is set on a constant surface temperature identical (T=673K) to the CFD settings. The maximum iterations is set to 499.

7.2.2 Fluid Model

The fluid domain comprises the inner part of the duct heater (Table 1.)

The setting used for fluid model are in the following Table 2

<table>
<thead>
<tr>
<th>FLUENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A standard $k - \epsilon$ model with enhanced wall treatment is applied.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STAR-CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A $k - \epsilon$ high Reynolds turbulence models is applied.</td>
</tr>
</tbody>
</table>

1. Start pro-STAR and resume the model file "star_model.mdl".

2. In the STAR GUIde go to Analysis Preparation/Running→Set Run Time Controls. Set Number of iterations to “500”.

3. Save the model "star_model.mdl" and exit pro-STAR.

<table>
<thead>
<tr>
<th>STAR-CD 3.26</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Create a subdirectory &quot;STARCD/ufile&quot; by renaming the provided ufile directory &quot;ufile_326&quot;.</td>
</tr>
</tbody>
</table>

2. Check if the user subroutine "bcdefw.f" is located in "STARCD/ufile". Otherwise copy subroutine "bcdefw.f" from "<MpCCI_home>/tutorial/DuctHeater/STARCD/ufile_326". |
<table>
<thead>
<tr>
<th>Density</th>
<th>1.205 $\frac{kg}{m^3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conductivity</td>
<td>0.0237 $\frac{W}{m\cdot K}$</td>
</tr>
<tr>
<td>Specific heat</td>
<td>1006 $\frac{J}{kg\cdot K}$</td>
</tr>
<tr>
<td>Inlet</td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td>0.5 $\frac{m}{s}$</td>
</tr>
<tr>
<td>Temperature</td>
<td>293.0 K</td>
</tr>
<tr>
<td>Turbulent Intensity</td>
<td>0.1</td>
</tr>
<tr>
<td>Turbulent Length</td>
<td>0.02 m</td>
</tr>
<tr>
<td>Outlet</td>
<td>standard setting</td>
</tr>
<tr>
<td>Cube</td>
<td></td>
</tr>
<tr>
<td>wall heat</td>
<td>fixed</td>
</tr>
<tr>
<td>Temperature</td>
<td>293.0 K</td>
</tr>
<tr>
<td>Duct</td>
<td></td>
</tr>
<tr>
<td>wall heat</td>
<td>fixed</td>
</tr>
<tr>
<td>Temperature</td>
<td>293.0 K</td>
</tr>
<tr>
<td>Heater</td>
<td></td>
</tr>
<tr>
<td>wall heat</td>
<td>fixed</td>
</tr>
<tr>
<td>Temperature</td>
<td>673.0 K</td>
</tr>
</tbody>
</table>

Table 1: Duct Heater: Boundary conditions for fluid model

<table>
<thead>
<tr>
<th>Time Domain</th>
<th>Steady state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gravity</td>
<td>$-9.81 \frac{m}{s^2}$</td>
</tr>
<tr>
<td>Thermal option</td>
<td>off</td>
</tr>
<tr>
<td>Buoyancy for AIR</td>
<td>On</td>
</tr>
</tbody>
</table>

Table 2: Duct Heater: Parameter setting for fluid model

The routine "bcdefw.f" is needed for retrieving the received wall temperature from the scalar array for STAR-CD 3.26. All other subroutines which are needed for a coupled simulation will be copied automatically by MpCCI.
STAR-CD 4.0x

For STAR-CD 4.0x the data transfer and management is handled via plug-ins and no user subroutines are needed for a coupled simulation.

7.3 Models Step

Start the MpCCI GUI by running the command `mpcci gui`. In the first step, the Models Step, the input files of the coupled codes are defined and scanned.

- Select RadTherm as the first code to couple.
- Select RadTherm 9.0.1 release or higher.
- Choose the input file "Radtherm/radtherm_model.tdf".
- Please press the Start Scanner button and a green check mark should appear at the top as shown on the left.
FLUENT

- Select FLUENT as the second code to couple.
- The FLUENT release should be set to 6.3.26.
- The FLUENT version should be set to 3ddp.
- Select "FLUENT/fluent_model.mdl" as input file and press the Start Scanner button.
- Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.

STAR-CD

- Select STAR-CD as the second code to couple.
- The STAR-CD release should be set to 3.26 or higher.
- Select "STARCD/star_model.mdl" as input file and press the Start Scanner button.
- Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.

7.4 Coupling Step

Press the Next button at the bottom of the Models Step to get to the Coupling Step.

In this example the coupling region corresponds to the surface of the duct and rectangle. MpCCI treats this region as a “Face” because it represents a 2D structure.

Select Face (2D) tab, and then configure the corresponding coupling components as depicted in Figure 2 and described in the following.
Figure 2: Duct Heater: Coupling of “FilmTemp”, “WallHTCoeff” and “WallTemp”

**RadTherm**

Double-click or drag down the coupling component “duct_leftside”, “duct_topside”, “duct_rightside”, “duct_bottomside”, “cube_backside”, “cube_leftside”, “cube_rightside” “cube_topside”, “cube_bottomside”, “cube_frontside” which characterizes the surface of the duct and the rectangle. The component will appear in the coupled field of the solver.

**FLUENT**

Double-click or drag down the coupling component “duct_leftside”, “duct_topside”, “duct_rightside”, “duct_bottomside”, “cube_backside”, “cube_leftside”, “cube_rightside” “cube_topside”, “cube_bottomside”, “cube_frontside” which characterizes the surface of the duct and the rectangle. The component will appear in the coupled field of the solver.

**STAR-CD**

Double-click or drag down the coupling component “duct_leftside”, “duct_topside”, “duct_rightside”, “duct_bottomside”, “cube_backside”, “cube_leftside”, “cube_rightside” “cube_topside”, “cube_bottomside”, “cube_frontside” which characterizes the surface of the duct and the rectangle. The component will appear in the coupled field of the solver.
In the quantities field choose as coupling type “steady state radiative heat transfer” and activate the quantities by pressing [Set] as marked red in Figure 2. The quantities “FilmTemp,” “WallHTCoeff” and “WallTemp” are activated. MpCCI will set the sender of the corresponding quantities.

No further settings are required, proceed to the Edit Step by pressing [Next].

7.5 Edit Step

No changes are required in the Edit Step, proceed to the Go Step by pressing [Next].

7.6 Go Step

In the server panel, no changes are necessary. The code panels are described for each code.
- The Initial quantities transfer is set to receive.
- All other options should not be changed.
- Set the Initial quantities transfer to send.
- Use the default settings.
• Set the Initial quantities transfer to send.

• As startup procedure select Prepare case and start to accomplish the necessary modifications in the STAR-CD model for a coupled simulation.

• The activated option Rebuild shared library will cause a compilation of the user subroutines, which are copied to the ufile directory by MpCCI.

• Disable Use MpCCI plugin for V4 to deactivate MpCCI plug-in, which is not available for STAR-CD 3.26.

• As MpCCI will build a new model file, save the original model file by pushing the Backup model file button.

• Enable the Double precision mode.

⚠️ Make sure, that you have already copied "bcdefw.f" to the ufile directory in step ▷7.2 Model Preparation <
STAR-CD 4.0x

- Set the Initial quantities transfer to skip.
- As startup procedure select Prepare case and start to accomplish the necessary modifications in the STAR-CD model for a coupled simulation.
- Enable Use MpCCI plugin for V4 to activate MpCCI plug-in in STAR-CD for data transfer.
- Deactivate option Rebuild shared library as no user subroutines for data transfer are needed (plug-in version).
- As MpCCI will build an new model file, save the original model file by pushing the Backup model file button.
- Enable the Double precision mode.

7.7 Running the Computation

7.7.1 Starting the Simulation

Save the MpCCI project file with name "duct.heater.csp" over the MpCCI GUI menu File→Save Project As.
Press the three Start buttons in the Go Step and the simulation codes should start. Some codes require additional actions:

RadTherm

Before starting the RadTherm computation the model has to be correctly setup to use the MpCCI Adapter by using the RadTherm GUI.(› VI-11.2.5.1 Hooks Functions ◄)
You have to select the Analyse→Params section in the RadTherm GUI in order to access the hooks functions setup Figure 3 (part VI). By clicking on the button HookFunctions... the setup windows will appear. You have to setup these functions for the following available RadTherm hooks:

Solution Start  Select the routine radthermmmpcci::TaiMpCCI_solution_start().
Iteration Start  Select the routine radthermmmpcci::TaiMpCCI_iteration_start().
VII Tutorial

7.7 Running the Computation

Figure 3: RadTherm Hooks Functions Setup.

**IterationEnd** Select the routine `radthermmpci::TaiMpCCI_iteration_end()`.

**Solution End** Select the routine `radthermmpci::TaiMpCCI_solution_end()`.

After having hooked the functions press the **OK** button to exit the dialog. You may check that the number of iterations is correctly set to 499. Then you may click on the **Run** button to start the computation. RadTherm will ask to save the model.

⚠️ The related hook information is saved and depends on your local MpCCI installation.

### FLUENT

The FLUENT Graphical interface is started.

1. Open **Solve**→**Initialize**→**Initialize** and press the button **init**.

2. Open **Solve**→**Iterate** and start 500 iterations.

During the simulation, FLUENT will display the residuals. At the end of the simulation, save the data file by selecting **File**→**write**→**data**.
The STAR-CD calculation is started in batch mode.

7.7.2 End of the Simulation

RadTherm
After performing 499 iterations RadTherm will ask to do some additional iterations. At this step you may accept the solution and the thermal computation is terminated. You may save the results for a later post-processing and exit the RadTherm GUI to close the coupled simulation.

7.8 Discussion of Results

RadTherm
Temperature results may be visualized in RadTherm GUI. (Figure 4)

FLUENT
FLUENT result file "fluent_model.dat" may be post processed by FLUENT.

STAR-CD
STAR-CD result file "star_model.pst" or "star_model.ccmp" may be post processed with pro-STAR.

By using the MpCCI Visualizer you may visualize the coupled values from the "tracefile.ccv". You may open the tools from the MpCCI GUI Tools→Visualizer. This will open automatically the "tracefile.ccv" file.
Figure 4: Duct Heater: Temperature section plot
Figure 5: Duct Heater: FLUENT wall temperature contour plot
Figure 6: Duct Heater: FLUENT total temperature contour plot
8 Y-Junction

8.1 Problem Description

Figure 1: Y-Junction: Fluid geometry and network boundaries

Topics of this Tutorial

- 1D network model
- 3D-model
- Steady State

Simulation Codes

- Network: Flowmaster 7.5.1, Flowmaster 7.6.0
- Fluid Mechanics: FLUENT 6.3.26, STAR-CD 4.04 or higher

8.2 Model Preparation

The simulation couples a network model with a fluid mechanics model. The files which you need for the simulation are included in the MpCCI distribution. Create a new directory "Y-Junction" and copy the
subdirectories from "<MpCCI_home>/tutorial/Y-Junction" which correspond to the simulation codes you want to use.

8.2.1 Network Model

The Flowmaster model (Figure 2) is composed of three networks. Each network will prescribe a flow (inlet/outlet) or pressure boundaries to the fluid model. The boundaries in a fluid model are represented within a Flowmaster network by a source.

1. Start Flowmaster then you need to log on the Flowmaster database.

2. Unpack the appropriate "Y_Junction.FMpack" pack file from the "<MpCCI_home>/tutorial/Y-Junction/Flowmaster " folder. Unpack the file directly under the root of your project tree.

3. Each source boundary has to activate its External Model Boundary property. For the following boundary IDs “13”, “15”, “26” you have to follow these instructions:

Figure 2: Y-Junction: Flowmaster network
a) Select the boundary component.

b) Edit the boundary with the **Edit** mouse menu context.

c) Select in the property list the External Model Boundary item.

d) Click on the Sub Form... button.

e) Change the property Boundary Active value to Yes.

f) Click on the button **Return**.

4. Run a standalone incompressible steady state (SS) Flowmaster analysis to completion.

5. The next step is to export the network boundary for the co-simulation.

- In the Network Views select the Data tab option.
- Click on the icon button that Generate a network data or analysis results report.
- Select the MpCCI Link File report type and click on the button **OK**.
• From the list of **Active Boundary Components**, select each item and click on the button \[\text{Add} >\].

• All active boundary components are in the **Selected Boundaries**.

• Select a boundary and provide a description by filling the text field **Add/Edit Boundary Description**.

• For the boundary “13”, enter the name “mass-in”.

• For the boundary “15”, enter the name “p-out1”.

• For the boundary “26”, enter the name “p-out2”.

• Click on the button \[\text{...}\] to select an output directory. (e.g. \\
\"<MpCCI\_home>/tutorial/Y-Junction/Flowmaster\"")

• In the **File Name** field provide the name "Y-Junction.fmlink".

• Click on the button \[\text{Create ASCII File...}\]

• The file has been created, the dialog can be closed.

• Exit the **Flowmaster** graphical interface.

⚠️ Do not renumber any Network used for co-simulation as component IDs are used to identify boundaries in **MpCCI** link file.

### 8.2.2 Fluid Model

The fluid model is a 3D model using the following settings:

• Turbulence model: k-epsilon RNG

• Time: Steady State

• Fluid: water

• Initial velocity 2 \(m.s^{-1}\)
Figure 3: Y-Junction: mesh
8.3 Models Step

Start the MpCCI GUI by running the command `mpcci gui`. In the first step, the Models Step, the input files of the coupled codes are defined and scanned.

**Flowmaster**

- Select **Flowmaster** as the first code to couple.
- Select **Flowmaster 7.5** release or higher.
- Choose the input file "Flowmaster/YJunction.fmlink".
- Please press the **Start Scanner** button and a green check mark should appear at the top as shown on the left.

**FLUENT**

- Select **FLUENT** as the second code to couple.
- The **FLUENT** release should be set to 6.3.26.
- Select "FLUENT/Y-junction.cas.gz" as input file and press the **Start Scanner** button.
- Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.
Select STAR-CD as the second code to couple.

The STAR-CD release should be set to 4.04 or higher.

Select "StarCD/Y-junction.mdl" as input file and press the [Start Scanner] button.

Finally a green check mark should appear, as shown at the left, which means everything is set up correctly.

Click on the button [Next >] to continue the co-simulation setup. A warning window will appear and complains about the model dimensions. You can continue by clicking on [Continue].
8.4 Coupling Step

In this example the coupling region corresponds to the surface of the inlet and outlets boundaries.

Select Face (2D) tab, and then configure the corresponding coupling components as depicted in Figure 4.

1. Select the Face 2D tab.

2. Select the coupling components for the first coupling region Face 1 by double-clicking them or dragging them into the Coupled boxes.

**Flowmaster**

Select the coupling component “mass-in”.

**FLUENT**

Select the coupling component “mass-in”.

**STAR-CD**

Select the coupling component “mass-in”.

---

Figure 4: Y-Junction: Coupling of “BoundaryMassFlux”, “BoundaryTotalPressure”, “MassFlowRate”, “TotalPressure”
3. Select the quantity \textit{BoundaryMassFlux} and the network Code is the sender. 
Select the quantity \textit{TotalPressure} and the CFD Code is the sender.

\begin{description}
\item[FLUENT] 
FLUENT receives the quantity \textit{BoundaryMassFlux} at the UDM index 0.
\item[STAR-CD] 
STAR-CD receives the quantity \textit{BoundaryMassFlux} at the SCALAR index 1.
\end{description}

\begin{itemize}
\item[...] 
A first coupled region “Face.1” has been created.
\end{itemize}

1. For the next coupled region click on the [Add] button in the Regions part. This will insert a new region Face.2 to couple.

2. Select the coupling components by double-clicking them or dragging them into the Coupled boxes.

\begin{description}
\item[Flowmaster] 
Select the coupling component “p-out1”.
\item[FLUENT] 
Select the coupling component “p-out1”.
\item[STAR-CD] 
Select the coupling component “p-out1”.
\end{description}

\begin{itemize}
\item[...] 
3. Select the quantity \textit{BoundaryTotalPressure} and the network Code is the sender. 
Select the quantity \textit{MassFlowRate} and the CFD Code is the sender.

\begin{description}
\item[FLUENT] 
FLUENT receives the quantity \textit{BoundaryTotalPressure} at the UDM index 1.
\item[STAR-CD] 
STAR-CD receives the quantity \textit{BoundaryTotalPressure} at the SCALAR index 2.
\end{description}

\begin{itemize}
\item[...] 
Repeat the step 1 to 3 for the component “p-out2” in the region Face.3.
\end{itemize}

No further settings are required, proceed to the Edit Step by pressing Next >.
8.5 Edit Step

Figure 5: Y-Junction: MpCCI Edit Step

- Deselect the CheckBoundingBox option.
- Select the MinimalDistance option from the tree and put for the Theta2 value zero.

The MpCCI server is now configured for the 1D-3D co-simulation. Proceed to the Go Step by pressing \[\text{Next }\].
8.6 Go Step

In the server panel, no changes are necessary. The code panels are described for each code.
- The Initial quantities transfer is set to receive.
- Enter the User Name to log in the database. In this tutorial it was “Admin”.
- Change the name of the Working Project Name corresponding to the root of your project list view in Flowmaster if necessary.
- Please select the analysis type to SS from the list.
- Let other settings as default.
- Set the Initial quantities transfer to exchange.
- Select an Optional journal files "Steady.jou" with the browse button. This file is located under 
  
  "<MpCCI_home>/tutorial/Y-Junction/FLUENT"

- Deactivate the Auto hook functions.
- Deactivate the Auto set MDM zones.
8.7 Running the Computation

8.7.1 Starting the Simulation

Save the MpCCI project file with name "y_junction.csp" over the MpCCI GUI menu File→Save Project As.

Press the three Start buttons in the Go Step and the simulation codes should start. Some codes require additional actions:

**Flowmaster**

The Flowmaster calculation is started in batch mode.

**FLUENT**

The FLUENT GUI is started and the calculation starts automatically. The FLUENT journal file "Steady.jou" defines the following procedures:

```plaintext
;; initialize flow
/solve/initialize initialize-flow
```
FLOWENT will perform 1000 iterations with 20 iterations without coupling.

STAR-CD

The STAR-CD calculation is started in batch mode. STAR-CD will perform 1000 iterations with 20 iterations without coupling.

8.8 Discussion of Results

Flowmaster

You can open the Flowmaster GUI and open the Y_Junction project. You can select some last results and plot a chart of the pressure or mass flow for one boundary component for example.

FLOWENT

After FLOWENT finished the 1000 iterations, you could save the data and post process the results with FLOWENT.

STAR-CD

STAR-CD result file "Y-junction.ccmp" may be post processed with pro-STAR. In order to visualize the quantity value stored in the user scalar you have to select in pro-STAR the data type Cell & Wall/Bound (Smotth).

On the fluid model we expect that the pressure at the inlet and outlet is closed to the prescribed pressure values imposed by the network model. At the inlet boundary “mass-in” a pressure of 4 bar was prescribed on the network model and on the fluid a total pressure value around 2.3 bar is expected. At the outlet boundary “p-out1” and “p-out2” a total pressure value of 1.5 bar respectively 1.2 bar is expected.

By using the MpCCI Visualizer you may visualize the coupled values from the "tracefile.ccv". You may open the tools from the MpCCI GUI Tools→Visualizer. This will open automatically the "tracefile.ccv"
file.

Figure 6: Y-Junction: Fluent Residuals plot
Figure 7: Y-Junction: Fluent Total Pressure plot
Figure 8: Y-Junction: Fluent Velocity vector section plot
VIII Programmers Guide
VIII Programmers Guide – Contents

1 Introduction .......................................................... 6

2 Code API ................................................................. 7
  2.1 Code Integration and Simulation Code Requirements ........................................ 8
    2.1.1 Data Exchange and Data Access .............................................................. 8
    2.1.2 MpCCI Interface for Code Integration ....................................................... 9
  2.2 Code Integration with the MpCCI API Kit ......................................................... 11
    2.2.1 A Simple Example .................................................................................... 11
    2.2.2 Step-by-Step Procedure for Code Integration ............................................. 14
    2.2.3 Code Coupling with the Example ............................................................... 25
  2.3 Code Configuration Directory ........................................................................ 28
  2.4 MpCCI GUI Configuration File gui.xcf ............................................................. 29
    2.4.1 Code Information: <CodeInfo> ................................................................ 29
    2.4.2 Codes Menu: <CodesMenuEntries> ............................................................. 29
    2.4.3 Models Step: <ModelsMenuEntries> ................................................................ 30
    2.4.4 Component Types: <ComponentTypeDimensions> ...................................... 31
    2.4.5 List of quantities: <SupportedQuantities> ................................................... 32
    2.4.6 Go Step: <GoMenuEntries> ....................................................................... 33
    2.4.7 Environments for Scanner, Starter, Stopper and Killer ............................... 35
    2.4.8 General MpCCI GUI Elements ..................................................................... 36
    2.4.9 Testing gui.xcf ......................................................................................... 42
  2.5 Perl Scripts ......................................................................................................... 43
    2.5.1 Using Information from gui.xcf in Scripts ................................................. 43
    2.5.2 Scanner.pm .............................................................................................. 43
    2.5.3 Starter.pm ............................................................................................... 45
    2.5.4 Stopper.pm .............................................................................................. 45
    2.5.5 Info.pm .................................................................................................... 45
    2.5.6 Subcmd.pm ............................................................................................... 46
    2.5.7 Testing the Perl Scripts ............................................................................. 47
  2.6 MpCCI Coupling Manager Functions ................................................................. 48
## Contents

2.6.1 Definition of Output Functions: `MpCCI_Message_init` ........................................... 49
2.6.2 Initialization: `MpCCI_Init` .................................................................................... 50
2.6.3 Get Initial Exchange Mode: `MpCCI_Get_init_actions` ........................................... 52
2.6.4 Data Exchange: `MpCCI_Transfer` ........................................................................... 53
2.6.5 End of Coupled Simulation: `MpCCI_Exit` ............................................................... 54
2.6.6 Definition of Nodes: `MpCCI_Def_nodes` ................................................................. 55
2.6.7 Definition of Elements: `MpCCI_Def_elems` ............................................................. 56

2.7 `MpCCI` Driver Functions ......................................................................................... 58
  2.7.1 Description Values .................................................................................................. 60
  2.7.2 Methods Called before/after some Action ............................................................ 61
  2.7.3 Mesh Definitions ................................................................................................... 62
  2.7.4 Data Exchange ..................................................................................................... 63

2.8 Data Structures and Predefined Macros .................................................................... 64
  2.8.1 Coupling Components ......................................................................................... 64
  2.8.2 Quantities ............................................................................................................. 65
  2.8.3 Loop Functions .................................................................................................... 66

3 `MpCCI` SDK Code Coupling Library ........................................................................... 68
  3.1 The `MpCCI` SDK Concepts ..................................................................................... 68
    3.1.1 Communication levels ......................................................................................... 68
    3.1.2 Coupling quantities ............................................................................................ 68
    3.1.3 Synchronization concepts and data transfer ....................................................... 69
    3.1.4 Coupling Regions ............................................................................................... 70
    3.1.5 Neighborhood Search and Interpolation ............................................................ 70
    3.1.6 The `MpCCI` SDK coupling server scheme ......................................................... 72
  3.2 `MpCCI` SDK Functions .......................................................................................... 74
    3.2.1 Naming Conventions and Terminology ............................................................... 74
    3.2.2 `MpCCI` SDK Data Types .................................................................................. 75
    3.2.3 Initialization and Coupling Definition ................................................................ 75
    3.2.4 Coupling Communication .................................................................................. 106
    3.2.5 Remeshing ......................................................................................................... 127
    3.2.6 Termination ....................................................................................................... 130
### 3.2.7 Control
3.2.8 Miscellaneous functions
3.2.9 Overview of the MpCCI SDK functions

#### 3.3 MpCCI Input File
- 3.3.1 Structure of the Input File
- 3.3.2 Code block
- 3.3.3 Quantities block
- 3.3.4 Control block
- 3.3.5 Contact block
- 3.3.6 Switches block
- 3.3.7 Jobs block
- 3.3.8 Parameters block
- 3.3.9 Coupling block
- 3.3.10 Additional block
- 3.3.11 Include Mechanism

#### 3.4 An Example
- 3.4.1 Start-up and Initialization
- 3.4.2 Coupling Definition
- 3.4.3 Coupled Computation
- 3.4.4 Termination
1 Introduction

This “Programmers Guide” addresses to engineers who have developed their own simulation code and plan to use it within the MpCCI coupling environment. The current MpCCI version allows two levels of interfacing own codes with the coupling environment:

- the MpCCI SDK is a lower level subroutine interface and
- the MpCCI Adapter level used since MpCCI 3.0 for commercial codes.

The MpCCI SDK interface level provides various basic functions to define coupling regions, control communication between the coupled codes and to handle other MpCCI parameters. The usage of MpCCI SDK is very similar to the usage of the MPI Message Passing Interface - there is no fixed protocol when to communicate which data with which other components. This MPI-like flexibility allows the implementation of very different coupling algorithms. On the other hand there is no guarantee that the MpCCI SDK integration of a code A has a compatible communication protocol as that of any other code B. A detailed description of MpCCI SDK can be found on \ref{MpCCI SDK Code Coupling Library}.

To avoid such protocol incompatibilities between different code integrations MpCCI 3.0 has introduced the concept of code adapters. These adapters have internal mechanisms to control the current coupling state and actions of each integrated code. These ‘Coupling Control Managers’ guarantee a consistent behavior and communication protocol for all adapted codes. Chapter \ref{Code API} describes the integration of new simulation codes using a code adapter.

The current MpCCI 3.1 allows both levels of integration. However, it should be noted that for reasons of compatibility the MpCCI Adapter Level should be preferred. In midterm the MpCCI SDK interface will be regarded as a pure internal MpCCI and thus might be subject to larger changes.
2 Code API

This section describes how to establish MpCCI support for your code, i.e. the code can then be coupled with all other codes, which are already supported by MpCCI.

This section is organized as follows:

- General description of code integration. Read this to get an idea how code integration works.
  > 2.1 Code Integration and Simulation Code Requirements

- Description of the MpCCI API Kit, which contains template files and an example. A step-by-step procedure for code integration is given here.
  > 2.2 Code Integration with the MpCCI API Kit

- Reference for MpCCI GUI integration.
  > 2.3 Code Configuration Directory
  > 2.4 MpCCI GUI Configuration File gui.xcf
  > 2.5 Perl Scripts

- Reference for the code adapter.
  > 2.6 MpCCI Coupling Manager Functions
  > 2.7 MpCCI Driver Functions
  > 2.8 Data Structures and Predefined Macros

Before starting the integration of a new code, you should contact the MpCCI support, mpcci@scai.fraunhofer.de, to obtain:

- The MpCCI API Kit, a set of template files and an example.
- A license for an adapter to your code.
2.1 Code Integration and Simulation Code Requirements

In order to understand code integration, you should be aware of the general procedure of a coupled simulation with MpCCI, which is described in IV-1.3 Code Coupling with MpCCI.

2.1.1 Data Exchange and Data Access

![Diagram of data exchange and access](image)

Figure 1: Calls of adapter library routines from the simulation code. The call of `MpCCI_Transfer` can be inserted before or after the solver.

For the coupling process, the analysis code must be able to send and receive data based on the mesh of a coupling component. This requires access to the following data of a simulation code:

**Mesh information** MpCCI handles exchanges of data between non-matching grids, thus it needs to know the grids on both sides. Therefore, the mesh data of the coupling component must be transferred to MpCCI. This includes

- Number of nodes, number of elements, type of floating point values
- Nodes: ID, coordinates,
- Elements: ID, type, nodes.

**Physical values to be exchanged** During the coupling process, data is exchanged, which must be transferred to MpCCI or received from MpCCI. This requires local

- Reading access to values of data to be transferred,
- Allocation of memory for receiving data,
Method to put received data on local mesh.

The data is sent or received by “driver functions”, which must be implemented as part of the code adapter. In addition coupling manager functions of the adapter library must be called at certain stages of the simulation as depicted in Figure 1.

MpCCI provides a C-Interface, which can be used from C, C++ and FORTRAN codes.

In addition to this actual adapter, a number of files can be provided to allow MpCCI to manage the coupled simulation and integrate the code into the MpCCI GUI.

2.1.2 MpCCI Interface for Code Integration

Two basic things are necessary for coupling a simulation code with MpCCI, see also Figure 2.

**GUI integration.** The integration into the MpCCI GUI is realized with a set of configuration files which describe the properties and capabilities of a code, how to scan input files and start the code. An overview of these files is given in §2.3 Code Configuration Directory.

**Code adapter.** The code adapter is needed to handle the data exchange. The adapter is a plug-in into the simulation code, which can be realized in different ways: It can be included directly into the code or be based on user functions which are provided by the code. MpCCI provides a basic C-interface for the development of code adapters. The structure of the code adapter is sketched in Figure 3. Its main parts are:

- **Driver functions** which perform the basic data exchange with the code. It is the passive part of the adapter, as the driver functions are called by the adapter library.

- **The Coupling Manager** is a set of functions for the communication between the code adapter and the MpCCI server. It is part of the MpCCI software package. The functions are defined in the header file "mpcci.h", the object files in "libmpcci*.a" must be linked with the simulation
code. The coupling manager functions call the provided driver functions. The exact procedure is sketched exemplarily for `MpCCI_Init` in Figure 18.

![Diagram of code adapter structure](image)

Figure 3: Code adapter structure: The code adapter is a plug-in of the analysis code, which consists of two parts: Driver functions and Coupling Manager
2.2 Code Integration with the MpCCI API Kit

The MpCCI API Kit contains templates as well as a simple example to demonstrate the integration of a simulation code into MpCCI.

The MpCCI API Kit contains the following files:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;adapter&quot;</td>
<td>Templates for code adapter</td>
</tr>
<tr>
<td>&quot;configuration&quot;</td>
<td>Templates for configuration directory</td>
</tr>
<tr>
<td>&quot;example/configuration&quot;</td>
<td>Configuration Directory files for the example</td>
</tr>
<tr>
<td>&quot;example/C/src&quot;</td>
<td>Source code of the example in C</td>
</tr>
<tr>
<td>&quot;example/C/src-nompcci&quot;</td>
<td>Source code of the example without code adapter in C</td>
</tr>
<tr>
<td>&quot;example/FORTRAN/src&quot;</td>
<td>Source code of the example in FORTRAN</td>
</tr>
<tr>
<td>&quot;example/FORTRAN/src-nompcci&quot;</td>
<td>Source code of the example without code adapter in FORTRAN</td>
</tr>
<tr>
<td>&quot;example/test&quot;</td>
<td>Files for testing the example</td>
</tr>
<tr>
<td>&quot;example/test-nompcci&quot;</td>
<td>Files for testing the example without code adapter</td>
</tr>
</tbody>
</table>

2.2.1 A Simple Example

As a simple example, the – simplified – simulation of an elastic foundation is included in the MpCCI API Kit. The original source code is included in the directory "example/C/src-nompcci", a FORTRAN version can be found in "example/FORTRAN/src-nompcci". The code can be used for two- and three-dimensional computations. The functionality is implemented in "main.c" ("main.f"), data structures are defined in "data.h" and "data.c" ("data.f") together with a simple file input routine. The code is kept simple and thus does not contain any input file checks etc.

Only two element types are supported: Line elements with two nodes and quadrilateral elements with four nodes. For each element an example input file is included in "example/test-nompcci".

The structure modeled in "2dexample.fnd" is depicted in Figure 4 on the left, its content is:

```
EF bed1 2 1 -0.5
NODES 3
  0 0 0
  1 1 0
  2 2 0
ELEMENTS 2
  0 0 1
  1 1 2
```

The acronym [EF] in the first line starts the definition of an elastic foundation (one file could contain more than one!). Its name is “bed1”, with space dimension “2” and springs in direction “1”, i.e. the y-direction.
The spring constant of all springs is “-0.5”. The minus sign is added because it is loaded from top, i.e., a positive pressure should result in a motion in negative y-direction.

Running “foundation” with this input file yields:

```
> ../bin/foundation-nompcci 2dexample.fnd
Foundation - computation
Reading file >>2dexample.fnd<<...
foundation >bed1<   dim=2 direction=1 stiffness=-0.5
    nodes...
    0 : 0 0
    1 : 1 0
    2 : 2 0
    elements...
    0 : 0 1
    1 : 1 2
    allocating memory...
read 1 foundations.
Step 0 of 3
Please enter pressure: 1.0

results for foundation 'bed1':
node 0:  force= 0.5  displacement= -1
node 1:  force= 1    displacement= -2
node 2:  force= 0.5  displacement= -1
```
Step 1 of 3
Please enter pressure: 2

results for foundation 'bed1':

<table>
<thead>
<tr>
<th>node</th>
<th>force</th>
<th>displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>-2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-2</td>
</tr>
</tbody>
</table>

In each step, the user is asked for a pressure value which is then applied and the resulting forces and displacements are printed to stdout. The number of “steps” can also be given as a second command line argument after the input file name, three steps are made if nothing is specified.

Please have a look at the files of the original code in "src-nompcci". The header file "data.h" contains the definition of the data structure `FOUNDATION` and of a list of such structures `fndlist` which is filled by the values from the input file.

In "main.c" ("main.f") all actual computations are contained, a number of functions is called from `main`, which controls what is done.

The example was created on a Linux 32 Bit machine but should also run on other Linux systems. The C version was compiled using the GNU compiler `gcc`, the FORTRAN version was tested with GNU, Absoft 9.0, Intel 9.1 and PGI 6.1 compilers, see the "Makefile".

⚠️ For combining C and FORTRAN sources, as it is done in the FORTRAN example, the naming conventions depend on the compiler. In the sources of the example an underscore is added to the C function names and lowercase names are used. Please consult the documentation of your compiler to find how to combine C and FORTRAN sources.
2.2.2 Step-by-Step Procedure for Code Integration

The creation of a code adapter can be performed on a step-by-step basis:
Step 1: Preparations.
Step 2: Create the configuration directory.
Step 3: Test the configuration.
Step 4: Create the code adapter.
Step 5: Test the code adapter.

For each step, the corresponding changes in the example are mentioned as well. To understand the changes you can compare (“diff”) the files with the original versions:

<table>
<thead>
<tr>
<th>original</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;configuration/*&quot;</td>
<td>&quot;example/configuration/*&quot;</td>
</tr>
<tr>
<td>&quot;example/C/src-nompcci/data.h&quot;</td>
<td>&quot;example/C/src/data.h&quot;</td>
</tr>
<tr>
<td>&quot;example/C/src-nompcci/data.c&quot;</td>
<td>&quot;example/C/src/data.c&quot;</td>
</tr>
<tr>
<td>&quot;example/C/src-nompcci/main.c&quot;</td>
<td>&quot;example/C/src/main.c&quot;</td>
</tr>
<tr>
<td>&quot;example/C/src-nompcci/Makefile&quot;</td>
<td>&quot;example/C/src/Makefile&quot;</td>
</tr>
<tr>
<td>&quot;adapter/adapter.h&quot;</td>
<td>&quot;example/C/src/adapter.h&quot;</td>
</tr>
<tr>
<td>&quot;adapter/adapter.c&quot;</td>
<td>&quot;example/C/src/adapter.c&quot;</td>
</tr>
</tbody>
</table>

All changes are commented and marked with CHANGED:. (For FORTRAN see the corresponding files in "example/FORTRAN".)
STEP 1: Preparations.

Understand the basic concepts of MpCCI. If you have no experience with MpCCI, it is recommended to run some of the Tutorial examples first. Try to understand the way MpCCI works by reading V-3 Code Coupling.

Decide which quantities shall be exchanged. Just select a basic set of quantities, more can be added later. It is sufficient to select one quantity for receiving and one for sending.

A complete list of quantities is given in the Appendix.

Identify possible coupling components. Decide whether data is exchanged on surfaces or volume and how they can be found in model files.

The example “foundation” code computes displacements due to an external pressure. Thus the following quantities will be exchanged:

- foundation code (elements) ← OverPressure ← partner code
- foundation code (nodes) → NPosition → partner code
STEP 2: Create the configuration directory.

Copy files from MpCCI API Kit  The configuration directories for simulation codes are all located in the subdirectory "<MpCCI_home>/codes" of the MpCCI home directory, which you can find with the command `mpcci home`.

In the "codes" directory, there is one configuration directory for each code. You should already find some of commercial simulation codes there.

Create a directory with the name of your code there and copy all template files from the MpCCI API Kit, subdirectory "configuration", to your new code configuration directory.

"gui.xcf" Change options to your needs, a detailed description is given in \[2.4\] MpCCI GUI Configuration File gui.xcf.<.

For the "foundation" example, the codename is set to “foundation”. The foundation code does not use any units, so as default we select “SI”. The code type is set to SolidStructure, as deformations of solid structures are computed. A list of available types is given in \[V-3.1.1\] Physical Domains.<. In the section <ModelsMenuEntries>, only “1.0” is added as version, the file extension is set to .fnd and the unit selection menus are kept as no fixed set of units is used in the code.

The <ComponentTypeDimensions> are a list of names for the coupling components. We actually only need Face, but add names Global, Line, Face and Volume.

From the supported quantities, OverPressure and NPosition are kept, the locations are set appropriately and send and receive options are set to Direct as all data will be directly written to and read from the code’s data structures. Both quantities can be exchanged on faces. In the 2D case, a line also represents a face!

As an additional Go-Menu entry, a selection of the number of steps is added. This will later be passed as a command line argument to the code.

The Scanner only needs to know the model file, the Starter also gets the number of steps, the stopper gets the model file name.

Perl scripts: "Info.pm", "Scanner.pm", "Starter.pm", "Stopper.pm", "Subcmd.pm"

The Perl scripts are needed to get information at run time and interact with your code. See \[2.5\] Perl Scripts< for a detailed description, each template file is commented.

Most important are "Scanner.pm" to scan the input file and "Starter.pm" to start the code. It is recommended to also use "Info.pm" to gather basic information and "Stopper.pm" to trigger a graceful exit of your code.

The MpCCI API Kit contains an additional "external.Scanner.pm" if you do not want to use Perl to scan the model file, but e.g. the file reader of your simulation code.
In the information module "Info.pm" for foundation, the only thing which is really done is to find the executable in the path and return the requested information.

The "Scanner.pm" scans the model file "*.find" for coupling component definitions, in our case these are all elastic foundations which are defined. These start with the keyword EF which is followed by the name. The input file is simply searched for lines containing this information, which is then returned to the calling MpCCI function.

The "Starter.pm" obtains the model file name and the number of steps from the MpCCI GUI, creates an argument list @argv with the corresponding command line arguments and starts “foundation”. 
STEP 3: Test the configuration.

Testing the Perl scripts. Before testing the GUI integration, you should ensure that the Perl scripts work.

- Start with the scanner, a test-script "test_Scanner.pl" is included in the "configuration" directory.
  
  To run a test, you must provide the environment variables, which are needed by the scanner. For this please edit "test_Scanner.pl" and change the assignments. To select a model file, e.g. provide

  ```
  $ENV{'_MPCCI_MODEL_FILE'} = '/home/fred/adapter-test/example.input';
  ```

  If all necessary variables are set, you can simply run "test_Scanner.pl".
  
  The results of the scanning process are saved in a scan file "mpcci_<original filestem>.scan", i.e. for the above example the file would be named "mpcci_example.scan".
  
  All information your scanner found should be contained in this file. The model info is listed first, followed by the list of components. Please check if all information is present.

- Test the starter using "test_Starter.pl" in the same way. The starter should not do more than start your code with the appropriate command line options.

- You can also test the stopper with "test_Stopper.pl".

Testing "gui.xcf"  For testing "gui.xcf" you need to have a license for a code adapter, which you usually receive together with the MpCCI API Kit.

For testing, you must prepare a simple input file and one for the partner code and proceed as follows.

1. Go to the testing directory and start the MpCCI GUI with `mpcci gui`.
   → If you get an error message right after starting the MpCCI GUI, there is a syntax error in your "gui.xcf". Make sure, that the file is a proper XML-file.

2. Your newly added code should now appear in the list of codes on the left side.
   → If you cannot find your new code, ensure that you have created a new subdirectory in "<MpCCI_home>/codes" with the correct name.

3. Click on your code, and you should see what you defined as elements of the model step.
   → If some elements are missing or wrong, check the `<ModelsMenuEntries>` section.

4. Fill in the required values in the form, and press the [Start Scanner]. → The scanner should start, otherwise you will receive an error message. If the scanner finished, click on the box which appears and check the return values of the scanner.

5. Select and scan the partner code and proceed to the Coupling Step. → Here, you should be able to select the quantities as defined in "gui.xcf". If you cannot select anything there are no quantities which are supported on the same geometric entity by both codes. If some quantities are missing check the `<SupportedQuantities>` block. Remember that only quantities will be shown, which are also supported by the partner code!
6. Finally, proceed to the Go step and check the buttons in the panel of your code.
   → If entries are wrong check the `<GoMenuEntries>` section of "gui.xcf".

7. You can also start your code – it should simply run as usually. However, no coupling will occur
    as no code adapter is present.

<i>For “foundation” the testing scripts "test_Scanner.pl" and "test_Starter.pl" were changed: The
    scanner only needs the model file, the starter model file and number of steps for testing.</i>
STEP 4: Create the code adapter.

- The adapter template files are located in the subdirectory "adapter". Add "adapter.h" and "adapter.c" to your source code.

- Change the definition of the list of driver functions `MpCCIDriverFunctions` in "adapter.c" to fit your needs: Add name of your code and select which data exchange functions you need. Usually the block with functions called before/after some actions can be left empty.

- Look if the interface functions are also useful for your code and make appropriate changes. In most cases they should already be OK.

- Adjust the driver functions. All driver functions which are declared in `MpCCIDriverFunctions` must be defined. See ▶ 2.7 MpCCI Driver Functions ◁ for a description.

- Change "adapter.h" to fit "adapter.c"

- Insert the calls of the interface functions at appropriate places in your code as described in ▶ 2.1 Code Integration and Simulation Code Requirements ◁.

- Add "adapter.c", the MpCCI include directory "<MpCCI_home>/include" and the MpCCI client library for your platform to your "Makefile".

- If your code is compiled and linked, you can proceed with testing the adapter.

For “foundation” the most important changes are (please see source code files for details):

**Driver Functions** Only a small set of driver functions, `MpCCI_Driver_updateComponents`, `MpCCI_Driver_defineGrid`, `MpCCI_Driver_getFaceNodeValues` and `MpCCI_Driver_putFaceElemValues` are used, all other functions were set to `NULL` in the `MpCCIDriverFunctions` structure.

**Additional helper functions** Additional helper functions are required: `getSurfaceID` to identify coupling components by their names and for `adapterOutput` and `error` for data output. In the C version they are included in "adapter.c" in the FORTRAN version they are located in "helpers.f".

**Data exchange** In the C version the data access is directly written into the driver functions `MpCCI_Driver_getFaceNodeValues` and `MpCCI_Driver_putFaceElemValues`. The FORTRAN version uses additional subroutines in "helpers.f": `getInfo` to obtain basic information, `getMesh` to obtain mesh information and `getNPosition` and `putPressure` for data exchange. The FORTRAN helper functions are called from the C routines in "adapter.c".

**Makefiles** The Makefile must be changed to include "mpcci.h" and the MpCCI library. For FORTRAN, additionally a C compiler is required to compile "adapter.c" which can be linked directly with the FORTRAN objects.
STEP 5: Test the code adapter.

Please ensure first that the files in the configuration directory are correct (Step 3). So we can assume now, that you already have set up a sample problem.

- Install the license for your code - either on a license server or on your local machine, see III-5 Licensing for details. You can check the license status with `mpcci license mpcci` or from the MpCCI GUI in License→Check the MpCCI license status.

- Open your project (*.csp*) in the MpCCI GUI.

- In the coupling step, set the output level to 3 to obtain maximum output.

- In the Go step, select Run server processes inside xterm.

- Start the MpCCI server and both simulation codes by pressing the [Start] buttons in the Go step. One window for each code server (and the control process if enabled) and for both codes should pop up.

- Check the output of your code and the corresponding MpCCI server. The output of the servers should help you to find any errors in the code adapter functions.

For the foundation code, the code’s output (in the yellow window) should look as follows:

```
Starting: foundation elasticwall.fnd 2 1> mpcci_elasticwall.log 2>&1
Waiting for logfile "mpcci_elasticwall.log"....
```

This means MpCCI is waiting for the code’s output. This message should be followed by the usual output of the code, here:

```
Foundation - computation
Reading file >>elasticwall.fnd<<....
foundation >elasticwall< dim=3 direction=2 stiffness=-5
nodes...
```

and so on. Further below you should find the call of `MpCCI_Init`, with

```
Initializing the coupling process!
Partner job#1("FLUENT") is code "FLUENT".
```

followed by the list of coupled regions
Coupled regions for code "foundation"

- Component #0: "elasticwall"
  - Dimension(2), MeshId(1), PartId(1), Nodes(9), Cells/Elements(4)
  - 2 Quantities.
    - Recv: 1012, 1D, loc(Elem), type(Field), sm(Direct/0), mean(BC/Value), name(OverPressure)
    - Send: 3041, 3D, loc(Node), type(Field), sm(Direct/0), mean(Grid/coord), name(NPosition)

For each exchange you should see an output like

```plaintext
### MpCCI_Transfer: starting a new transfer.
### newActions(wait=1): >PUTCQ>ISEND>RECV>WAIT>GETCQ
entered get_values...
finished get_values...
MpCCI_ISend_recv() called 1 times:
  ITER:  0 sec.
  SEND:  0 sec.
  RECV:  1 sec.
  XCHG:  1 sec.
  WAIT:  1 sec.
entered put_values...
finished put_values...
```

Remember that for the first exchange the value of Initial quantities transfer is used, thus – depending on your choice the output get_values or put_values may not appear.

The output in the server window is much more detailed (because the output level is set to 3). It starts with the DEBUG OUTPUT OF THE INPUT FILE DATA which is a summary of the settings in the MpCCI GUI. This is followed by the actual server output, starting with

```plaintext
foundation:2: Entered CCI_Init.
```

which is followed by many more CCI... calls. Important is the section

```plaintext
foundation:2:0: Entered CCI_Def_nodes.
foundation:2:0:  IN meshId       = 1
foundation:2:0:  IN partId      = 1
foundation:2:0:  IN globalDim   = 3
foundation:2:0:  IN nNodes      = 9
foundation:2:0:  IN nNodeIds    = 9
```
which reflects the node coordinates. Check whether they correspond to the coordinates given in the input file! This is followed by a block starting with

```
CCI_Demo:
1:2:0: Entered CCI_Def_elems.
```

with the element definitions. Next follows the neighborhood computation, which lists the neighborhood information (see also $\triangleright$ V-3 Code Coupling $\triangleleft$), starting with

```
CCI_Demo:
1:2:0: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
CCI_Demo:
1:2:0: NEIGHBORHOOD COMPUTATION
CCI_Demo:
1:2:0: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Each exchange of quantities is reflected by a number of entries,
```
CCI_Demo:
1:2:0: Entered CCI_Recv,
```
foundation:2:0: Entered CCI_Get_elems.
and
foundation:2:0: Entered CCI.Put_nodes.
foundation:2:0: Entered CCI.Isend.
All values which are send or received are listed. Please check if the quantities are transferred correctly.
2.2.3 Code Coupling with the Example

For testing the code adapter for the example code, a set of sample problems is provided in "example/test" for coupling “foundation” with some commercial codes.

![FLUENT – foundation sample problem. A cube filled with an ideal gas is coupled with an elastic foundation which deforms due to the gas pressure.](image)

The sample problem is sketched in Figure 5. Both input files are given, "foundation/elasticwall.fnd" for “foundation” and for FLUENT "FLUENT/cube.cas".

To run the sample problem do the following:

- Go to the "example/test" directory and start the MpCCI GUI.

- Select “foundation” as the first code (you should first finish creating the configuration and code adapter for the foundation code, see §2.2.2 Step-by-Step Procedure for Code Integration) and "foundation/elasticwall.fnd" as model file. Keep the unit system set to SI.

- Start the Scanner for “foundation”, if you click on the green check mark, you should get

```
# MpCCI relevant information:
# Model dimensions : 3D
```
Select FLUENT as the second code, the FLUENT version 3d, and the latest FLUENT release. Finally choose "FLUENT/cube.cas" as model file.

Start the Scanner for FLUENT.

Proceed to the Coupling Step – only the Face (2D)-card should be enabled. Select elasticwall and bottom as coupling components and NPosition and OverPressure as quantities.

Proceed to the Edit step and set OutputLevel→Global to 3.

Proceed to the Go step. Select Run server processes inside xterm for the server. For “foundation” select receive for the initial transfer and 3 as number of steps. For FLUENT select exchange as initial transfer and do not change further options.

Save the project and start the processes. Three server windows should pop up, for foundations, FLUENT and the control process. A yellow window with the output of the foundation code should pop up and the FLUENT graphical interface.

In the FLUENT window select Solve→Initialize →Initialize and press the Init to initialize the FLUENT solution.

Select Solve→Iterate and set the Number of Time Steps to 2. and press Iterate.

Now, FLUENT should perform 2 iterations while “foundation” finishes its computation. The FLUENT result is depicted in Figure 6. You should clearly recognize the deformation of the bottom where the elastic foundation is coupled.
Figure 6: FLUENT mesh and result of the cube example.
2.3 Code Configuration Directory

The files which are required to integrate the code into MpCCI must located in a specific directory, as already described in ▷ 2.2 Code Integration with the MpCCI API Kit ◁. The directory "<MpCCI_home>/codes" has one subdirectory for each code which is supported by MpCCI. At least six files, which always have the same file names, should be inside any code subdirectory:

"gui.xcf" This file contains all definitions which are required to fit the code into the MpCCI GUI, which includes
- code name and version information,
- extension of input files,
- additional options for the Model-, Go- menus,
- and a list of supported quantities.

▷ 2.4 MpCCI GUI Configuration File gui.xcf ◁

"Scanner.pm" This Perl script is started to scan the input file of the simulation code for information which is needed for the coupling process, mainly to identify possible coupling components.

▷ 2.5.2 Scanner.pm ◁

"Starter.pm" The starter script starts the simulation code with appropriate command line options, which can be selected in the MpCCI GUI.

▷ 2.5.3 Starter.pm ◁

"Stopper.pm" The stopper script is called if the code shall be stopped, i.e. if the stop in the MpCCI GUI is clicked.

▷ 2.5.4 Stopper.pm ◁

"Info.pm" should collect application specific information like code release.

▷ 2.5.5 Info.pm ◁

"Subcmd.pm" can be used to define MpCCI code specific subcommands.

▷ 2.5.6 Subcmd.pm ◁
2.4 MpCCI GUI Configuration File "gui.xcf"

The file "gui.xcf" is an XML-file, which contains definitions for the MpCCI GUI. Entries for the Model and Go step and for the Codes menu in the menu bar can be defined, and the selected options can be passed on to the scanner, starter, stopper and killer scripts.

It consists of several sections, which are discussed in the following.

2.4.1 Code Information: <CodeInfo>

The code information block contains the basic information of a code:

- **Units**: Unit system used by the code
- **Type**: Type of code - needed to determine possible coupling types
- **SelfCoupler**: Indicator if a code can be coupled with itself (default is false)

The type of the code can be one or several of the code types CFD, ElectroMagnetism, FluidPlasma, FluidThermal, InjectionMoulding, Radiation, SolidStructure, SolidThermal. Usually one code can support different analysis types, these can be given separated by spaces, e.g. "CFD FluidThermal".

2.4.2 Codes Menu: <CodesMenuEntries>

For each code commands can be defined which are provided in the menu bar beneath the codename item. If more than three codes are offered the code menus are collected under the [Codes] item in the menu bar.

![Figure 7: Codes Menus for two (Code_A and Code_B) and more than three Codes](image)

Each provided command is defined as a menu element

```xml
<Release type="menu"
text="Releases"
tooltip="Display all installed releases located by MpCCI."
command="mpcci Codename releases" />
```

- the tag Release is used to identify the selected value later. Its name can be arbitrary but should not begin with a number and it should not contain special characters.
• the type is "menu".

• the provided text is displayed as the menu entry.

• the tooltip is shown as tooltip for the menu entry.

• the command will be passed to the local system to be executed.

The result of the executed command is shown in a dialog box.

2.4.3 Models Step: <ModelsMenuEntries>

For each code additional options can be added to the Models step in the MpCCI GUI. The information given here is only partly evaluated by the MpCCI GUI itself. Most options are for use in the scanner or starter to hand it over to the simulation code. The definitions in the "gui.xcf" actually completely determine the appearance in the MpCCI GUI.

The template file from the MpCCI API Kit contains already two examples. An overview of all possible elements is given in ⊳2.4.8 General MpCCI GUI Elements⊿.

The first element in the example is an enumeration element

```xml
<Version type="enum" default="1.0" description="Please select the version:'">
  <enum value="1.0" />
  <enum value="1.1" />
  <enum value="latest" />
</Version>
```

The tag Version is used to identify the selected value later. Its name can be arbitrary but should not begin with a number and it should not contain special characters.

The second element is a file selector, which is usually part of every Model step entry:

```xml
<ModelFile type="filename" required="true" default="" 
  description="Please select the model file:'">
  <filename suffix=".mod" />
</ModelFile>
```

Further <filename> lines can be added to allow a selection of different suffixes.

An example of a configuration is shown in Figure 8.
<ModelsMenuEntries>
  <Version type="enum" default="latest"
    description="Version:">
    <enum value="0.9" />
    <enum value="0.999" />
    <enum value="latest" />
  </Version>

  <ModelFile type="filename" required="true"
    default="" description="Model file:">
    <filename suffix=".suf" />
    <filename suffix=".newsuf" />
  </ModelFile>

  <Units type="enum" default="SI"
    description="Unit system:">
    <enum value="British" />
    <enum value="SI" />
    <enum value="variable" />
  </Units>
</ModelsMenuEntries>

Figure 8: Example of <ModelsMenuEntries> definition in "gui.xcf" and resulting Model step buttons.

### 2.4.4 Component Types: <ComponentTypeDimensions>

Here, you specify the dimension for component types. The components and their types are returned by the scanner in the scanner output file. Because the types for the components differ from code to code each type name used in the scanner output file has to be associated with a dimension. These dimensions correspond with the labels of the element collections in the MpCCI GUI Coupling Step.

- 0 means the component is a data structure and its elements are 0D global values.
- 1 means the component comprises 1D line elements.
- 2 means the component comprises 2D face elements.
- 3 means the component comprises 3D volume elements.

For each component type add a line in the <ComponentTypeDimensions> block.

<ComponentTypeDimensions>
  <Typename type="int" default="0 | 1 | 2 | 3" />
</ComponentTypeDimensions>
The tag **Typename** has to be replaced by the type name used in the scanner output file. The **type** is "int" and **default** should be set to the associated dimension: "0", "1", "2" or "3".

### 2.4.5 List of quantities: <SupportedQuantities>

Before the actual list of quantities, the storage options are defined in the element **<StorageOptions>**. You need not change the definition given in the template. More definitions are only useful, if your code supports different storage locations of the quantities and the user should select them in the **MpCCI GUI**.

The tag **<SupportedQuantities>** contains a list of all quantities which are supported by the simulation code. The template which you unpacked in your code subdirectory contains a complete list of all quantities, which can be handled by **MpCCI**. For a description of the quantities see also the quantities list in the **Appendix**.

So please remove or comment all quantities your code does not support.

The remaining lines must be fitted to the simulation code. Each line has four attributes, e.g. :

```xml
<Temperature type="quantity" loc="node elem" so="Direct" ro="Direct"/>
```

- The **type** is "quantity".
- The attribute **loc** which defines the location of the quantity can be "node", "elem" or "global" or a combination of these e.g. "node elem", which means it can be of either type.
  - "node" Nodal quantity, i.e. the values are defined for each node.
  - "elem" Element quantity, i.e. the values are defined per element, also for quantities defined at special points of the element, e.g. at integration points.
  - "global" Global quantity, which is not related to nodes or elements, e.g. a time step size.
- The attributes **so** and **ro** which stand for “send option” and “receive option” can be set to any of the storage methods defined in **<StorageOptions>** or combinations like "Direct Usrmem". Usually **so** and **ro** are either set to the empty value ",", which means no receiving or sending of this quantity is possible, or to "Direct", which means the values are read and written directly to the nodes or elements in the simulation code. The predefined storage methods are also listed in the description of the code API macros in ▷2.8 Data Structures and Predefined Macros◁.

Now the quantities must be assigned to the element types |Global|, |Line|, |Face| and |Volume| which are the labels of the element collections in the **MpCCI GUI Coupling Step**.

Finally a quantity definition block should look like

```xml
<Quantities>
  <DeltaTime type="quantity" loc="global" so="Direct" ro="Direct"/>
  <WallForce type="quantity" loc="elem" so="Direct" ro=""/>
  <NPosition type="quantity" loc="node" so="" ro="Direct"/>
</Quantities>
```
which means that the simulation code can exchange three different quantities. The time step size is a global quantity and can be sent or received, wall forces are defined at elements and can only be sent, while the node positions can only be received. The wall forces and node positions can only be exchanged on 2D surfaces.

### 2.4.6 Go Step: `<GoMenuEntries>`

Similar to the definitions for the Models step, the appearance of the go step can be defined in "gui.xcf".

**Additional Configuration for Parallel Code**

A minimum set of definitions have to be provided in order to describe the parallel configuration of the code.

This is the set of definitions to insert in `<GoMenuEntries>`:

```xml
<ParallelRun type="panel" default="false" description="Run parallel">
  <NumProcs type="int" default="1" min="1" max="512" description="No. of parallel processes" />
  <SharedFS type="bool" default="false" description="Shared file system (no file copy)" />
  <HostList type="hostlist" default="" description="Optional 'host host ...' to be used" />
  <HostFile type="filename" default="" description="Optional hostlist file">
    <filename suffix=".hosts" />
    <filename suffix=".hostlist" />
    <filename suffix=".hostfile" />
  </HostFile>
  <DefaultHosts type="bool" default="false" description="Use default hostfile" />
</ParallelRun>
```
The parallel configuration is encapsulated in a panel element. The following information may be configured:

- The number of parallel processes.
- Specify if it is a shared file system.
- Specify a host list.

Additionally you have to add some environments in the `<Starter>`.

```xml
<_MPCCI_<code name>_PARA_RUN type="string"
    default="%(GoMenuEntries.ParallelRun)" />
<_MPCCI_<code name>_PARA_NPROCS type="string"
    default="%(GoMenuEntries.ParallelRun.NumProcs)"/>
<_MPCCI_<code name>_PARA_SHAREDFS type="string"
    default="%(GoMenuEntries.ParallelRun.SharedFS)"/>
<_MPCCI_<code name>_PARA_HOSTLIST type="string"
    default="%(GoMenuEntries.ParallelRun.HostList)"/>
<_MPCCI_<code name>_PARA_HOSTFILE type="string"
    default="%(GoMenuEntries.ParallelRun.HostFile)"/>
<_MPCCI_<code name>_PARA_DEFHOSTS type="string"
    default="%(GoMenuEntries.ParallelRun.DefaultHosts)"/>
```

This information will be evaluated by the function `code_start_prepare`.

```perl
my ($numProcs,$sharedFS,@hostList) = code_start_prepare($codeName, # required
    $numProcsRequested, # optional
    $nameDefaultHost, # optional
    $nameFirstHost, # optional
    $compressHostList, # optional
    $printHostList, # optional
    $checkHostList, # optional
    $copyClientrc # optional);
```

This is a helper function which has to be called in your "Starter.pm" script. This will assist the preparation of the parallel run. The function will create a list of hosts where we fire up the processes and return the number of processes and the hostlist. You may post process the return values to parameter the start of your code.
2.4.7 Environments for Scanner, Starter, Stopper and Killer

The last sections of "gui.xcf" define which information is passed to the Perl scripts which are called by the MpCCI GUI. Values are transferred in form of environment variables, which are defined in "gui.xcf". Each of the elements <Scanner>, <Starter>, <Stopper> and <Killer> contains one sub-element <Environment> in which the variables are set to values which are defined in other sections of "gui.xcf". There are two classes of values:

**Required values** must be set. These are:

<table>
<thead>
<tr>
<th>scripts</th>
<th>variable</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanner, Starter, Stopper, Killer</td>
<td>_MPCCI_MODEL_FILE</td>
<td>The name of the model file.</td>
</tr>
<tr>
<td>Starter</td>
<td>MPCCI_INITIAL_EXCHANGE</td>
<td>Value for initial exchange (see ( \triangle V-3.3 ) Coupling Algorithms).</td>
</tr>
</tbody>
</table>

**Optional values** can be added as necessary. All variables should follow the name convention and start with _MPCCI_<code name> to avoid conflicts with other variables.

All variables must be of type "string".

A short definition of the environment of the starter script for the code “example” could look as follows:

```
<Starter>
  <Environment>
    <MPCCI_INITIAL_EXCHANGE type="string" default="%(GoMenuEntries.InitialExchange)" />
    <_MPCCI_MODEL_FILE type="string" default="%(ModelsMenuEntries.ModelFile)" />
    <_MPCCI_EXAMPLE_VERSION type="string" default="%(ModelsMenuEntries.Version)" />
    <_MPCCI_EXAMPLE_MODE type="string" default="%(GoMenuEntries.batchMode)" />
  </Environment>
</Starter>
```

**References**

Often in "gui.xcf" values which were defined in another section of "gui.xcf" or other references shall be handed over to a script, which is e.g. called by a command button. The following references are possible:

- "%(<element>.<subelement>)" value of an element specified within this configuration file e.g. "%(ModelsMenuEntries.ModelFile)".
- "%(mpcciserver:<element>.<subelement>)" the same as before but the element will be taken from the MpCCI server configuration file.
- "%(#codename)" references the instance name of the code which is also used in the MpCCI GUI to specify the code.
- "$(<environment variable name>)" value of the specified environment variable on the local machine.
2.4.8 General MpCCI GUI Elements

There is a choice of MpCCI GUI elements, which can be used in "gui.xcf" within `<ModelsMenuEntries>` or `<GoMenuEntries>`. For each element applies:

- **Identifier** is a unique identifier for that element.
- **type** defines the type for the special element.
- **description** is always used as title for the respective element.

⚠️ The tag `Identifier` used in the following examples has to be replaced by a unique name which can be arbitrary but should not begin with a number and it should not contain special characters.

**Text Field**

```xml
<Identifier type="string" default="opt1 opt2"
   description="Additional Code_B options"/>
```

default is the default value of the text.

![Text Field representation](image9.png)

Figure 9: Text Field representation

**File Selector**

```xml
<Identifier type="filename" default="" description="Data file (optional)"
   <filename value=".suffix1"/>
   <filename value=".suffix2"/>
</Identifier>
```

default should be left empty.

value is one accepted and selectable file suffix in addition to the predefined selectable "All Files" option.

![File Selector representation](image10.png)

Figure 10: File Selector representation
**Enumeration**

There are three ways for getting the enumeration values:

- List them in a value list.
- Specify a local file as source which holds the values line by line.
- Specify a command which will be executed at runtime by selecting a value and which will provide the values line by line to standard output.

**Value list:**

```xml
<Identifier type="enum" default="SI" description="Select unit system">
    <enum value="British"/>
    <enum value="cgs"/>
    <enum value="variable"/>
</Identifier>
```

default is the default and will be added as `enum value` if it doesn’t exist in the value list. 
value is one possible value of the value list.

![Figure 11: Enumeration representation](image)

**Source file:**

```xml
<Identifier type="enum" default="typeA" description="Title"
            source="fileWithEnumValues"/>
```

default is the default and will be added as `enum value` if it doesn’t exist in the value list. 
source is the name of the file which holds the value list. The file will be looked for in the current user directory and in the user home directory. Each line in the source file corresponds to an `enum value` in the value list.

**Command:**
**Identifier**

*default* is the default and will be added as *enum value* if it doesn’t exist in the value list.

*command* is the command which will be executed to get the value list. References as described in 2.4.7 are allowed to be used.

*hostref* is an optional specification of the host on which the command shall be executed. Therefore the referenced object has to be a file. The host of this file will be used to execute the command. If no *hostref* is specified the command is executed on the local host.

**Range Value**

There are two types of range values. You may create a range value of

- integer value, in that case you have to use the type *int*.

- floating value, in that case you have to use the type *float*.

```
<Identifier type="int" default="1"
    min="1" max="512"
    description="No. of parallel processes"/>
```

```
<Identifier type="float" default="0.01"
    min="1e-10" max="1e10"
    description="Coupling time step"/>
```

*default* is the initial default value.

*min* defines the lower limit.

*max* defines the upper limit.

The *min* and *max* options are optional. If one of them lacks the minimum respectively maximum value of the system will be taken. If neither *min* or *max* is given the value is set to the default value and may not be changed.

![Figure 12: Range value representation for int and float types](image-url)
Checkbox

```xml
<Identifier type="bool" default="true"
    description="Start additional control process"/>
```

default must be initialized with true or false.

![Checkbox representation](image1.png)

Figure 13: Checkbox representation

Command

A command is an element which executes a defined command. It is represented by a button which has to be click to execute the command.

```xml
<Identifier type="command" default="cmd arg1 %(GoMenueEntries.Jobname)"
    description="define some user files"
    hostref="%(ModelsMenuEntries.ModelFile)"
    addComponentEnvironment="true"/>
```

default is the command to be executed with its arguments. References to other elements are allowed and will be resolved.

hostref is an optional specification of the host on which the command shall be executed. Therefore the referenced object has to be a file. The host of this file will be used to execute the command. If no hostref is specified the command is executed on the local host.

addComponentEnvironment is an optional specification whether the environment with the components and quantities (as used for the starter) shall be set or not. If no addComponentEnvironment is specified the environment won’t be set.

![Command representation](image2.png)

Figure 14: Command representation

Hostlist

A hostlist is an element which checks the list of host names. It verifies the resolvability of the host and if it is alive.
Whitespace, comma or semicolon may be used as delimiters to define the list of host names. A host name may be given as “[user@]host”.

```xml
<Identifier type="hostlist" default=""
    description="Optional 'host host ...' to be used"/>
```

default is the initial default value and may be left empty.

![Optional 'host host ...' to be used](Figure 15: Hostlist representation)

**Panel**

A panel is used to group elements which only have to be set if a special feature is used. The panel itself consists of a checkbox and indicates if the special feature is used or not. If the checkbox is set the panel expands and shows its subelements. Now these subelements can be set and will be evaluated if required. If the checkbox is unset, the panel will disappear and hide its subelements.

```xml
<Identifier type="panel" default="false" description="Run parallel">
    ... subelements ...
</Identifier>
```

default must be initialized with true or false.

**Common Feature: Dependency**

For each of the previous elements a dependency may be added. This means that an element is only shown in the MpCCI GUI if the set dependency is complied.

```xml
<Identifier type="<element type>" default="<default value>" description="Title"
    dependsOn="%(ModelsMenuEntries.Units)" dependingValue="variable"
    dependingCondition="<true | false>"/>
```
dependsOn

is a reference to the element this element depends on.

dependingValue

is the value the referenced value will be compared to. If this element depends on more than one value of the referenced element this depending value may be a list of values separated by a [ ] (i.e.: dependingValue="value 1 | value 5")

dependingCondition

provides the condition for the comparison between the depending value and the referenced value. If the condition is true this depending element is only shown if the compared values (resp. one of the compared values if the depending value is a list of values) are equal. On false this element is shown if the compared values (resp. all of the compared values if the depending value is a list of values) are not equal. The default condition true is taken when the dependingCondition is omitted.

If the dependency is used in a panel the panel cannot be set or unset by the user anymore because this will be done automatically in reliance on its dependency then.

An example can be viewed in the template which you unpacked in your code subdirectory. There the element GridLengthUnit in the <ModelsMenuEntries> block depends on the Units element of the same block. The GridLengthUnit element is only shown if the value of Units is set to "variable".

**Common Option: Required**

Another option which is common to all previously described MpCCI GUI elements is the required option.

```xml
<Identifier type="<element type>" default="" description="Title"
required="true | false" />
```
required states that this element must be set before going on with the next step in the MpCCI GUI. Normally the default value for the element is left empty so that the user must choose a value explicitly e.g. as with the `<ModelFile>`. In MpCCI GUI required elements are marked with a (*) at the end of the title. A dialog box is shown if required elements are not set while going on with the next step. The default for the required option is false.

![Figure 17: Required representation for a file selector element](image)

### 2.4.9 Testing "gui.xcf"

Testing of the files in the configuration directory is described in §2.2.2 Step-by-Step Procedure for Code Integration §.
2.5 Perl Scripts

You need not really learn Perl to write the scripts to run with your code. Basic documentation on the language (Linux/UNIX: man perl) is included in the Perl distributions, Schwartz et al. [2005] is recommended as a book for beginners.

So just edit the provided templates.

2.5.1 Using Information from "gui.xcf" in Scripts

As mentioned in the description of "gui.xcf" you need to pass information from the MpCCI GUI to the scanner, starter and stopper scripts. This information is exported from the MpCCI GUI into the Perl world via environment variables. These variable need to be defined in the "gui.xcf" as described in 2.4.7 Environments for Scanner, Starter, Stopper and Killer and are then evaluated within the Perl scripts. As a Perl programmer you can access the environment variables via the global Perl hash %ENV

```
$value = $ENV{'VARIABLE_NAME'};
```

During testing it may be helpful to read the variables with an automatic error check. For this purpose some Perl routines are provided in a Perl module MpCCICore::Env which needs to be included (use or require) in your Perl scripts.

- **env_optional_value(<variable name>)** get value of a variable, return an empty string if not defined.
- **env_required_value(<variable name>)** same as above, but exit with an error if variable is not defined.
- **env_boolean_value(<variable name>)** interpret value of the variable as Boolean value, the variable must be one of t\_true or 1 to yield “true” and f\_false or 0 to yield “false”.
- **env_optional_int(<variable name>, <default>, [<min>, [<max>]])** get value of a variable as an integer value. If the variable is not defined, the given default value is returned. If <min> and <max> are given, it is checked if the value lies within the given limits.
- **env_required_int(<variable name>, [<min>, [<max>]])** Same as above, but the variable must be defined, i.e. also a default value is not needed.

2.5.2 "Scanner.pm"

This file contains one subroutine which is called by MpCCI to scan the model file, which is named sub code_scanner($$$). It has three arguments:
$codename  The name of your code.

$modelFile  The name of the model file which shall be scanned.

$tmpName  The name of a temporary file, which can be used during the scanning process if necessary.

Sometimes the scanner does not only retrieve information but also applies some changes to the model file.

It returns two hashes (i.e. associative arrays):

%regionList contains all possible coupling components. The hash with the $regionName as key has the following structure:

```perl
%regionList{$regionName} = [$regionName, $regionType, $regionId];
$regionName = string with the name of the component for later use in adapter
$regionType = type of the region (references to Component Types in gui.xcf)
$regionID = number with ID of the component for later use in the adapter
```

See ▶ 2.4.4 Component Types: <ComponentTypeDimensions> for definition of the dimension of $regionType (=Typename) in "gui.xcf".

%modelInfo contains some general information on the model file and has the following structure:

```perl
MDIM => '3', # Model dimension: 1,2,3
CSYS => 'C', # Coordinate system: C=cartesian, S=Spherical, A=axis symmetric
SOLU => 'S', # Solution type: S=Static, T=Transient, D=Dynamic, C=Coupled
LCAS => '?', # Load cases: Number of loaded cases
UNIT => '?', # Unit system
PREC => 'D' # Precision: S=Single precision (32 bit),
D=Double precision (64 bit),
L=Long double precision (128 bit)
```

The "Scanner.pm" consists of three steps:

1. Parsing the model file. The file is opened, and a loop is run over all lines of the file. In the loop the file is scanned for coupling components. The search process must be adapted to your model file format to get the name, type and id of the component. A numerical id can also be omitted (specify 0) if your file does not use ids.

2. Adding further variables, which are not contained in the model file, but always present. Usually this is only the case for global variables.

3. Defining the basic model information, which can be retrieved during the scan or given as fixed values.
2.5.3 "Starter.pm"

"Starter.pm" contains one subroutine, which does not actually start the code, but provides the necessary information. The starter function obtains two arguments, \$codeName and \$modelFile. It must provide the command line arguments for starting the code and put it into the variable \@arg, which is the first return value.

Additionally, %infoHash is required to determine how the output of the simulation code is handled. The following options can be chosen:

<table>
<thead>
<tr>
<th>option</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STDOUT</td>
<td>null</td>
<td>What to do with data written to stdout, can be any of \xterm (i.e.) write to an xterm), \mpcci (i.e.) output handled by MpCCI, null (ignore output), combined with the keyword file if it shall also written to a log file.</td>
</tr>
<tr>
<td>STDERR</td>
<td>null</td>
<td>Same as STDOUT, but for stderr.</td>
</tr>
<tr>
<td>STDLOG</td>
<td>&lt;empty string&gt;</td>
<td>Name of logfile</td>
</tr>
<tr>
<td>BACKGR</td>
<td>white</td>
<td>Background color of xterm.</td>
</tr>
<tr>
<td>FOREGR</td>
<td>black</td>
<td>Foreground color of xterm.</td>
</tr>
<tr>
<td>ATSTART</td>
<td>&lt;not set&gt;</td>
<td>Reference to a Perl subroutine which is called before the code starts. You can define this routine in &quot;Starter.pm&quot;.</td>
</tr>
<tr>
<td>ATEXIT</td>
<td>&lt;not set&gt;</td>
<td>Reference to a Perl subroutine which is called as soon as the code exits. You can define this routine in &quot;Starter.pm&quot;.</td>
</tr>
</tbody>
</table>

As for the scanner, a testing script is provided for "Starter.pm", which is named "test_Starter.pl".

2.5.4 "Stopper.pm"

The stopper script should cause the code to perform a regular exit, i.e. not simply kill the code, but save data and quit. This is often achieved with a stop file, which is read by the code.

A test for the stopper is provided in "test_Stopper.pl" of the MpCCI API Kit.

2.5.5 "Info.pm"

In the script "Info.pm" the subroutine \sub code_information($) is defined. It has one argument:

\$codename The name of your code.

This subroutine will be called by MpCCI, e.g. when \sub code_print_releases or \sub code_print_info is called in "Subcmd.pm". It returns one hash (i.e. associative array) %infoHash:

\textbf{CODE_NAME} the official name of the code.

\textbf{CODE_HOME} the home path of the code installation.
**CODE_EXEC** the full pathname of executable to start the application.

**CODE_RLSE** the code internal release token.

**CODE_ARCH** the architecture token of the application.

**MPCCI_RLSE** the adapter release token (should be equal **CODE_RLSE** or **'STATIC'**).

**MPCCI_ARCH** the adapter architecture token (should be == **CODE_ARCH**).

Example:

```perl
$infoHash{$codeName} = [
    $codeName='solverxy',
    $codeDirectory='/opt/code',
    $pathToExecutable='$codeDirectory/bin/code.exe',
    $release='1.2.4',
    $architecture='linux_x86',
    $adapterRelease='$release',
    $arch='$architecture'
];
```

### 2.5.6 "Subcmd.pm"

In this Perl module exactly one public subroutine `sub code_subcommand($)` is defined. This subroutine is called from the MpCCI command as a code specific subcommand like `mpcci <codename> releases`.

This file is optional, but it is recommended to use it! The subroutine has one argument:

- **$codename** The name of your code.

You need to inspect the global Perl array `@ARGV` and then make decisions based on its contents. The simplest way of implementing a command line parser is to use the parsing tools which comes with MpCCI. You just define a Perl hash `%hash3` with the command line options as the key. The value for each command line option is a reference to an array containing three entries:

- the MpCCI environment setup level 0=no, 1=arch 2=core, 3=gui
- a short help message
- the name of an environment variable or a code reference

The Perl subroutine `sub code_subcommand($)` then has to call `hash3ref_run('option',%hash3)` to process the hash. `hash3ref_run` then either prints the help text, displays the value of the environment variable or calls the subroutine specified via the code reference.
The returned value of `sub code_subcommand($)` is irrelevant. `sub code_subcommand($)` may either call `exit(0)` or `exit(1)` in case of a failure or simply return.

### 2.5.7 Testing the Perl Scripts

For each script, a "test_* .pl" script is included in the MpCCI API Kit, which can be used to test the scripts separately.

Testing of the scripts in the configuration directory is described in [2.2.2 Step-by-Step Procedure for Code Integration](#).
2.6 MpCCI Coupling Manager Functions

The coupling manager functions are called by the calling code. They are defined in some MpCCI header files and you should add `#include "mpcci.h"` to the source files, from which you call the coupling manager functions.

Example calls are given "adapter.c" in the MpCCI API Kit.

The coupling manager functions are:

- 2.6.1 Definition of Output Functions: `MpCCI_Message_init`< on page 49
- 2.6.2 Initialization: `MpCCI_Init`< on page 50
- 2.6.3 Get Initial Exchange Mode: `MpCCI_Get_init_actions`< on page 52
- 2.6.4 Data Exchange: `MpCCI_Transfer`< on page 53
- 2.6.5 End of Coupled Simulation: `MpCCI_Exit`< on page 54
- 2.6.6 Definition of Nodes: `MpCCI_Def_nodes`< on page 55
- 2.6.7 Definition of Elements: `MpCCI_Def elems`< on page 56
2.6.1 Definition of Output Functions: `MpCCI_Message_init`

```c
void MpCCI_Commit_message_init( void (*printFullMessage)(const char *str, int len),
                                 void (*printFullWarning)(const char *str, int len),
                                 void (*printFullFatal  )(const char *str, int len),
                                 void (*printLineMessage)(const char *str, int len),
                                 void (*printLineWarning)(const char *str, int len),
                                 void (*printLineFatal  )(const char *str, int len),
                                 void (*atExitHandler)(void));
```

Before calling any other coupling manager function, the functions for input and output should be defined. There are seven different functions, the three first functions get multi-line strings for ordinary output messages, warnings and fatal messages together with the string length.

The second set of functions only receive one line at a time, i.e. the strings contain no newline characters (line-printer mode), which is useful for calling from FORTRAN.

It is not necessary to define all functions (give `NULL` pointers instead), but you should define either the first set of three functions or the second set of three functions.

⚠️ Do not use `printf` for output as the strings are already formatted and may contain percent (%) signs!
2.6.2 Initialization: \texttt{MpCCI\_Init}

The initialization function must be called from your code before the beginning of the iteration or time loop. However, the geometry of the mesh should already be known, as it is passed to \texttt{MpCCI} at this point.

```c
int MpCCI_Init(const char *jobId, const char *codeId, MPCCI_DRIVER_t *codeDriver);
```

The arguments are:
- `const char *jobId` : Job name, only needed for coupling a code with itself, should be \texttt{NULL} in other cases.
- `const char *codeId` : Name of your code.
- `MPCCI_DRIVER_t *codeDriver` : Pointer to list of driver functions. See \texttt{§ 2.7 MpCCI Driver Functions}.

The return value is a Boolean value and true if a connection to the server could be established. This can be used to detect whether a coupled or uncoupled simulation is run.

The initialization routine performs the following tasks:

- Check if initialization is already done and return in that case to avoid repeated initializations.
- Check if the supplied driver functions match the \texttt{MpCCI} version.
- Register driver functions, i.e. the pointers to the function pointers are saved to internal data structures.
- Set the code ID string to \texttt{codeName} if it is not set already via the \texttt{MpCCI} GUI (see also the description of \_\texttt{MPCCI\_CODE\_IDSTRING} in \texttt{§ 2.4.7 Environments for Scanner, Starter, Stopper and Killer}).
- Establish socket connection to the \texttt{MpCCI} server.
- Receive information from the server, i.e. all information defined in the \texttt{MpCCI} GUI, including the coupling components.
- Call the driver function \texttt{MpCCI\_Driver\_appendComponents} if it is defined.
- Close the list of coupling components, i.e. it cannot be altered any longer. If the list is empty, the connection to the server is closed as no coupling is necessary.
- Calls the driver function \texttt{MpCCI\_Driver\_updateComponents}, in which the numbers of nodes and elements of the coupling components can be defined.
- Loops over all coupling components:
  - Calls \texttt{MpCCI\_Driver\_selectComponent} (if defined) for each component.
  - Calls \texttt{MpCCI\_Driver\_defineGrid} to get the mesh definitions.
- Print the list of coupling components and establish communication channels for each coupling component.

- Wait until neighborhood search is complete.

- Return to the calling code.

Figure 18: Call of MpCCI_Init
2.6.3 Get Initial Exchange Mode: \texttt{MpCCI\_Get\_init\_actions}

This function is used to determine which kind of exchange was selected. In the \texttt{MpCCI} GUI for each code the option \textit{Initial quantities transfer} is set to \texttt{exchange}, \texttt{none}, \texttt{receive} or \texttt{send}. This information is given to the executed code via the environment variable \texttt{MPCCI\_INITIAL\_EXCHANGE}. Each of the possible values corresponds to a bit mask, which can be retrieved with \texttt{MpCCI\_Get\_init\_actions}.

\begin{verbatim}
int MpCCI_Get_init_actions (const char *actStr, int *actBits);
\end{verbatim}

The arguments are:
\begin{description}
\item[\texttt{const char *actStr}] is the default value for the initial exchange, should be \texttt{NULL} if no default value is selected or one of \texttt{"exchange"}, \texttt{"none"}, \texttt{"receive"} or \texttt{"send"}.
\item[\texttt{int *actBits}] Pointer to an integer variable to which the bit set will be stored.
\end{description}

The return value is one if an exchange mode could be found and zero if no mode could be determined.
2.6.4 Data Exchange: MpCCI_Transfer

The transfer function must be called from your code before or after the solver.

```c
int MpCCI_Transfer(int actMask, int doWait)
```

The arguments are:

- **int actMask**  
  Action mask. A bit set which defines the actions to be performed: Send and/or receive data. The possible values are:
  - QTACT_SEND send bits.
  - QTACT_RECV receive bits.

- **int doWait**  
  Wait flag for receiving data:
  - 1 Wait until other code sends data.
  - 0 Do not wait, keep the Receive bits set in the return value if no data was available from the other code.

The return value is:

- **openActions**  
  Action mask, which is empty if all actions given in `actMask` were performed. Otherwise it contains the actions which should be performed in the next call of `MpCCI_Transfer`, i.e. the return value should be used as `actMask` argument for the next transfer.

⚠️ Usually `actMask` is set to the value obtained by calling `MpCCI_Get_init_actions` for the first exchange and then to `actMask = QTACT_SEND | QTACT_RECV`. The waiting flag should be set to `doWait = 1`, then it is not necessary to check the return value, as sending is always possible (values are buffered) and `MpCCI_Transfer` waits until it receives some data.

The transfer routine performs the following tasks:

- If `actMask` indicates a send, for each coupling component:
  - Get the quantities from the application by calling the appropriate driver function `get...Values`.
  - Send quantities to the MpCCI server.

- If `actMask` does not indicate a receive, return to the calling code.

- Test if the requested information (quantities) is already available at the server.

- If not `doWait` is not set, return immediately with return value QTACT_RECV.

- Wait until the partner code has send information to the server.

- For each coupling component:
  - Receive the interpolated quantities from the server.
  - Copy quantities into the application by calling the appropriate driver function `put...Values`.

- Return to calling code.
### 2.6.5 End of Coupled Simulation: \texttt{MpCCI Exit}

```c
int MpCCI_Exit(int status, const char message[]);
```

The arguments are:
- **status**: Exit status, i.e. a number to identify the error
  - 0 means no error occurred.
- **message**: Error message, which is written with the number to the corresponding log files and/or the screen.

The return value corresponds to the return value of \texttt{MPI} and indicates if the \texttt{MPI} communication could be terminated. The only task of \texttt{MpCCI Exit} is:

- Close connection to the coupling server.

If the status is zero, normal program termination is assumed and the message is ignored.
2.6.6 Definition of Nodes: \texttt{MpCCI\_Def\_nodes}

The node definition should be called from the driver function \texttt{MpCCI\_Driver\_defineGrid} to define the nodes of a coupling component, see also "adapter.c" of the MpCCI API Kit. Only those nodes who send or receive data during the coupling process should be given.

The syntax is:

```c
int MpCCI\_Def\_nodes(int meshId, int partId, int globalDim,
                       int nNodes, int nNodeIds, const int nodeIds[],
                       int coordsDataType, const void* coords )
```

The parameters are:

- **int meshId**: mesh ID, should be set to \texttt{CCP\_MESHID(ccp)}.
- **int partId**: part ID, should be set to \texttt{CCP\_PARTID(ccp)}.
- **int globalDim**: dimension of your mesh, usually 2 or 3, should be equal to number of coordinates for each node.
- **int nNodes**: number of nodes of the component, can be simply set to \texttt{CCP\_NNODES(ccp)}.
- **int nNodeIds**: number of node IDs, should be set to same value as \texttt{nNodes}, i.e. \texttt{CCP\_NNODES(ccp)}.
- **const int nodeIds[]**: Array of the node IDs. The array size must be equal to \texttt{nNodeIds}.
- **int coordsDataType**: The data type of the node coordinates, possible values are:
  - \texttt{MPCCI\_FLOAT} C \texttt{float} type.
  - \texttt{MPCCI\_DOUBLE} C \texttt{double} type.
  - \texttt{MPCCI\_LONG\_DOUBLE} C \texttt{long\ double} type.
  - \texttt{MPCCI\_REAL} FORTRAN \texttt{real} type.
  - \texttt{MPCCI\_REAL4} FORTRAN \texttt{real*4} type.
  - \texttt{MPCCI\_REAL8} FORTRAN \texttt{real*8} type.
- **const void* coords**: pointer to array of coordinates of type \texttt{coordsDataType}. The size of the array must be \texttt{nNodes} \times \texttt{globalDim}.
2.6.7 Definition of Elements: MpCCI_Defelems

The element definition should be called from the driver function MpCCI_Driver_defineGrid after the node definition to define the elements of a coupling component, see also "adapter.c" of the MpCCI API Kit.

⚠️ All nodes used in MpCCI_Def elems must be already defined with MpCCI_Def_nodes.

The syntax is:

```c
int MpCCI_Def_elems(int meshId, int partId,
                    int nElems, int nElemIds, const int elemIds[],
                    int nElemTypes, const int elemTypes[],
                    const int nNodesPerElem[], const int elemNodes[] )
```

The parameters are:
- `int meshId` mesh ID, should be set to CCP_MESHID(ccp).
- `int partId` part ID, should be set to CCP_PARTID(ccp).
- `int nElems` number of elements, can be set to CCP_NELEMS(ccp).
- `int nElemIds` number of element IDs, same as nElems, set to CCP_NELEMS(ccp).
- `const int elemIds[]` array of element IDs. Size should be nElemIds.
- `int nElemTypes` number of element types, usually the same as nElems.
- `const int elemTypes[]` Element types. This can be one of the predefined element types, which are given in Table 1.
- `const int nNodesPerElem[]` Array of numbers of nodes per element. See also the more detailed description below.
- `const int elemNodes[]` Array of the node numbers (your node numbers) of the element nodes. See description below.

**Definition of Element Nodes**

The nodes of each element are given in two arrays. The first, nNodesPerElem contains one integer value for each element, which corresponds to the number of nodes for the element. The nodes for all elements are simply listed in elemNodes, the size of this array is therefore difficult to give, but it does not matter if it is too big.

So, if you have a 4-node element with nodes 1, 2, 3, 4 and two 3-node elements with nodes 11, 12, 13 and 21, 22, 23, you should define:

```c
int nNodesPerElem[] = { 4, 3, 3};
int elemNodes[] = { 1, 2, 3, 4, 11, 12, 13, 21, 22, 23 };`
```

⚠️ It is important to specify the nodes in the correct order! The order which you must use is given in Table 1.
Definition of Element Types

The element types are listed in Table 1.

The element types can be given in two ways: Either you have a mesh, which consists of elements of the same type. In this case you can set `nElemTypes=1` and `elemTypes[] = <$/> Element type`, i.e. the list of element types only consists of one type, which is valid for all elements.

It is also possible to call `MpCCI_Def elems` several times from `MpCCI_Driver_defineGrid` to define elements of different types.

If you have a mesh which consists of different types you can give the type for each element. Then `nElemTypes` should have the same value as `nElems` and the array `elemTypes` should contain one type for each element.

Table 1: Element types in the MpCCI code API
2.7 MpCCI Driver Functions

The driver functions are responsible for the exchange of data between the code adapter and the application. To enable a coupled simulation a set of functions must be provided. Exemplary implementations of the necessary driver functions are given in "adapter.c" of the MpCCI API Kit. A description of all functions is given on the following pages.

The driver functions are called by the coupling manager functions as described in \ref{2.6 MpCCI Coupling Manager Functions}, therefore they can be regarded as the passive part of the adapter.

A structure of pointers to these functions is handed over to MpCCI via the \texttt{MpCCI_Init} routine, see also "adapter.c" of the MpCCI API Kit and \ref{2.6.2 Initialization: \texttt{MpCCI_Init}}. The names of the functions can be chosen arbitrarily. Only two functions are \textit{required} functions, which must be provided, all other functions need not be defined, a \texttt{NULL}-pointer can be given instead. The complete structure without \texttt{NULL} pointers is:

\begin{verbatim}
typedef struct _mpcci_driver_info
{
    /* REQUIRED: information to do a compatibility check */
    size_t this_size;    /* sizeof this "struct _mpcci_driver_info" */
    unsigned this_version; /* version 3.0.6=306, 3.1.0=310 */

    /* REQUIRED: pointer to a static string containing the name of the code */
    const char *code_name;

    /* REQUIRED: define symbolic names for the types of regions in code terminology
    const char *region_description[4];

    /* REQUIRED: sizeof(float|double): tell the manager the size of floating point
                * values that are arguments to the set of putXXXValues() functions */
    size_t values_size;

    /* REQUIRED: tell the manager what to do */
    unsigned ccm_action;    /* bitmask: __QTACT..... */

    /* OPTIONAL: methods called before/after some actions */
    void (*appendComponent) (int ndefined);    /* store/copy component names in CCLIST */
    void (*afterCloseSetup) (void);           /* method called after def_close */
    void (*componentSelect) (const CC_DESC_t *ccp);
    void (*beforeGetAndSend)(void);           /* method called before send operation */
    void (*afterGetAndSend) (void);           /* method called after send operation */
    void (*beforeRecvAndPut)(void);           /* method called before receive operation */
    void (*afterRecvAndPut) (void);           /* method called after receive operation */
\} _mpcci_driver_info;
\end{verbatim}
/* REQUIRED: methods to get information about a component
* REQUIRED: method to update and check component names - may be C or FORTRAN code
* returns != 0, if this is an MpCCI process == valid coupled component found
* REQUIRED: method used to define the grid (nodes & elements) */
int (*updateComponents)(void);
void (*defineGrid) (CC_DESC_t *ccp);
void (*moveNodes) (CC_DESC_t *ccp);

/* OPTIONAL(depends): methods used to get quantities from the application */
int (*getGlobalValues )(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
int (*getLineNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
int (*getLineElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
int (*getFaceNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
int (*getFaceElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
int (*getVoluNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
int (*getVoluElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);

/* OPTIONAL(depends): methods used to store quantities into the application */
void (*putGlobalValues )(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
void (*putLineNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
void (*putLineElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
void (*putFaceNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
void (*putFaceElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
void (*putVoluNodeValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
void (*putVoluElemValues)(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);

} MPCCI_DRIVER_t;
2.7.1 Description Values

`size_t this_size`
The size of the structure, must be set to `sizeof(MPCCI_DRIVER_t)`.

`unsigned this_version`
The version of the code adapter, i.e. the MpCCI version, without dots. Should currently be set to 306 to match this description.

`const char *code_name`
Should be set to the name of the simulation code.

`const char *region_description[4]`
An array of descriptions to improve output. You should give a set of names that match the terminology of your code, e.g.

```
{ "global value", /* name for global quantities */
  "edge",    /* name for lines */
  "surface", /* name for faces */
  "body" }    /* name for volumes */
```

`size_t values_size`
The size of the values given to the put functions, usually set to one of `sizeof(float)` or `sizeof(double)`.

`unsigned ccm_action`
This defines the actions of the code coupling manager (ccm). In most cases it should be the combination `QTACT_SEND|QTACT_RECV`, i.e. the adapter sends and receives data. Other values should only be used in special case, e.g. codes with several processes, where only one process communicates with MpCCI.
2.7.2 Methods Called before/after some Action

`void appendComponent(int mdefined)`

This method is intended for future use only. It is planned that coupling components can be added automatically and need not be selected in the MpCCI GUI.

⚠️ Please do not use this method yet. All components can be chosen in the MpCCI GUI.

`void afterCloseSetup(void)`

This method is called at the end of the setup of coupling components.

It is usually not necessary to define this function.

`void * componentSelect(const int *compId, const char *compName, int len)`

You can use this driver function to remember which components were selected in the MpCCI GUI for coupling, it is called once for each component, which is given in `ccp`. The id and name and the length of the name are given as arguments.

In most codes this function is not needed.

`void beforeGetAndSend(void)`

This method is called before any get functions are called and data is send. Define this method if you need to perform any actions before the get functions can read data.

`void afterGetAndSend(void)`

This method is called after all required get functions were called and the data was transferred to the MpCCI server.

`void beforeRecvAndPut(void)`

This method is called before data is received and before appropriate put functions are called.

`void afterRecvAndPut(void)`

This method is called after the put functions were called.
2.7.3 Mesh Definitions

The mesh definition functions are called if information on the meshes is needed.

```c
int updateComponents(void)
```

This function is called during `MpCCI_Init`. You should update the components in this function, namely the number of nodes and elements of each component, for details please see "adapter.c" of the MpCCI API Kit. The tasks are:

- Loop over all coupling components (with `CCP_LOOP_ALL`)
  - Set number of elements with the command `CCP_NELEMS(ccp) = ...` and the number of nodes with `CCP_NNODES(ccp) = ...` for the coupling component. This often depends on the dimension of the component which can be checked with `CCP_DIMENSION(ccp)`.

- If any valid coupling components were found, return 1, otherwise 0. If a zero is returned, the process is not coupled, which can appear in parallel processes, where only some processes have coupling components.

```c
void defineGrid(CC_DESC_t *ccp)
```

This function is required. You should define nodes and elements of the coupling component given by `ccp`:

- Set number of nodes with `MpCCI_Def_nodes` (see §2.6.6 Definition of Nodes: `MpCCI_Def_nodes`).

- Set number of elements with `MpCCI_Def_elems` (see §2.6.7 Definition of Elements: `MpCCI_Def_elems`).

```c
void moveNodes(CC_DESC_t *ccp)
```

This function is intended for future use. Do not define it!
2.7.4 Data Exchange

There are two kinds of exchange functions. The get functions read data from the code and are called before sending data to the partner code via the MpCCI servers. The put functions are called after data was received and must write the data to the code.

The get functions are all similar and have the same syntax:

```c
int get...Values(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
```

The parameters are:
- `const CC_DESC_t *ccp`  Pointer to coupling component structure.
- `const CQ_DESC_t *cqp`  Pointer to quantity structure.
- `void *values`          Pointer to data array.

Return value:
- `int floatSize`  Size of the floating point values which are written to `values`.
  - Typically this is `sizeof(float)` or `sizeof(double)`.

In the get functions, the required values must be to the compact data array, which is already allocated. The pointer is given as parameter. You can give the data in any floating point format. The length of the array is given by `CCP_NNODES(ccp) * CQP_DIM(cqp)` for nodal values and `CCP_NELEMS(ccp) * CQP_DIM(cqp)` for element values. `CQP_DIM(cqp)` is the dimension of each value, e.g. 3 for a vector.

The get functions are distinguished by the component dimension, `Global`, `Line`, `Face` or `Volu`, and by the location, i.e. `Node` or `Elem`. You only need to define those functions, which are supported by your code, i.e. if your code only supports coupling on faces with nodal values, only `MPCCI_DRIVER_getFaceNodeValues` must be defined.

In each function, loop over nodes or elements and collect the values. These values must be written to the array, which `values` points to. The size of the array is given by `CCP_NNODES(ccp) * CQP_DIM(cqp)` for nodal values and `CCP_NELEMS(ccp) * CQP_DIM(cqp)` for element values. `CQP_DIM(cqp)` is the dimension of each value, e.g. 3 for a vector.

The order of nodes and elements is given by the order in which they are defined in `CCI_Def_nodes` and `CCI_Def_elems` in `MpCCI_Init`.

```c
void put...Values(const CC_DESC_t *ccp, const CQ_DESC_t *cqp, void *values);
```

The parameters are the same as for the get functions. The data array is now filled with values which can be read and transferred to the mesh. The values are of the type given as `values.size` in the driver function structure, see 2.7 MpCCI Driver Functions.
2.8 Data Structures and Predefined Macros

MpCCI uses some predefined macros, which can be used in function calls and are described here briefly. Most of the macros described below are defined in the files "<MpCCI_home>/include/ccllib.h" and "<MpCCI_home>/include/mpcciconst.h", which include short descriptions as well.

2.8.1 Coupling Components

The coupling component descriptor `CC_DESC_t` represents a coupling component for the code. Many driver functions obtain an argument `ccp` which is a pointer to the coupling component (coupling component pointer).

The structure contains the basic properties of a component and a list of the quantities, which shall be exchanged. The most important macros for `CC_DESC_t` are listed below, all macros take an argument of type `CC_DESC_t *`, thus they can be applied directly to the pointer `ccp`. E.g. `CCP_NNODES(ccp)` should be used to set or determine the number of nodes of a component.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char *</td>
<td>CCP_NAME(ccp)</td>
<td>name of coupling component</td>
</tr>
<tr>
<td><code>CC_DIM_t</code></td>
<td>CCP_DIMENSION(ccp)</td>
<td>dimension of component. Possible values are: <code>CC_DIM_DUMMY</code> (dummy value), <code>CC_DIM_GLOBAL</code> = 0 (scalar), <code>CC_DIM_LINE</code> = 1 (line), <code>CC_DIM_FACE</code> = 2 (face), <code>CC_DIM_VOLUME</code> = 3 (volume)</td>
</tr>
<tr>
<td>int</td>
<td>CCP_NQUANT(ccp)</td>
<td>number of quantities to transfer</td>
</tr>
<tr>
<td>int</td>
<td>CCP_NNODES(ccp)</td>
<td>number of nodes/points in component</td>
</tr>
<tr>
<td>int</td>
<td>CCP_NELEMS(ccp)</td>
<td>number of faces/elements/cells in component</td>
</tr>
<tr>
<td>int</td>
<td>CCP_NODMAX(ccp)</td>
<td>max. number of nodes per element in this component</td>
</tr>
<tr>
<td>int</td>
<td>CCP_MESHID(ccp)</td>
<td>MpCCI mesh id</td>
</tr>
<tr>
<td>int</td>
<td>CCP_PARTID(ccp)</td>
<td>MpCCI partition id</td>
</tr>
<tr>
<td>void *</td>
<td>CCP_AUXPTR(ccp)</td>
<td>pointer to auxiliary optional information defined and used by the application</td>
</tr>
<tr>
<td><code>CQ_DESC_t</code>*</td>
<td>CCP_GETCQP(ccp)</td>
<td>list of quantities to transfer</td>
</tr>
<tr>
<td>bool</td>
<td>CCP_ISGLOBAL(ccp)</td>
<td>check whether component represents global values</td>
</tr>
<tr>
<td>bool</td>
<td>CCP_ISLINE(ccp)</td>
<td>check whether component is a line</td>
</tr>
<tr>
<td>bool</td>
<td>CCP_ISFACE(ccp)</td>
<td>check whether component is a face</td>
</tr>
<tr>
<td>bool</td>
<td>CCP_ISVOLUME(ccp)</td>
<td>check whether component is a volume</td>
</tr>
</tbody>
</table>

See also ▶2.8.3 Loop Functions◀ for loops over components.
## 2.8.2 Quantities

The quantity descriptor \texttt{CQ\_DESC\_t} represents one quantity. It contains the name and properties of the quantity.

<table>
<thead>
<tr>
<th>Return value</th>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{const char *}</td>
<td>CQP_NAME(cqp)</td>
<td>symbolic name e.g. “temp”, “JHeat” . . .</td>
</tr>
<tr>
<td>int</td>
<td>CQP_IDC(cqp)</td>
<td>ID code of quantity (see also the Quantity Reference in the Appendix)</td>
</tr>
<tr>
<td>int</td>
<td>CQP_DIM(cqp)</td>
<td>dimension of quantity: 1=scalar, 3=vector etc. .</td>
</tr>
<tr>
<td>int</td>
<td>CQP_LOC(cqp)</td>
<td>location of value, one of: \texttt{CCI_QUANTITY_LOCATION_NODE} \texttt{CCI_QUANTITY_LOCATION_ELEM} \texttt{CCI_QUANTITY_LOCATION_GLOBAL}</td>
</tr>
<tr>
<td>int</td>
<td>CQP_FTYP(cqp)</td>
<td>type of quantity, one of \texttt{CCI_QUANTITY_TYPE_MESH} \texttt{CCI_QUANTITY_TYPE_FIELD} \texttt{CCI_QUANTITY_TYPE_FLUX} \texttt{CCI_QUANTITY_TYPE_USERDEF}</td>
</tr>
<tr>
<td>int</td>
<td>CQP_IPOL(cqp)</td>
<td>default type of interpolation, one of: \texttt{CCI_QUANTITY_IPOL_MAX} maximum value is taken \texttt{CCI_QUANTITY_IPOL_MIN} minimum value is taken \texttt{CCI_QUANTITY_IPOL_SUM} sum \texttt{CCI_QUANTITY_IPOL_PROD} product</td>
</tr>
<tr>
<td>int</td>
<td>CQP_PMEAN(cqp)</td>
<td>physical meaning of this quantity, one of \texttt{MPCCI_QPM_UNDEF} physical meaning of quantity really undefined \texttt{MPCCI_QPM_ANY} quantity is not further specified \texttt{MPCCI_QPM_SRC_MASS} a mass sink/source \texttt{MPCCI_QPM_SRC_MOM} a momentum sink/source \texttt{MPCCI_QPM_SRC_ENTH} an energy sink/source \texttt{MPCCI_QPM_PROP_MAT} a property (material) \texttt{MPCCI_QPM_BC_VALUE} a boundary condition value \texttt{MPCCI_QPM_BC_GRAD} a boundary condition wall normal gradient \texttt{MPCCI_QPM_GRID} a grid coordinate/displacement \texttt{MPCCI_QPM_CHEMCOMP} a chemical component concentration</td>
</tr>
<tr>
<td>int</td>
<td>CQP_TDIR(cqp)</td>
<td>transfer direction, one of \texttt{CQ_TDIR_NONE} no transfer \texttt{CQ_TDIR_SEND} quantity is sent \texttt{CQ_TDIR_RECV} quantity is received</td>
</tr>
<tr>
<td>int</td>
<td>CQP_TSTATE(cqp)</td>
<td>state of transfer, one of \texttt{CQ_TSTATE_UNDEF} no transfer was done before \texttt{CQ_TSTATE_SEND} quantity was sent</td>
</tr>
<tr>
<td>Return value</td>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>int</td>
<td>CQP_SMETHOD(cqp)</td>
<td>how to store received quantities, one of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CQ_TSTATE_RECV quantity was received</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPCCI_QSM_UNDEF invalid send method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPCCI_QSM_DIRECT directly written into buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPCCI_QSM_BUFFER quantity is buffered locally and copied later</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPCCI_QSM_USRMEM quantity to store in users indexed memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPCCI_QSM_SCALAR quantity to store in users index scalars</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPCCI_QSM_SPECIES quantity to store the chemical species</td>
</tr>
<tr>
<td>int</td>
<td>CQP_SINDEX(cqp)</td>
<td>index to user supplied memory or user scalars</td>
</tr>
<tr>
<td>void *</td>
<td>CQP_SBUFFER(cqp)</td>
<td>quantities local transfer buffer</td>
</tr>
<tr>
<td>int</td>
<td>CQP_SBFSIZE(cqp)</td>
<td>size of allocated local buffer</td>
</tr>
</tbody>
</table>

### 2.8.3 Loop Functions

In addition to the data access functions, a number of predefined loops are available:

**CCP_LOOP_ALL(ccp,i){ ... }**

Loop over all coupling components.

Parameters:
- CC_DESC_t* ccp  
  Is set to coupling components during loop.
- int i            
  Loop counter.

**CCP_LOOP_VALID(ccp,i){ ... }**

Loop over all valid coupling components, i.e. all components with dimension other than CC_DIM_DUMMY.

Parameters:
- CC_DESC_t* ccp  
  Is set to coupling components during loop.
- int i            
  Loop counter.

**CQP_LOOP_ALL(ccp,cqp,cqpEnd){ ... }**

Loop over all quantities of a coupling component given by ccp.

Parameters:
- CC_DESC_t* ccp  
  Coupling component, must be given.
- CQ_DESC_t* cqp  
  Variable is set to quantities during loop.
- CQ_DESC_t* cqpEnd  
  Variable is set to end of loop (not used).
CQP_LOOP_RECV (ccp, cqp, cqpEnd) { ... }  
Like \texttt{CQP_LOOP\_ALL}, but loop goes only over quantities which are received.

CQP_LOOP_SEND (ccp, cqp, cqpEnd) { ... }  
Like \texttt{CQP\_LOOP\_ALL}, but loop goes only over quantities which are sent.
3 MpCCI SDK Code Coupling Library

3.1 The MpCCI SDK Concepts

3.1.1 Communication levels

In the layered approach described in detail below we have in mind that MpCCI SDK is situated between the application and the MPI library. Similar to the Message Passing Interface MPI, MpCCI SDK uses the communicator concept. Instead of just transferring data from one process to another MpCCI SDK takes into account the grids on which the data are located. Thus tools for neighborhood search and interpolation are provided. Additionally, MpCCI SDK is able to support several grids used within each of the codes. This concept is obviously required in the multigrid context.

MpCCI SDK is independent of the parallelization strategy used by a code. It integrates SPMD (single program multiple data) as well as MPMD (multiple program multiple data) programs. The user can use the codes he is familiar with. No training for new application software packages is required. Only minimal source code changes of the simulation codes themselves are necessary.

While the codes internally may use a communication platform of their own choice (e.g. MPI, PVM or Shared Memory) MpCCI SDK uses a TCP/IP based socket connection for communication between the codes and the MpCCI SDK Coupling Server. Internally the MpCCI SDK Coupling Server uses MPI as its parallelization platform.

For each parallel process in the coupled application codes there is exactly one corresponding mirror process in the MpCCI SDK Coupling Server. Communication between code process and MpCCI SDK process is handled bidirectional - an exclusive socket connection is established a the beginning of the coupled job for each pair of code process and MpCCI SDK mirror process.

3.1.2 Coupling quantities

The values transferred by MpCCI SDK are referred to as quantities. A quantity can be any type of variable used in any of the codes. An arbitrary dimension can be assigned to these quantities. Velocity, temperature, size of time-steps are typical examples for quantities. Obviously there are two types of quantities: On the one hand, some quantities are represented by discrete functions, evaluated during the computation on an arbitrary mesh. Those values are assigned to a specific geometric position. On the other hand, there are quantities represented by one or more single values, which are independent of a geometrical position. The former are referred to as grid-based quantities, whereas the latter are described as global quantities.

A grid in MpCCI SDK consists of a certain amount of parts, called coupling partitions. Each partition consists of a certain amount of geometrical positions referred to as nodes. Nodes can be combined using various types of elements in order to specify connectivity information used in the simulation codes. In the multigrid context this structure is applied to any of the grid levels separately.

Each grid-based quantity has to be assigned to exactly one grid. On the other hand, more than one quantity can be defined on each grid. Further, more than one grid can exist in each of the simulation codes.
participating in the coupled computation. Application of a multigrid algorithm could be one reason for using several grids covering the same geometrical region. Chimera techniques can be considered a more prominent example for the existence of several grids in one code. A staggered grid can be seen as a number of grids covering the same geometrical position.

### 3.1.3 Synchronization concepts and data transfer

Data transfer is one of the main tasks of MpCCI SDK. Coupling values are transferred both transparently and efficiently. Full flexibility as well as high comfort are offered to the user. To ensure a high degree of scalability and performance communication takes place exclusively among processes participating in the coupled computation.

MpCCI SDK supports two kinds of concepts for the data exchange.

Blocking and non-blocking communication can be invoked by routines similar to the according routines in MPI. This concept is MPI-like based on send and receive actions and will not be described in this paragraph. For more information see the descriptions of the subroutines (CCI\texttt{send} and CCI\texttt{recv}) in §3.2 MpCCI SDK Functions or read the example in §3.4 An Example.

The second concept is based on so-called synchronization points which offers the opportunity of bi-directional communication: Reaching a synchronization point a process may send and receive data. Only those quantities are transferred to other processes, that are required by the remote processes. An illustration is given in Figure 1. Codes taking part in a synchronization point may be defined in a flexible way.

![Figure 1: Exchange of data via synchronization points](image)

An example for a simple algorithmic use of synchronization points is given in Figure 2.

For a more detailed description see the definitions of the subroutines (CCI\texttt{defsync}, CCI\texttt{syncinfo} and CCI\texttt{reachsync}) in §3.2 MpCCI SDK Functions and §3.3.9 Coupling block.
3.1.4 Coupling Regions

MpCCI SDK supports different kinds of coupling regions and spaces. In the two-dimensional and three-dimensional space line, surface and volume coupling based on element definitions are possible. Meshes consisting of spherical triangles or (exclusive) quadrilaterals are treated as a special case. Furthermore MpCCI SDK is able to perform data transfer on nodes without element definition.

3.1.5 Neighborhood Search and Interpolation

The exchange of grid based coupling quantities is a non-trivial task. Due to the basic discretization in the simulation codes the coupling data are usually arranged on different structured or unstructured meshes. As an example the grid of one multigrid level of each code is shown in Figure 3. Therefore, MpCCI SDK has to perform neighborhood search and interpolation between the meshes to realize a sensible exchange of coupling data. Each of the meshes is usually distributed on several parallel processes. The complex problem raised by this distribution is solved by MpCCI SDK efficiently.

The main problem is to determine how these surfaces fit together, the so-called neighborhood computation. This information is used then to establish the inter-processor communication, and to assist the codes in
the interpolation of coupling values between the different grids.

The simulation codes must specify their coupling region to MpCCI. The coupling regions may consist of the element types described in \ref{CCIdefelems}. The coordinate systems must be global in the MpCCI SDK standard version to enable MpCCI SDK to detect the distances and relations between the specified regions. It is allowed that the coupling regions are not identical but distant from each other. The maximum distance allowed is adjustable by a parameter in the input file of MpCCI. In the case of non-matching grids the data arranged on the nodes of the source grid are interpolated by MpCCI SDK on the nodes of the target grid. In MpCCI SDK advanced and professional version coordinate systems can be transformed so that they match each other.

In the coupling definition phase each process of a code announces its parts of the coupling region with their nodes and elements of the coupling surface to MpCCI. Each coupling partition has to be assigned with a partition identifier (see \ref{MpciSDKFunctions}) which must be used in subsequent MpCCI SDK calls to indicate the involved part of the coupling region. When each code has specified the grid data of its side of the coupling surface, MpCCI SDK performs the contact or neighborhood search which is necessary for

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3}
\caption{Data exchange between two non-matching grids (distance exaggerated)}
\end{figure}
the interpolation.

A linear and a bilinear interpolation depending on the type of the element is implemented. Also, the user can choose between a conservative and a non-conservative interpolation (see &gt; V-3.2.5 Flux and Field Interpolation &lt; for details).

With respect to the whole coupling process the main computation is carried out in the communication phase. In this phase the coupling data are specified to the library. MpCCI SDK sends the data to the receiver code, interpolates the data onto the nodes of the target grid and distributes the interpolated data to the specified arrays. Then the computations of the receiver code, which depend on the coupling data, can start.

Based on this information the communication is optimized in the following way: Only those processors exchange data, that have grid points for which a connectivity was detected, instead of distributing all data to all processors.

3.1.6 The MpCCI SDK coupling server scheme

Since MpCCI 3.0.5 MpCCI SDK only works with the concept of "coupling servers", where each code lives in its own MPI world (if it is parallel in itself) and is started independently of the others.

This chapter describes the general idea of the coupling server scheme and names more advanced features.

3.1.6.1 General Idea of the Coupling Server Scheme

The idea of the coupling server scheme is to separate the codes into their own independent communication worlds: each code uses its own MPI or another communication platform for its internal parallelization. There are no restrictions or interferences with the MPIs of the remote codes. The codes are started independently in the same way as in the standalone case. The coupling of codes is realized via additional "coupling server" processes (one for each code process). MpCCI SDK internally uses MPI for communication between its parallel server processes. In case of two coupled codes there are three a priori independent "communication and parallelization worlds":

- the processes of code A
- the processes of code B
- the MpCCI SDK coupling server processes, one process for each code process of A and B

The "coupling server" processes execute the special binary "mpcci-server" which is provided by the MpCCI SDK distribution. Each coupling server process is associated to exactly one code process. The coupling server does nothing else but listen for MpCCI SDK requests from its code process. If such an MpCCI SDK request arrives, the coupling server calls the corresponding MpCCI SDK subroutine, and communicates the results of the MpCCI SDK subroutine back to its code process.
The communication between a coupling server and its code process is not MPI based (because they do not live in the same MPI world), but instead based on sockets. The code processes themselves need to be linked with the "libmpccisdk_32.a" or "libmpccisdk_64.a" communication library.

### 3.1.6.2 Advanced Features

Let us begin with a few technical words about the setup phase of the MpCCI SDK Coupling Server. This will help you understand some of the advanced configuration parameters.

How do the server processes find their clients? This is non-trivial because the server processes and the clients are started independently of each other.

In our implementation, the first MpCCI-server process (the so called ”leading MpCCI-server”) plays a special role: it mediates the communication setup between each code process and its corresponding MpCCI-server. This is done in the following way:

- The leading MpCCI-server knows from the MpCCI SDK input file, how many clients are to be expected. It opens a socket connection and listens to a specific port for client registration messages.

- The clients, i.e. the processes of the codes read a setup file "${HOME}/.cciclientrc" (if the variable CCI_CLIENT_HOME is set and a writable directory: "${CCI_CLIENT_HOME}/.cciclientrc") which has been written by the `ccirun` script when starting the MpCCI SDK world of the MpCCI-servers. This setup file contains information about the IP address and the port number of the leading MpCCI-server. So the first action of each client during `CCIinit` is to contact the leading MpCCI-server for registration.

- After receiving the registration message from a code process, the leading MpCCI-server associates one of the MpCCI-servers to the client. It sends the IP address and a port number to both the client and the corresponding MpCCI-server. The relevant MpCCI-server then listens to this port, and the client connects to this port. In this way, the client sets up a connection to its MpCCI-server. When it is set up, all subsequent communication between client and server will be handled via this connection.

⚠️ The hand-over of port information to establish socket connections will change with MpCCI 3.0.6. Instead of creating a file "${HOME}/.cciclientrc" for each client process the user has to define environment variables describing the host and port number of the leading MpCCI-server.

In addition to this setup mechanism where the leading MpCCI-server mediates the connections, there is an alternative setup where clients and servers find themselves directly. This direct setup requires additional information both for the MpCCI-servers and the clients.

There are a lot of configuration switches by means of which the user can adjust the behavior of the coupling servers. They are described in ❯ 3.2 MpCCI SDK Functions ❮
3.2 MpCCI SDK Functions

This chapter describes the basic MpCCI SDK subroutines needed to perform the coupling algorithms described previously. Before we start with a detailed description of the individual calls we give a brief introduction to the terminology used. The reader who is familiar with FORTRAN, C and MPI may skip 3.2.1 Naming Conventions and Terminology. For a brief overview of the complete set of MpCCI functions the reader is referred to Table 4 on page 146 and to the index list at the end of this manual.

3.2.1 Naming Conventions and Terminology

The naming conventions for MpCCI SDK subroutines are similar to those in MPI. Each subroutine name consists of a list of words separated by underscores ".". Subroutine names start with the word “CCI” as in “Code Coupling Interface”. The first letter of the second word is capitalized as in CCI_Send. The FORTRAN versions will have names that are all upper case, e.g. FCCsend. To achieve portability, the names of MpCCI SDK subroutines will be at most 30 characters long. The names of non-blocking routines start with CCI_ (followed by lower case letters in C). For example, the non-blocking counterpart of CCI_send is CCI_isend, in FORTRAN it is FCCisend.

According to Snir et al. [1996] we will use the following terminology in the description of MpCCI SDK subroutines:

- **local**: If the procedure of the subroutine depends only on the local executing process. Such an operation does not require an explicit communication with another process.

- **non-local**: If completion of the procedure may require the execution of a MpCCI SDK procedure on another process.

- **blocking**: If return from a MpCCI SDK procedure indicates the user is allowed to re-use resources specified in the call. Any visible change in the state of the calling process is completed before the call returns.

- **non-blocking**: If the procedure may return before the operation initiated by the call completes, and before the user is allowed to re-use resources (such as arrays specified in the call).

- **collective**: If all processes in a certain process group must invoke the procedure.

In addition to MPI, the following terminology is used:

- **one-side collective**: If completion of the operation initiated by the call is a collective operation over the MpCCI SDK processes of the local code.

- **two-side collective**: If completion of the operation initiated by the call depends on the MpCCI SDK processes of both the local and remote code.
MpCCI SDK procedures are specified in language independent notation. Subroutine arguments are marked as

- **IN**, for arguments whose values may be used but are not changed,
- **OUT**, for arguments whose values are not used on input, but may be changed by the call, or
- **INOUT**, for arguments whose values may be used on input to the call, and may be changed by the call.

Arguments of type OUT may have arbitrary input values. Aliasing of arguments in procedure calls is only allowed if these arguments are all IN variables. Otherwise, the results are undefined.

Concrete C and FORTRAN descriptions are added for MpCCI SDK subroutines that have been implemented. Note that all MpCCI SDK subroutines in FORTRAN have an additional argument **ierror** at the end of the argument list. This argument is a FORTRAN integer and has the same meaning as the return value of the routine in C. In FORTRAN, MpCCI SDK and MPI routines are subroutines, and are invoked with the call statement.

The index order of array elements differs in FORTRAN and C: In our examples we refer to the FORTRAN index ordering where the first element of an array **ary** is **ary(1)** 1.

MpCCI SDK subroutines for receiving coupling values have a return status output argument. This argument contains more detailed information about the information received.

In C, **status** is a structure of type **CCIstatus**. In FORTRAN, it is an integer array which has the length **CCIstatussize**. The constant **CCIstatussize** is defined in "ccif.h".

### 3.2.2 MpCCI SDK Data Types

Some of the MpCCI SDK functions described in the following have an argument concerning the data type of values. To avoid problems with the different data types on various platforms MpCCI SDK defines constants of its own that should be used.

The different data types for C and FORTRAN are collected in the tables below.

**Table 1** lists the available MpCCI SDK data types for C:

**Table 2** lists the available MpCCI SDK data types for FORTRAN.

### 3.2.3 Initialization and Coupling Definition

Connections between different codes are established by MpCCI SDK in the start-up and initialization phase.

---

1In C it would be **ary[0]**.
In the MpCCI SDK initialization phase, several constants are computed. To access these constants from C, one must use:

```c
#include "mpcci.h"
```

To access these constants from FORTRAN, one must use the following:

```c
c        --- include common block containing MpCCI constants
#include 'mpccif.h'
```

In the coupling definition phase, each code specifies its coupling region to MpCCI. This coupling region in general consists of the part of the grid coordinates that take part at the coupling. After that the initial neighborhood computation takes place. The search algorithm is explained in \( \text{V-3.2.1 Pre-Contact Search} \).
CCIinit

\begin{verbatim}
int CCI_Init( int* pArgc, char*** pArgv )

subroutine CCI_INIT( ierror )
integer ierror
\end{verbatim}

This subroutine enrolls a code in the coupled computation. CCIinit initializes MpCCI SDK for both C and FORTRAN/FORTRAN 90 usage. Afterwards, the complete environment for the caller has been set as specified in the MpCCI SDK input file (working directory, environment variables).

CCIinit initializes the following external variables defined in "mpcci.h" (for C), "mpccif.h" (for FORTRAN):

- **CCIncodes**
  The number of codes (jobs) involved in the current computation.

- **CCImycodeid**
  The code id of the current code (job), \( 1 \leq \text{CCImycodeid} \leq \text{CCIncodes} \).

A code id CCImycodeid is always in the range \( 1 \leq \text{CCImycodeid} \leq \text{CCIncodes} \). If the same code occurs twice in the coupled computation, CCImycodeid will have different values at the different instances of this code.

CCIinit is a blocking collective operation over all processes of all codes, to be performed after start-up of a code, and before any other MpCCI SDK subroutines are called. No MPI calls should take place between call to MPI_Init and CCIinit, and no I/O should take place before CCIinit. Also the communicator MPIcommworld should not be used for any communication in the whole coupled computation.

- Class: non-local, blocking
- Position: first MpCCI SDK call
### CCIinitwithidstring

- **pArgc**: INOUT pointer to pass the argument count to `MPI_Init`
- **pArgv**: INOUT pointer to pass the arguments to `MPI_Init`
- **idString**: IN id string

```c
int CCI_Init_with_id_string( int* pArgc, int*** pArgv, const char* idString )
```

```fortran
subroutine CCI_INIT_WITH_ID_STRING( idString, ierror )
```

- `character(*) idString`
- `integer ierror`

In addition to the task of the subroutine `CCIinit` this subroutine offers the possibility to overwrite the id string of the code block of the MpCCI SDK input file with the string `idString`. The subroutines `CCIinit` and `CCIinitwithidstring` must not be called both in one code.

- **Class**: non-local, blocking
- **Position**: first MpCCI SDK call
**CCIdefmesh**

meshId IN grid identifier within each code
idString IN identifier(s) of the mesh

```c
int CCI_Def_mesh( int meshId, const char idString[] )

subroutine CCI_DEF_MESH( meshId, idStringLen, idString, ierror )

integer meshId, idStringLen, ierror
caller*() idString
```

Subroutine **CCIdefmesh** enables the user to define expressive names for the different meshes. These names can be used in the coupling block of the MpCCI input file.

The argument **idString** may contain different names separated by comma. All sub-strings are compared with those of the other code to determine the matching meshes.

Note: this subroutine has to be called before **CCIsyncinfo** and **CCIcomminfo**. Otherwise the meshes defined in the latter calls are unknown to MpCCI.
Figure 4 illustrates the naming conventions of CCIdefmesh:

Meshes of the first code

1 part_1
part_2
part_3

2 part_3

Meshes of the second code

3 part_1

12 part_2

1 part_3

Figure 4: Naming conventions for CCIdefmesh.

<table>
<thead>
<tr>
<th>Class</th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>after closing the initial phase, first definition</td>
</tr>
</tbody>
</table>
**CCIdefpart**

<table>
<thead>
<tr>
<th>meshId</th>
<th>IN</th>
<th>grid identifier within each code</th>
</tr>
</thead>
<tbody>
<tr>
<td>partId</td>
<td>IN</td>
<td>partition identifier of each grid</td>
</tr>
</tbody>
</table>

```c
int CCI_Def_partition( int meshId, int partId )
```

```fortran
subroutine CCI_DEF_PARTITION( meshId, partId, ierror )
integer    meshId, partId, ierror
```

With **CCIdefpart** a partition of a grid is defined which will take part in the coupling process. The call of **CCIdefpart** with the parameters `meshId` and `partId` defines a partition with the identifier `partId` on the grid with identifier `meshId`.

Each code may use several (N) grids for solving the equations (multigrid, staggered grid). Every grid of each code is identified by the `meshId`. This numbering of the different grids is arbitrary.

Each grid identified by the `meshId` may consist of several partitions. From the view of **MpCCI** the grid partitions are the smallest parts of a grid, the “grid atoms”. Every **MpCCI** process may work on a set of partitions.
The following example defines two grids with three partitions on both grids like in Figure 5 (the number of partitions on each grid may not be equal):

```c
CCI_Def_partition( meshId1, partId1 );
CCI_Def_partition( meshId1, partId2 );
CCI_Def_partition( meshId1, partId3 );
CCI_Def_partition( meshId2, partId1 );
CCI_Def_partition( meshId2, partId2 );
CCI_Def_partition( meshId2, partId3 );
```
<table>
<thead>
<tr>
<th><strong>Class</strong></th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Position</strong></td>
<td>after closing the initial phase, second definition (after calls of <code>CCIdefmesh</code>, if used)</td>
</tr>
</tbody>
</table>
The purpose of subroutine `CCI_defnodes` is to specify the nodes (grid points) within each partition that will take part in the coupling. `CCI_defnodes` is a strictly local operation. Based on these nodes the elements are defined.

There are three different ways of numbering the nodes. They can be selected by the actual choice of variable `nNodeIds`:

1. `nNodeIds = 0` The nodes are numbered consecutively from 1 to `nNodes`. `nodeIds` is ignored.
2. `nNodeIds = 1` The nodes are numbered consecutively from `nodeIds(1)` to `nodeIds(1)+nNodes`. `nodeIds(1)` has to be specified of course.
3. `nNodeIds = nNodes` The numbering of the nodes will be according to the elements of array `nodeIds`. The number of the i-th node is `nodeIds(i)`. 

The layout of array `coords` containing the coordinates of the nodes is the same in `FORTRAN/ FORTRAN 90`
and C. If, for $\text{globalDim} = 3$, $x_i$, $y_i$, and $z_i$ are the coordinates of the $i$-th point with $1 \leq i \leq n\text{Nodes}$, then the points are stored in the following way:

$$x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_{n\text{Nodes}}, y_{n\text{Nodes}}, z_{n\text{Nodes}}.$$  

For the value of variable `realType` please refer to Table 1 and Table 2.

It is possible to redefine coordinates for any $\text{nodeId}$ later in the program. But this has to be done by a call of $\text{CCImodnodes}$ within a remeshing operation (and not in a second call to $\text{CCIdefnodes}$).

$\text{CCIdefnodes}$ has to be called for each partition defined. This implies that $\text{CCIdefpart}$ has to be called first before invoking $\text{CCIdefnodes}$ for each partition. For each partition $\text{CCIdefnodes}$ must only be called once. It is not possible to add nodes to a partition by calling $\text{CCIdefnodes}$ a second time.

**Example:** The following example defines the nodes of partition $\text{partId}$ on grid $\text{meshId}$ (see Figure 20):

```c
/**
 * Variables for CCI_Def_nodes:
 */

/* Global dimension: 3 coordinates per node. */
int globalDim = 3;
/* Number of nodes for the partition: */
int nNodes = 9;
/* Node numbering according to the first item: */
int nNodeIds = 0;
int* nodeIds = 0;
/* Data type of the coordinates of the nodes: */
int realType = CCI_FLOAT;
/* Coordinates of the nodes: */
float coordsData[] = { 0, 0, 0, /* node 1 */
                     0.5, 0, 0, /* node 2 */
                     1, 0, 0, /* node 3 */
                     0, 0.5, 0, /* node 4 */
                     0.5, 0.5, 0, /* node 5 */
                     1, 0.5, 0, /* node 6 */
                     0, 1, 0, /* node 7 */
                     0.5, 1, 0, /* node 8 */
                     1, 1, 0 }; /* node 9 */
...

/* Call of the subroutine */
CCI_Def_nodes( meshId,
```
partId,
globalDim,
nNodes,
nNodeIds,
nodeIds,
realType,
coordsData );

<table>
<thead>
<tr>
<th>Class</th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>behind CCIdefpart (or within a remeshing)</td>
</tr>
</tbody>
</table>
**CCI_def elems**

- **meshId**  
  IN  
  grid identifier within each code

- **partId**  
  IN  
  partition identifier of each grid

- **nElems**  
  IN  
  number of elements

- **nElemIds**  
  IN  
  switch for element numbering, dimension of array `elemIds`

- **elemIds**  
  IN  
  element identifier

- **nElemTypes**  
  IN  
  length of array `elemTypes`

- **elemTypes**  
  IN  
  element type of the specified elements

- **nNodesPerElem**  
  IN  
  number of nodes needed to describe one specific element

- **elemNodes**  
  IN  
  indices of the nodes which define the elements

---

```c
int CCI_Def_elems( int meshId, int partId, int nElems,
                   int nElemIds, const int elemIds[],
                   int nElemTypes, const int elemTypes[],
                   const int nNodesPerElem[], const int elemNodes[] )
```

```fortran
subroutine CCI_DEF_ELEMS( meshId, partId, nElems, nElemIds,
                          elemIds, nElemTypes, elemTypes,
                          nNodesPerElem, elemNodes, ierror )
```

```fortran
integer   meshId, partId, nElems,
          nElemIds, elemIds(nElemIds), nElemTypes,
          elemTypes(nElemTypes), nNodesPerElem( ), elemNodes( ), ierror
```

This strictly local operation is used to specify the topology of a coupling partition by specifying the number of elements, element type, and for each element the indices of the nodes that define it. The node indices correspond to the ordering of the node specification in the call of `CCI_defnodes` (see 3.2.3 on page 84).

|nElems| elements can be defined. These may consist of only one type (|nElemTypes| = 1) up to |nElems| types with different element types `elemTypes`. Thus, different element types can be specified for the same coupling partition with just one call of this subroutine.

For nearest-neighbor interpolation from node to node `CCI_def elems` can be omitted.

In MpCCI, the interpolation type depends on the element type which is chosen and the settings in the MpCCI input file. For every element type it is possible to define element-based and node-based quantities.

Seven types of surface elements, where two element types can also be used for volume coupling in the two-dimensional space, four types of volume elements in the three-dimensional space, one line element,
i.e. surface element in the two-dimensional space, and two element types for coupling on the sphere are provided in the current version:

1. **CCIelemquad**: The number of nodes is four. The nodes can be enumerated anti-clockwise or clockwise, but the direction has to be the same for the whole partition. This element type can be used for surface coupling in the three-dimensional space and volume coupling in the two-dimensional space.

   ![Figure 6: CCIelemquad](image)

2. **CCIelemtri**: The number of nodes is three. This element type can be used for surface coupling in the three-dimensional space and volume coupling in the two-dimensional space.

   ![Figure 7: CCIelemtri](image)

3. **CCIelemquaeight**: The number of nodes is eight. The first node stored in `elemNodes` must be a node at a vertex of the element, the other nodes must be enumerated anti-clock-wise.

   ![Figure 8: CCIelemquaeight](image)

4. **CCIelemquanine**: The number of nodes is nine. The first node stored in `elemNodes` must be a node at a vertex of the element, the other nodes must be enumerated in rows.
5. **CCIelemtrsix**: The number of nodes is six. The first node stored in `elemNodes` must be a node at a vertex of the element, the other nodes must be enumerated anti-clock-wise.

6. **CCIelempentagon**: The number of nodes is five. The nodes are enumerated anti-clock-wise. The interpolation will be done via triangles.

7. **CCIelemhexagon**: The number of nodes is six. The nodes are enumerated anti-clock-wise. The interpolation will be done via quadrilaterals.

For line coupling in the 2-dimensional space there is currently implemented only one element type: **CCIelemline**. This element type represents the line segment between two nodes.

The volume elements are described in the following enumeration:
1. CCIelemtetrahedron: The number of nodes is four. The interpolation defined for this element is linear.

2. CCIelemhexahedron: The number of nodes is eight. The numbering of the nodes is shown in the figure. The interpolation defined for this element is linear.

3. CCIelemprism: The number of nodes is six. The interpolation defined for this element is linear.

4. CCIelempyramid: The number of nodes is five. The interpolation is non-linear.
The element types for surface coupling, line coupling and volume coupling must not be mingled on the same mesh. The global dimension of nodes for line coupling and volume coupling in the two-dimensional space is 2, for surface and volume coupling in the three-dimensional space the global dimension is 3.

\texttt{MpCCI} supports coupling on the sphere, but several restrictions must be regarded. The midpoint of the sphere must be the origin of the coordinate system. All meshes must consist of the same spherical element type. This is due to the fact that the spherical quadrilateral cannot be decomposed in two spherical triangles. The spherical element types are defined as follows:

- \texttt{CCIelemsphtri} : The number of nodes is three. The nodes are connected by curves on the sphere which have the minimal length.

- \texttt{CCIelemsphqua} : The number of nodes is four. The nodes can be enumerated anti-clockwise or clockwise. Degenerated quadrilaterals may be defined if the "north" or the "south pole" of the sphere is a doubly defined node. The north and the south pole must not be an inner point or lie
on an edge of the spherical quadrilateral. The edges of the spherical quadrilateral must be a line of "latitude" or "longitude" on the sphere.

![Spherical Quadrilateral](image)

Figure 19: CCIelemsphqua.

The sphere or parts of the sphere may also be treated as a "normal" surface in the three-dimensional space and all surface elements may be used to define it.

Based on the nodes specified by the user, MpCCI automatically computes the centers of the elements (open circles). Coupling quantities can be assigned to the nodes by CCIputnodes. CCIgetnodes is used to get values located at the nodes. The centers are referred to by CCIputelems and CCIgetelems. Further element types are intended to be included in later versions of MpCCI.

nElemIds must be set either to 0 or to nElems. The elements are numbered consecutively if you choose nElemIds = 0. If e.g. the first 10 triangles (CCIelemtri) and thereafter 20 quadrilaterals (CCIelemquad) are specified, then the 11th element is the first quadrilateral. If you choose nElemIds = nElems the elements will be numbered according to elemIds (see also CCIdefnodes).

All nodes defined by CCIdefnodes must be used in the topology specification calls for a coupling partition. The various elements can be defined by more than one call to CCIdefelems. It is the caller’s responsibility to ensure that no degenerate elements occur and that different elements do not overlap.

The ordering of array elemNodes is according to the order specified with routine CCIdefnodes for array coords.

**Example:** In the following example we refer to the definitions made in the example of CCIdefnodes. With the help of the nodes defined there we build a mesh consisting of three quadrilaterals with the ids 1, 2, 3 and two triangles with the ids 4, 5.

```c
/**
 * Variables for CCI_Def_elms:
 */

/* Number of quadrilaterals: */
int nElems1 = 3;

/* Numbering of the quadrilaterals: 1, 2, 3 */
int nElemIds1 = 0;
```
Figure 20: Mesh for the example.

```c
int* elemIds1 = 0;
/* Elementtypes ( must be 1 ): */
int nElemTypes1 = 1;
int elemTypes1[] = { CCI_ELEM_QUAD };
/* Nodes per quadrilateral: */
int nNodesPerElem1[] = { 4 };
/* Node ids of the quadrilaterals: */
int elemNodes1[] = { 1, 2, 5, 4, /* quadrilateral 1 */
                    2, 3, 6, 5, /* quadrilateral 2 */
                    5, 6, 9, 8 }; /* quadrilateral 3 */

/* Number of triangles: */
int nElems2 = 2;
/* Numbering of the triangles: 4, 5 */
int nElemIds2 = 2; /* or = 0 */
int elemIds2[] = { 4, 5 };
/* Elementtypes ( must be one ): */
int nElemTypes2 = 1;
int elemTypes2[] = { CCI_ELEM_TRIANGLE };
/* Nodes per element: */
int nNodesPerElem2[] = { 3 };
/* Node ids of the triangles: */
int elemNodes2[] = { 4, 5, 8, /* triangle 4 */
                    4, 8, 7 }; /* triangle 5 */
```

...
/ * Call of the subroutine to define the quadrilaterals: */
CCI_Def_elems( meshId, partId, nElems1, nElemIds1, elemIds1, nElemTypes1, elemTypes1, nNodesPerElem1, elemNodes1 );

/* Call of the subroutine to define the triangles: */
CCI_Def_elems( meshId, partId, nElems2, nElemIds2, elemIds2, nElemTypes2, elemTypes2, nNodesPerElem2, elemNodes2 );

Class local, non-blocking
Position behind CCIdefnodes
### CCI_defcomm

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>localCommId</td>
<td>IN</td>
<td>communicator of local code</td>
</tr>
<tr>
<td>remoteCodeId</td>
<td>IN</td>
<td>id of the remote code</td>
</tr>
<tr>
<td>remoteCommId</td>
<td>IN</td>
<td>communicator of remote code</td>
</tr>
<tr>
<td>nQuantityIds</td>
<td>IN</td>
<td>length of array \texttt{quantityIds}</td>
</tr>
<tr>
<td>quantityIds</td>
<td>IN</td>
<td>ids as defined in the input file</td>
</tr>
<tr>
<td>nLocalMeshIds</td>
<td>IN</td>
<td>length of array \texttt{localMeshIds}</td>
</tr>
<tr>
<td>localMeshIds</td>
<td>IN</td>
<td>id of local meshes</td>
</tr>
</tbody>
</table>

```c
int CCI_Def_comm( int localCommId, int remoteCodeId, 
                  int remoteCommId, 
                  int nQuantityIds, const int quantityIds[], 
                  int nLocalMeshIds, const int localMeshIds[] )
```

```fortran
subroutine CCI_DEF_COMM( localCommId, remoteCodeId, remoteCommId, 
                         nQuantityIds, quantityIds, nLocalMeshIds, 
                         localMeshIds, ierror )
```

```fortran
integer localCommId, remoteCodeId, remoteCommId, nQuantityIds, 
         quantityIds(nQuantityIds), nLocalMeshIds, 
         localMeshIds(nLocalMeshIds), ierror
```

By calling \texttt{CCI_defcomm} communicators are built which group processes for later communication.

\texttt{CCI_defcomm} has to be called by all local MpCCI SDK processes that take part at least at one of the local grids that are defined by \texttt{localMeshIds}. Besides these requirements any other local process may call \texttt{CCI_defcomm} as well. This routine collects all the information which is needed to build up the communication but does not interpret it in contrast to \texttt{MPIcommcreate}. It is also possible to specify communicator parameters in the coupling block of the MpCCI SDK input file. To get those information \texttt{CCI_comminfo} has to be called before calling \texttt{CCI_defcomm}.

By calling \texttt{CCI_defcomm} an MpCCI SDK communicator is created on the calling side. In the MpCCI SDK communication routines (e.g. \texttt{CCIsend}, \texttt{CCIrecv}, see \S 3.2.4 \texttt{CCIsend} on page 112 and \S 3.2.4 \texttt{CCIrecv} on page 116) this communicator will be addressed by the variable \texttt{localCommId}. Every MpCCI SDK communicator is finally based on an MPI inter-communicator. Any communication like \texttt{CCIsend} or \texttt{CCIrecv} has to be called by all processes that share the same identifier \texttt{localCommId}.

\texttt{localCommId} is a local variable in the sense that it is only known on every process within the local code. On the remote side it finds its counterpart in the variable \texttt{remoteCommId}. In combination with the variable \texttt{localCommId} an MpCCI SDK communicator is defined unmistakably as a pair consisting of...
a communicator ID and a code ID \texttt{[remoteCommId, remoteCodeId]} or \texttt{[localCommId, localCodeId]}. Predefined constants for these identifiers are \texttt{CCIncodes} and \texttt{CCImycodeid}. Note that with every MpCCI SDK communicator a MPI communicator is defined.

If the input file is not used to specify the parameters of the communicators all variables \texttt{localCommId} should be documented since these values are required by the remote code when calling \texttt{CCIdefcomm}.

Both variables \texttt{localCommId} and \texttt{remoteCommId} need to be greater than zero.

In the communicator the \texttt{nQuantityIds} quantities which are stored in \texttt{quantityIds} may be communicated. The quantities and their identifiers must be defined in the MpCCI SDK input file. Instead of naming all quantities, a wild card \texttt{CCIanyquantity} can be used. This constant is also very helpful when coupling more than two codes.

In \texttt{localMeshIds} the corresponding mesh identifiers are to be stored. There are two ways to do this. If there is only one mesh involved for all quantities of \texttt{quantityIds}, \texttt{nlocalMeshIds} may be set to 1 and the ( only ) entry of \texttt{localMeshIds} is the identifier of that mesh. If the quantities are to be communicated on different meshes \texttt{nLocalMeshIds} equals \texttt{nQuantityIds} and the corresponding mesh identifiers must be stored in \texttt{localMeshIds}.

In the MpCCI SDK input file the quantities of the codes which match each other must be specified in the quantities block. Make sure that the local communicator ( \texttt{localCommId} ) and the communicator of the remote code ( \texttt{remoteCommId} ) communicate matching quantities. It is possible to communicate one quantity on different meshes in one communicator. In this case the order of those meshes in \texttt{localMeshIds} in the local communicator definition and in the definition of the remote communicator in the remote code are used to put or get the values.

**Example:** In the following example we define a communicator for a code with the identifier 1. The variable \texttt{localCommId} is 112 and the corresponding communicator of the remote code which is code 2 in this case has the identifier 221. The identifier of the quantities are 1, 2, 3, so in the MpCCI SDK input file those quantities must be defined in the code block of code 1. We assume that there is only one mesh involved in the communication of this communicator. The identifier of that mesh is 1.

```c
/**
 * Variables for CCI_Def_comm:
 */

/* Identifier of the communicator: */
int localCommId = 112;

/* Identifier of the remote code: */
int remoteCodeId = 2;

/* Identifier of the communicator of the remote code: */
int remoteCommId = 221;

/* Number of quantities: */
int nQuantityIds = 3;

/* Identifier of the quantities: */
```
```c
int quantityIds[] = { 1, 2, 3 }; /* Number of mesh identifiers: */
int nLocalMeshIds = 1; /* or = 3 */ /* Identifiers of the meshes: */
int localMeshIds[] = { 1 }; /* or = { 1, 1, 1} */

/* Call of the subroutine: */
CCI_Def_comm( localCommId,
    remoteCodeId,
    remoteCommId,
    nQuantityIds,
    quantityIds,
    nLocalMeshIds,
    localMeshIds );
```

**Class**
local, non-blocking

**Position**
before CCIclosesetup
CCI_defsync

syncPointId IN identifier of the synchronization point
nQuantitiesToSend IN number of quantities to be sent
quantitiesToSend IN array of quantity–identifiers, length \( n\text{QuantitiesToSend} \)
nMeshIdsToSend IN length of the array meshIdsToSend, either 1 or \( n\text{QuantitiesToSend} \)
meshIdsToSend IN array of mesh–identifiers, length \( n\text{MeshIdsToSend} \)
nQuantitiesToRecv IN number of quantities to be received
quantitiesToRecv IN array of quantity–identifiers, length \( n\text{QuantitiesToRecv} \)
nMeshIdsToRecv IN length of the array meshIdsToRecv, either 1 or \( n\text{QuantitiesToRecv} \)
meshIdsToRecv IN array of mesh–identifiers, length \( n\text{MeshIdsToRecv} \)

```c
int CCI_Def_sync_point( int syncPointId,
                        int nQuantitiesToSend, const int quantitiesToSend[],
                        int nMeshIdsToSend, const int meshIdsToSend[],
                        int nQuantitiesToRecv, const int quantitiesToRecv[],
                        int nMeshIdsToRecv, const int meshIdsToRecv[],
                        int ierror )

subroutine CCI_DEF_SYNC_POINT( syncPointId, nQuantitiesToSend,
                                quantitiesToSend, nMeshIdsToSend,
                                meshIdsToSend, nQuantitiesToRecv,
                                quantitiesToRecv, nMeshIdsToRecv,
                                meshIdsToRecv, ierror )

integer syncPointId, nQuantitiesToSend, quantitiesToSend( nQuantitiesToSend ),
         nMeshIdsToSend, meshIdsToSend( nMeshIdsToSend ), nQuantitiesToRecv,
         quantitiesToRecv( nQuantitiesToRecv ), nMeshIdsToRecv,
         meshIdsToRecv( nMeshIdsToRecv ), ierror
```

A synchronization point initiates a number of send and receive operations. As it is obvious from the parameter list above the receivers of the send operation and the senders of the receive operations are not to be specified. All synchronization points which are going to be used in the code must be defined in the initialization phase. To start the action see CCI_reachsync, to get the synchronization point information specified via the input file see CCI_syncinfo (the coupling block of the input file is concerned with synchronization points). Due to the matching criterion from the input file or if no matching criterion was given in the input file due to the numbering of the synchronization points of the different codes MpCCI automatically determines the sending and receiving codes for the quantities. If there is no sender resp.
receiver found there will be no error message, so the user has to care about that. But if there is more than
one sender found an error message will be given.

Every code which takes part in a synchronization point communication has to call CCIdefsync. The
identifier of the synchronization point (i.e. syncPointId) may be different in different codes, the
matching of the synchronization points of the codes must then be specified in the coupling block of the
input file, otherwise synchronization points with the same identifier match each other.

The identifiers of the quantities the code is going to send (receive) at the synchronization point are stored in
quantitiesToSend (quantitiesToRecv) of length nQuantitiesToSend (nQuantitiesToRecv). The
 corresponding mesh identifiers are stored in the arrays meshIdsToSend (meshIdsToRecv) of length
nMeshIdsToSend (nMeshIdsToRecv). The value of nMeshIdsToSend (nMeshIdsToRecv) must be
equal to nQuantitiesToSend (nQuantitiesToRecv) or 1, if the same mesh is valid for all quantities
to be sent (received). The values nQuantitiesToSend and nQuantitiesToRecv must be non-negative
and may not exceed CCImaxnquant. The value 0 is allowed here and implies that there are no send or no
receive operations for the code at the current synchronization point.

Example: A synchronization point will be defined without using the MpCCI input file. The quantities 2
and 3 on mesh 1 are to be sent and the quantity 1 is to be received on mesh 1. (An example for defining
a synchronization point with the help of the input file can be found in ▸3.3 MpCCI Input File◁.)

```c
/**
 * Variables for CCI_Def_sync_point:
 */

/* Identifier of the synchronization point */
int syncPointId = 1;
/* Number of quantities to be sent: */
int nQuantitiesToSend = 2;
/* Identifier of the quantities to be sent: */
int quantitiesToSend[] = { 2, 3 };
/* Number of the mesh identifiers: */
int nMeshIdsToSend = 1;
/* Identifiers of the meshes: */
int meshIdsToSend[] = { 1 };
/* Number of quantities to be received: */
int nQuantitiesToRecv = 1;
/* Identifier of the quantities to be received: */
int quantitiesToRecv[] = { 1 };
/* Number of the mesh identifiers: */
int nMeshIdsToRecv = 1;
/* Identifiers of the meshes: */
int meshIdsToRecv[] = { 1 };
```
/* Call of the subroutine: */
CCI_Def_sync_point( syncPointId,
    nQuantitiesToSend,
    quantitiesToSend,
    nMeshIdsToSend,
    meshIdsToSend,
    nQuantitiesToRecv,
    quantitiesToRecv,
    nMeshIdsToRecv,
    meshIdsToRecv );

Class local, non-blocking
Position before CCIclosesetup
For the coupled computation the neighborhood of a local and remote grids must be determined. If the subroutine \texttt{CCIdefsearchtags} is not used \texttt{MpCCI} determines the neighborhood between all grids and partitions which take part in the coupled computation. \texttt{CCIdefsearchtags} allows the user to decide between which partitions the neighborhood must be computed. The tags stored in \texttt{searchTags} are global identifiers. If \texttt{CCIdefsearchtags} is called in the initialization phase \texttt{MpCCI} only determines the neighborhood between such partitions which have a search tag (an element of \texttt{searchTags}) in common. If \texttt{CCIdefsearchtags} is not called by a code \texttt{MpCCI} will interpret that all possible search tags are set.

\begin{itemize}
  \item \textbf{Class} \hspace{1cm} \text{local, non-blocking}
  \item \textbf{Position} \hspace{1cm} \text{before \texttt{CCIclosesetup}}
\end{itemize}
In coupled computation the different codes may define meshes with different underlying coordinate systems. They can differ in the location of the origin, in the unit vectors and even in the types of the coordinates such as Cartesian, spherical or cylindrical coordinates. The coordinate transformation module supports different kinds of coordinate systems in the two- and three-dimensional space. The computations carried out by MpCCI are based on Cartesian coordinate systems so that spherical and cylindrical coordinates are always transformed to Cartesian coordinates internally.

Doubly defined nodes may occur after the transformation from e.g. a cylindrical coordinate system to a Cartesian coordinate system. These nodes are mapped on one node by MpCCI and the element definitions are adjusted: Quadrilaterals e.g. may be deformed to triangles in the Cartesian coordinate system.

MpCCI supports Cartesian, cylindrical and spherical coordinate systems whereas cylindrical coordinates must not be used in the two-dimensional case (see Table 3). Currently there is only a C-interface implemented for coordinate transformations.

<table>
<thead>
<tr>
<th>Type</th>
<th>two-dimensional space</th>
<th>three-dimensional space</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartesian coordinate system</td>
<td>yes</td>
<td>yes</td>
<td>CCIcoordcart</td>
</tr>
<tr>
<td>cylindrical coordinate system</td>
<td>no</td>
<td>yes</td>
<td>CCIcoordcyl</td>
</tr>
<tr>
<td>spherical coordinate system</td>
<td>yes</td>
<td>yes</td>
<td>CCIcoordsph</td>
</tr>
</tbody>
</table>

Table 3: Types of coordinate systems.

The mesh `meshId` is prepared for a coordinate transformation. The parameter `globalDim` must equal 2 for the two-dimensional space and must equal 3 for the three-dimensional space. The parameter `type` determines the type of the original coordinate system:
• **CCIcoordcart** Abbreviation for Cartesian coordinate systems.

• **CCIcoordcyl** Abbreviation for cylindrical coordinate systems. A point is determined by \((r, \varphi, z)\) where the radius \(r\) is a non-negative real number, the height \(z\) is a real number, and the angle \(\varphi\) varies between 0 and \(2\pi\). The coordinates are transformed to Cartesian coordinates \((x, y, z)\) according to

\[
\begin{align*}
  x &= r \cos \varphi \\
  y &= r \sin \varphi \\
  z &= z
\end{align*}
\]

in the three-dimensional space. Cylindrical coordinates are not defined for the two-dimensional space. Afterwards \texttt{origin} and \texttt{unitVectors} are transformed according to the parameters as described below.

• **CCIcoordsph** Abbreviation for spherical coordinate systems. A point is determined by \((r, \varphi, \psi)\) where the radius \(r\) is a non-negative real number, the angle \(\varphi\) varies between 0 and \(2\pi\) and the angle \(\psi\) varies between 0 and \(\pi\). The coordinates are transformed to Cartesian coordinates \((x, y, z)\) according to

\[
\begin{align*}
  x &= r \cos \varphi \sin \psi \\
  y &= r \sin \varphi \sin \psi \\
  z &= r \cos \psi
\end{align*}
\]

in the three-dimensional space and according to

\[
\begin{align*}
  x &= r \cos \varphi \\
  y &= r \sin \varphi
\end{align*}
\]

in the two-dimensional space. Afterwards \texttt{origin} and \texttt{unitVectors} are transformed according to the parameters as described below.

The coordinates of the origin of the original Cartesian coordinate system in the "new" coordinates are given by \texttt{origin}. The array \texttt{unitVectors} consists of two two-dimensional vectors \(w_1, w_2\) if \texttt{globalDim} equals 2 or of three three-dimensional vectors \(v_1, v_2, v_3\) in the three-dimensional case. In the two-dimensional space the unit vectors \((1, 0)\), and \((0, 1)\) of the original Cartesian coordinate systems are mapped on \(w_1, w_2\). In the three-dimensional space the unit vectors \((1, 0, 0), (0, 1, 0), \text{ and } (0, 0, 1)\) of the original Cartesian coordinate systems are mapped on \(v_1, v_2, v_3\).

The parameter \texttt{equalsZero} is used to determine whether two points have the same coordinates after the coordinate transformation.

If a coordinate transformation is defined for a mesh, then the elements of a partition of that mesh must be defined by only one call of the subroutine \texttt{CCIdefelem}. This is necessary because the type of some elements may change during the transformation.
The vector-valued quantities are transformed to the new coordinate system and re-transformed to the
original coordinate system automatically during the communication.

The subroutine **CCIdefctf** is only one of two methods to define a coordinate transformation. The
additional block of the **MpCCI** input file can also be used.

![Coordinate transformation](image)

**Figure 21: Coordinate transformation**

The following example defines a partition with a Cartesian coordinate system:

```c
int meshId = 1;
int globalDim = 3;
int type = CCI_COORD_CART;
float equalsZero = 1.0e-6;
double origin[] = { 1.0, 1.0, 1.0 };
double unitVectors[] = { 1.0, 1.0, 1.0,
                         1.0, 0.0, 0.0,
                         0.0, 2.0, 0.0 };;
CCI_Def_coord_transform( meshId,
                        globalDim,
                        type,
                        equalsZero,
                        origin,
                        unitVectors );
```

**Class**    local, non-blocking  
**Position** before node definition on the partitions
**CCIclosesetup**

cciProcess  IN  flag for MpCCI SDK process

```c
int CCI_Close_setup( int cciProcess )
```

```fortran
subroutine CCI_CLOSE_SETUP( cciProcess, ierr )
```

```fortran
integer cciProcess, ierr
```

Subroutine CCIclosesetup terminates the coupling and definition phase. MpCCI SDK processes must call this subroutine with \texttt{cciProcess} \neq 0, other processes with \texttt{cciProcess} = 0. The exact value of \texttt{cciProcess} \neq 0 is not important.

CCIclosesetup must be called collectively by all processes of all codes, this means, by all processes that called CCIinit.

<table>
<thead>
<tr>
<th>Class</th>
<th>non-local, blocking, collective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>following the last definition routine, e.g.</td>
</tr>
<tr>
<td></td>
<td>CCIdefpart, CCIdefnodes, CCIdefelems,</td>
</tr>
<tr>
<td></td>
<td>CCIdefcomm, CCIdefsift</td>
</tr>
</tbody>
</table>
3.2.4 Coupling Communication

**CCIputnodes**

- **meshId**: IN grid identifier within each code
- **partId**: IN partition identifier of each grid
- **quantityId**: IN identifier specified in the input file
- **quantityDim**: IN dimension of quantity
- **nNodes**: IN number of nodes
- **nNodeIds**: IN switch node numbering, dimension of array `nodeIds`
- **nodeIds**: IN node identifier
- **valuesDataType**: IN identifier of input data
- **values**: IN returned data values

```c
int CCI_Put_nodes( int meshId, int partId, int quantityId, int quantityDim, int nNodes, int nNodeIds, const int nodeIds[], int valuesDataType, const void* values )
```

```fortran
subroutine CCI_PUT_NODES( meshId, partId, quantityId, quantityDim, nNodes, nNodeIds, nodeIds, valuesDataType, values, ierror )
integer meshId, partId, quantityId, quantityDim, nNodes, nNodeIds, nodeIds(nNodeIds), valuesDataType, ierror
<real type> values(quantityDim*nNodes)
```

Subroutine **CCIputnodes**, a strictly local operation, is used for specifying coupling values at the nodes to **MpCCI**.

The data items in array `values` are ordered according to the node identifier specified in array `nodeIds`. The specified coupling values are later sent to the remote code by using **CCIsend** or **CCIisend** (see ⊳ 3.2.4 **CCIsend** ⊳ on page 112 and ⊳ 3.2.4 **CCIisend** ⊳ on page 114). According to the type of `values` the parameter `valuesDataType` must be chosen: for the value of variable `valuesDataType` please refer to Table 1 and Table 2.

The quantity with the identifier `quantityId` must be defined in the relevant code block of the **MpCCI** input file with the location specification `loc = nodes`. (If `loc = elem` was chosen **CCIputelems** must be applied.) The dimension `quantityDim` of the quantity also depends on the specification of the code.
block of the input file. If scalar was chosen the value of quantityDim must be 1, if vector was chosen the value of quantityDim must equal the dimension of the vector.

The number of nodes nNodes is at most the total number of the nodes of the partition partId specified via CCIdefnodes. It is possible to call CCIputnodes for a subset of the nodes of a partition.

The value of nNodeIds and nodeIds may be chosen for grid based quantities according to the enumeration in the description of CCIdefnodes.

Example: In the description of CCIdefnodes an example is given which will be extended now. There a mesh is defined with nine nodes numbered from 1 to 9. The mesh with identifier meshId consists of one partition partId.

```c
/**
 * Variables for CCI_Put_nodes:
 */

/* Quantity identifier: This quantity must be specified in the input file. */
int     quantityId     = 1;
/* Dimension of the quantity: We assume that quantity 1 is of scalar type. */
int     quantityDim    = 1;
/* Number of nodes to which the subroutine will be applied: */
int     nNodes         = 9;
/* The following choice selects all nodes of the partition. */
int     nNodeIds       = 0;
int*    nodeIds        = 0;
/* Data type for the values: */
int     valuesDataType = CCI_FLOAT;
/* Values at the nodes: */
float    values[]      = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

/* Call of the subroutine:
CCI_Put_nodes( meshId, partId, quantityId, quantityDim, nNodes, nNodeIds, nodeIds, valuesDataType, values );
```
<table>
<thead>
<tr>
<th><strong>Class</strong></th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Position</strong></td>
<td>following the definition phase, after <strong>CCIclosesetup</strong> and before a corresponding <strong>CCIisend</strong> of <strong>CCIisend</strong></td>
</tr>
</tbody>
</table>
**CCIPutElems**

- **meshId**  `IN`  grid identifier within each code
- **partId**  `IN`  partition identifier of each grid
- **quantityId**  `IN`  quantity identifier specified in the input file
- **quantityDim**  `IN`  dimension of quantity
- **nElems**  `IN`  number of elements
- **nElemIds**  `IN`  switch for element numbering, dimension of array `elemIds`
- **elemIds**  `IN`  element identifier
- **nDataPointsPerElem**  `IN`  length of array `dataPointsPerElem`
- **dataPointsPerElem**  `IN`  number of data belonging to one element
- **valuesDataType**  `IN`  identifier of input data
- **values**  `IN`  coupling values

```c
int CCI_Put_elements( int meshId, int partId, int quantityId, int quantityDim
                      int nElems, int nElemIds, const int elemIds[],
                      int nDataPointsPerElem, const int dataPointsPerElem[],
                      int valuesDataType, const void* values )

subroutine CCI_PUT_ELEMENTS( meshId, partId, quantityId, quantityDim,
                              nElems, nElemIds, elemIds, nDataPointsPerElem,
                              dataPointsPerElem, valuesDataType, values, ierror )

integer          meshId, partId, quantityId, quantityDim, nElems,
                  nElemIds, elemIds(nElemIds), nDataPointsPerElem,
                  dataPointsPerElem(nDataPointsPerElem), valuesDataType, ierror

<real type>      values(quantityDim*nPoints)
```

Subroutine **CCIPutElems**, which is a strictly local operation, is used for specifying coupling values at the data locations of an element to **MpCCl** which can be communicated by **CCIsend** or **CCIisend**.

The quantity with the identifier `quantityId` must be defined in the relevant code block of the **MpCCl** input file with the location specification `loc = elem`. (If `loc = node` was chosen **CCIPutNodes** must be applied.) The `quantityDim` also depends on the specification of the code block of the input file. If `scalar` was chosen the value of `quantityDim` must be 1, if `vector` was chosen the value of `quantityDim` must equal the dimension of the vector.

The number of elements `nElems` is at most the total number of elements specified to **MpCCl** for the coupling partition in different calls of **CCIdefElems** for this partition. It is possible to call **CCIPutElems**.
for a subset of the elements of a partition.
The value of \texttt{nElemIds} and \texttt{elemIds} may be chosen according to the three possibilities of the enumeration listed in the description of \texttt{CCIdefnodes}.

In the current version \texttt{nDataPointsPerElem} must be 1 and the ( only ) entry of \texttt{dataPointsPerElem} must be 1, too. This is due to the interpolations which are currently implemented in \texttt{MpCCI}.

For further information about \texttt{CCIPutelems} see also the description of the subroutine \texttt{CCIPutnodes}. \texttt{CCIPutelems} specifies the coupling values at the element centers to \texttt{MpCCI} in the same way as \texttt{CCIPutnodes} specifies the coupling values at the nodes to \texttt{MpCCI}.

**Example:** We now continue with the example which was started in the description of \texttt{CCIdefnodes} and \texttt{CCIdefelems}. It should be recalled that five elements were defined for partition \texttt{partId} on mesh \texttt{meshId}.

```c
/**
 * Variables for CCI_Put elems:
 */

/* Quantity identifier: This quantity must be specified in the input file. */
int quantityId = 7;
/* Dimension of the quantity: In the input file quantity 7 was defined 'vector'. */
int quantityDim = 3;
/* Number of elements which are involved in the call of the subroutine: Must be smaller or equal the number of the elements of the partition which is 5. */
int nElems = 5;
/* Ids of the elements involved. The following choice selects all elements of this example. */
int nElemIds = 0;
int* elemIds = 0;
/* Number of data points per element: There is no other choice than to use the following. */
int nDataPointPerElem = 1;
int dataPointPerElem[] = { 1 };
/* Data type of the values: */
int valuesDataType = CCI_FLOAT;
/* Values which are associated with the data points: */
float values[] = { 1, 1, 1, /* vector 1 */
  2, 2, 2, /* vector 2 */
  3, 3, 3, /* vector 3 */
  4, 4, 4, /* vector 4 */
```
5, 5, 5 }; /* vector 5 */

/* Call of the subroutine: */
CCI_Put_elems( meshId,
    partId,
    quantityId,
    quantityDim,
    nElems,
    nElemIds,
    elemIds,
    nDataPointsPerElem,
    dataPointsPerElem,
    valuesDataType,
    values );

Class       local, non-blocking
Position    as for CCIputnodes
### CCI_send

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nQuantityIds</td>
<td>IN</td>
<td>length of array</td>
</tr>
<tr>
<td>quantityIds</td>
<td>IN</td>
<td>ids as defined in the input file</td>
</tr>
<tr>
<td>nLocalMeshIds</td>
<td>IN</td>
<td>length of array</td>
</tr>
<tr>
<td>localMeshIds</td>
<td>IN</td>
<td>id of local meshes</td>
</tr>
<tr>
<td>comm</td>
<td>IN</td>
<td>MpCCI communicator</td>
</tr>
</tbody>
</table>

```c
int CCI_Send( int nQuantityIds, const int quantityIds[],
             int nLocalMeshIds, const int localMeshIds[],
             int comm )
```

```fortran
subroutine CCI_SEND( nQuantityIds, quantityIds, nLocalMeshIds,
                     localMeshIds, comm, ierror )
```

```c
integer nQuantityIds, quantityIds(nQuantityIds),
        nLocalMeshIds, localMeshIds(nLocalMeshIds),
        comm, ierror
```

### CCI_send

CCI_send is a blocking two-side collective operation over the MpCCI processes of the local and remote code (see also §3.2.4 CCI_isend on page 114).

This subroutine sends the computed coupling values (specified using CCI_putnodes and CCI_putelems) to a remote code with coupling partitions through which coupling occurs with the calling code.

All MpCCI processes of the calling code must call this routine, even if they have no coupling partitions through which coupling occurs with the other code. This enables MpCCI to do the computations required for keeping the neighborhood information up to date in parallel using all MpCCI processes of a code.

**Example:** In the description of CCI_defnodes a communicator with the identifier 112 was defined. Here it is used to send some quantities.

```c
/**
 * Variables for CCI_Send:
 */

/* Number of quantities: */
int nQuantityIds = 3;
/* Identifier of the quantities: */
int quantityIds[] = { 1, 2, 3 };
/* Number of mesh identifier: Here the same mesh for all. */
int nLocalMeshIds = 1;
/* Identifier of the mesh: */
int localMeshIds[] = { 1 };
/* Identifier of the communicator: */
int comm = 112;

/* Call of the subroutine: */
CCI_Send( nQuantityIds, quantityIds, nLocalMeshIds, localMeshIds, comm );

Figure 22: Data transfer between meshes of different codes

<table>
<thead>
<tr>
<th>Class</th>
<th>non-local, blocking, collective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>following the CCIput routines</td>
</tr>
</tbody>
</table>
**CCIisend**

nQuantityIds  IN  length of array quantityIds
quantityIds   IN  ids as defined in the input file
nLocalMeshIds IN  length of array localMeshIds
localMeshIds IN  id of local meshes
comm         IN  MpCCI communicator
request      OUT  request handle

```c
int CCI_Isend( int nQuantityIds, const int quantityIds[],
              int nLocalMeshIds, const int localMeshIds[],
              int comm, CCI_Request* request )
```

```fortran
subroutine CCI_ISEND( nQuantityIds, quantityIds, nLocalMeshIds,
                      localMeshIds, comm, request, ierror )
```

```c
integer nQuantityIds, quantityIds(nQuantityIds),
        nLocalMeshIds, localMeshIds(nLocalMeshIds),
        comm, request, ierror
```

**CCIisend** is a non-blocking one-side collective operation over the MpCCI processes of the local code (see also \[3.2.4 \text{CCIisend} \] on page 112). Its completion is independent of other processes. After this call, new calls to other MpCCI subroutines may be issued. The subroutine returns a handle to a MpCCI request object, which must be explicitly tested for completion by a code using **CCIwait** (see page \[3.2.4 \text{CCIwait} \] on page 118). If a code does not explicitly test for requests to finish, then in the course of computation many such request objects will be created, and the system can run out of resources.

**Class**  local, non-blocking, collective

**Position**  behind a set of **CCIput** routines
CCIprobe

- **nQuantityIds**: IN length of array `quantityIds`
- **quantityIds**: IN ids as defined in the input file
- **nLocalMeshIds**: IN length of array `localMeshIds`
- **localMeshIds**: IN id of local meshes
- **comm**: IN MpCCI communicator
- **flag**: OUT Boolean
- **status**: OUT status object

```c
int CCI_Iprobe( int nQuantityIds, const int quantityIds[],
                 int nLocalMeshIds, const int localMeshIds[],
                 int comm, int* flag, CCI_Status* status )
```

```fortran
subroutine CCI_IPROBE( nQuantityIds, quantityIds, nLocalMeshIds,
                      localMeshIds, comm, flag, status, ierror )
```

```c
integer nQuantityIds, quantityIds(nQuantityIds),
        nLocalMeshIds, localMeshIds(nLocalMeshIds),
        comm, status(CCI_STATUS_SIZE), ierror
```

```fortran
integer nQuantityIds, quantityIds(nQuantityIds),
        nLocalMeshIds, localMeshIds(nLocalMeshIds),
        comm, status(CCI_STATUS_SIZE), ierror
```

This subroutine tests whether there is a message matching the first five parameters (see also `CCIrecv`) which can be received. The value of `flag` will be one (i.e. `true`) if all of the processes of communicator `comm` can receive that message, otherwise `flag` will be zero (i.e. `false`). In the first case the status parameter `status` contains the information of the message as it is described in the description of `CCIrecv`.

- **Class**: non-local, non-blocking, collective
- **Position**: before a call of `CCIrecv`
CCIrecv

nQuantityIds       IN  length of array quantityIds
quantityIds         IN  ids as defined in the input file
nLocalMeshIds       IN  length of array localMeshIds
localMeshIds        IN  id of local meshes
comm                IN  MpCCI communicator
status              OUT status object

```
int CCI_Recv( int nQuantityIds, const int quantityIds[],
             int nLocalMeshIds, const int localMeshIds[],
             int comm, CCI_Status* status )
```

```
subroutine CCI_RECV( nQuantityIds, quantityIds, nLocalMeshIds,
                       localMeshIds, comm, status, ierror )
```

```
integer nQuantityIds, quantityIds(nQuantityIds),
        nLocalMeshIds, localMeshIds(nLocalMeshIds),
        comm, status(CCI_STATUS_SIZE), ierror
```

CCIrecv is a blocking two-side collective operation over the MpCCI processes of the local code. This subroutine is used for receiving coupling values from a remote code with coupling partitions through which coupling occurs with the calling code (to be obtained using CCIgetnodes and CCIgetelems). All MpCCI processes of the calling code must call this routine, even if they have no partitions through which coupling occurs with the other code.

If \( n\text{QuantityIds} \) is set to a value greater than 0, the next message containing at least the given quantities is received.

In case \( n\text{QuantityIds} \) equals zero, quantities are received in the same order as they have been sent. The status variable contains the identifier of received quantities.

In C, status is a structure of type CCIstatus that contains fields constants \text{nQuantities}, \text{quantityIds} and \text{localMeshIds}. Thus status.nQuantities contains the number of quantities, status.quantityIds the ids of the received quantities and status.localMeshIds the identifier for the local mesh on which the received quantities have to be placed.

In FORTRAN, status is an array of length CCIstatuussize, which is set to \( 1 + 2 \times \text{CCImaxnquant} \). Analogous to the C programming syntax status(CCInqu Quantities) contains the number of quantities, status.

\(^2\)With CCImaxnquant the maximum number of quantities is specified that can be transferred with MpCCI. In the current implementation this value is set to 30.
(CCIquantityids) the identifier of the received quantities and status (CCIlocalmeshid) the identification for the local mesh on which the received quantities have to be placed.

Example: The example in the description of CCIsend is extended here. Code 1 has sent three quantities 1, 2, 3. We assume that the matching quantities of code 2 are 7, 8, 9 resp.. This must have been specified in the quantities block of the MpCCI input file. In the example of CCIdefcomm the communicator 112 of code 1 matches with the communicator 221 of the remote code 2. We assume that code 2 is prepared to receive the quantities 7, 8, 9 via the communicator 221. So the following is a part of code 2.

```c
/**
 * Variables for CCI_Recv:
 */

/* Number of quantities: */
int nQuantityIds = 3; /* or = 0 */

/* Identifiers of the quantities: */
int quantityIds[] = { 7, 8, 9 };;

/* Number of mesh identifiers: Here different meshes. */
int nLocalMeshIds = 3;

/* Identifiers of the meshes: */
int localMeshIds[] = { 1, 1, 2 };

/* Identifier of the communicator: */
int comm = 221;

/* Status object: */
CCI_Status status;

/* Call of the subroutine: */
CCI_Recv( nQuantityIds, 
          quantityIds, 
          nLocalMeshIds, 
          localMeshIds, 
          comm, 
          &status );
```

Class non-local, blocking, collective

Position before CCIgetnodes and/or CCIgetelems
The function \texttt{CCIwait} is a one-side collective operation over the MpCCI processes of the local code. It blocks all processes until all MPI requests associated with the non-blocking MpCCI operation have finished. It is used to complete non-blocking MpCCI sends (see \ref{CCIisend} on page 114).

Completion of a non-blocking MpCCI send indicates that all non-blocking MPI sends associated with this call have terminated. On completion of a non-blocking MpCCI send, MpCCI performs some final tasks like unpacking and storing received messages, (possibly) interpolation, and deletion of buffers allocated internally. If \texttt{CCIwait} is never called for requests, then the system can run out of resources during coupled computation.

\texttt{CCIwait} deletes the request object pointed to by \texttt{request} and sets the request handle to the value \texttt{CCIrequestnull}. Information on the completed operation is returned in \texttt{status}. \texttt{CCIwait} with \texttt{request} = \texttt{CCIrequestnull} is a no-operation.

<table>
<thead>
<tr>
<th>Class</th>
<th>non-local, blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>after a corresponding \texttt{CCIisend}</td>
</tr>
</tbody>
</table>
CCIreachsync

syncPointId      IN    identifier of a synchronization point
status          OUT   status object

```
int CCI_Reach_sync_point ( int syncPointId, CCI_Status* status )

subroutine CCI_REACH_SYNC_POINT( syncPointId, status, ierror )

integer syncPointId, status( CCI_STATUS_SIZE ), ierror
```

CCIreachsync performs the communication between those codes which are specified in the `matchsyncpt` statement of the coupling block of the input file or if this statement was not used between those codes which have defined a synchronization point with the identifier `syncPointId`. It must therefore be called by all of the concerned codes. The send and receive operations are determined by the synchronization point definition (see `CCIdfssync`) of the codes.

The communication at the synchronization point with the identifier `syncPointId` is carried out. The identifier `syncPointId` is a code local identifier if the `matchsyncpt` statement was used, otherwise the identifier is global.

For the interpretation of the parameter `status` see `CCIrecv`.

<table>
<thead>
<tr>
<th>Class</th>
<th>non-local, blocking, collective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>between a couple of CCIput and CCIget routines</td>
</tr>
</tbody>
</table>
### CCIgetnodes

- **meshId** **IN** grid identifier within each code
- **partId** **IN** partition identifier of each grid
- **quantityId** **IN** identifier specified in the input file
- **quantityDim** **IN** dimension of quantity
- **nNodes** **IN** number of nodes
- **nNodeIds** **IN** switch for node numbering, dimension of array \( \text{nodeIds} \)
- **nodeIds** **IN** node identifier
- **valuesDataType** **IN** identifier of input data
- **values** **OUT** returned data values
- **maxnEmptyNodes** **IN** maximum number of nodes with(out) data
- **emptyNodes** **OUT** returned nodes containing (no) data
- **nEmptyNodes** **OUT** length of array \( \text{emptyNodes} \)

```c
int CCI_Get_nodes( int meshId, int partId, int quantityId, 
                  int quantityDim, int nNodes, int nNodeIds, 
                  const int nodeIds[], int valuesDataType, 
                  void* values, int maxnEmptyNodes, 
                  int emptyNodes[], int* nEmptyNodes )
```

```fortran
subroutine CCI_GET_NODES( meshId, partId, quantityId, quantityDim, 
                          nNodes, nNodeIds, nodeIds(nNodeIds), valuesDataType, 
                          values, maxnEmptyNodes, emptyNodes, 
                          nEmptyNodes, ierror )
```

```fortran
integer meshId, partId, quantityId, quantityDim, nNodes, nNodeIds, 
       nodeIds(nNodeIds), valuesDataType, maxnEmptyNodes, 
       emptyNodes(maxnEmptyNodes), nEmptyNodes, ierror

<real type> values(quantityDim*nNodes)
```

This subroutine CCIgetnodes, a strictly local operation, is used for obtaining coupling values received from the code by the previous CCIrecv. According to the type of \( \text{values} \) the parameter \( \text{valuesDataType} \) must be chosen: for the value of variable \( \text{valuesDataType} \) please refer to Table 1 and Table 2. 

\( \text{nNodes} \) is the number of nodes specified for this coupling partition in the call of subroutine CCIdefnodes. The array \( \text{emptyNodes} \) indicates the nodes where no new values were obtained or where new values were obtained depending on the sign of \( \text{maxnEmptyNodes} \). If \( \text{maxnEmptyNodes} \) is positive the MpCCI internal
identifiers of the empty nodes are returned and if \( \text{maxnEmptyNodes} \) is negative the internal identifiers of the non-orphaned nodes are determined. The user is advised to check whether the returned value \( nEmptyNodes \) exceeds absolute value of the user defined value of \( \text{maxnEmptyNodes} \). In cases where \( nEmptyNodes \geq |\text{maxnEmptyNodes}| \) some information is lost about the location of data gaps. If a coordinate transformation is defined the user defined node identifiers are returned instead of the MpCCI internal identifiers. The orphaned and non-orphaned nodes cannot be retrieved if \( \text{overlap} = \text{partial} \) is specified in the contact block of the MpCCI input file unless \( \text{warn_orphaned_points} = \text{on} \) is specified in the contact block or the additional block contains a \( \text{non_orphaned_objects} \) statement for the mesh \( \text{meshId} \).

**Example:** In the example at the end of the description of CCIrecv among other quantity 7 was received by code 2. Now we apply CCIgetnodes to obtain the values for quantity 7 on partition 1 of mesh 1. We assume that the number of nodes of that partition is 6 and that all nodes should obtain a value. We assume that in the quantities block for code 2 of the MpCCI input file the location \( \text{loc} = \text{node} \) was chosen. Otherwise CCIgetelems must be applied. Quantity 7 of code 2 matches with quantity 1 of code 1, which was a scalar quantity. This implies that the dimension of quantity 7 of code 2 has dimension 1.

```c
/**
 * Variables for CCI_Get_nodes:
 */

/* Identifier of the mesh: */
int meshId = 1;
/* Identifier of the partition: */
int partId = 1;
/* Identifier of the quantity: */
int quantityId = 7;
/* Dimension of the quantity: */
int quantityDim = 1;
/* Number of nodes: */
const int nNodes = 6;
/* Node numbering: All nodes of the partition are chosen. */
int nNodeIds = 0;
int* nodeIds = 0;
/* Data type of the values: */
int valuesDataType = CCI_FLOAT;
/* Data values: */
float values[nNodes];
/* Maximum number of nodes with(out) data: */
int maxnEmptyNodes = 0;
/* Identifiers of nodes with(out) data: */
int* emptyNodes = 0;
/* Number of nodes with(out) data: */
int nEmptyNodes;
```
/* Call of the subroutine: 
CCI_Get_nodes( meshId,
    partId,
    quantityId,
    quantityDim,
    nNodes,
    nNodeIds,
    nodeIds,
    valuesDataType,
    values,
    maxnEmptyNodes,
    emptyNodes,
    &nEmptyNodes );

Class    local, non-blocking
Position  behind CCIrecv

CCI_getelems

meshId     IN       grid identifier within each code
partId     IN       partition identifier of each grid
quantityId IN       identifier specified in the input file
quantityDim IN       dimension of quantity
nElems     IN       number of elements
nElemIds   IN       switch for element numbering, dimension of array elemIds
elemIds    IN       element identifier
nDataPointsPerElem IN    number of elements specified for the given coupling partition
dataPointsPerElem IN    number of data belonging to one element
valuesDataType IN    identifier of input data
values     OUT      coupling values
maxnEmptyElems IN    maximum number of elements without data
emptyElems  OUT      returned elements containing no data
nEmptyElems OUT      length of array emptyElems

int CCI_Getelems( int meshId, int partId, int quantityId,
    int quantityDim, int nElems, int nElemIds,
Subroutine $\text{CCIgetelems}$ is a strictly local operation. It retrieves coupling values from element centers received from the code by the previous $\text{CCIrecv}$. The number of elements $\text{nElems}$ is the total number of elements specified for this coupling partition in consecutive calls of $\text{CCIdefeelems}$.

$\text{nEmptyElems}$ tells the number of elements that did not receive a value and the array $\text{emptyElems}$ describes these elements.

In the current version $\text{nDataPointsPerElem}$ must be 1 and the ( only ) entry of $\text{dataPointsPerElem}$ must be 1, too. See $\text{CCIputelems}$ (▷3.2.4 $\text{CCIputelems}$ on page 109).

For further information about $\text{CCIgetelems}$ see also the description of the subroutine $\text{CCIgetnodes}$ (▷3.2.4 $\text{CCIgetnodes}$ on page 120). $\text{CCIgetelems}$ is used for retrieving the coupling values at the element centers in the same way as $\text{CCIgetnodes}$ is used for retrieving the coupling values at the nodes.

**Example:** For all 3 elements of the partition 1 of mesh 1 values for quantity 11 will be obtained. We assume that quantity 11 is a scalar quantity.

```c
/**
 * Variables for CCI_Get_elems:
 */

/* Identifier of the mesh: */
int meshId = 1;
/* Identifier of the partition: */
```
int partId = 1; /* Identifier of the quantity: */
int quantityId = 11; /* Dimension of the quantity: */
int quantityDim = 1; /* Number of elements: Here all 3 elements are chosen. */
const int nElems = 3; /* Node numbering: Here all elements are chosen. */
int nElemIds = 0;
int* elemIds = 0; /* There is no choice for the following settings: */
int nDataPointsPerElems = 1;
int* dataPointsPerElems[] = { 1 }; /* Data type of the values: */
int valuesDataType = CCI_FLOAT; /* Data values: */
float values[nElems]; /* Currently the following variables have no relevance: */
/* Maximum number of elements without data: */
int maxnEmptyElems = 0; /* Identifiers of elements without data: */
int* emptyElems = 0; /* Number of elements without data: */
int nEmptyElems;

/* Call of the subroutine: */
CCI_Get_elems( meshId,
            partId,
            quantityId,
            quantityDim,
            nElems,
            nElemIds,
            elemIds,
            nDataPointsPerElems,
            dataPointsPerElems,
            valuesDataType,
            values,
            maxnEmptyElems,
            emptyElems,
            &nEmptyElems );
<table>
<thead>
<tr>
<th>Class</th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>following a CCIrecv</td>
</tr>
</tbody>
</table>
**CCIcheckconv**

- **myConvergence** (*IN*) code of local convergence state
- **globalConvergence** (*OUT*) global convergence state
- **comm** (*IN*) MpCCI communicator

```c
int CCI_Check_convergence( int myConvergence, int *globalConvergence,
                          int comm )
```

```fortran
subroutine CCI_CHECK_CONVERGENCE( myConvergence, globalConvergence,
                                  comm, ierror )
```

```fortran
integer myConvergence, globalConvergence, comm, ierror
```

CCIcheckconv is a blocking two-side collective operation over the MpCCI processes of the given communicator `comm`. In addition, the constant `CCIanycode` can be used as the `comm` argument. In this case, CCIcheckconv must be called by all codes and exchanges the convergence states.

Each code specifies its local convergence state `myConvergence` by an integer. These integers are exchanged between the codes and the maximum of all integers is returned to each code in `globalConvergence`.

MpCCI defines four constants for the local convergence state with the following meaning:

- **CCIconverged** ( = 0 ) I have converged and could stop here, but I can also do a further coupling step.
- **CCIcontinue** ( = 1 ) I have not yet converged and would like to do a further coupling step.
- **CCISTOP** ( = 2 ) I have converged and I must stop here. No further coupling step is possible for me!
- **CCIdiverged** ( = 3 ) I have diverged and I must stop here.

This subroutine has to be called to start the remeshing (see \(\text{3.2.5 Remeshing} \)).

Another important meaning of this subroutine is the creation of coupling steps when writing a MpCCI tracefile. CCIcheckconv introduces a new coupling step in the tracefile. With a suitable visualization tool for MpCCI tracefiles the user can observe the data sent and received in the different steps in an animation.

**Class** non-local, blocking, collective

**Position** following the CCI_send or CCI_recv routines to initiate a new coupling step; after the remeshing subroutines to actually start the remeshing
3.2.5 Remeshing

In the current version MpCCI SDK supports the remeshing in a straight-forward and simple way only. First, changes of the mesh of one code can be defined after a coupling step by subsequent calls of \texttt{CCIremesh}, \texttt{CCIdefnodes} and \texttt{CCIdefelems} or \texttt{CCImodnodes}, and finally \texttt{CCICloseremesh}. Second, the remote code has to be informed about the changes by calling \texttt{CCICheckconv}. This will lead to a new setup of the coupling, i.e. a complete neighborhood search from scratch.

It is planned to introduce a faster remeshing that takes into account the existing neighborhood information into the next version.

\texttt{CCIremesh}

\begin{verbatim}
int CCI_Remesh()

subroutine CCI_REMESH( ierror )

integer ierror

\end{verbatim}

The subroutine \texttt{CCIremesh} introduces a remeshing operation (mesh deformation or mesh redefinition). Within this operation the nodes of the meshes can be totally redefined with \texttt{CCIdefnodes} and \texttt{CCIdefelems} as in the definition phase. If the nodes have new coordinates only and the elements are unchanged, the nodes can be modified with \texttt{CCImodnodes}. At the end of the remeshing operation, a call of \texttt{CCICloseremesh} is necessary. It is not allowed to call an other MpCCI subroutine in between.

Note that the remeshing is a local operation. In order to inform the remote code about the remeshing the subroutine \texttt{CCICheckconv} has to be called after the remeshing definition has been finished with \texttt{CCICloseremesh}.

\begin{itemize}
    \item \textbf{Class} \hspace{1cm} local, non-blocking
    \item \textbf{Position} \hspace{1cm} before \texttt{CCICloseremesh}
\end{itemize}
**CCImodnodes**

- **meshId** IN: grid identifier within each code
- **partId** IN: partition identifier of each grid
- **globalDim** IN: dimension of the global coordinate system
- **nNodes** IN: number of nodes
- **nNodeIds** IN: switch node numbering, dimension of array nodeIds
- **nodeIds** IN: node identifier
- **realType** IN: data type of elements of coords
- **coords** IN: address of coordinate array

```c
int CCI_Mod_nodes( int meshId, int partId, int globalDim, int nNodes,
                   int nNodeIds, const int nodeIds[], int realType,
                   const void* coords )

subroutine CCI_MOD_NODES( meshId, partId, globalDim, nNodes,
                           nNodeIds, nodeIds, realType,
                           coords, ierror)

integer meshId, partId, globalDim, nNodes, nNodeIds,
        nodeIds(nNodeIds), realType, ierror
<real type> coords(globalDim*nNodes)
```

The subroutine **CCImodnodes** defines the nodes within each partition that are to be modified. For the arguments of the subroutine, please refer to **CCIdefnodes**.

**Class** local, non-blocking

**Position** between **CCIremesh** and **CCIcloseremesh**
CCIcloseremesh

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nMeshIds</td>
<td>IN</td>
<td>length of array meshIds</td>
</tr>
<tr>
<td>meshIds</td>
<td>IN</td>
<td>id of meshes to remesh</td>
</tr>
<tr>
<td>nRemeshingModes</td>
<td>IN</td>
<td>length of array remeshingModes</td>
</tr>
<tr>
<td>remeshingMode</td>
<td>IN</td>
<td>modes for remeshing</td>
</tr>
</tbody>
</table>

```c
int CCI_Close_remesh( int nMeshIds, const int meshIds[],
                      int nRemeshingModes, const int remeshingMode[])
```

```fortran
subroutine CCI_CLOSE_REMESH( nMeshIds, meshIds, nRemeshingModes,
                            remeshingMode, ierror )
```

```c
integer nMeshIds, meshIds(nMeshIds), nRemeshingModes,
       remeshingMode(nRemeshingModes), ierror
```

With the subroutine CCIcloseremesh the remeshing definition is closed. nMeshIds is the length of the array meshIds that holds the numbers of the meshes. The array remeshingMode contains the mode of remeshing for every mesh in meshIds.

In the current version, the only remeshing mode is CCItotalremesh.

<table>
<thead>
<tr>
<th>Class</th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>after CCIremesh and CCImodnodes (or CCIdefnodes and CCIdefelems), before CCIcheckconv</td>
</tr>
</tbody>
</table>
3.2.6 Termination

CCI finalize

```c
int CCI_Finalize()

subroutine CCI_FINALIZE( ierror )

integer ierror
```

Subroutine CCI finalize terminates the coupling and has to be called collectively by all those processes of any code that called CCI init. If MPI init has been called from within CCI init (see 3.2.3 CCI init on page 77) MPI finalize will also called by CCI finalize. If MPI init has been called explicitly before invoking CCI init, MPI finalize has to be called accordingly after CCI finalize. No MpCCI subroutines may be called after this subroutine (even MPI init).

When the codes have collectively determined that the coupled computation has converged and must be terminated, they must collectively notify MpCCI of this by calling subroutine CCI finalize. This must be done by all processes of all codes (not only the MpCCI processes). If the coupled computation has been started by simultaneously invoking a control process CCI finalize sends a message to the control process telling that the coupled computation has finished, and it cleans up the local MpCCI layers. After the control process (when present) has received termination messages from all processes it performs its local cleanup, writes its output files, and calls MPI finalize.

The subroutine CCI finalize returns control to the code, which can continue its own termination phase, such as writing output files (different to CCI init). The order in which the code-local termination proceeds and when CCI finalize is called is insignificant. A code can just as well first write all its output files and then call CCI finalize.

- **Class**: non-local, blocking
- **Position**: final MpCCI call
### 3.2.7 Control

Sometimes the user finds it necessary to have the opportunity to read in parameter values or any other information that might be helpful to gain some control over the coupled computation. Such functionality is provided by the routine `CCIparaminfo`.

#### CCIparaminfo

<table>
<thead>
<tr>
<th>idString</th>
<th>IN</th>
<th>string to identify input value</th>
</tr>
</thead>
<tbody>
<tr>
<td>valueDataType</td>
<td>IN</td>
<td>data type of value</td>
</tr>
<tr>
<td>value</td>
<td>OUT</td>
<td>return value set in input file</td>
</tr>
<tr>
<td>flag</td>
<td>OUT</td>
<td>return flag</td>
</tr>
</tbody>
</table>

```c
int CCI_Parameter_info( const char* idString, int valueDataType,
                        void* value, int* flag )
```

In the MpCCI input file (see ▶ 3.3 MpCCI Input File) arbitrary code specific values can be specified which can be copied into the address space of the corresponding program with the help of routine `CCIparaminfo`. Each input value is identified by a character string `idString` of arbitrary length. Note that spaces are not allowed to define the identifier string. The value associated with `idString` can be of all data types named in **Table 1** and **Table 2**. `valueDataType` has to be set according to these types. In case of the type `CCIstring` the argument `value` should be defined with the length `CCImaxstringlen` to ensure that the complete value can be returned.

The return flag indicates whether the requested value has been defined in the input file (`flag = 1`) or not (`flag = 0`).

- **Class**: local
- **Position**: between `CCIinit` and `CCIfinalize`
CCIgeneralinfo

- **item**: IN identifier of the type of information
- **subItem**: IN identifier of the required sub-item
- **maxnIntParams**: IN maximum number of elements of `intParams`
- **intParams**: OUT array of information of integer type
- **nIntParams**: OUT number of elements of `intParams`
- **maxnRealParams**: IN maximum number of elements of `realParams`
- **realParams**: OUT array of information of real type
- **nRealParams**: OUT number of elements of `realParams`
- **maxStringParamLength**: IN maximum length of `stringParam`
- **stringParam**: OUT string
- **stringParamLength**: OUT length of `stringParam`

```c
int CCI_General_info( int item, int subItem, int maxnIntParams,
    int intParams[], int* nIntParams,
    int maxnRealParams, double realParams[],
    int* nRealParams, int maxStringParamLength,
    char stringParam[], int* stringParamLength )

subroutine CCI_GENERAL_INFO( item, subItem,
    maxnIntParams, intParams, nIntParams,
    maxnRealParams, realParams, nRealParams,
    stringParam, stringParamLength, ierror )

integer      item, subItem,
    maxnIntParams, intParams( maxnIntParams ), nIntParams,
    maxnRealParams, nRealParams,
    stringParamLength, ierror
double precision realParams( maxnRealParams )
character*(1024) stringParam
```

This subroutine extracts the settings of the sub-item `subItem` of type `item` from the input file. Currently possible types are information about quantities (CCIquantinfo) and about the matching of the grids (CCImatchinginfo).

For the item CCIquantinfo there is a quantity identifier required for `subItem`. `maxnIntParams` should be set to CCIquantinfolen. In case of a failure `nIntParams` is 0. Otherwise, the result is stored in the
arrays \textit{intParams} and \textit{stringParam} which have the following entries:

- \textit{intParams}[CCIquantdim] contains the dimension.

- \textit{intParams}[CCIquanttype] contains the type:
  - \textit{CCIquanttypemesh},
  - \textit{CCIquanttypefield},
  - \textit{CCIquanttypeflux} or
  - \textit{CCIquanttypeuser}.

- \textit{intParams}[CCIquantloc] contains the location:
  - \textit{CCIquantlocnode},
  - \textit{CCIquantlocelm} or
  - \textit{CCIquantlocglobal}

- \textit{intParams}[CCIquantipol] contains the \textit{ipol} entry of the quantities block of the input file.

- \textit{stringParam} consists of the name of the quantity.

\textit{nIntParams} is set to 4 and \textit{stringParamLength} contains the length of the name of the quantity.

For the item \textit{CCImatchinginfo} the value of \textit{subItem} is ignored, because no further choices are possible. The result is stored in \textit{intParams} which has the length 1 in this case:

- \textit{intParams}[0] contains the matching information about the grids:
  - \textit{CCImatchgrids} or
  - \textit{CCInonmatchgrids}

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Class} & local \\
\textbf{Position} & between \textit{CCIinit} and \textit{CCIfinalize} \\
\hline
\end{tabular}
\end{center}
### CCIsyncinfo

- **syncPointId** (IN) - identifier of the synchronization point
- **maxnQuantToSend** (IN) - maximum number of quantities to be sent
- **quantitiesToSend** (OUT) - array of quantity identifiers, length `maxnQuantToSend`
- **meshIdsToSend** (OUT) - array of the mesh identifiers, length `maxnQuantToSend`
- **nQuantitiesToSend** (OUT) - number of relevant entries in the arrays `quantitiesToSend` and `meshIdsToSend`
- **maxnQuantToRecv** (IN) - maximum number of quantities to be received
- **quantitiesToRecv** (OUT) - array of quantity identifiers, length `maxnQuantToRecv`
- **meshIdsToRecv** (OUT) - array of the mesh identifiers, length `maxnQuantToRecv`
- **nQuantitiesToRecv** (OUT) - number of relevant entries in the arrays `quantitiesToRecv` and `meshIdsToRecv`

```c
int CCI_Sync_point_info( int syncPointId, int maxnQuantToSend,
                         int quantitiesToSend[], int meshIdsToSend[],
                         int* nQuantitiesToSend, int maxnQuantToRecv,
                         int quantitiesToRecv[], int meshIdsToRecv[],
                         int* nQuantitiesToRecv )

subroutine CCI_SYNC_POINT_INFO( syncPointId, maxnQuantToSend,
                                 quantitiesToSend, meshIdsToSend,
                                 nQuantitiesToSend, maxnQuantToRecv,
                                 quantitiesToRecv, meshIdsToRecv,
                                 nQuantitiesToRecv, ierror )

integer syncPointId, maxnQuantToSend,
         quantitiesToSend( maxnQuantToSend ),
         meshIdsToSend( maxnQuantToSend ),
         nQuantitiesToSend, maxnQuantToRecv,
         quantitiesToRecv( maxnQuantToRecv ),
         meshIdsToRecv( maxnQuantToRecv ),
         nQuantitiesToRecv, ierror
```
CCIsyncinfo gets information from the syncpt statement of the code and coupling blocks of the MpCCI input file for a synchronization point. Only the information for the local code is read.

For the synchronization point with the identifier syncPointId the communication information is extracted from the input file. The integer values maxnQuantToSend and maxnQuantToRecv specify the maximal number of quantities which may be sent or received. A natural choice for these parameters is CCImaxn quant as this value is not a restriction (see CCIdefsync). The identifiers for the quantities which are going to be sent (received) are stored in the array quantitiesToSend (quantitiesToRecv). The corresponding mesh identifiers are stored in the array meshIdsToSend (meshIdsToRecv). The number of the relevant entries of the arrays are given by nQuantitiesToSend (nQuantitiesToRecv).

To define the synchronization point CCIsyncinfo must be followed by a call of CCIdefsync with the same identifier syncPointId.

<table>
<thead>
<tr>
<th>Class</th>
<th>local, non–blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>between CCIinit and CCIclosesetup</td>
</tr>
</tbody>
</table>
**CCIcomminfo**

- **localCommId**: IN identifier of the communicator
- **remoteCodeId**: OUT identifier of the remote code
- **remoteCommId**: OUT identifier of the remote communicator
- **maxnQuantToSend**: IN maximum number of quantities to be sent
- **quantitiesToSend**: OUT array of quantity identifiers, length `maxnQuantToSend`
- **meshIdsToSend**: OUT array of the mesh identifiers, length `maxnQuantToSend`
- **nQuantitiesToSend**: OUT number of relevant entries in the arrays `quantitiesToSend` and `meshIdsToSend`
- **maxnQuantToRecv**: IN maximum number of quantities to be received
- **quantitiesToRecv**: OUT array of quantity identifiers, length `maxnQuantToRecv`
- **meshIdsToRecv**: OUT array of the mesh identifiers, length `maxnQuantToRecv`
- **nQuantitiesToRecv**: OUT number of relevant entries in the arrays `quantitiesToRecv` and `meshIdsToRecv`

```c
int CCI_Comm_info( int localCommId, int* remoteCodeId, int* remoteCommId,
                   int maxnQuantToSend, int quantitiesToSend[],
                   int meshIdsToSend[], int* nQuantitiesToSend,
                   int maxnQuantToRecv, int quantitiesToRecv[],
                   int meshIdsToRecv[], int* nQuantitiesToRecv )
```

```fortran
subroutine CCI_COMM_INFO( localCommId, remoteCodeId, remoteCommId,
                          maxnQuantToSend, quantitiesToSend, 
                          meshIdsToSend, nQuantitiesToSend, 
                          maxnQuantToRecv, quantitiesToRecv, 
                          meshIdsToRecv, nQuantitiesToRecv, ierror )
```

```fortran
integer localCommId, remoteCodeId, remoteCommId,
        maxnQuantToSend, quantitiesToSend(maxnQuantToSend),
        meshIdsToSend(maxnQuantToSend), nQuantitiesToSend,
        maxnQuantToRecv, quantitiesToRecv(maxnQuantToRecv),
        meshIdsToRecv(maxnQuantToRecv), nQuantitiesToRecv, ierror
```
CCIcomminfo gets information from the `comm` statement of the coupling block of the MpCCI input file about a communicator. Only the information for the local code is read.

For the communicator with the identifier `localcommId` the communication information is extracted from the input file. The identifiers of the remote code and the remote communicator are to be stored in `remoteCodeId` and `remoteCommId`. The integer values `maxnQuantToSend` and `maxnQuantToRecv` specify the maximum number of quantities which may be sent or received. A natural choice for these parameters is `CCImaxnquant` as this value is not a restriction (see `CCIdefcomm`). The identifiers for the quantities which are going to be sent (received) are stored in the array `quantitiesToSend` (`quantitiesToRecv`). The corresponding mesh identifiers are stored in the array `meshIdsToSend` (`meshIdsToRecv`). The number of the relevant entries of the arrays are given by `nQuantitiesToSend` (`nQuantitiesToRecv`).

To define the communicator `CCIcomminfo` must be followed by a call of `CCIdefcomm` with the same identifier `localCommId`. There is only one exception to this. There may be some specifications for the standard communicator `CCIcommrcode` in the input file. In this case no call of `CCIdefcomm` is necessary to define the communicator.

<table>
<thead>
<tr>
<th>Class</th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>between <code>CCIinit</code> and <code>CCIClosesetup</code></td>
</tr>
</tbody>
</table>
CCIgetidstring

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>IN</td>
<td>identifier of the job</td>
</tr>
<tr>
<td>maxCodeIdStringLen</td>
<td>IN</td>
<td>maximal length of codeIdString</td>
</tr>
<tr>
<td>codeIdString</td>
<td>OUT</td>
<td>id string from the code block</td>
</tr>
<tr>
<td>codeIdStringLen</td>
<td>OUT</td>
<td>length of codeIdString</td>
</tr>
<tr>
<td>maxJobIdStringLen</td>
<td>IN</td>
<td>maximal length of jobIdString</td>
</tr>
<tr>
<td>jobIdString</td>
<td>OUT</td>
<td>id string from the job block</td>
</tr>
<tr>
<td>jobIdStringLen</td>
<td>OUT</td>
<td>length of jobIdString</td>
</tr>
</tbody>
</table>

```c
int CCI_Get_id_string( int jobId, int maxCodeIdStringLen, char codeIdString[], int* codeIdStringLen, int maxJobIdStringLen, char jobIdString[], int* jobIdStringLen )
```

```fortran
subroutine CCI_GET_ID_STRING( jobId, codeIdString, codeIdStringLen, jobIdString, jobIdStringLen, ierror )
character*(*) codeIdString, jobIdString
integer jobId, codeIdStringLen, jobIdStringLen, ierror
```

The subroutine **CCIgetidstring** extracts the id string from the code block and the job block of the **MpCCI** input file and stores them in `codeIdString` and `jobIdString` resp. In the call of the **FORTRAN** subroutine the number of parameters is smaller than in the call of the **C** subroutine: `maxCodeIdStringLen` and `maxJobIdStringLen` are missing.

- **Class**: local, non–blocking
- **Position**: between **CCIinit** and **CCIfinalize**
CCIgetisp

meshId IN grid identifier within each code
partId IN partition identifier of each grid
remoteJobId IN identifier of a remote job
remoteMeshId IN identifier of a mesh of the remote job
globalDim IN global dimension
nElems IN number of elements
nElemIds IN length of identifier array beneath
elemIds IN identifier of the elements
nRemoteElemsPerLocalElem OUT array of numbers of intersecting remote elements
nTotalElemPairs IN/OUT length of array beneath
nIntersPointsPerElemPair OUT array of numbers of intersection points per element pair
nTotalIntersPoints IN/OUT length of the coordinates array
coordsValueType IN type of the coordinates array below
intersPointsCoords OUT array of numbers of intersection points per element pair

int CCI_Get_intersection_points( int meshId, int partId,
    int remoteJobId, int remoteMeshId,
    int globalDim, int nElems,
    int nElemIds, const int elemIds[],
    int nRemoteElemsPerLocalElem[],
    int* nTotalElemPairs,
    int nIntersPointsPerElemPair[],
    int* nTotalIntersPoints,
    int coordsValueType,
    void* intersPointsCoords )

The subroutine CCIgetisp retrieves the intersection points computed in the neighborhood search of type EE to determine the size of the overlap regions of the remote elements w.r.t a local element. Please note that this subroutine is defined in "cciext.h" and has a C-Interface only.

To use this function two specifications in the MpCCI input file are necessary:

- In the [contact] block an EE neighborhood search algorithm must be specified for the mesh pair meshId for the local job and remoteMeshId for the remote job with the identifier remoteJobId.
- There must be an [inters_points] statement in the [additional] block for this mesh pair.
**globalDim** is the number of coordinates of a point. The parameters **nElems**, **nElemIds** and **elemIds** have the same meaning as in **CCIdefelems**.

According to the type of **intersPointsCoords** the parameter **coordsValueType** must be chosen: for the value of variable **coordsValueType** please refer to Table 1 and Table 2.

The function **CCIgetisp** must be called twice:

- The first call is required to retrieve the length of the arrays **nIntersPointsPerElemPair** and **intersPointsCoords**, here the parameters **nTotalElemPairs** and **nTotalIntersPoints** must have a negative value when the subroutine is called. Both parameters are IN+OUT parameters.

- The second call is necessary to fill the data arrays **nTotalElemPairs**, **nIntersPointsPerElemPair** and **intersPointsCoords**. The subroutine must be called with the integer values retrieved in the first call. **nTotalElemPairs** and **nTotalIntersPoints** are IN parameters now.

After the second call the coordinates of the intersection points are stored in **intersPointsCoords**. These intersection points can be grouped into intersection points of element pairs according to the entries of **nIntersPointsPerElemPair** and finally these element pairs can be associated with the local elements with the help of the information stored in **nRemoteElemsPerLocalElem**.

Notice the following items:

- The intersection points are computed by projecting elements of the remote mesh onto the local mesh elements in a surface coupling, so the elements are not intersecting in the intersection points but one projected element and a non-projected element.

- During the neighborhood computation the elements are decomposed into triangles, tetrahedrons or line segments, so all projections, computation of overlap regions etc. are done w.r.t. these three types. The computed intersection points are just approximations to the ’real’ intersection points in two ways: the usual numerical approximation and the approximation of an element by decomposing it into triangles, tetrahedrons or line segments (Figure 24).

- The underlying algorithms are implemented to compute the size of the element overlaps and not to determine the minimal number of points to describe the figure. Caused by this the number of intersection points may be bigger than necessary. It is possible that a point on an edge of the intersection figure is retrieved by this subroutine even if this point is not necessary to describe the figure (Figure 23 and Figure 24).

**Example:** The following very simple meshes are defined:

After the first call of the function with

<table>
<thead>
<tr>
<th>nElems</th>
<th>= 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>globalDim</td>
<td>= 2</td>
</tr>
<tr>
<td>nElemIds</td>
<td>= 0</td>
</tr>
</tbody>
</table>
Figure 23: Intersection points.

Figure 24: Intersection points. The lilac point is not necessary to describe the intersection figure.

\begin{align*}
\text{nTotalElemPairs} &= -1 \\
\text{nTotalIntersPoints} &= -1
\end{align*}
we retrieve the values

\[
\begin{align*}
\text{nTotalElemPairs} & = 4 \\
\text{nTotalIntersPoints} & = 12
\end{align*}
\]

Now it is known the total number of element pairs is 4 and the total number of intersection points is 12. The second call of the subroutine will retrieve for example the following array values:

\[
\begin{align*}
\text{nRemoteElemsPerLocalElem} & = \{ 2, 2 \} \\
\text{nIntersPointsPerElemPair} & = \{ 3, 3, 3, 3 \}
\end{align*}
\]

Both elements on the local mesh intersect with two elements on the remote mesh.

Each intersection figure is described by three points.

\[
\begin{align*}
\text{intersPointsCoords} & = \{ 0.0, 0.0, 0.5, 0.5, 0.0, 1.0, \\
& \quad 0.0, 0.0, 1.0, 0.0, 0.5, 0.5, \\
& \quad 0.0, 1.0, 0.5, 0.5, 1.0, 1.0, \\
& \quad 1.0, 0.0, 0.5, 0.5, 1.0, 1.0 \}
\end{align*}
\]

The coordinate array depends on which element was defined first and the order of the nodes defining the elements in \text{CCIdefelems}. So the result of the coordinate array may be given in another order than above.

- **Class**: local, non-blocking
- **Position**: after \text{CCIclosesetup}
3.2.8 Miscellaneous functions

There are a few miscellaneous functions that might be helpful in some situations. These functions are counterparts to the corresponding MPI functions. They are listed in a separate section as they do not belong to the MpCCI SDK core functions.

CCIabort

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commId</td>
<td>IN</td>
<td>id of the communicator - currently unused</td>
</tr>
<tr>
<td>errorCode</td>
<td>IN</td>
<td>number of error code</td>
</tr>
<tr>
<td>message</td>
<td>IN</td>
<td>message text</td>
</tr>
</tbody>
</table>

```c
int CCI_Abort( int commId, int errorCode, char message[] )
```

```fortran
subroutine CCI_ABORT( commId, errorCode, message, ierror )
```

```fortran
character(*) message
```

```fortran
integer commId, errorCode, ierror
```

The subroutine **CCIabort** prints the string **message** and stops the process by calling **MPI.Abort**. The argument **commId** is currently not used.

**Class**   local, non–blocking
**Position** between CCIclosesetup and CCIfinalize
The function `CCI_wtick` returns the resolution of `CCI_wtime` in seconds (as `MPI_Wtick`).

<table>
<thead>
<tr>
<th>Class</th>
<th>local, non-blocking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>between <code>CCIClosesetup</code> and <code>CCIFinalize</code></td>
</tr>
</tbody>
</table>
The function `CCIWtime` returns a floating-point number of seconds, representing the elapsed wall-clock time since some time in the past. It enables the user to easily measure the time spent in a certain part of the program (as `MPI_Wtick`). The resolution is given by `CCIWtick`.

An example shows the usage of this function:

```c
{  
double starttime, endtime;
  starttime = CCI_Wtime();
  ... stuff to be timed ...
  endtime = CCI_Wtime();
  printf("stuff took %.2f seconds\n", endtime - starttime);
}
```

**Class**  
local, non–blocking

**Position**  
between `CCIclosesetup` and `CCIfinalize`
### 3.2.9 Overview of the MpCCI SDK functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page References</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCIinit</td>
<td>Enroll in the coupled computation.</td>
<td>3.2.3 CCIinit ◄ on page 77</td>
</tr>
<tr>
<td>CCIinitwithidstring</td>
<td>Enroll in the coupled computation.</td>
<td>3.2.3 CCIinitwithidstring ◄ on page 78</td>
</tr>
<tr>
<td>CCIdefmesh</td>
<td>Define a name for a mesh.</td>
<td>3.2.3 CCIdefmesh ◄ on page 79</td>
</tr>
<tr>
<td>CCIdefpart</td>
<td>Define the partitions of each grid.</td>
<td>3.2.3 CCIdefpart ◄ on page 81</td>
</tr>
<tr>
<td>CCIdefnodes</td>
<td>Specify the nodes of a coupling partition.</td>
<td>3.2.3 CCIdefnodes ◄ on page 84</td>
</tr>
<tr>
<td>CCIdefelem</td>
<td>Specify the elements of a coupling partition.</td>
<td>3.2.3 CCIdefelements ◄ on page 87</td>
</tr>
<tr>
<td>CCIdefcomm</td>
<td>Define communicators for process groups.</td>
<td>3.2.3 CCIdefcomm ◄ on page 95</td>
</tr>
<tr>
<td>CCIdefsych</td>
<td>Define synchronization point.</td>
<td>3.2.3 CCIdefsych ◄ on page 98</td>
</tr>
<tr>
<td>CCIdefsearchtags</td>
<td>Define tags for neighborhood search.</td>
<td>3.2.3 CCIdefsearchtags ◄ on page 101</td>
</tr>
<tr>
<td>CCIdefctf</td>
<td>Specify a coordinate transformation.</td>
<td>3.2.3 CCIdefctf ◄ on page 102</td>
</tr>
<tr>
<td>CCIcloseup</td>
<td>Terminate the initialization phase.</td>
<td>3.2.3 CCIcloseup ◄ on page 105</td>
</tr>
<tr>
<td>CCIputnodes</td>
<td>Put coupling values node wise.</td>
<td>3.2.4 CCIputnodes ◄ on page 106</td>
</tr>
<tr>
<td>CCIputelems</td>
<td>Put coupling values at element centers.</td>
<td>3.2.4 CCIputelem ◄ on page 109</td>
</tr>
<tr>
<td>CCIsend</td>
<td>Send coupling values.</td>
<td>3.2.4 CCIsend ◄ on page 112</td>
</tr>
<tr>
<td>CIIsend</td>
<td>Send coupling values (non-blocking).</td>
<td>3.2.4 CIIsend ◄ on page 114</td>
</tr>
<tr>
<td>CCIiprobe</td>
<td>Test of incoming messages.</td>
<td>3.2.4 CCIiprobe ◄ on page 115</td>
</tr>
<tr>
<td>CCIrecv</td>
<td>Receive coupling values.</td>
<td>3.2.4 CCIrecv ◄ on page 116</td>
</tr>
<tr>
<td>CCIwait</td>
<td>Wait for MpCCI SDK request to finish.</td>
<td>3.2.4 CCIwait ◄ on page 118</td>
</tr>
<tr>
<td>CCIreachsync</td>
<td>Reach synchronization point.</td>
<td>3.2.4 CCIreachsync ◄ on page 119</td>
</tr>
<tr>
<td>CCIgetnodes</td>
<td>Get coupling values node wise.</td>
<td>3.2.4 CCIgetnodes ◄ on page 120</td>
</tr>
<tr>
<td>CCIgetelem</td>
<td>Get coupling values from element centers.</td>
<td>3.2.4 CCIgetelem ◄ on page 122</td>
</tr>
<tr>
<td>CCIcheckconv</td>
<td>Check state of convergence.</td>
<td>3.2.4 CCIcheckconv ◄ on page 126</td>
</tr>
</tbody>
</table>

Table 4: Overview of the MpCCI SDK functions (1).
Remeshing

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCIremesh</td>
<td>Begin the remeshing definition.</td>
<td>3.2.5</td>
</tr>
<tr>
<td>CCImodnodes</td>
<td>Specify the nodes to be modified.</td>
<td>3.2.5</td>
</tr>
<tr>
<td>CCIcloseremesh</td>
<td>Terminate the remeshing definition.</td>
<td>3.2.5</td>
</tr>
</tbody>
</table>

Termination

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCIfinalize</td>
<td>Terminate the coupling.</td>
<td>3.2.6</td>
</tr>
</tbody>
</table>

Control

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCIparaminfo</td>
<td>Read in input values.</td>
<td>3.2.7</td>
</tr>
<tr>
<td>CCIgeneralinfo</td>
<td>Read in input values.</td>
<td>3.2.7</td>
</tr>
<tr>
<td>CCIsyncinfo</td>
<td>Get synchronization point info.</td>
<td>3.2.7</td>
</tr>
<tr>
<td>CCIcomminfo</td>
<td>Get communicator info.</td>
<td>3.2.7</td>
</tr>
<tr>
<td>CCIgetidstring</td>
<td>Get id strings from the input file.</td>
<td>3.2.7</td>
</tr>
<tr>
<td>CCIgetisp</td>
<td>Get intersection points.</td>
<td>3.2.7</td>
</tr>
</tbody>
</table>

Miscellaneous

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCIabort</td>
<td>Abort MpCCI SDK processes.</td>
<td>3.2.8</td>
</tr>
<tr>
<td>CCIwtick</td>
<td>Give resolution of ( \text{wtime} ).</td>
<td>3.2.8</td>
</tr>
<tr>
<td>CCIwtime</td>
<td>Give elapsed wall-clock time.</td>
<td>3.2.8</td>
</tr>
</tbody>
</table>

Table 5: Overview of the MpCCI SDK functions (2).

### 3.3 MpCCI Input File

In the coupled computation different types of information have to be provided to the MpCCI library. At first, MpCCI needs to know the application codes that take part in the coupling, their default environment and the way to start them. Of great importance are the different coupling quantities the codes can send and receive, and how the codes refer to them. The information so far can be described as “code configuration file” that should be shipped with the distribution of the application codes. Then, an important information is the mapping between the coupling quantities that allows each code to use different numbers for the same physical quantity. Finally there are a lot of settings for MpCCI itself, like

- parameters for algorithms, e.g. search algorithms or mesh quality checks,
- switches controlling the output level for debugging, or
- information about the coupling algorithm.

MpCCI provides defaults for almost all settings thus reducing the work for the user. On the other hand all such default values can be changed via the input file. In order to keep the input file readable and taking into account that settings of the codes can be supplied by the code owners, an input file can consist of several files. An effective mechanism for collecting input information distributed to different files is described in 3.3.11 Include Mechanism.
Note that all MpCCI input files are case-sensitive. After reading the user input file, the control process, when active, checks if the coupling is possible.

In the following description <key> indicates mandatory keywords or identifiers and [args] indicates optional arguments. A loose syntax description will be used. For instance, <key1>, ..., <keyn> denotes a comma-separated list of an arbitrary number of keywords.

### 3.3.1 Structure of the Input File

The MpCCI input file consists of several blocks:

- code block(s)
- quantities block(s)
- control block
- contact block
- switches block
- jobs block
- parameters block
- coupling block
- additional block

Each of these blocks starts with a main keyword and ends with the word end:

```
<MainKeyword>
...
end
```

The different blocks in the MpCCI input file are described below. Some of these blocks are mandatory, others are optional. The blocks must occur in the input file in the same ordering as listed above.

When working with the MpCCI GUI an other block named addcontrol ("additional control") might appear at the end of the input file. This block is generated by MpCCI and used internally only.
3.3.2 Code block

code codename

% coupling quantities of the code, at least one quantity:
quantity1(no = no1, loc = node|elem, dim = scalar|vector, type = field|flux);
quantity2(no = no2, loc = node|elem, dim = scalar|vector, type = field|flux);
quantity3(no = no3, loc = node|elem, dim = scalar|vector|< n >, type = userdef,
ipol = max|min|sum|prod);
quantity4(no = no4, loc = global, dim = scalar|vector, ipol = max|min|sum|prod);
mesh(no = no5);
quantity5(no = no5, loc = < .. >, dim = < .. >, type = < .. >, SI = a * Q + b);
% information about start-up
env(env_var_name) = "env_var_value";
pre_exec = "pre_exec_string";
exec = "exec_string";
post_exec = "post_exec_string";
% additional command line arguments for the exec command:
add_exec_args = "add_exec_arguments";
% id string for this code:
id_string = "id_string";
% enumeration of the parameters for the current code block (for MpCCI GUI only):
parameter parameter_name1( type = string|integer|real|logical,
default = default_value1, doc = "doc_string1" );
parameter parameter_name2( type = string|integer|real|logical,
default = default_value2 );
parameter parameter_name3;
% enumeration of the synchronization points (for MpCCI GUI only):
syncpt no : send( send_quantity_id1, send_quantity_id2, send_quantity_idn ),
recv( recv_quantity_id1, recv_quantity_id2, recv_quantity_idn );
end

For each code in the coupling there must be a code block, i.e. in a coupling of three different codes, there will be three code blocks, whereas in a coupling of the same code one code block is sufficient.

The code block describes what quantities there are, their name, number, location in the grid, dimension and type. At present, five types of quantities are supported, namely [field, flux, userdef, global and mesh].

For the first two types (field or flux), an arbitrary name must be specified together with a number, the location in the grid (node or elem) with the attribute loc (in former MpCCI versions: attribute where), the dimension and the type. Type field is recommended for quantities located at the nodes on the sending side, i.e. sending values from nodes to nodes or elements. Type flux is recommended for quantities located at the nodes on the receiving side, i.e. sending values from nodes or elements to nodes.
For the user-defined quantities (type \texttt{userdef}) it is possible to specify extended dimensions besides \texttt{scalar} and \texttt{vector}, i.e. the integer names the dimensions. In this case MpCCI needs the interpolation method \texttt{(ipol)}. An other type are \texttt{global} quantities that are defined with the attribute \texttt{loc}. Currently, there are four possibilities for the interpolation of global quantities: \texttt{max}, \texttt{min}, \texttt{sum} and \texttt{prod}.

Finally the type \texttt{mesh} is recognized by MpCCI and handled in a special way, therefore, only the numbers by which a code refers to them must be specified.

The \texttt{SI} setting supports transformations of quantities when two codes refer to the same quantity in different units. For example, code A may use ”mm” (millimeter) for the quantity ”displacement”, whereas code B uses ”m” (meter). With $\texttt{SI = 1000 * Q}$ in the quantity setting of code A, the values of both codes will have the same unit. The \texttt{SI} statement has the general form $\texttt{SI = a * Q + b}$ where $a$ and $b$ are real numbers and $a$ is nonzero. It means that the relation between the quantity value w.r.t. the unit used by the code ($Q$) and the quantity value w.r.t. the corresponding reference unit ($SI$) is given by the equation $\texttt{SI = a * Q + b}$. Typically, the reference unit will be the corresponding \texttt{SI} unit, but this is not necessary: it is only necessary that all codes use the same reference unit for a given quantity. The linear shift $b$ in the equation is provided for a quantity such as ”temperature” with several common units (Celsius and Fahrenheit).

For all types of quantities holds: different numbers must be used for different quantities. Negative numbers are allowed.

The instructions \texttt{env, pre_exec, exec, post_exec} specify the default environment for a code, the \texttt{startup_commands}, and are identical in meaning to those fields in the \texttt{jobs} block of the input file. This means, that one can specify general code-specific but job-independent parameters for a code in the \texttt{code} block instead of repeating them for each computation in latter blocks. This will make the MpCCI input file much more readable.

By using \texttt{add_exec_args} additional command line arguments can be given for the exec command of each process which belongs to the code \texttt{codename}. In \texttt{id_string} an identifier for the code can be stored.

The parameter information is only relevant for the MpCCI GUI, so the description is omitted here.

The information concerning the synchronization points of the code are only relevant for the MpCCI GUI.

Example:

```plaintext
code struc
  % Define coupling quantities of the code:
  temperature (no = 7, loc = node, dim = scalar, type = field);
  pressure (no = 13, loc = elem, dim = scalar, type = field);
  geschwindigkeit(no = 100, loc = node, dim = vector, type = field);
  kraft (no = 1000, loc = elem, dim = vector, type = flux);
  mesh (no = 101);

  % Define specific environment variables for the code
  % (may be overwritten in the "jobs" block):
  env(STRUC_LICENSE_DIR) = "/home/struc/license_dir";
```

exec = "/home/struc/bin/bin/struc";
% Id string for this code:
id_string = "FEM_CODE 2.4.5";
end

3.3.3 Quantities block

quantities
% how do the quantities correspond to each other?
code_i.quantity_j = code_k.quantity_l;
% enumeration of the parameters for all code blocks (for MpCCI GUI only):
parameter parameter_name_1( default = default_value_1, doc = "doc_string_1" );
parameter parameter_name_2( default = default_value_2 );
parameter parameter_name_3;
end

This section describes how the coupling quantities defined for each code in the code blocks match. The instruction code_i.quantity_j = code_k.quantity_l specifies that quantity_j of code_i is the same physical quantity as quantity_l of code code_k.

This mechanism enables new codes of different application areas to use MpCCI, because the knowledge about quantities and their type is not present in the MpCCI source. Also, this mechanism enables different codes to use their own local numbering of coupling quantities, independent of the other ones. MpCCI will perform the mapping of coupling types internally.

For instance, if we couple a fluid code requiring the mesh velocity with a structures code requiring forces, then the quantities section looks like:

quantities
% how do the quantities correspond to each other?
cfd.velocity = struc.geschwindigkeit;
cfd.force = struc.kraft;
end

In the above example, the structures code happens to use the German words for velocity and force, so that it has to be specified that 'velocity = geschwindigkeit'. An equality of two quantities is valid when they have the same type and dimension. They may reside at different positions of the computational grids, however.

Currently there are some restrictions due to the interpolation algorithms: The above matching conditions for the quantities are necessary for the communication between the codes. Values for quantities of type \texttt{flux} must be sent to quantities with the location specification \texttt{loc = node} and values for quantities of type \texttt{field} must be located at the nodes on the sending side.
The parameter information is only relevant for the MpCCI GUI, so the description is omitted here.

### 3.3.4 Control block

```plaintext
control
  tracefile ( name = "file_name", trace_par2, ..., trace_par_n );
  check_mesh_quality ( on|off );
  mesh_quality_nodes ( tol = ϵ );
  mesh_quality_quad ( quad_par1, ..., quad_par_n );
  mesh_quality_quad8 ( quad8_par1, ..., quad8_par_n );
  mesh_quality_quad9 ( quad9_par1, ..., quad9_par_n );
  mesh_quality_triangle ( triangle_par1, ..., triangle_par_n );
  mesh_quality_triangle6 ( triangle6_par1, ..., triangle6_par_n );
  skip_element_checks = false|true;
  timeout = seconds;
  allow_core_dump ( off|on );
  restart_file_name = "restart_file_name_stem";
  restart_file_access = none|read|write;
  contact_detection ( contact_par1, ..., contact_par_m );
end
```

The optional control block specifies some general steering parameters and some parameters for algorithms used by MpCCI internally. It can be used to specify:

- **tracefile**: Information for the tracefile, currently written in format HDF5[^1]. The tracefile can be used to visualize the coupled computation with a suitable visualization tool. For more information about tracefiles, please see V-6 MpCCI Visualizer. The name of the tracefile can be specified through

```plaintext
control
  tracefile ( name = "test.ccv" );
end
```

If no name or an empty name is specified, no tracefile will be written. Besides the name, there is a number of switches (value: [on] or [off]) for the tracefile to be generated. With `trace_comm_values` MpCCI will write, in addition to the geometry data, communication values into the tracefile, default is [on]. The switches `trace_mesh_values` and `trace_mesh_change_values` will lead communication values of coupling type ”mesh” and ”mesh_change” into the tracefile (unless `trace_comm_values` is switched off). With `close_after_writing` MpCCI will close the tracefile each time after writing data into it, default is [off]. And finally, the switch `implicit_coupling_steps` tells MpCCI whether or not to generate a new coupling step in the tracefile during a call of the subroutine `CCIcheckconv`, default is [on]. The parameter `format` is

[^1]: Hierarchical Data Format, [www.hdfgroup.org](http://www.hdfgroup.org)
tells MpCCI the format of the tracefile to be written. [hdf5] is the current HDF format and the default setting. It is recommended for large tracefiles. The older HDF format [hdf4] is provided for compatibility reason to former MpCCI version.

```plaintext
control
  tracefile( name = "test.ccv",
             close_after_writing = on, % default: off
             trace_comm_values = on, % default: on
             trace_mesh_values = on, % default: off
             trace_mesh_change_values = off, % default: on
             implicit_coupling_steps = on, % default: on
             format = hdf4 % default: hdf5
           );
end
```

**check_mesh_quality:** This statements turns on (or off) the mesh quality checks during CCIclosesetup. The checks give a warning if there are badly shaped elements or other inconsistencies of the mesh. By default it is turned on and some parameters of the mesh tests can be set in the subsequent items

- mesh_quality_nodes,
- mesh_quality_triangle,
- mesh_quality_triangle6,
- mesh_quality_quad,
- mesh_quality_quad8 and
- mesh_quality_quad9.

The setting **check_mesh_quality(on)** may cause an MpCCI internal problem if the coordinates of the nodes are big. This problem can be avoided by turning the checks off or by raising the tolerance with **mesh_quality_nodes**.

**mesh_quality_nodes:** MpCCI checks for each coupling partition if the specified nodes are all different. That is, no two different nodes $v_1$ and $v_2$ may be the same within a given tolerance $\text{tol}$, i.e. $\|v_1 - v_2\| < \text{tol}$.

This statement can be used to specify an alternate tolerance for this check. The default is $\text{tol} = 1^{-7}$.

```plaintext
mesh_quality_nodes( tol = 1e-5 ); % default 1e-7
```

**mesh_quality_quad:** Specify parameters for the mesh quality checks for quadrilaterals.
mesh_quality_quad(taper = 0.4, % default 0.35
aspect_ratio = 30.0, % default 39.0
skew = 60.0, % default 60.0 degrees
warp = 5.0 ); % default 5.0 degrees

**mesh_quality_triangle:** Specify parameters for the mesh quality checks for triangles.

mesh_quality_triangle( aspect_ratio = 30.0, % default 39.0
skew = 45.0 ); % default 45.0 degrees

**mesh_quality_quad8, mesh_quality_quad9, mesh_quality_triangle6:** Specify parameters for the mesh quality checks for \texttt{CCIelemquad8}, \texttt{CCIelemquad9} and \texttt{CCIelemtriangle6}. The syntax and the meaning of the parameters are the same for these three element types, so we only give an example for \texttt{CCIelemquad8}.

mesh_quality_quad8(taper = 0.4, % default 0.35
angle = 150.0, % default 150.0 degrees
area = 0.0 ); % default 0.0

There are two more surface element types in MpCCI: \texttt{CCIelempentagon} and \texttt{CCIelemhexagon}. For interpolation these elements are decomposed into triangles (\texttt{CCIelempentagon}) or into quadrilaterals (\texttt{CCIelemhexagon}). The element checks are therefore done on the elements of the decomposition. So the parameters set in \texttt{mesh_quality_triangle} and \texttt{mesh_quality_quad} are relevant for \texttt{CCIelempentagon} and \texttt{CCIelemhexagon} as well. The shape of the faces of the three-dimensional elements are checked in the same way.

**skip_element_checks:** If this parameter is set no element checks will be performed. The default value is \textit{false}.

**timeout:** The \texttt{timeout} statement specifies the maximum allowed time in seconds between two consecutive messages arriving at the control process. This is a primitive method for dead-lock detection. The default is 0, which means that this mechanism is not active.

**allow_core_dump:** If this parameter is set a core dump will be written in case of abnormal termination. The default value is \textit{off}.

**restart_file_name** and **restart_file_access:** With the \texttt{restart file} settings the neighborhood relations can be stored in files to be reused and skip the neighborhood search which can be very time-consuming. The \texttt{restart_file_name} defines the base of the file names: if \texttt{’xxx’} is set, \texttt{’xxx.1’} will keep the data of process 1, \texttt{’xxx.2’} that of process 2, etc. If \texttt{restart_file_access} is set to \texttt{[write]}, the neighborhood relations will be written to files. Then, \texttt{restart_file_access} can be set to \texttt{[read]} in order to reuse this information. Default setting is \texttt{[none]}.  

**contact_detection:** This statement could be used to specify attributes of the grids and to define the contact detection algorithm and matching criterion to use in former MpCCI versions. The user should use
the contact block instead. See [3.3.5 Contact block] about the contact block for information about the parameters.

### 3.3.5 Contact block

This optional block is used to specify attributes of the grids and to define the contact detection algorithm and matching criterion to use. This block substitutes the `contact_detection` statement of the control block.

```plaintext
contact
  type = PP|PE|EE|nearest_neighbor;
  overlap = full|partial;
  warn_orphaned_points = on|off;
  user_performs_contact_search = false|true;
  global_dim = n;
  coupling_dim = m;
  pre_contact_search = search_alg ( 
    search_par_name1 = search_par_value1,
    ... 
    search_par_name_n = search_par_value_n 
  );
  matching_criterion = match_crit_name ( 
    match_par_name1 = match_par_value1,
    ... 
    match_par_name_n = match_par_value_n 
  );
  default_alg ( 
    default_alg_par_name1 = default_alg_par_value1,
    ... 
    default_alg_par_name_n = default_alg_par_value_n 
  );
  alg alg_name1 ( 
    alg_par_name1 = alg_par_value1,
    ... 
    alg_par_name_n = alg_par_value_n 
  );
  mesh_pair ( job_name1/meshId_m, job_name2/meshId_n ) : alg_name1;
end
```

**type:** Default setting for the grid type is PE that switches to the standard interpolation for non-matching grids. For matching grids, i.e. the nodes of one grid are (almost) the same of the other, the type must be
PP. With setting the `type` to `EE` for non-matching grids the user can switch to a neighborhood search and an interpolation based on intersection methods. If `type` is set to `nearest_neighbor` interpolation and neighborhood search is substituted by the nearest neighbor algorithm: For each node of the local grid the node of the remote grid is computed which has the minimal distance from the local grid point. The value on this node will then be transferred to the node of the remote grid. The value of `skip_element_checks` of the control block is set to `off` automatically.

**overlap:** The `overlap` region can be `partial` or `full`, which is the default value. If the `overlap` is `partial`, no brute force search for orphaned nodes or elements is executed. This can be useful for grid type `EE`.

**warn_orphaned_points:** The setting `warn_orphaned_points` defines if the test for orphaned points should be made. A warning will be given if orphaned points are found. The test can take some time, so if the overlap is partial this switch should be set to `off`. Default is `on`.

**user_performs_contact_search:** The setting `user_performs_contact_search` enables the user to switch to a user-defined contact search algorithm, default is `false`. The definition of a contact search algorithm by the user is only possible if the interface ”user def. interpolation” is installed. This interface is not available by now.

**global_dim:** The setting defines the global dimension, default is 3 (see ▷ 3.1.4 Coupling Regions ◄).

**coupling_dim:** The setting defines the dimension of the coupling region, default is 2 (see ▷ 3.1.4 Coupling Regions ◄). For instance, for a coupling of two codes in the 2-dimensional space (along a 1-dimensional coupling line), the contact block has to be added with the following two statements:

```plaintext
contact
  ...
  global_dim  = 2;
  coupling_dim = 1;
  ...
end
```

**pre_contact_search and matching_criterion:** For the `pre_contact_search` algorithm and for the `matching_criterion`, parameters may be specified in brackets. If no parameters for the pre-contact search algorithm or matching criterion are specified, then defaults are used. The specification of pre-contact search algorithms and matching criteria may also be omitted entirely; in this case, default for `pre_contact_search` algorithm is `bucket` and for the `matching_criterion` is `minimal_distance`. The possible algorithms are described in more detail in ▷ V-3.2 Data Exchange ◄.

**default_alg:** The item `default_alg` allows the user to redefine the internal defaults, e.g. `type`, `global_dim` or `coupling_dim`, for the neighborhood search algorithm. There are four different combinations possible for the global and the coupling dimension: (3,2), (3,3), (2,2), and (2,1).

**alg and mesh_pair:** The item `alg` allows the user to define specific neighborhood search algorithms. Settings of the contact block, e.g. `type`, `global_dim` or `pre_contact_search`, can be summarized with a name for the algorithm. By giving this name in the `mesh_pair` definition the defined neighborhood search
algorithm can be applied for a pair of meshes each identified by `jobname` and `meshId`. Thus, different neighborhood search algorithms can be used within a coupling of codes with different meshes.

In the following example an algorithm “mySpherContactAlg” is defined with different settings for the contact search of grids consisting of spherical quadrilaterals as one can see at the `pre_contact_search` item (`bucket_sphquad`). The dimension items are left out as they are properly defaulted. Then, the algorithm defined is set to be used for the neighborhood search between mesh 2 of “myJob1” and mesh 4 of “Jobname2”.

```
contact
  ...
  alg mySpherContactAlg (  
    type = EE,
    overlap = partial,
    warn_orphaned_points = off,
    pre_contact_search = 
      bucket_sphquad(
        bbox_expansion = 2, 
        bucket_expansion = 1.5  
      ),
    matching_criterion = 
      intersection_sphquad( 
        rejection = 0.01,
        equals_zero = 2.0e-6,
        perform_tests = false,
        scale_result = false,
      ),
  );
  mesh_pair ( myJob1/2, Jobname2/4 ) : mySpherContactAlg;  
end
```

### 3.3.6 Switches block

```
switches
  switch_name_i = value_i;
  ...
  group_name_j :  
    switch_name_k = value_k;
  ...
end
```
The optional switches block defines the general behavior of MpCCI. For this purpose we distinguish between groups and switches. A simple example of a switch is the integer switch `output_level`, which determines how much output MpCCI writes:

```plaintext
switches
  output_level = 3;
end
```

The following switches are available:

- **output_level**
  - 0: no output, default value
  - 1: print message when entering and leaving a MpCCI subroutine
  - 2: print input and output arrays as well
  - 3: print maximal output
- **debug_level**
  - 0: no debugging output, default value
  - 1: produce debugging output

A group identifies a part of the program with a given name. The group mechanism is used to specify for which part of MpCCI the switch holds. For instance,

```plaintext
switches
  cci_put_nodes:
    output_level = 3;
end
```

specifies that an output level of 3 must be used for MpCCI subroutine `CCIputnodes`.

Some groups are subdivided into subgroups so that it is also possible to define a switch value for several parts of MpCCI at the same time. It is also possible to do this, and define a special switch value for one subgroup which overwrites the default. For instance,

```plaintext
switches
  cci_subroutines:
    output_level = 3;
  cci_put_nodes:
    output_level = 2;
end
```

specifies that for all MpCCI subroutines an output level of 3 must be used, except for MpCCI subroutine `CCIputnodes`, for which an output level of 2 is used.

The ordering in which switches are specified for groups and subgroups is irrelevant. So, in the above example, it would have been equivalent to specify the switch `output_level` for group `cci_put_nodes` before that for group `cci_subroutines` instead. More than one switch may be specified for a group. The construct `group_name:` merely specifies that all subsequent switches apply to the indicated group.

If a switch is specified at the start of a `switches` block, then it applies to all groups, except those for which another value has been specified. So for example,

```plaintext
switches
```

158 VIII MpCCI 3.1.1-1
output_level = 3;
cci_put_nodes:
    output_level = 2;
end

specifies a default output level of 3 for all groups. The output level of MpCCI subroutine `CCIputnodes` will be 2.

The following group names are available:

```plaintext
cci_subroutines
    cci_init
    cci_init_with_id_string
    cci_setup
        cci_def_partition
        cci_def_nodes
        cci_def elems
        cci_def_comm
        cci_def_sync_point
        cci_def_search_tags
        cci_close_setup
    cci_putget
        cci_put_nodes
        cci_get_nodes
        cci_put elems
        cci_get elems
    cci_sendrecv
        cci_send
        cci_isend
        cci_recv
        cci_reach_sync_point
        cci_wait
        cci_iprobe
        cci_check_convergence
    cci_remeshing
        cci_remesh
        cci_mod_nodes
        cci_close_remesh
    cci_info
        cci_parameter_info
        cci_general_info
        cci_sync_point_info
        cci_comm_info
        cci_get_id_string
```
The indentation in this list shows the hierarchy of groups and subgroups. The group \texttt{cci\_subroutines} comprises some artificial subgroups and all MpCCI subroutines starting with the \texttt{CCI} prefix. On the other hand, the subgroup \texttt{cci\_send} belongs to the subgroup \texttt{cci\_sendrecv} and of course to the group \texttt{cci\_subroutines}. An other top-level group is \texttt{neighborhood\_search} that refers to information about the used search algorithms. Finally, a switch \texttt{>0} for the group \texttt{timing} will generate runtime information of the subroutines and output it in \texttt{CCIfinalize}.

The interpretation of a switch by a group depends on the group. Some parts of MpCCI may not use a particular switch at all.

### 3.3.7 Jobs block

This section specifies the codes that participate in the coupled computation, and how they are to be started. We use the term 'job' instead of 'code' to allow for multiple occurrences of a single code in the coupled computation.

To allow for any type of programming model, different processes of the parallel code may be started in different ways, which is indicated by the process groups. For an SPMD code there is only one process group. For a host-node parallelization there are two: one containing only one process called the host, and one containing the node processes. A job may also consist of more than two process groups.

A simple example for a jobs block is:

```plaintext
jobs
  struc1 = struc(
    nprocs = 10,
    env(STRUC_JOB) = "flap"
  );

  ... \% job2

end
```

The process group name \texttt{struc} after the arbitrary job name "struc1" identifies the code information already specified for \texttt{struc} in a code block. Since the start-up information for the code is already known, one does not need to repeat it. It is allowed, however, to override the defaults specified in the code block.
For each process group the following specifications can be given: `pwd`, `pre_exec`, `exec`, `post_exec`, `nprocs` and `env`.

The `pwd` describes in what directory the job will run, and this directory will be set by the `CCIinit` call. The execution string `exec` specifies the command needed to start up the process. The command will be searched for first in the directory indicated by `pwd`, and then in the directories in the normal search path for executables. The `nprocs` command describes how many processes are to be started.

Environment variables can be specified optionally using the `env` construct. This design allows for the case where different process groups or codes require different values for the same environment variable.

Optionally one can use the `pre_exec` and `post_exec` scripts to be executed before and after the coupled computation runs. The scripts should not read standard input. When the script is run, only the working directory is set as specified in the input file. Other environmental settings are not in effect (system defaults). Because the scripts are run prior to starting the coupled computation, any environment variables that are set are not in effect anymore during computation.

The `pre_exec` and `post_exec` facilities are useful for example when a code requires its input files to be distributed to the local disks of the hosts where it will run. By doing so, I/O is optimized because different processes will perform I/O on different hard disks. The scripts are executed with the specified arguments. The name of a host list file for this process group is added by the start-up procedure as final argument. The tasks specified by the `pre_exec` and `post_exec` must be done during the MpCCI start-up procedure since it is unknown beforehand on what hosts the process group will run, and because with some scheduling systems (e.g. EASY), one does not have access to the remote hosts until the processes are available.

The specified environment variables and working directories are in effect after the `CCIinit` call. The default environment variables and working directories depend on the platform and MPI implementation used.

An example job specification that shows all settings is as follows:

```plaintext
jobs

  % Define environment variables which are set in all jobs
  % (may be overwritten in each job):
  env( ENVVAR5 ) = "value5";
  env( ENVVAR6 ) = "value6";

  % Define the directory of the MpCCI distribution (for license mechanism).
  % Valid for all jobs (unless overwritten there). Overwrites the default
  % value from the "addcontrol" block.
  cci_home_dir = "/home/user/mpcci";

  % Job 1 consists of two groups of processes: 1 host and 3 nodes.
  % It executes the code "CodeName1" (this must be the name of one of
```
% the code blocks above).

JobName1 = CodeName1 (  
    % Define startup information and environment variables  
    % for all process groups of this job. They overwrite  
    % the defaults from the corresponding code block "CodeName1"  
    % and from the above environment variables which are set  
    % in all jobs, but the settings here may be overwritten  
    % in each separate process group below.  
    pwd = ".",
    pre_exec = "ex3.pre arg1",
    exec = "ex3Binary1",
    post_exec = "ex3.post arg1 arg2",
    nprocs = 1,
    env( ENVVAR7 ) = "value7",
    env( ENVVAR8 ) = "value8"
)

: host (  
    % Define start-up information and environment variables  
    % for the 'host' process group. This overwrites all  
    % previous defaults.  
    pwd = ".",
    pre_exec = "host.pre arg1",
    exec = "host",
    post_exec = "host.post arg1 arg2",
    nprocs = 1,
    env( ENVVAR7 ) = "value7a",
    env( ENVVAR8 ) = "value8a"
)
+ nodes (  
    pwd = ".",
    pre_exec = "ex3.pre arg1",
    exec = "node",
    post_exec = "ex3.post arg1 arg2",
    nprocs = 2,
    env( ENVVAR7 ) = "value7b",
    env( ENVVAR8 ) = "value8b"
);

% Additional process groups would be allowed here.  
% You would add them as further summands.

% Job 2 consists of one process group only and executes the code  
% "CodeName2".

162 VIII MpCCI 3.1.1-1
The above example defines two jobs `JobName1` and `JobName2`. The first job uses a host-node parallelization with 1 host process and 2 node processes. The second job is an SPMD job with 1 process.

The code ids given to jobs during coupled computation are determined by the order in which they were specified in the \texttt{MpCCI} input file. The first job gets code id 1, and the $i$-th job gets id $i$.

\texttt{MpCCI} needs the home directory of the \texttt{MpCCI} distribution to check the validity of the license. Usually the name is available through the environment variable \texttt{CCIhome}. With \texttt{cci_home_dir} the search for the license file can be redirected.

### 3.3.8 Parameters block

The parameters block is optional. It enables the user to provide values of different kind (integer, float, string) to the codes involved. A value is identified by a string given in the form

```
parameter
   \text{string}_i = \text{value}_i;
end
```

The codes can query these values via \texttt{CCIparaminfo}. An example illustrates the use of the parameters block:

```
parameters

% Default settings for all jobs:
temperature = 12.34;
pressure    = 67.89;
nSteps      = 1000;

% Specific settings for job 1 (overwrite the above defaults):
JobName1( temperature = 1234.5678,
          pressure  = 5678.12,
          mode      = "sequential",
          additional = 5678 );
```
In this example the integer value 1000 is assigned to the parameter `nSteps`. This setting is valid and accessible by the `MpCCI` subroutine `CCEparaminfo` for all codes. Then, there is a specification of values for `JobName1`. The job name has to be the one already used in the previous `jobs` block. This statement defines four parameters. The value of parameter ”temperature” is changed to 1234.5678 for `JobName1` because this setting is more specific. The parameter ”mode” is of type ”string”, so the value has to be written in quotation-marks. `JobName2` will operate with a temperature of 1200.0. The names of parameters are not allowed to contain any blanks.

Note: the `MpCCI GUI` will present the parameters only if they are defined in the code or quantities blocks before!

### 3.3.9 Coupling block

The coupling block is optional. In this block synchronization points and communicators can be defined and the matching of the definitions can be specified.

```
coupling
  % Definition of synchronization points:
syncpt job_name1( sync_point_id1 ) :
  send( send_quantity_id1/send_mesh_id1, send_quantity_id2/send_mesh_id2, ...),
  recv( recv_quantity_id1/recv_mesh_id1, recv_quantity_id2/recv_mesh_id2, ...);
  % matching of the synchronization points of the different jobs:
match_syncpt
  job_name1( sync_point_id1, sync_point_id2, ...)
  = job_name2( sync_point_id_a, sync_point_id_b, ...)
  = job_name3( sync_point_id_A, sync_point_id_B, ...)
  = ... ;
  % Redefinition of the predefined communicators:
comm job_name1( with job_name2 ) :
  send( send_quantity_id1/send_mesh_id1, send_quantity_id2/send_mesh_id2, ...),
  recv( recv_quantity_id1/recv_mesh_id1, recv_quantity_id2/recv_mesh_id2, ...);
  % Switch for the redefined communicators:
set_communicators = true;
  % User-defined communicators:
comm job_name1( comm_id1 ) :
  send( send_quantity_id1/send_mesh_id1, send_quantity_id2/send_mesh_id2, ...),
```
recv( recv_quantity_id_1/recv_mesh_id_1, recv_quantity_id_2/recv_mesh_id_2, ...);
comm job_name_1( comm_id_2 ) :
    sendrecv( sendrecv_quantity_id_a/mesh_id_a, sendrecv_quantity_id_b/mesh_id_b, ...);
% matching of communicators of the different jobs:
match_comm job_name_1( comm_id_1 ) = job_name_2( comm_id_a );
end

**Synchronization points**

A synchronization point initiates a number of send and receive operations defined here for all codes which take part in a synchronization point. This has to be done for all these codes in the following way:

```
syncpt JobName( SyncPointId ) : send( quantityId1/meshId1, quantityId2/meshId2, ...),
                                recv( QuantityId1/MeshId1, QuantityId2/MeshId2, ...);
```

It is possible to define a synchronization point containing e.g. a number of send (or receive) operations only. In this case, the `recv` part (or the `send` part) can be left out. The meshes can be named by `meshId` or by `idString` as defined with `CCIdefmesh`. If the codes have just one mesh, the mesh specification can be omitted.

To define the synchronization point with id `syncPointId` for job `jobName` the subroutines `CCIsyncinfo( syncPointId, ...)` and `CCIdefsync( syncPointId, ...)` must be called in the initialization phase of the code. To start the action `CCIreachsync( syncPointId, ...)` must be called (after the call of `CCIclosesetup`). Instead of defining a synchronization point in the input file the user may call `CCIdefsync` in the initialization part of each code concerned.

If synchronization points are used for communication one can specify which synchronization points of the codes match each other even if they are not defined in the coupling block. If no `match_syncpt` is given although there are synchronization points defined it is assumed that matching synchronization points of the different codes have the same identifier. If the matching is done explicitly this must then be done with the help of the `match_syncpt` statement:

```
match_syncpt jobName1( syncPointId1, syncPointId2, ...)
    = jobName2( idSyncPoint1, idSyncPoint2, ...)
    = jobName3( syncPoint1, syncPoint2, ...)
    = ... ;
```

This statement is interpreted as follows: The synchronization point `syncPointId1` of code `jobName1` matches with the synchronization point `idSyncPoint1` of code `jobName2` etc. and the synchronization
point \( \text{syncPointId2} \) of code \( \text{jobName1} \) matches with the synchronization point \( \text{idSyncPoint2} \) of code \( \text{jobName2} \) etc. **Caution:** Every code from the job block must be in that enumeration! It is not allowed to repeat the \text{match_syncpt} statement, i.e. every synchronization point you want to use in your code must be in that enumeration, even those you are going to define in your code without the \text{syncpt} statement described above. If a code does not take part in a synchronization communication you must write '-' in the enumeration instead of a synchronization point id. For example: \( \text{syncPoint1} \) in the enumeration belonging to the code \( \text{jobName3} \) may be equal to '-' instead of being an integer.

If synchronization points are defined in the code and no matching criterion is given in the input file then matching synchronization points of different codes must have the same identifier.

We give an example for synchronization point definition via the coupling block:

```
coupling

  syncpt JobName1( 2 ) : send( temperature1/1, temperature2/1 ),
                  recv( temperature0/2 );
  syncpt JobName2( 1 ) : send( Temperatur0/1 ),
                  recv( Temperatur1/1, Temperatur2/2 );

  match_syncpt JobName1( 2, 5 )
      = JobName2( 1, 6 );

end
```

In this example the quantities \text{temperature1} and \text{temperature2} will be sent by code \text{JobName1} at the synchronization point with the id 2. The \text{meshId} is 1 in both cases. In addition, the code \text{JobName1} may receive the quantity \text{temperature0} on mesh 2 at the same synchronization point 2.

**Caution:** No check will be done whether there is a sending code for a corresponding quantity due to the quantities block. That means that it is up to the user to decide whether code \text{JobName1} will actually receive values for \text{temperature0}. The user can be sure that the code does not receive values, which were not intended to be received. Whether there is a receiving code for the quantities which are sent will not be checked either. But, if there is more than one code sending a quantity an error will occur. For there are only two codes involved in the coupled computation of the example above no error message will be given.

For code \text{JobName2} the following communication is defined at synchronization point 1: \text{Temperatur0} on mesh 1 will be sent and \text{Temperatur1} on mesh 1 and \text{Temperatur2} on mesh 2 may be received. How the quantities of the codes match each other is up to the quantities block.

In this example there are two synchronization points defined for each code: synchronization points 2 and 5 for code \text{JobName1} and synchronization point 1 and 6 for code \text{JobName2}. For there is only one synchronization point defined for each code in the input file the other synchronization points can only be defined in the codes themselves.
The `match_syncpt` statement implies that the synchronization point 2 of code `JobName1` matches the synchronization point 1 of code `JobName2`, and the synchronization point 5 of code `JobName1` matches the synchronization point 6 of code `JobName2`.

**Communicators**

For the definition of communicators we distinguish between two types of communicators:

- predefined communicators (i.e. `CCIcommrcode`)
- user-defined communicators.

For the specification of the predefined communicators we use the following type of the `comm` statement:

```
comm JobName1( with JobName2 ) : send( quantityId1/meshId1, quantityId2/meshId2, ...),
    recv( QuantityId1/MeshId1, QuantityId2/MeshId2, ...);
```

It is possible to define a communicator for send (or receive) operations only. In this case, the `recv` part (or the `send` part) can be left out. Moreover, a `*` can be used instead of naming all quantities. This wild card is also very helpful when coupling more than two codes. As another easement, it is possible to name the mesh by the `meshId` or by `idString` as defined with `CCIdefmesh`. If the codes have just one mesh, the mesh specification can be omitted. Finally, the send and the receive part can be combined with `sendrecv` if the quantities are exactly the same for both directions.

The `set_communicators` statement tells MpCCI that the predefined communicators `CCIcommrcode` are redefined according to the above settings, i.e. `CCIdefcomm` is called implicitly during `CCIclosesetup` for these communicators (unless the code itself has called `CCIdefcomm`).

```
set_communicators = true;
```

For user-defined communicators the `comm` statement must be used in the following way:

```
comm JobName1( CommId1 ) : send( quantityId1/meshId1, quantityId2/meshId2, ...),
    recv( QuantityId1/MeshId1, QuantityId2/MeshId2, ...);
```

In both cases the communicators are then prepared to send the quantities specified in the `send` part of the statement and prepared to receive the quantities specified in the `recv` part of the statement.
The possibilities mentioned above for naming the quantities easily can be used for user-defined communicators too. The following definition for instance covers the case that the send and the receive quantities are equal:

```plaintext
comm JobName1( CommId2 ) : sendrecv( quantityIdA/meshA, quantityIdB/meshB, ...);
```

To define a user-defined communicator with id \( \text{commId} \) for job \( \text{jobName} \) the subroutines \( \text{CCIcomminfo}( \text{commId}, ...) \) and \( \text{CCIdefcomm}( \text{commId}, ...) \) must be called in the initialization phase of the code. For predefined communicators this can be omitted.

For user-defined communicators it is necessary to specify how the defined communicators match each other (the predefined communicators provide this information by definition). We use the \( \text{match_comm} \) statement to store the matching information:

```plaintext
match_comm     jobName1( commId1 )
               = jobName2( commIdA );
```

The parameters \( \text{commId}* \) must be identifiers of user-defined communicators. This statement is interpreted as follows: The communicator \( \text{commId1} \) of job \( \text{jobName1} \) matches with the communicator \( \text{commIdA} \) of job \( \text{jobName2} \). The \( \text{match_comm} \) must be called separately for every pair of user-defined communicators which are specified in the input file.

Communicators and synchronization points can also be defined in the initialization phase of the codes with the help of the subroutines \( \text{CCIdefcomm} \) and \( \text{CCIdefsync} \).

### 3.3.10 Additional block

The \textit{additional} block is optional. It can be used to define and initiate some additional features for the coupling. In the current \textit{MpCCI} version the following features are available:

- Coordinate transformation
- Computation of the intersection points between the elements (or their projection) of two meshes.
- Determination of the non-orphaned nodes and elements of the meshes

The \textit{additional} block of the \textit{MpCCI} input file has the following form:

```
additional
```
Coordinate transformation

For a mesh of a certain code a coordinate transformation for the node coordinates can be defined. For the parameters are the same as in the \texttt{MpCCI} subroutine \texttt{CCIdefctf} a description is left out here. A coordinate transformation for job \texttt{JobName1} for mesh 1 can be defined in the following way:

\begin{verbatim}
coord_system JobName1 ( mesh_id = 1, 
global_dim = 3, 
type = cart, 
equals_zero = 1.0e-6, 
origin = ( 0.5, 0.5, 0.5), 
unit_vectors = ( 0.5, 0.0, 0.0, 
  0.0, 0.5, 0.0, 
  0.0, 0.0, 1.0 ) );
\end{verbatim}

Intersection points

The subroutine \texttt{CCIgetisp} must not be used without the \texttt{inters_points} statement in the \texttt{additional} block. For mesh pairs for which an EE neighborhood algorithm was defined an \texttt{inters_points} statement may be defined.

\begin{verbatim}
inters_points ( JobName1/1, JobName2/1 );
\end{verbatim}

This statement holds for both meshes, so getting intersection points is prepared for both meshes specified here. In the current version one mesh must not compute the intersection points with more than five remote meshes.

Non-orphaned objects
If the identifier of the non-orphaned nodes or elements of a mesh should be given in `CCIgetnodes` or `CCIgetelems` the `non_orphaned_objects` statement must be used to prepare this. This statement can also be used to get the identifiers of the orphaned nodes and elements. This holds for `partial` and `full` overlaps (see `contact` block). The `non_orphaned_objects` statement overwrites the `warn_orphaned_points` statement of the control block.

If the user-defined node identifiers shall be returned in the call of `CCIgetnodes` in addition a coordinate transformation must be defined otherwise the `MpCCI` internal identifiers (i.e. the position in the coordinate array in the call of `CCIdefnodes`) is returned there. Please remember that all elements must be defined in one call of `CCIdefelems` if a coordinate transformation is defined.

```plaintext
coord_system JobName2 ( mesh_id = 1,
                         global_dim = 3,
                         type = cart,
                         equals_zero = 1.0e-6,
                         origin = ( 0.0, 0.0, 0.0 ),
                         unit_vectors = ( 1.0, 0.0, 0.0,
                                          0.0, 1.0, 0.0,
                                          0.0, 0.0, 1.0 ) );
non_orphaned_objects ( JobName2/1 );
```

### 3.3.11 Include Mechanism

Typically the different input blocks described above that refer to the configuration of a code are located in different files, which have to be merged into one using an include mechanism: In the `MpCCI` input file, one includes the required configuration files. The `MpCCI` environment variable `CCIpath` is the search path used for finding the configuration files.

A typical `MpCCI` input file for a coupling between a structure code (`struc`) and a fluid code (`cfd`) will look like this:

```plaintext
include "struc-config.cci"; % specific settings for struc
include "cfd-config.cci"; % specific settings for cfd
include "struc-cfd-config.cci"; % how do the quantities of
                                 % struc and cfd match

% first block of the input file defined by the user
control
... end

% second block of the input file defined by the user
... etc.
```
In this example, "struc-config.cci" and "cfd-config.cci" describe the code configuration files for code struc and cfd, and struc-cfd-config.cci describes additional information for the coupling of struc and cfd. The latter will include a definition of how coupling quantities used by the different codes match.

3.4 An Example

This section describes an example of two codes coupled with each other. For ease of exposition, we consider a two-dimensional example with a one-dimensional coupling region. The term coupling surface or coupling region used before, now refers to this coupling line.

The example is the coupling of a fluid (F) with a structures code (S). The coupling method considered is one where the structures code requires forces and heat fluxes at the nodes, that are computed by the fluid code at the cell-centers. The fluid code on the other hand requires the new mesh coordinates and velocities from the structures code at its nodes. The structures code computes these quantities at its nodes. The grids do not match at the grid interfaces. The coupling is depicted in Figure 26. The example code fragments are in FORTRAN.

![Figure 26: Two-dimensional coupling example of a structures with a fluid code.](image-url)
3.4.1 Start-up and Initialization

In the coupled computation, both code $S$ and $F$ use 4 processes. Code $S$ only wants to call MpCCI subroutines from the two processes that lie at the grid interface. The fluid code wants to call MpCCI subroutines from all its processes, which implies of course that some processes of code $F$ do not have a coupling partition. The jobs block of the MpCCI input file may look the following way:

```
jobs
  JobName1 = S (
    pwd = ".",
    exec = "binary1",
    nprocs = 2
  );
  JobName2 = F (
    pwd = ".",
    exec = "binary2",
    nprocs = 4
  );
end
```

where $binary1$ and $binary2$ are the names of the executables.

The initialization of the coupled computation starts with the enrolling of both codes the coupled computation:

```
c    --- enroll in the coupled computation
    call CCI_INIT(ierr)
```

3.4.2 Coupling Definition

In the coupling definition phase, each of the codes specifies its coupling region to MpCCI. In this example only one mesh with one partition on each processor is used. For process $S4$ this looks like:

```
c    --- specify partitions
    meshId = 1
    partId = 1
    call CCI_DEF_PARTITION(meshId, partId, ierr)
```

```
c    --- specify nodes
```
globalDim = 2
nNodes   = 3
nNodeIds = 0
nodeIds  = 0
dataType  = CCI_REAL

coreds(1)  = 0.0e0  ! x coordinate of the first node
coreds(2)  = 0.0e0  ! y coordinate of the first node
coreds(3)  = 0.0e0  ! x coordinate of the second node
coreds(4)  = 0.5e0  ! y coordinate of the second node
coreds(5)  = 0.0e0  ! x coordinate of the third node
coreds(6)  = 1.0e0  ! y coordinate of the third node

call CCI_DEF_NODES(meshId, partId, globalDim, nNodes,
                   +       nNodeIds, nodeIds, dataType, coords, ierror)

For the interpolation chosen in this example information concerning the topology of elements is necessary. This will be defined as follows:

c --- specify elements by defining their nodes

nElems       = 2
nElemIds     = 0
elemIds      = 0
nElemTypes   = 1
elemTypes    = CCI_ELEM_LINE
nNodesPerElem = 2

elemNodes(1) = 1
elemNodes(2) = 2
elemNodes(3) = 2
elemNodes(4) = 3

call CCI_DEF_ELEMS(meshId, partId, nElems, nElemIds,
                    + elemIds, nElemTypes, elemTypes, nNodesPerElem,
                    + elemNodes, ierr)
The processes \( F_2 \) and \( F_4 \) need not define a mesh. Note that in Table 6, the nodes at the centers of the cells of code \( F \) have not been specified. This is because these nodes are derived nodes, whose coordinates can be computed from the other nodes. The definition of the derived nodes as a function of the other nodes is part of the element definition.

Besides the definition of the topology of the coupling region, each code has to specify the locations of the coupling quantities for the coupling with each of the other codes separately. It can be specified in the \texttt{MpCCI} input file if coupling values reside at the element centers. The code blocks and the quantities block of the \texttt{MpCCI} input file should look the following way:

\begin{verbatim}

code S
  mesh( id = 6 );
  velocity( id = 7, loc = node, dim = scalar, type = field );
  force( id = 8, loc = node, dim = 2, type = flux );
  heat( id = 9, loc = node, dim = scalar, type = flux );
end

code F
  mesh( id = 10 );
  velocity( id = 11, loc = node, dim = scalar, type = field );
  force( id = 12, loc = elem, dim = 2, type = flux );
  heat( id = 13, loc = elem, dim = scalar, type = flux );
end

quantities
  S.velocity = F.velocity;
  S.force = F.force;
  S.heat = F.heat;
end
\end{verbatim}

\texttt{MpCCI} can only map fluxes to the nodes and from elements to elements. Fields can be interpolated from nodes and from elements to elements.

In the definition phase other subroutines may be called for defining synchronization points, communicators, search tags etc. For instance the definition of a communicator in the structure code could look as follows:

\begin{verbatim}

VIII MpCCI 3.1.1-1
The remote communicator 321 must be defined in the remote code which is the fluid code in this example. The communicator 321 should be able to communicate the quantities 10 and 11 which match the quantities 6 and 7 of the structure code as defined in the input file above.

The third step of the initialization phase is to define the \texttt{MpCCI} processes to \texttt{MpCCI}. Thus after the coupling regions have been completely specified, \texttt{FCCclosesetup} must be called to commit to the definition of the regions and perform the initial neighborhood computation. This is done by calling subroutine \texttt{FCCclosesetup} with an argument that indicates if the calling process is a \texttt{MpCCI} process or not.

In this example the processes \texttt{S2}, \texttt{S4}, \texttt{F1} and \texttt{F3} must call \texttt{FCCclosesetup} with the value \texttt{included = 1}. \texttt{F2} and \texttt{F4} must call \texttt{FCCclosesetup} with the value \texttt{included = 0}.

After this call, only the above defined \texttt{MpCCI} processes \texttt{S2}, \texttt{S4}, \texttt{F1} and \texttt{F3} may call \texttt{MpCCI} subroutines, with the exception of \texttt{FCCfinalize}, which must be called by all processes of the coupling. After this the coupled computation can start.
3.4.3 Coupled Computation

The exchange of coupling values through a coupling partition is simple. Here we only give pseudo-code for sending and receiving of coupling values between two codes. More details are algorithm-dependent and therefore omitted.

Code \[S\] specifies coupling values at the nodes, and can therefore call \texttt{FCCputnodes} for node-wise specification:

\begin{verbatim}
c    --- specify values at the nodes for the first quantity

    meshId     = 1
    partId     = 1
    quantityId = 6
    quantityDim = 2
    nNodes     = 3
    nNodeIds   = 0
    nodeIds    = 0
    dataType   = CCI_REAL
    values(1)  = ...
    values(2)  = ...
    ...
    values(6)  = ...

    call CCI_PUT_NODES(meshId, partId, quantityId, quantityDim,
                      + nNodes, nNodeIds, nodeIds, dataType,
                      + values, ierr)

c    --- and for the second quantity

    quantityId = 7
    quantityDim = 1
    values(1)  = ...
    values(2)  = ...
    values(3)  = ...

    call CCI_PUT_NODES(meshId, partId, quantityId, quantityDim,
                      + nNodes, nNodeIds, nodeIds, dataType,
                      + values, ierr)

c    --- send values

    nQuantityIds = 2
    quantityIds(1) = 6
\end{verbatim}
quantityIds(2) = 7
nLocalMeshIds = 2
localMeshIds(1) = 1
localMeshIds(2) = 1
comm = 123

call CCI_SEND(nQuantityIds, quantityIds,
+ nLocalMeshIds, localMeshIds, comm, ierror)

The line \texttt{comm = 123} could be exchanged by \texttt{comm = CCI_COMM_RCODE(2)} in this example.

The specification of coupling values by code \texttt{F} is different because there the values reside at the cell-centers:

c --- specify values at the elements

meshId = 1
partId = 1
quantityId = 12
quantityDim = 2
nElems = 2
nElemIds = 0
elemIds = 0
nDataPointsPerElem = 1
dataPointsPerElem = 1
dataType = CCI_REAL
values(1) = ...
values(2) = ...
...
values(6) = ...

call CCI_PUT_ELEMS(meshId, partId, quantityId, quantityDim,
+ nElems, nElemIds, elemIds,
+ nDataPointsPerElem, dataPointsPerElem,
+ dataType, values, ierror)

c --- fill array with the second quantity

quantityId = 13
quantityDim = 1
values(1) = ...
values(2) = ...
values(3) = ...
call CCI_PUT_ELEMS(meshId, partId, quantityId, quantityDim,
+               nElems, nElemIds, elemIds,
+               nDataPointsPerElem, dataPointsPerElem,
+               dataType, values, ierr)

--- send values

nQuantityIds    = 2
quantityIds(1)  = 12
quantityIds(2)  = 13
nLocalMeshIds   = 2
localMeshIds(1) = 1
localMeshIds(2) = 1
remoteCodeId    = 1
comm            = CCI_COMM_RCODE(remoteCodeId)

call CCI_SEND(nQuantityIds, quantityIds,
+               nLocalMeshIds, localMeshIds, comm, ierr)

For the quantities 12 and 13 there is no user defined communicator, so the predefined communicator
CCI_COMM_RCODE(1) must be used.

The pseudo-code for receiving coupling values is analogous.

### 3.4.4 Termination

Termination of the coupled computation must be done by calling subroutine [FCCfinalize](FCC_finalize). After calling
this subroutine, the calling process is still alive, but may not use MpCCI calls anymore. The caller can
continue its own termination sequence by closing output files and so on. Of course, any output file of a
code may also be closed before calling [FCCfinalize](FCC_finalize). In pseudo-code:

--- termination sequence of a code

... some statements for cleaning up, closing files etc....

--- exit MpCCI

call CCI_FINALIZE(ierr)

--- continue termination sequence

... last statements for cleaning up, closing files etc....
In this design, the termination sequence of the codes can be used without modification. Only the \texttt{FCCfinalize} call must be added. The \texttt{MpCCI} control process will not kill the codes, but just terminate the coupling.
IX Appendix – Contents

Quantity Reference 4

Literature 41

Glossary 42

Keyword Index 46
## Quantity Reference

<table>
<thead>
<tr>
<th>Quantity</th>
<th>ANSYS</th>
<th>Abaqus</th>
<th>CFX</th>
<th>FINE/Hexa</th>
<th>FINE/Turbo</th>
<th>FLUENT</th>
<th>FLUX</th>
<th>Flowmaster</th>
<th>ICEPAK</th>
<th>MSC. Marc</th>
<th>MSC. NASTRAN</th>
<th>PERMAS</th>
<th>PosRad</th>
<th>RadTherm</th>
<th>STAR-CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsPressure</td>
<td>s/r</td>
<td>r</td>
<td>s/r</td>
<td>s</td>
<td>s</td>
<td>s/r</td>
<td></td>
<td></td>
<td>r</td>
<td>r</td>
<td>s</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AcstPressure</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BodyForce</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoundaryMassFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoundaryMassFlux</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoundaryStaticPressure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoundaryTemp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoundaryTotalPressure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoundaryVelocityMagnitude</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoundaryVolumeFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGAngle</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGOmega</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGPosition</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGVelocity</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChargeDensity</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Current1 – Current4</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CurrentDensity</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeltaTime</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Density</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynPressure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrCond1</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrCond3</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrCondX</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrCondY</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrCondZ</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectricField</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectricFlux</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrRes1</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrRes3</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrResX</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrResY</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ElectrResZ</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enthalpy</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantity</td>
<td>ANSYS</td>
<td>Abaqus</td>
<td>CFX</td>
<td>FINE/Hexa</td>
<td>FINE/Turbo</td>
<td>FLUENT</td>
<td>FLUX</td>
<td>Flowmaster</td>
<td>ICEPAK</td>
<td>MSC.Marc</td>
<td>MSC.NASTRAN</td>
<td>PERM</td>
<td>PosRad</td>
<td>RadTherm</td>
<td>STAR-CD</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>--------</td>
<td>------</td>
<td>-----------</td>
<td>------------</td>
<td>--------</td>
<td>------</td>
<td>------------</td>
<td>--------</td>
<td>----------</td>
<td>-------------</td>
<td>------</td>
<td>--------</td>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>Film2Temp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FilmTemp</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gs00 – gs19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gv00 – gv19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HeatFlux</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HeatSource</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntFlag</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IterationNo</td>
<td>s/r</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JouleHeat</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JouleHeatLin</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LorentzForce</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MagneticField</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MagneticFlux</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MassFlowRate</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MassFlowVect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MassFluxRate</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDisplacement</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td></td>
<td>s</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NPosition</td>
<td>s</td>
<td>s</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>r</td>
<td></td>
<td>s</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OverPressure</td>
<td>s/r</td>
<td>r</td>
<td>s/r</td>
<td>s</td>
<td>s</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PhysicalTime</td>
<td>s/r</td>
<td>s</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>r</td>
<td>s/r</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PorePressure</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PorousFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RealFlag</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RefPressure</td>
<td>s/r</td>
<td>s</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RelWall2Force</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RelWallForce</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SpecificHeat</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThermCond1</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThermCond3</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThermCondX</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThermCondY</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ThermCondZ</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantity</td>
<td>ANSYS</td>
<td>Abaqus</td>
<td>CFX</td>
<td>FINE/Hexa</td>
<td>FINE/Turbo</td>
<td>FLUENT</td>
<td>FLUX</td>
<td>Flowmaster</td>
<td>ICEPAK</td>
<td>MSC.Marc</td>
<td>MSC.NASTRAN</td>
<td>PERMAS</td>
<td>PosRad</td>
<td>RadTherm</td>
<td>STAR-CD</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
<td>--------</td>
<td>-----</td>
<td>-----------</td>
<td>------------</td>
<td>--------</td>
<td>------</td>
<td>-------------</td>
<td>--------</td>
<td>-----------</td>
<td>-------------</td>
<td>--------</td>
<td>--------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>TimeStepNo</td>
<td>s/r</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
</tr>
<tr>
<td>TotalPressure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td>s</td>
<td></td>
<td>s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TotalTemp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>udm00 – udm19</td>
<td>s</td>
<td></td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uds00 – uds19</td>
<td>s</td>
<td></td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td>r</td>
<td>s</td>
<td></td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VelocityMagnitude</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voltage1 – Voltage4</td>
<td>s/r</td>
<td></td>
<td></td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VolumeFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VolumeFlowRate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wall2Force</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wall2HeatFlux</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wall2HTCoeff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wall2Temp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WallForce</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
</tr>
<tr>
<td>WallHeatFlux</td>
<td>s/r</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
</tr>
<tr>
<td>WallHTCoeff</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td>r</td>
<td>r</td>
<td>s/r</td>
<td>s/r</td>
<td></td>
<td>s/r</td>
</tr>
<tr>
<td>WallTemp</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s</td>
<td>s</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
<td>s/r</td>
</tr>
</tbody>
</table>

Table 7: Codes and Quantities: s = “code can send quantity”, r = “code can receive quantity”
### AbsPressure

Absolute pressure \([ \text{N/m}^2 ]\)

- **Code API symbol:** `MPCCI_QID_ABSPRESSURE`
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

### AcstPressure

Acoustic pressure \([ \text{N/m}^2 ]\)

- **Code API symbol:** `MPCCI_QID_ACSTPRESSURE`
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**BodyForce**  
General body force density vector \[ \text{[N/m}^3 \text{]} \]

- Code API symbol: MPCCI_QID_BODYFORCE
- Default value: 0.0
- Dimension: Vector
- Physical meaning: Momentum source
- Interpolation type: Flux density
- Coupling Dimensions: Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**BoundaryMassFlow**  
Boundary mass flow \[ \text{[kg/s]} \]

- Code API symbol: MPCCI_QID_BNDMASSFLOW
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Flux
- Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

**BoundaryMassFlux**  
Boundary mass flux \[ \text{[kg/m}^2 \text{ s]} \]

- Code API symbol: MPCCI_QID_BNDMASSFLUX
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Flux density
- Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>UDM</td>
<td></td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>SCALAR</td>
<td></td>
</tr>
</tbody>
</table>
**BoundaryStaticPressure**  
Boundary static pressure [N/m²]

- Code API symbol: `MPCCI_QID_BNDSTATICPRESSURE`
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Flux density
- Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>UDM</td>
<td></td>
</tr>
</tbody>
</table>

---

**BoundaryTemp**  
Boundary temperature [K]

- Code API symbol: `MPCCI_QID_BNDTEMPERATURE`
- Default value: 300.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Field
- Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>UDM</td>
<td></td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>SCALAR</td>
<td></td>
</tr>
</tbody>
</table>

---

**BoundaryTotalPressure**  
Boundary total pressure [N/m²]

- Code API symbol: `MPCCI_QID_BNDTOTALPRESSURE`
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Flux density
- Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>UDM</td>
<td></td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>SCALAR</td>
<td></td>
</tr>
</tbody>
</table>
BoundaryVelocityMagnitude  Boundary velocity magnitude [m/s]

- Code API symbol: MPCCI_QID_BNDVELOCITYMAG
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Field
- Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td></td>
<td>UDM</td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

BoundaryVolumeFlow  Boundary volume flow [m³/s]

- Code API symbol: MPCCI_QID_BNDVOLFLOW
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Flux
- Coupling Dimensions: Face

The quantity is currently not supported by any standard code.

CGAngle  Moving obstacle CG angle [rad]

- Code API symbol: MPCCI_QID_MO_ANGLE
- Default value: 0.0
- Dimension: Vector
- Physical meaning: Grid displacement/coordinate
- Interpolation type: max/min/sum/prod
- Coupling Dimensions: Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
**CGOmega**  
Moving obstacle CG angular velocity [rad/s]

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**CGPosition**  
Moving obstacle CG position [m]

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**CGVelocity**  
Moving obstacle CG velocity [m/s]

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
**ChargeDensity**  
Charge density [C/m³]

- **Code API symbol:** MPCCI_QID_CHARGEDENSITY
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** General
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**Current1 – Current4**  
Electric current - phase 1 – 4 [A]

- **Code API symbol:** MPCCI_QID_CURRENT1
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** General
- **Interpolation type:** max/min/sum/prod
- **Coupling Dimensions:** Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**CurrentDensity**  
Electric current density vector [A/m²]

- **Code API symbol:** MPCCI_QID_CURRENTDENSITY
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Boundary condition: face normal gradient
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM UDS</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct SPATIAL</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
### DeltaTime

Time step size [s]

- **Code API symbol:** `MPCCI_QID_TIMESTEP_SIZE`
- **Default value:** 1.0
- **Dimension:** Scalar
- **Physical meaning:** General
- **Interpolation type:** max/min/sum/prod
- **Coupling Dimensions:** Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>Abaqus</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

### Density

Density [kg/m³]

- **Code API symbol:** `MPCCI_QID_DENSITY`
- **Default value:** 1.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**DynPressure**  
Dynamic pressure \([\text{N/m}^2]\)

- **Code API symbol:** MPCCI_QID_DYNPRESSURE
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

**ElectrCond1**  
Electric conductivity - xyz \([\text{S/m}]\)

- **Code API symbol:** MPCCI_QID_ELECTCOND1
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**ElectrCond3**  
Electric conductivity - \((x,y,z)\) \([\text{S/m}]\)

- **Code API symbol:** MPCCI_QID_ELECTCOND3
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
### ElectrCondX
Electric conductivity - x [S/m]
- **Code API symbol:** MPCCI_QID_ELECTCONDX
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

### ElectrCondY
Electric conductivity - y [S/m]
- **Code API symbol:** MPCCI_QID_ELECTCONDY
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

### ElectrCondZ
Electric conductivity - z [S/m]
- **Code API symbol:** MPCCI_QID_ELECTCONDZ
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**ElectricField**

Electric field vector [V/m]

- **Code API symbol:** MPCCI_QID_ELECTRICFIELD
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Boundary condition: face normal gradient
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM UDS</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct SPATIAL</td>
</tr>
</tbody>
</table>

**ElectricFlux**

Electric flux vector [C/m²]

- **Code API symbol:** MPCCI_QID_ELECTRICFLUX
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Boundary condition: face normal gradient
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM UDS</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**ElectrRes1**

Electric resistivity - xyz [ohm m]

- **Code API symbol:** MPCCI_QID_ELECTRESV1
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct SPATIAL</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
Electric resistivity - (x,y,z) [ohm m]

**ElectrRes3**
- Code API symbol: MPCCI_QID_ELECTRESV3
- Default value: 0.0
- Dimension: Vector
- Physical meaning: Material property/general property
- Interpolation type: Field
- Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

Electric resistivity - x [ohm m]

**ElectrResX**
- Code API symbol: MPCCI_QID_ELECTRESVX
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Material property/general property
- Interpolation type: Field
- Coupling Dimensions: Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

Electric resistivity - y [ohm m]

**ElectrResY**
- Code API symbol: MPCCI_QID_ELECTRESVY
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Material property/general property
- Interpolation type: Field
- Coupling Dimensions: Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**ElectrResZ**

Electric resistivity - $z$ [ohm m]

- **Code API symbol:** MPCCI_QID_ELECTRESVZ
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**Enthalpy**

Enthalpy density [W/m$^3$]

- **Code API symbol:** MPCCI_QID_ENTHALPY
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** General
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**Film2Temp**

2 sided wall film temperature [K]

- **Code API symbol:** MPCCI_QID_FILM2TEMPERATURE
- **Default value:** 300.0
- **Dimension:** Biscalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Field
- **Coupling Dimensions:** Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
**FilmTemp**  
Film temperature [K]  
Code API symbol: **MPCCI_QID_FILMTEMPERATURE**  
Default value: 300.0  
Dimension: Scalar  
Physical meaning: Boundary condition: value  
Interpolation type: Field  
Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>RadTherm</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**gs00 – gs19**  
Global scalar 00 – 19 [-]  
Code API symbol: **MPCCI_QID_UGS_00**  
Default value: 0.0  
Dimension: Scalar  
Physical meaning: General  
Interpolation type: max/min/sum/prod  
Coupling Dimensions: Global

The quantity is currently not supported by any standard code.

**gv00 – gv19**  
Global vector 00 – 19 [-]  
Code API symbol: **MPCCI_QID_UGV_00**  
Default value: 0.0  
Dimension: Vector  
Physical meaning: General  
Interpolation type: max/min/sum/prod  
Coupling Dimensions: Global

The quantity is currently not supported by any standard code.
**HeatFlux**  
Heat flux density vector $[\text{W/m}^2]$  
Code API symbol: `MPCCI_QID_HEATFLUX`  
Default value: 0.0  
Dimension: Vector  
Physical meaning: Boundary condition: face normal gradient  
Interpolation type: Flux density  
Coupling Dimensions: Volume  

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**HeatSource**  
General heat source density $[\text{W/m}^3]$  
Code API symbol: `MPCCI_QID_HEATSOURCE`  
Default value: 0.0  
Dimension: Scalar  
Physical meaning: Energy source  
Interpolation type: Flux density  
Coupling Dimensions: Volume  

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
### IntFlag

Control switch (Int) [-]

**Code API symbol:** MPCCI_QID_INT_SWITCH  
**Default value:** 0  
**Dimension:** Scalar  
**Physical meaning:** General  
**Interpolation type:** max/min/sum/prod  
**Coupling Dimensions:** Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

### IterationNo

Iteration number [-]

**Code API symbol:** MPCCI_QID_ITERATION_COUNT  
**Default value:** 0  
**Dimension:** Scalar  
**Physical meaning:** General  
**Interpolation type:** max/min/sum/prod  
**Coupling Dimensions:** Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
**JouleHeat**  
Joule heat density [W/m³]

- Code API symbol: MPCCI_QID_JOULEHEAT
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Energy source
- Interpolation type: Flux density
- Coupling Dimensions: Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct SPATIAL</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**JouleHeatLin**  
Joule heat linearization [W/m³ K]

- Code API symbol: MPCCI_QID_JOULEHEATLIN
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Energy source
- Interpolation type: Flux density
- Coupling Dimensions: Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
</tbody>
</table>
LorentzForce

Lorentz force density vector \([N/m^3]\)

- **Code API symbol:** MPCCI_QID_LORENTZFORCE
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Momentum source
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Node, Element</td>
<td>Direct ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct SPATIAL</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

MagneticField

Magnetic field vector \([A/m]\)

- **Code API symbol:** MPCCI_QID_MAGNETICFIELD
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Boundary condition: face normal gradient
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>UDM UDS</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct SPATIAL</td>
</tr>
</tbody>
</table>
**Magnetic Flux**  
Magnetic flux density vector [T]

- **Code API symbol:** MPCCI_QID_MAGNETICFLUX
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Boundary condition: face normal gradient
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM UDS</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>SCALAR</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**Mass Flow Rate**  
Mass flow rate [kg/s]

- **Code API symbol:** MPCCI_QID_MASSFLOWRATE
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Mass source
- **Interpolation type:** Flux
- **Coupling Dimensions:** Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td></td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td></td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

**Mass Flow Vector**  
Mass flow vector [kg/s]

- **Code API symbol:** MPCCI_QID_MASSFLOWVECT
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Mass source
- **Interpolation type:** Flux
- **Coupling Dimensions:** Volume

The quantity is currently not supported by any standard code.
**MassFluxRate**  
Mass flux rate [kg/m² s]

- **Code API symbol**: MPCCI_QID_MASSFLUXRATE
- **Default value**: 0.0
- **Dimension**: Scalar
- **Physical meaning**: Mass source
- **Interpolation type**: Flux density
- **Coupling Dimensions**: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td></td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

**MassFluxVect**  
Mass flux vector [kg/m² s]

- **Code API symbol**: MPCCI_QID_MASSFLUXVECT
- **Default value**: 0.0
- **Dimension**: Vector
- **Physical meaning**: Mass source
- **Interpolation type**: Flux density
- **Coupling Dimensions**: Volume

The quantity is currently not supported by any standard code.

**NDisplacement**  
Nodal displacement [m]

- **Code API symbol**: MPCCI_QID_NODE_DIS
- **Default value**: 0.0
- **Dimension**: Vector
- **Physical meaning**: Grid displacement/coordinate
- **Interpolation type**: Field
- **Coupling Dimensions**: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PERMAS</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Buffered</td>
</tr>
</tbody>
</table>
### NPosition

Nodal position \([\text{m}]\)

- **Code API symbol:** MPCCI_QID_NODE_POS
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Grid displacement/coordinate
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Buf</td>
<td></td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Buffered</td>
<td></td>
</tr>
</tbody>
</table>

### OverPressure

Relative pressure \([\text{N/m}^2]\)

- **Code API symbol:** MPCCI_QID_OVERPRESSURE
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>ETAB</td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**PhysicalTime**

- **Physical time** [s]
- **Code API symbol:** MPCCI_QID_PHYSICAL_TIME
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** General
- **Interpolation type:** max/min/sum/prod
- **Coupling Dimensions:** Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**PorePressure**

- **Pore pressure** [N/m²]
- **Code API symbol:** MPCCI_QID_POREPRESSURE
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**PorousFlow**

- **Pore fluid flow** [m/s]
- **Code API symbol:** MPCCI_QID_POREFLOW
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: face normal gradient
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Face, Volume

The quantity is currently not supported by any standard code.
**RealFlag**

Control switch (Real) [-]

- Code API symbol: `MPCCI_QID_REAL_SWITCH`
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: General
- Interpolation type: max/min/sum/prod
- Coupling Dimensions: Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**RefPressure**

Reference pressure [N/m²]

- Code API symbol: `MPCCI_QID_REF_PRESSURE`
- Default value: 1.12e5
- Dimension: Scalar
- Physical meaning: General
- Interpolation type: max/min/sum/prod
- Coupling Dimensions: Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**RelWall2Force**

2 sided wall boundary relative force vector [N]

- Code API symbol: `MPCCI_QID_RELWALL2FORCE`
- Default value: 0.0
- Dimension: Symmetric tensor
- Physical meaning: Boundary condition: value
- Interpolation type: Flux
- Coupling Dimensions: Face

The quantity is currently not supported by any standard code.
### RelWallForce

Boundary relative force vector \([N]\)

**Code API symbol:** MPCCI_QID_RELWALLFORCE

**Default value:** 0.0

**Dimension:** Vector

**Physical meaning:** Boundary condition: value

**Interpolation type:** Flux

**Coupling Dimensions:** Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

### Residual

Global residual \([-]\)

**Code API symbol:** MPCCI_QID_GLOBAL_RESIDUAL

**Default value:** 0.0

**Dimension:** Scalar

**Physical meaning:** General

**Interpolation type:** max/min/sum/prod

**Coupling Dimensions:** Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
**SpecificHeat**  
Specific heat \([\text{J/kg K}]\)

- **Code API symbol:** MPCCI,QID_SPECHEAT
- **Default value:** 1.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**Temperature**  
Temperature \([\text{K}]\)

- **Code API symbol:** MPCCI,QID_TEMPERATURE
- **Default value:** 300.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Volume</td>
<td>Node, Element</td>
<td>Direct ETAB</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct SPATIAL</td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**ThermCond1**  
Thermal conductivity - xyz [W/m K]  
Code API symbol: MPCCI_QID_THERMCOND1  
Default value: 0.0  
Dimension: Scalar  
Physical meaning: Material property/general property  
Interpolation type: Field  
Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Line, Face, Volume</td>
<td>Node</td>
<td>Direct</td>
<td>SPATIAL</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**ThermCond3**  
Thermal conductivity -(x,y,z) [W/m K]  
Code API symbol: MPCCI_QID_THERMCOND3  
Default value: 0.0  
Dimension: Vector  
Physical meaning: Material property/general property  
Interpolation type: Field  
Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**ThermCondX**  
Thermal conductivity - x [W/m K]

- **Code API symbol:** MPCCI_QID_THERMCONDX
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume Element</td>
<td>Direct</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>Volume Node Dir</td>
<td></td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume Element Dir UDM</td>
<td></td>
<td></td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume Element Dir UDM</td>
<td></td>
<td></td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume Element Direct SCALAR</td>
<td></td>
<td></td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**ThermCondY**  
Thermal conductivity - y [W/m K]

- **Code API symbol:** MPCCI_QID_THERMCONDY
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Material property/general property
- **Interpolation type:** Field
- **Coupling Dimensions:** Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume Element</td>
<td>Direct</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>Volume Node Dir</td>
<td></td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume Element Dir UDM</td>
<td></td>
<td></td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume Element Dir UDM</td>
<td></td>
<td></td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume Element Direct SCALAR</td>
<td></td>
<td></td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
ThermCondZ
Thermal conductivity - z [W/m K]
Code API symbol: MPCCI_QID_THERMCONDZ
Default value: 0.0
Dimension: Scalar
Physical meaning: Material property/general property
Interpolation type: Field
Coupling Dimensions: Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Line, Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Volume</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

TimeStepNo
Time step number [-]
Code API symbol: MPCCI_QID_TIMESTEP_COUNT
Default value: 0
Dimension: Scalar
Physical meaning: General
Interpolation type: max/min/sum/prod
Coupling Dimensions: Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>
**TotalPressure**  
Total pressure [N/m²]

- Code API symbol: MPCCI_QID_TOTALPRESSURE
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Flux density
- Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td></td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>

**TotalTemp**  
Total Temperature [K]

- Code API symbol: MPCCI_QID_TOTALTEMP
- Default value: 300.0
- Dimension: Scalar
- Physical meaning: Boundary condition: value
- Interpolation type: Field
- Coupling Dimensions: Line, Face, Volume

The quantity is currently not supported by any standard code.

**udm00 – udm19**  
User defined memory 00 – 19 [-]

- Code API symbol: MPCCI_QID_UMEM_00
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: General
- Interpolation type: Field
- Coupling Dimensions: Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Element</td>
<td>ETAB</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDM</td>
<td>UDM</td>
</tr>
<tr>
<td>FLUX</td>
<td>Face, Volume</td>
<td>Node</td>
<td>SPATIAL</td>
<td>SPATIAL</td>
</tr>
</tbody>
</table>
**uds00 – uds19**  
User defined scalar 00 – 19 [-]

- **Code API symbol:** MPCCI_QID_USCL_00
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** General
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Element</td>
<td>ETAB</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>UDS</td>
<td>UDS</td>
</tr>
</tbody>
</table>

**Velocity**  
Velocity vector [m/s]

- **Code API symbol:** MPCCI_QID_VELOCITY
- **Default value:** 0.0
- **Dimension:** Vector
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Volume</td>
<td>Element</td>
<td></td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Hexa</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face, Volume</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**VelocityMagnitude**  
Velocity magnitude [m/s]

- **Code API symbol:** MPCCI_QID_VELOCITYMAG
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Field
- **Coupling Dimensions:** Line, Face, Volume

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td></td>
</tr>
<tr>
<td>Flowmaster</td>
<td>Face</td>
<td>Element</td>
<td></td>
<td>Direct</td>
</tr>
</tbody>
</table>
Voltage1 – Voltage4

Electric voltage - phase 1 – 4 [V]

- Code API symbol: MPCCI_QID_VOLTAGE1
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: General
- Interpolation type: max/min/sum/prod
- Coupling Dimensions: Global

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Global</td>
<td>global</td>
<td>APDL</td>
<td>APDL</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Global</td>
<td>global</td>
<td>Dir</td>
<td>Dir</td>
</tr>
<tr>
<td>FLUX</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Global</td>
<td>global</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

VolumeFlow

Volume flow vector [m³/s]

- Code API symbol: MPCCI_QID_VOLFLOWVECT
- Default value: 0.0
- Dimension: Vector
- Physical meaning: Mass source
- Interpolation type: Flux
- Coupling Dimensions: Volume

The quantity is currently not supported by any standard code.

VolumeFlowRate

Volume flow rate [m³/s]

- Code API symbol: MPCCI_QID_VOLFLOWRATE
- Default value: 0.0
- Dimension: Scalar
- Physical meaning: Mass source
- Interpolation type: Flux
- Coupling Dimensions: Face

The quantity is currently not supported by any standard code.
**Wall2Force** 2 sided wall boundary absolute force vector [N]

- Code API symbol: MPCCI_QID_WALL2FORCE
- Default value: 0.0
- Dimension: Symmetric tensor
- Physical meaning: Boundary condition: value
- Interpolation type: Flux
- Coupling Dimensions: Face

The quantity is currently not supported by any standard code.

**Wall2HeatFlux** 2 sided wall boundary normal heat flux density [W/m²]

- Code API symbol: MPCCI_QID_WALL2HEATFLUX
- Default value: 0.0
- Dimension: Biscalar
- Physical meaning: Boundary condition: face normal gradient
- Interpolation type: Flux density
- Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PosRad</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
</tbody>
</table>

**Wall2HTCoeff** 2 sided wall boundary heat transfer coefficient [W/m² K]

- Code API symbol: MPCCI_QID_WALL2HTCOEFF
- Default value: 0.0
- Dimension: Biscalar
- Physical meaning: Boundary condition: value
- Interpolation type: Field
- Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
</tbody>
</table>
Wall2Temp  2 sided wall boundary temperature [K]
  Code API symbol: MPCCI_QID_WALL2TEMPERATURE
  Default value: 300.0
  Dimension: Biscalar
  Physical meaning: Boundary condition: value
  Interpolation type: Field
  Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PosRad</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

WallForce  Boundary absolute force vector [N]
  Code API symbol: MPCCI_QID_WALLFORCE
  Default value: 0.0
  Dimension: Vector
  Physical meaning: Boundary condition: value
  Interpolation type: Flux
  Coupling Dimensions: Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**WallHeatFlux**

Boundary normal heat flux density \([\text{W/m}^2]\)

- **Code API symbol:** MPCCI_QID_WALLHEATFLUX
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: face normal gradient
- **Interpolation type:** Flux density
- **Coupling Dimensions:** Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PosRad</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>

**WallHTCoeff**

Boundary heat transfer coefficient \([\text{W/m}^2\text{K}]\)

- **Code API symbol:** MPCCI_QID_WALLHTCOEFF
- **Default value:** 0.0
- **Dimension:** Scalar
- **Physical meaning:** Boundary condition: value
- **Interpolation type:** Field
- **Coupling Dimensions:** Face

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PosRad</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
**WallTemp**  
Boundary temperature [K]

<table>
<thead>
<tr>
<th>Code</th>
<th>Coupling Dimensions</th>
<th>Location</th>
<th>Send option</th>
<th>Receive option</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSYS</td>
<td>Face</td>
<td>Node, Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>Abaqus</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>FieldVariable</td>
</tr>
<tr>
<td>CFX</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FINE/Turbo</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>FLUENT</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>ICEPAK</td>
<td>Face</td>
<td>Element</td>
<td>Dir</td>
<td>UDM</td>
</tr>
<tr>
<td>MSC.Marc</td>
<td>Line, Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>MSC.NASTRAN</td>
<td>Line, Face</td>
<td>Node</td>
<td>Direct</td>
<td></td>
</tr>
<tr>
<td>PERMAS</td>
<td>Face</td>
<td>Node</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>PosRad</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>RadTherm</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>Direct</td>
</tr>
<tr>
<td>STAR-CD</td>
<td>Face</td>
<td>Element</td>
<td>Direct</td>
<td>SCALAR</td>
</tr>
</tbody>
</table>
Literature


Glossary

The aim of this glossary is to give a short description of the most important technical terms which are used in this manual and the context of MpCCI. For further reference, links to the corresponding manual sections are given.

architecture
In the MpCCI context, architecture refers to the computer system, i.e. the combination of operating system and system hardware. Each characteristic combination supported by MpCCI is denoted by an architecture token, see V-2.3.1 MPCCI.ARCH - Architecture Tokens. A list of current architecture tokens is given in the Release Notes.

association
Association is the search for nodes/elements which lie close to a node/element of the partner mesh, also referred to as contact search or neighborhood search. The association is carried through during the initialization phase of MpCCI as a preparation for data interpolation. See V-3.2.1 Pre-Contact Search.

client
A simulation code with a code adapter acts as a client of the MpCCI server, i.e. the term client refers to a code or adapter, see V-3.4.1 Client-Server Structure of MpCCI.

code adapter
A code adapter is a plug-in into a simulation code which allows coupling of the code with other codes via MpCCI. Code adapters can be based on user-defined functions and are usually linked with the simulation code, preferably as a shared library, see V-3.4.1 Client-Server Structure of MpCCI. Code adapters are provided for all codes which are officially supported by MpCCI, further code adapters can be added using the code API (see IX-2 Code API).

control process
The control process is an optional process, which is started to write a tracefile (see tracefile below).

coupling algorithm
A coupling algorithm is the procedure of a coupled simulation. It determines when data is transferred, which code starts computing a new time step, etc. An overview of possible coupling algorithms is given in V-3.3 Coupling Algorithms.

coupling component
A coupling component is a part of a coupling region, typically a set of elements, see description of coupling region.

coupling region
Coupling regions define a part of a mesh which is coupled. In one coupling region, certain quantities are exchanged with the partner code. Each coupling region has a counterpart in the mesh of the partner code, which should have roughly the same geometry. A coupling region is composed of one or
several coupling components. Coupling regions are defined in the MpCCI GUI (see ▶ IV-2.5 Coupling Step – Definition of Coupling Regions and Quantities ◀ and ▶ V-4.4 Coupling Step ◀).

coupling type
A set of typical coupling types can be selected in the Coupling Step of the MpCCI GUI. Each type corresponds to a predefined selection of quantities for a coupling dimension, see ▶ V-4.4 Coupling Step ◀ for details.

Edit Step
The Edit Step is the third panel in the wizard-like MpCCI GUI. The control parameters for MpCCI can be set in this panel as described in ▶ V-4.5 Edit Step ◀.

element-element (EE)
Old name for the intersection algorithm.

exchange
Other term for transfer.

field
One of the two interpolation principles (see also flux). Field interpolation is applied for quantities which do not depend on the element area, e.g. density, electric resistivity or temperature. See ▶ V-3.2.5 Flux and Field Interpolation ◀.

flux
One of the two interpolation principles (see also field). Flux interpolation is applied for quantities for which the integral over an area/volume must be preserved, e.g. forces or mass flux. See ▶ V-3.2.5 Flux and Field Interpolation ◀.

grid
See mesh.

Go Step
The Go Step is the fourth and last panel in the wizard-like MpCCI GUI, where options for starting the simulation codes can be set and the coupled simulation is started (and stopped). See ▶ V-4.6 Go Step ◀.

initialization
The initialization is carried through at the beginning of a coupled simulation. Both codes send their mesh data to MpCCI and the meshes are associated. See ▶ V-3.3.1 Course of the Coupling Process ◀ for details.

interpolation
In most coupled simulations, data is transferred between non-matching grids. I.e. the data is given on one mesh and a corresponding set of data for a different mesh is expected by the other code. The process of finding such a distribution is called interpolation. Different methods for this procedure are described in ▶ V-3.2 Data Exchange ◀.
intersection

The *intersection* method is one of the interpolation schemes offered by *MpCCI*. It is based on overlapping surfaces or penetrating volumes of elements, see ▷ V-3.2.3 Intersection ◁.

mesh

The *mesh* (also called *grid*) consists of a set of nodes which are given by their spatial coordinates and a set of elements each which is connected to a typical number of nodes. The connection of nodes and elements is known as *mesh topology*. *MpCCI* can transfer data between mesh-based simulation codes, e.g. Finite Element (FE) or Finite Volume (FV) codes.

minimal distance

The *minimal distance* method is one of the interpolation schemes offered by *MpCCI*. It is based on an association of nodes of one mesh to elements of the other mesh. The value which is transferred depends on the location of the node (or its projection) inside the element, see ▷ V-3.2.3 Intersection ◁.

Models Step

The Models Step is the first panel in the wizard-like *MpCCI* GUI, where the models which shall be coupled are selected. In addition some code-specific options can be selected, see ▷ V-4.3 Models Step ◁.

multi-physics

The purpose of *MpCCI* is to couple codes for the solution of *multi-physics* problems. Such problems consist of partial problems each of which can be classified into a different physical domain. Typical domains are *fluid mechanics*, *solid mechanics*, *heat transfer* or *electromagnetism*. See ▷ V-3.1 Multi-Physics ◁ for details.

neighborhood search

Other term for *association*.

orphaned nodes

If mesh *association* fails, some nodes cannot be associated with nodes or elements of the other mesh. This means they cannot be considered during data transfer, which usually yields severe errors. See ▷ V-3.2.4 Orphaned Nodes and Elements ◁.

point-element (PE)

Other name for *minimal distance* interpolation.

quantity

Physical *quantities* must be exchanged between meshes for a coupled simulation. For each coupling case a typical set must be selected in the *Coupling Step* of *MpCCI*. A quantity has a typical SI-unit.

remote file browser

*MpCCI* uses a *remote file browser* for model file selection, which allows to connect to a remote computer via a remote shell. By selecting model file on a remote computer, a user determines where to run a simulation code. See ▷ V-4.7 Remote File Browser ◁ for a description of the *MpCCI* remote file browser.
**scanner**

In the *Models Step* of the MpCCI GUI the scanner must be started to scan the model file. The scanner searches the file for definition of possible coupling regions and further code-dependent information. See &gt;IX-2.5.2 Scanner.pm &lt; for a detailed description of what the scanner does.

**server**

The communication of a *simulation code* with MpCCI is based on a client-server model, i.e. during a computation a set of MpCCI servers is started, which are connected to the simulation codes (*clients*). The servers perform all tasks required for data transfer between two codes. See &gt;IV-1.3 Code Coupling with MpCCI &lt; and &gt;V-3.4.1 Client-Server Structure of MpCCI &lt; for a description.

**simulation code**

All software programs which can be coupled by MpCCI are commonly referred to as *simulation codes*. A description of all codes is given in the Codes Manual.

**staggered method**

MpCCI uses the *staggered* approach to solve multi-physics problems: Each code computes one time step independently and data is only exchanged after such a partial solution is obtained. This is also known as *weak coupling*. The solution procedure is discussed in &gt;IV-1.2 Solution of Coupled Problems &lt;.

**tracefile**

The *tracefile* can be best described as graphical log file. If a tracefile is written during a computation (by an additional control process), the mesh geometry and transferred data is stored. The contents of the *tracefile* can be viewed with the MpCCI Visualizer as described in &gt;V-6 MpCCI Visualizer &lt;.
Keyword Index

All page numbers are preceded by the roman part numbers:

I Overview
II Release Notes
III Installation Guide
IV Getting Started
V User Manual
VI Codes Manual
VII Tutorial
VIII Programmers Guide
IX Appendix

Important entries are bold, italic entries refer to the glossary.

Abaqus, VI-13
acoustics, V-22
ActivePerl, III-9
addcontrol block (SDK), VIII-148
additional block (SDK), VIII-168
Additional Scalars, VI-121
align, VI-11
ANSYS, VI-25
API Kit, VIII-11
architecture, V-13, V-13, V-40, IX-42, V-98, V-100
supported by MpCCI, II-9
association, IV-5, VIII-23, V-30, IX-42, VIII-51, V-74
axisymmetry, VII-80

backup, V-116
batch execution, V-42, V-58, V-117
block (SDK)
addcontrol, VIII-148
additional, VIII-168
code, VIII-149
contact, VIII-155
control, VIII-152
coupling, VIII-164
jobs, VIII-160
parameters, VIII-163
quantities, VIII-151
switches, VIII-157
bounding box check, V-57

CCI-functions
CCIabort, VIII-143
CCIanycode, VIII-126
CCIanyquantity, VIII-96
CCIcheckconv, VIII-126
CCIcloseremesh, VIII-129
CCIclosesetup, VIII-105
CCIcomminfo, VIII-136
CCIcontinue, VIII-126
CCIconverged, VIII-126
CCIcoordcart, VIII-102
CCIcoordcyl, VIII-103
CCIcoordsph, VIII-103
CCIdefcomm, VIII-95
CCIdefctf, VIII-102
CCIdefelems, VIII-87
CCIdefmesh, VIII-79
CCIdefnodes, VIII-84
CCIdefpart, VIII-81
CCIdefsearchtags, VIII-101
CCIdefsync, VIII-98
CCIdiverged, VIII-126
CCIelemhexagon, VIII-89
CCIelemhexahedron, VIII-90
CCIelemline, VIII-89
CCIlempentagon, VIII-89
CCIlemprism, VIII-90
CCIlempyramid, VIII-90
CCIlemquad, VIII-88
CCIlemquaeight, VIII-88
CCIlemsquaneight, VIII-88
CCIlemsphqua, VIII-92
CCIlemsphtrihedron, VIII-91
CCIlemtetrahedron, VIII-90
CCIlemtriangle, VIII-88
CCIlemtrsix, VIII-89
CCIfinalize, VIII-130
CCIgeneralinfo, VIII-132
CCIgetelems, VIII-122
CCIgetidstring, VIII-138
CCIgetisp, VIII-139
CCIinit, VIII-77
CCIinitwithidstring, VIII-78
CCIiprobe, VIII-115
CCIisend, VIII-114, VIII-118
CCIlocalmeshid, VIII-115, VIII-117
CCImatchgrids, VIII-133
CCImatchinginfo, VIII-132
CCImaxnquant, VIII-99, VIII-115, VIII-117, VIII-135
CCImaxstringlen, VIII-131
CCImodnodes, VIII-128
CCImycodeid, VIII-96
CCIincodes, VIII-96
CCIinonmatchgrids, VIII-133
CCIinquantities, VIII-115, VIII-117
CCIparaminfo, VIII-131
CCIputtelems, VIII-109
CCIputnodes, VIII-106
CCIquantdim, VIII-133
CCIquantinfo, VIII-132
CCIquantinfoelen, VIII-132
CCIquantipol, VIII-133
CCIquantityids, VIII-115, VIII-117
CCIquantloc, VIII-133
CCIquantlocelem, VIII-133
CCIquantlocglobal, VIII-133
CCIquantlocnode, VIII-133
CCIquanttype, VIII-133
CCIquanttypefield, VIII-133
CCIquanttypeflux, VIII-133
CCIquanttypemesh, VIII-133
CCIquanttypeuser, VIII-133
CIreachsync, VIII-119
CIrecv, VIII-116
CIremesh, VIII-127
CIrequestnull, VIII-118
CIsend, VIII-112
CIstatus, VIII-75
CIstatussize, VIII-75
CIstop, VIII-126
CIisyncinfo, VIII-134
CItotalremesh, VIII-129
CIwait, VIII-118
CIwtime, VIII-145
communicator, VIII-95
coupling steps, VIII-126
data types, VIII-75
Fortran data types, VIII-75
overview, VIII-146, VIII-146
tracefile, VIII-126
CFD, IV-3, IV-8, IX-9, IV-9, V-22
cgs units, VII-8
check
mesh, V-57
mesh quality, V-73, VIII-153
clean
temporary files, V-124
client, V-39, IX-42
home (SDK), VIII-73
cluster, V-42
code adapter, IV-6, VI-9, IX-42
code API, VIII-7
code block (SDK), VIII-149
code configuration directory, VIII-28
code integration, VIII-7, VIII-8
keyword index ix appendix

step-by-step, viii-14
command line interface, v-84
communicator (sdk), viii-164
complex start, v-38
computational fluid dynamics, see cfd
configuration directory, viii-28
contact, v-74, viii-156
contact block (sdk), viii-155
control block (sdk), viii-152
control parameters, v-71
control process, ix-42, see tracefile
control window, v-137
coordinate systems, viii-102
coordinate transformation, vii-11
coordinate transformation (sdk), viii-169
coupled system, iv-4
coupling, v-21
coupling algorithm, v-30, ix-42
coupling block (sdk), viii-164
coupling component, ix-42
coupling manager functions, viii-48
coupling region, ix-42
coupling server (sdk), viii-72
coupling step, iv-13, see gui
coupling type, v-22, ix-43, v-67
data flow
data visualizer, v-136
data structure, viii-64
data types (sdk), viii-75
debugging, v-14, v-102, viii-158
directory
dpci resource, v-17
code configuration, viii-28
domain
domain physical, v-21
download, iii-6
driver function, viii-58
duc heater, vii-90
duc heater, vii-90
edit step, ix-43, see gui
ee, v-27
elastic flap, vii-26
elastic foundation, viii-11
electromagnetic, vii-64
electromagnetism, v-22
electrothermal analysis, v-24
element check, v-57
element definition, viii-56
element types, viii-57
element-element (ee), ix-43
element-element relationship, see intersection
endianness, v-40
environment variable, v-12, iii-16, viii-35, v-97, v-102
example
code integration, viii-11
exchange, ix-43, viii-53, viii-63
before solution, iv-22
initial, v-15, viii-52
exhaust, vii-47
fe, iv-3, iv-8, i-9, iv-10
fem, v-21
field, ix-43
file system, v-81
files, v-16
temporary, v-18
finalization, v-30
fine/hexa, vi-41
fine/turbo, vi-48
finite element, see fe
firewall, v-41
flap
elastic, vii-26
flexlm, see license, v-109
flowmaster, vi-54
 Fluent, vi-62
fluid domain, iv-9
fluid dynamics, see cfd
fluid-structure interaction, see fsi
flux, v-27, ix-43
flux, v-27, ix-43
flux, v-27, ix-43
flx, vi-80
foundation example, viii-11
fsi, vii-8, v-23, vii-26, vii-80
full overlap (sdk), viii-156
GLOBUS, V-52, V-123
glossary, IX-42
Go Step, IX-43, see GUI
Graphical User Interface, see GUI
grid, IX-43
grid morpher, V-145
grid type (SDK), VIII-155
group switches, VIII-158
GUI, V-58, V-89
  Coupling Step, IV-13
  Edit Step, IV-16
    options, V-71
  Go Step, IV-18, V-76
    menus, V-59
  Models Step, IV-11
  title, V-59
configuration file, VIII-29
Go Step, VIII-33
Models Step, VIII-30
Start, IV-10

HDF (SDK), VIII-152
heat transfer, V-22
HOME, III-16
home
  MpCCI, V-11
home directory, V-13
hostlist file, V-41

include mechanism, VIII-170
info, VI-10, VIII-45
information, V-97
initial exchange, V-15, IV-19, VIII-52
initialization, V-30, IX-43, VIII-50
input file
  SDK, VIII-147
insideOnly, V-27
installation, III-5, V-106
  Java, III-39
  Perl, III-37
  Windows, III-13
multi-platform, III-12
prerequisites, III-6
interpolation, IV-6, IX-43
intersection, V-27, IX-44, V-74, V-76
intersection points (SDK), VIII-169
ipol (SDK), VIII-150
iteration, V-30

Java, III-9
  installation, III-39
job control, V-115
job scheduler, V-42
jobs block (SDK), VIII-160
killer, VIII-35
killing jobs, V-125
license, III-19, V-106, V-107
  manager, V-109
LoadLeveler, V-50, V-122
loop, VIII-66
LSF, V-44, V-119
macro, VIII-64
Magneto-Thermal, VII-64
manifold, VII-47
matching criterion (SDK), VIII-156
mesh, IX-44
mesh check, V-57
mesh definition, VIII-55, VIII-56, VIII-62
mesh quality check, V-57, V-73, VIII-153
Microsoft Windows, see Windows
minimal distance, V-26, IX-44, V-74, V-75
model file, VI-12
model preparation, IV-8
Models Step, IX-44, see GUI
morpher, VII-26, V-90, VI-130, V-145
mount, V-82
MpCCI
  Control Panel (FLUENT), VI-71
  GUI, see GUI
  Morpher, V-90, VI-130, V-145
  Observer, V-93
  Project Manager, V-146
  Visualizer, IV-24, V-95, V-136
point-element relationship, see minimal distance
port, V-40
post-processing, IV-25
prerequisites
installation, III-6
pressure
reference, V-23
project manager, V-94, V-146
projection distance, V-28
ptoi, V-128
pwd (SDK), VIII-161
quantities block (SDK), VIII-151
quantity, IV-4, VIII-32, IX-44
descriptor, VIII-65
reference, IX-4
visualizer, V-139
quantity (SDK), VIII-147
queuing system, V-42
radiation, V-22
RadTherm, VI-111
reference pressure, V-23
FLUENT, VI-63
region, see coupling region
releases, VI-10
remote computing, V-39
remote file browser, IX-44, V-80
remote shell, V-20, III-27, V-42, V-101, V-110
testing, III-28
requirements
code integration, VIII-8
resource directory, V-17
restart, V-38, V-73, VIII-154
results, IV-24
return status, VIII-75
MpCCI-RSH, III-11
Preparing the .rhosts file, III-33
rsh, see remote shell, V-82
runtime information (SDK), VIII-160
scanner, VI-12, VIII-35, VIII-43, IX-45
Start, IV-12
server, **V-39, IX-45, VIII-72, V-129**
SGE, **V-48, V-121**
SGEEE, **V-48, V-121**
simulation code, **VI-9, III-17, IX-45**
software
   MpCCI, **V-10**
      third party, **V-20**
solid mechanics, **V-21**
ssh, see remote shell, **V-82**
staggered method, **IV-5, IX-45**
STAR-CD, **VI-117**
   Additional Scalars, **VI-121**
   Plug-In, **VI-118**
   Restart, **VI-118, VI-123**
starter, **VIII-35, VIII-45**
starting MpCCI, **V-88**
step-by-step code integration, **VIII-14**
stopper, **VIII-35, VIII-45**
storage index, **VI-121**
subcmd, **VIII-46**
subcommands
   list, **V-86**
subcycling, **V-37**
surface coupling, **IV-4**
switches block, **VIII-157**
symmetry
   axial, **VII-80**
synchronization point (SDK), **VIII-69, VIII-98, VIII-164**
temporary files, **V-18**
terminal window, **V-96**
testing
   installation, **III-16**
   license server, **III-26**
   MpCCI installation, **V-111**
   remote shell, **III-28**
thermal coupling, **V-23, VII-64**
third party software, **V-20**
time step size
   exchange, **V-36**
timeout (SDK), **VIII-154**
timing (SDK), **VIII-160**
tolerance, **VIII-153**
Torque, **V-46, V-120**
tracefile, **IV-24, V-40, IX-45, V-72, V-136, VIII-152**
transient problems, **V-31**
type (contact) (SDK), **VIII-155**
unit, **VII-80**
units, **VII-8, VI-12**
units (SDK), **VIII-150**
update
   MpCCI, **V-114**
   user-defined function, **VII-80**
version
   mpcci, **V-101**
vibration, **VII-8**
viewer window, **V-139**
visualizer, see MpCCI Visualizer
volume coupling, **IV-4**
vortex street, **VII-8**
Windows
   rcp, **III-32**
   rsh, **III-32**
   rlogin, **III-32**
Windows, **III-16, III-32, III-32, III-35**
windows installer, **III-13**
xterm, **V-96**
Y-Junction, **VII-106**