Defining Phases and Interfaces

Cantera 1.5

David G. Goodwin

August 14, 2003

Division of Engineering and Applied Science
California Institute of Technology
Email: dgoodwin@caltech.edu

CONTENTS

CHAPTER

ONE

Introduction

Virtually every Cantera simulation involves one or more phases of matter. Depending on the calculation being performed, it may be necessary to evaluate thermodynamic properties, transport properties, and/or homogeneous reaction rates for the phase(s) present. In problems with multiple phases, the properties of the interfaces between phases, and the heterogeneous reaction rates at these interfaces, may also be required.

Before the properties can be evaluated, each phase must be *defined*, meaning that the models to use to compute its properties and reaction rates must be specified, along with any parameters the models require. For example, a solid phase might be defined as being incompressible, with a specified density and composition. A gaseous phase for a combustion simulation might be defined as an ideal gas consisting of a mixture of many species that react with one another via a specified set of reactions.

For phases containing multiple species and reactions, a large amount of data is required to define the phase, since the contribution of each species to the thermodynamic and transport properties must be specified, and rate information must be given for each reaction. While this *could* be done directly in an application program, a better approach is put the phase and interface definitions in a text file that can be read by the application, so that a given phase model can be re-used for other simulations.

This document describes how to write such files to define phases and interfaces for use in Cantera simulations. It also includes several complete examples that list the input file, the application program or script that uses the input file, and the resulting output, so that you can see exactly how the process of definining and using phase and interface models works in practice.

1.1 Prerequisites

To work with Cantera input files, you need Cantera version 1.5 or later, and must have installed the Cantera Python interface, since Cantera uses Python to parse the input files (even if you are using Cantera from some other language). If you are using a PC running Windows, binary installers are available that can install Cantera 1.5 and the Python interface in minutes. If you are using a unix, linux, or Mac OS X system, then the build process can automatically build and install the Cantera Python interface for you. See the installation instructions for more details.

1.2 A Simple Example

1.2.1 Pure Argon Gas

To get started, let's take a look at a definition for very simple phase: pure, gaseous argon:

```
ideal_gas(name = 'argon_gas',
          elements = 'Ar',
          species = 'argon',
          initial_state = state(
                    temperature = 1200.0,
                    pressure = OneAtm
                  )
         )

species(name = 'argon',
        atoms = 'Ar:1',
        thermo = const_cp( t0 = 298.15,
                           h0 = 0.0,
                           cp0 = 2.5*GasConstant,
                           s0 = (154.723, 'J/mol/K')
                         )
       )
```

The first entry defines a model for a gas that uses the ideal gas equation of state, is composed only of elemental argon, and contains one species named `argon`. This species is defined in the second entry, which states that each "molecule" of `argon` contains one atom of element Ar, and that the thermodynamic properties can be computed taking the heat capacity to be constant at $(5/2)\hat{R}$, with the specified values for $\hat{h}^0$ and $\hat{s}^0$ at 298.15 K.

To use this argon-gas model in an application, all that is required is to save the definition in a file, and then import it from the file into the application. Of course, if you use Cantera from Python or Matlab, the "application" could be an interactive session:

```
>>> from Cantera import *
>>> gas = importPhase('argon.cti')
>>> print gas

       temperature            1200  K
          pressure          101325  Pa
           density        0.405714  kg/m^3
  mean mol. weight          39.948  amu

                         1 kg             1 kmol
                      -----------      ------------
          enthalpy          469234        1.874e+07     J
   internal energy          219489        8.768e+06     J
           entropy         4597.62        1.837e+05     J/K
    Gibbs function    -5.04791e+06       -2.017e+08     J
 heat capacity c_p         520.301        2.078e+04     J/K
 heat capacity c_v         312.181        1.247e+04     J/K

                         X                 Y
                      -------------     ------------
             argon   1.000000e+00     1.000000e+00
>>>
```

Here the argon gas definition is contained in file 'argon.cti'[1]

The `gas` object returned by function `importPhase` implements the argon gas model, and can be used to compute any other desired thermodynamic property, or to carry out operations that require knowing thermodynamic properties. If, for example, we want to determine the temperature that results if argon is expanded adiabatically and reversibly (i.e. isentropically) in a nozzle to half the initial pressure, all we have to do is this:

```
>>> gas.setState_SP(gas.entropy_mass(), 0.5*gas.pressure())
>>> print gas.temperature()
909.429939906
```

The definition of argon gas given above allows us to use Cantera to compute virtually any thermodynamic property of pure argon gas. And of course we are not limited to using file 'argon.cti' in Cantera Python applications. We can use it in Matlab just as easily:

```
>> gas = importPhase('argon.cti');
>> t = temperature(gas)

t =

       1200

>> s = entropy_mole(gas)

s =

   1.8367e+05

>>
```

or in compiled Cantera application programs written in C++ or Fortran.

1.2.2 Adding Transport Properties

For some simulations, only thermodynamic properties are needed, but for others transport properties like the viscosity and thermal conductivity are also required. Since the definition of argon so far says nothing about the transport properties, we need to extend it a bit before we can compute the viscosity and thermal conductivity:

```
ideal_gas(name = 'argon_gas',
          elements = 'Ar',
          species = 'argon',
          transport = 'mix',
          initial_state = state(
                    temperature = 1200.0,
                    pressure = OneAtm
                  )
         )

species(name = 'argon',
        atoms = 'Ar:1',
```

[1] By convention, Cantera input file names usually have the extension '.cti', but in fact any extension (or none at all) is acceptable.

```
thermo = const_cp( t0 = 298.15, h0 = 0.0,
                   cp0 = 2.5*GasConstant,
                   s0 = (154.723, 'J/mol/K')),
transport = gas_transport(
                   geom = 'atom',
                   diam = 3.33,
                   well_depth = 136.50)
)
```

We've added a specification of a *transport model* to use to compute transport properties, and in the definition of species `argon`, we've added some information needed by the transport model — specifically, the geometry of the species (an atom), the Lennard-Jones collision diameter (3.33 A), and the Lennard-Jones well depth $\epsilon/k_B = 136.50$ K. With this extra information, Cantera can now compute the viscosity and thermal conductivity using the specified transport model.[2]

```
>>> gas = importPhase('argon.cti')
>>> print gas.viscosity(), gas.thermalConductivity()
6.27948725896e-05 0.0490083894103
```

As always, property values computed by Cantera are in SI units, so this viscosity has units of Pa-s, and the thermal conductivity has units of W/m^2/K.

At this point, we have a definition of a model for argon gas that can be used to determine its thermodynamic and transport properties, as long as the conditions are such that the ideal gas equation of state can be used. This model would be useful, for example, to compute the properties needed for a multidimensional, compressible simulation of a high-speed argon flow. Of course, argon is simply enough that such a model is hardly needed — most of the properties are either constants or simple functions of temperature (and in some cases pressure).

But if we wanted to do the same high-speed compressible simulation for air, computing the properties would be much less straightforward than for pure argon, and it would make sense to use Cantera for this purpose. As for argon, all that is required is an appropriate definition in a text file, but now the definition will be substantially more complex than that above. Air contains multiple species; at high temperature, not only N_2, O_2, and Ar are present, but also NO, NO_2, N, O, and possibly ionized species. The diatomic and triatomic species have temperature-dependent heat capacities, and the we may also need to allow for the possibility of reaction among these species.

As a result, an input file containing a defintion of high-temperature air will be longer than the simple 'argon.cti' file. But the basic structure of the file is the same, only with more species and with definitions of reactions.

1.3 Organization of this Document

In the following chapters, we'll cover how to define phases and interfaces with any number of species and reactions. We begin in Chapter 2 with a summary of some basic rules for writing input files, a discussion of how they are processed, and of how errors are handled. In Chapter 3, we will go over how to define phases and interfaces, including how to import species and reactions from external files. Then in Chapter 4, we'll look in depth at how to specify the component parts of phase and interface models — the elements, species, and reactions. Finally, in Chapter 5, we'll put it all together, and present some complete, realistic example problems, showing the input file containing the definitions of all phases and interfaces, the application code to use the input file to solve a problem, and the resulting output.

[2] The `'mix'` keyword selects a model that computes pure species properties in terms of certain collision integrals from kinetic theory, and then uses mixture rules to compute an appropriate weighted average of these values for ideal gas mixtures.

CHAPTER

TWO

Working with Input Files

Before we can describe how to define phases, interfaces, and their components (elements, species, and reactions), we need to go over a few points about the mechanics of writing and processing input files.

2.1 Input File Syntax

An input file consists of *entries* and *directives*, both of which have a syntax much like functions. An entry defines an object — for example, a reaction, or a species, or a phase. A directive sets options that affect how the entry parameters are interpreted, such as the default unit system, or how certain errors should be handled.

Entries have *fields* that can be assigned values. A **species** entry is shown below that has fields *name* and *atoms* (plus several others):

```
species( name = 'C60', atoms = 'C:60')
```

Most entries have some fields that are required; these must be assigned values, or else processing of the file will abort and an error message will be printed. Other fields may be optional, and take default values if not assigned.

An entry may be either a *top-level entry* or an *embedded entry*. Top-level entries specify a phase, an interface, an element, a species, or a reaction, and begin in the first (leftmost) column. Embedded entries specify a model, or a group of parameters for a top-level entry, and are usually embedded in a field of another entry.

The fields of an entry are specified in the form <*field_name*> = <*value*>, and may be listed on one line, or extend across several. For example, two entries for graphite are shown below. The first is compact:

```
stoichiometric_solid(name = 'graphite', species = 'C(gr)', elements = 'C',
             density = (2.2, 'g/cm3')))
```

and the second is formatted to be easier to read:

```
stoichiometric_solid(
       name         = 'graphite',
       elements     = 'C',
       species      = 'C(gr)',
       density      = (2.2, 'g/cm3')
     )
```

Both are completely equivalent.

The species `C(gr)` that appears in the definition of the graphite phase is also defined by a top-level entry. If the heat capacity of graphite is approximated as constant, then the following definition could be used:

```
species(
        name = 'C(gr)',
        atoms = 'C:1',
        thermo = const_cp(
                    t0 = 298.15,
                    h0 = 0.0,
                    s0 = (5.6, 'J/mol/K'),       # NIST
                    cp0 = (8.43, 'J/mol/K'))     # Taylor and Groot (1980)
       )
```

Note that the *thermo* field is assigned an embedded entry of type **const_cp**. Entries are stored as they are encountered when the file is read, and only processed once the end of the file has been reached. Therefore, the order in which they appear is unimportant.

2.1.1 Comments

The character `'#'` is the comment character. Everything to the right of this character on a line is ignored.

```
#
# set the default units
#
units( length    = 'cm',   # use centimeters for length
       quantity  = 'mol')  # use moles for quantity
```

2.1.2 Strings

Strings may be enclosed in single quotes or double quotes, but they must match. To create a string containing single quotes, enclose it in double quotes, and vice versa. If you want to create a string to extend over multiple lines, enclose it in triple double quotes.

```
string1 = 'A string.'
string2 = "Also a 'string'"
string3 = """This is
a
string too."""
```

The multi-line form is useful when specifying a phase containing a large number of species:

```
       species = """ H2  H  O  O2  OH  H2O  HO2  H2O2  C  CH
                     CH2  CH2(S)  CH3  CH4  CO  CO2  HCO  CH2O  CH2OH  CH3O
                     CH3OH  C2H  C2H2  C2H3  C2H4  C2H5  C2H6  HCCO  CH2CO  HCCOH
                     N  NH  NH2  NH3  NNH  NO  NO2  N2O  HNO  CN
                     HCN  H2CN  HCNN  HCNO  HOCN  HNCO  NCO  N2  AR  C3H7
                     C3H8  CH2CHO  CH3CHO """
```

2.1.3 Sequences

A sequence of multiple items is specified by separating the items by commas and enclosing them in square brackets or parentheses. The individual items can have any type – strings, integers, floating-point numbers (or even entries or other lists). Square brackets are often preferred, since parentheses are also used for other purposes in the input file, but either can be used.

```
s0 = (3.5, 'J/mol/K')      # these are
s0 = [3.5, 'J/mol/K']      # equivalent
```

2.1.4 Variables

Another way to specify the species `C(gr)` is shown here:

```
graphite_thermo = const_cp(t0 = 298.15,
                           h0 = 0.0,
                           s0 = (5.6, 'J/mol/K'),      # NIST
                           cp0 = (8.43, 'J/mol/K')     # Taylor and Groot (1980)

                          )
species(
        name = 'C(gr)',
        atoms = 'C:1',
        thermo = graphite_thermo
       )
```

In this form, the **const_cp** entry is stored in a variable, instead of being directly embedded within the **species** entry. The *thermo* field is assigned this variable.

Variables can also be used for any other parameter type. For example, if you are defining several phases in the file, and you want to set them all to the same initial pressure, you could define a pressure variable

```
P_initial = (2.0, 'atm')
```

and then set the *pressure* field in each embedded **state** entry to this variable.

2.1.5 Omitting Field Names

Field names may be omitted if the values are entered in the order specified in the entry declaration. (Entry declarations are the text printed on a colored background in the following chapters.) It is also possible to omit only some of the field names, as long as these fields are listed first, in order , before any named fields.

For example, the **element** entry is declared in Section 4.1 as

element*(symbol, atomic_mass)*
An atomic element or isotope.

symbol
The symbol for the element or isotope.

atomic_mass
The atomic mass in amu.

The first four entries below are equivalent, while the last two are incorrect and would generate an error when processed.

```
element(symbol = "Ar", atomic_mass = 39.948)     # OK
element(atomic_mass = 39.948, symbol = "Ar")     # OK
element("Ar", atomic_mass = 39.948)              # OK
element("Ar", 39.948)                            # OK

element(39.948, "Ar")                            # error
element(symbol = "Ar", 39.948)                   # error
```

2.1.6 Dimensional Values

Many fields have numerical values that represent dimensional quantities — a pressure, or a density, for example. If these are entered without specifying the units, the default units (set by the **units** directive described in Section 2.2) will be used. However, it is also possible to specify the units for each individual dimensional quantity (unless stated otherwise). All that is required is to group the value in parentheses or square brackets with a string specifying the units.

```
pressure = 1.0e5          # default is Pascals
pressure = (1.0, 'bar')   # this is equivalent

density = (4.0, 'g/cm3')
density = 4000.0          # kg/m3
```

Compound unit strings may be used, as long as a few rules are followed:

1. Units in the denominator follow `'/'`.
2. Units in the numerator follow `'-'`, except for the first one.
3. Numerical exponents follow the unit string without a `'^'` character, and must be in the range 2–6. Negative values are not allowed.

```
A = (1.0e20, 'cm6/mol2/s')     # OK
h = (6.626e-34, 'J-s')         # OK
density = (3.0, 'g/cm3')       # OK

A = (1.0e20, 'cm^6/mol/s')     # error (^)
A = (1.0e20, 'cm6/mol2-s')     # error ('s' should be in denominator)
density = (3.0, 'g-cm-3')      # error (negative exponent)
```

Table 2.1: Allowed values for the fields of the **units** directive.

field	allowed values
length	'cm','m', 'mm'
quantity	'mol', 'kmol', 'molec'
time	's', 'min', 'hr', 'ms'
energy	'J', 'kJ', 'cal', 'kcal'
act_energy	'kJ/mol', 'J/mol', 'J/kmol' 'kcal/mol', 'cal/mol', 'eV', 'K'

2.2 Setting the Default Units

The default unit system may be set with the **units** directive. Note that unit conversions are not done until the entire file has been read. Only one **units** directive should be present in a file, and the defaults it specifies apply to the entire file. If the file does not contain a **units** directive, the default units are meters, kilograms, kilomoles, and seconds.

Shown below are two equivalent ways of specifying the site density for an interface. In the first version, the site density is specified without a units string, and so its units are constructed from the default units for quantity and length, which are set with a **units** directive.

```
units(length = 'cm', quantity = 'molec')
interface(name = 'Si-100',
          site_density = 1.0e15,    # molecules/cm2 (default units)
          ...)
```

The second version uses a different default unit system, but overrides the default units by specifying an explicit units string for the site density.

```
units(length = 'cm', quantity = 'mol')
interface(name = 'Si-100',
          site_density = (1.0e15, 'molec/cm2')    # override default units
          ...)
```

The second version is equivalent to the first, but would be very different if the units of the site density were not specified!

The *length* and *time* units are used to construct the units for reaction pre-exponential factors. The *energy* units are used for molar thermodynamic properties, in combination with the units for quantity.

Since activation energies are often specified in units other than those used for thermodynamic properties, a separate field is devoted to the default units for activation energies.

```
units(length = 'cm', quantity = 'mol', act_energy = 'kcal/mol')
kf = Arrhenius(A = 1.0e14, b = 0.0, E = 54.0)    # E is 54 kcal/mol
```

The declaration of the **units** directive is shown in Fig. 2.1, and the allowed values for the fields of the **units** directive are listed In Table 2.1.

units*(length, mass, quantity, time, energy, act_energy)*
The default units.

length
The default units for length. Default: 'm'

mass
The default units for mass. Default: 'kg'

quantity
The default units to specify number of molecules. Default: 'kmol'

time
The default units for time. Default: 's'

energy
The default units for energies. Default: 'J'

act_energy
The default units for activation energies. Default: 'K'

Figure 2.1: The **units** directive. The allowed field values are listed in Table 2.1.

2.3 Processing the Input File

2.3.1 A Two-Step Process

From the point of view of the user, it appears that a Cantera application that imports a phase definition reads the input file, and uses the information there to construct the object representing the phase or interface in the application. While this is the net effect, it is actually a two-step process. When a function like `importPhase` is called to import a phase definition from a file, a *preprocessor* runs automatically to read the input file and create a *data file* that contains the same information but in an XML-based format called CTML. After the preprocessor finishes, Cantera imports the phase definition from the CTML data file.

The CTML file is saved in the same directory as the input file, and has the same name but with the extension changed to '.xml'. If the input file has the name 'propane.cti', for example, then the CTML file will be placed in the same directory with name 'propane.xml'. If you like, once the CTML file has been created, you can specify it rather than the '.cti' input file in calls to `importPhase` (or similar functions). This is slightly faster, since the preprocessing step can be skipped. It also allows Cantera simulations to be run on systems that do not have Python, which Cantera uses in the preprocessing step but does not require to read CTML files.

2.3.2 Two File Formats

Why two file formats? There are several reasons. XML is a widely-used standard for data files, and it is designed to be relatively easy to parse. This makes it possible for other applications to use Cantera CTML data files, without requiring the substantial chemical knowledge that would be required to use .cti files. For example, "web services" (small applications that run remotely over a network) are often designed to accept XML input data over the network, perform a calculation, and send the output in XML back across the network. Supporting an XML-based data file format facilitates using Cantera in web services or other network computing applications.

The difference between the high-level description in a .cti input file and the lower-level description in the CTML file may be illustrated by how reactions are handled. In the input file, the reaction stoichiometry and its reversibility or irreversibility are determined from the reaction equation. For example,

```
O + HCCO <=> H + 2 CO
```

specifies a reversible reaction between an oxygen atom and the ketenyl radical HCCO to produce one hydrogen atom and two carbon monoxide molecules. If `<=>` were replaced with `=>`, then it would specify that the reaction should be treated as irreversible.

Of course, this convention is not spelled out in the input file — the parser simply has to know it, and has to also know that a "reactant" appears on the left side of the equation, a "product" on the right, that the optional number in front of a species name is its stoichiometric coefficient (but if missing the value is one), etc. The preprocessor does know all this, but we cannot expect the same level of knowledge of chemical conventions by a generic XML parser.

Therefore, in the CTML file, reactions are explicitly specified to be reversible or irreversible, and the reactants and products are explicitly listed with their stoichiometric coefficients. The XML file is, in a sense, a "dumbed-down" version of the input file, spelling out explicitly things that are only implied in the input file syntax, so that "dumb" (i.e., easy to write) parsers can be used to read the data with minimal risk of misinterpretation.

The reaction definition

```
reaction( "O + HCCO <=> H + 2 CO",   [1.00000E+14, 0, 0])
```

in the input file is translated by the preprocessor to the following CTML text:

```
<reaction id="0028" reversible="yes">
  <equation>O + HCCO [=] H + 2 CO</equation>
  <rateCoeff>
    <Arrhenius>
      <A units="cm3/mol/s">  1.000000E+14</A>
      <b>0</b>
      <E units="cal/mol">0.000000</E>
    </Arrhenius>
  </rateCoeff>
  <reactants>HCCO:1 O:1</reactants>
  <products>H:1 CO:2</products>
</reaction>
```

The CTML version is much more verbose, and would be much more tedious to write by hand, but is much easier to parse, particularly since it is not necessary to write a custom parser — virtually any standard XML parser, of which there are many, can be used to read the CTML data. [1]

So in general files that are easy for knowledgable users (you) to *write* are more difficult for machines to *parse*, because they make use of high-level application-specific knowledge and conventions to simplify the notation. Conversely, files that are designed to be easily parsed are tedious to write because so much has to be spelled out explicitly. A natural solution is to use two formats, one designed for writing by humans, the other for reading by machines, and provide a preprocessor to convert the human-friendly format to the machine-friendly one.

2.3.3 Preprocessor Internals: the `ctml_writer` Module

If you are interested in seeing the internals of how the preprocessing works, take a look at file 'ctml_writer.py' in the Cantera Python package. Or simply start Python, and type:

[1] The CTML `reactants` and `products` elements require some extra parsing that is not standard XML. This may change in a future version of CTML.

```
from Cantera import ctml_writer
help(ctml_writer)
```

The ‘ctml_writer.py’ module can also be run as a script to convert input .cti files to CTML. For example, if you have input file ‘phasedefs.cti’, then simply type at the command line

```
python ctml_writer.py phasedefs.cti
```

to create CTML file ‘phasedefs.xml’.

Of course, most of the time creation of the CTML file will happen behind the scenes, and you will not need to be concerned with CTML files at all.

2.4 Error Handling

During processing of an input file, errors may be encountered. These could be syntax errors, or could be ones that are flagged as errors by Cantera due to some apparent inconsistency in the data — an unphysical value, a species that contains an undeclared element, a reaction that contains an undeclared species, missing species or element definitions, multiple definitions of elements, species, or reactions, and so on.

2.4.1 Syntax Errors

Syntax errors are caught by the Python preprocessor, not by Cantera, and must be corrected before proceeding further. Python prints a “traceback” that allows you to find the line that contains the error. For example, consider the following input file, which is intended to create a gas with the species and reactions of GRI-Mech 3.0, but has a misspelled the field name (*reactions*):

```
ideal_gas(name = 'gas',
          elements = 'H O',
          species = 'gri30: all',
          reactionss = 'gri30: all'
          )
```

When this definition is imported into an application, an error message like the following would be printed to the screen, and execution of the program or script would terminate.

```
 Traceback (most recent call last):
  File "/tmp/.cttmp.py", line 8, in ?
    execfile(file)
  File "./gas.cti", line 7, in ?
    reactionss = 'gri30: all' )
TypeError: __init__() got an unexpected keyword argument 'reactionss'

Traceback (most recent call last):
  File "tgas.py", line 2, in ?
    g = importPhase('gas.cti','gas')
  File "/sw/lib/python2.2/site-packages/Cantera/importFromFile.py", line 22, in importPhase
    return importPhases(file, [name])[0]
  File "/sw/lib/python2.2/site-packages/Cantera/importFromFile.py", line 30, in importPhases
    root = XML.XML_Node(name = 'doc', src = file, preprocess = 1)
  File "/sw/lib/python2.2/site-packages/Cantera/XML.py", line 28, in __init__
```

```
    _cantera.xml_build(self._xml_id, src, preprocess)
cantera.error:
Procedure: cti2ctml
Error:   could not convert input file to CTML
 command line was: python /tmp/.cttmp.py &> cti2ctml.log
Check error messages above for syntax errors.
```

The error message shows the chain of functions that were called before the error was encountered. For the most part, these are internal Cantera functions not of direct concern here. But near the top of this message is

```
      File "./gas.cti", line 7, in ?
        reactionss = 'gri30: all' )
    TypeError: __init__() got an unexpected keyword argument 'reactionss'
```

This message says that at line 7 in file 'gas.cti', the keyword (i.e. field name) `'reactionss'` was not recognized. Seeing this message, it is clear that the problem is that *reactions* is misspelled.

The traceback message is also written to file 'cti2ctml.log' in the local directory.

2.4.2 Cantera Errors

Now let's consider the other class of errors — ones that Cantera, not Python, detects. Continuing the example above, suppose that the misspelling is corrected, and the input file processed again. Again an error message results, but this time it is from Cantera.

```
cantera.error:
Procedure: installSpecies
Error:   species C contains undeclared element C
```

The problem is that the phase definition specifies that all species are to be imported from dataset `gri30`, but only the elements H and O are declared. The `gri30` datset contains species composed of the elements H, O, C, N, and Ar. If the definition is modified to declare these additional elements,

```
ideal_gas(name = 'gas',
          elements = 'H O C N Ar',
          species = 'gri30: all',
          reactions = 'gri30: all'
          )
```

it may be imported successfully.

Errors of this type do not have to be fatal, as long as you tell Cantera how you want to handle them. You can, for example, instruct Cantera to quitely skip importing any species that contain undeclared elements, instead of flagging them as errors. You can also specify that reactions containing undeclared species (also usually an error) should be skipped. This allows you to very easily extract a portion of a large reaction mechanism, as will be described in Section 3.1.8.

CHAPTER

THREE

Phases and Their Interfaces

Now that we have covered how to write syntactically-correct input files, we can turn our attention to the *content* of the file. We'll start by describing the entries for phases of various types, and the look at how to define interfaces between phases.

3.1 Phases

For each phase that appears in a problem, a corresponding entry should be present in the input file(s). For example, suppose we want to conduct a simulation with detailed chemistry of an idealized solid-oxide fuel cell shown in Fig. 3.1. The problem involves three solid phases (A nickel anode, a platinum cathode, and an oxygen-conducting yttrium-stabilized zirconia electrolyte), and two different gas phases (a fuel mixture on the anode side, and air on the cathode side). The problem also involves a number of interfaces at which heterogeneous chemistry may occur – two gas-metal interfaces, two gas-electrolyte interfaces, and two metal-electrolyte interfaces.

How to carry out this fuel cell simulation is beyond the scope of this document; we introduce it here only to give an example of the types of phases and interfaces that might need to be defined in order to carry out a simulation. (Of course, many simulations with Cantera only require defining a single phase.)

There are several different types of entries, corresponding to different types of phases. They share many common features, however, and so we will begin by discussing those aspects common to all entries for phases. The general declaration of any of the phase entries is shown in Fig. 3.2.

Here **<phase_type>** stands for one of the implemented phase types. Currently, these are **ideal_gas**, **stoichiometric_solid**, **stoichiometric_liquid**, and **ideal_solution**. Non-ideal models will be added in future releases.

3.1.1 The Phase Name

The name field is a string that identifies the phase. It must not contain any whitespace characters or reserved XML characters, and must be unique within the file among all phase definitions of any type.

Phases are referenced by name when importing them into an application program, or when defining an interface between phases.

3.1.2 Declaring the Elements

The elements that may be present in the phase are declared in the *elements* field. This must be a string of element symbols separated by spaces and/or commas. Each symbol must either match one listed in the database file 'elements.xml', or else match the symbol of an **element** entry defined elsewhere in the input file (see Section 4.1).

The 'elements.xml' database contains most elements of the periodic table, with their natural-abundance atomic masses (Appendix B). It also contains a few isotopes (D, Tr), and an "element" for an electron (E). This pseudo-element can

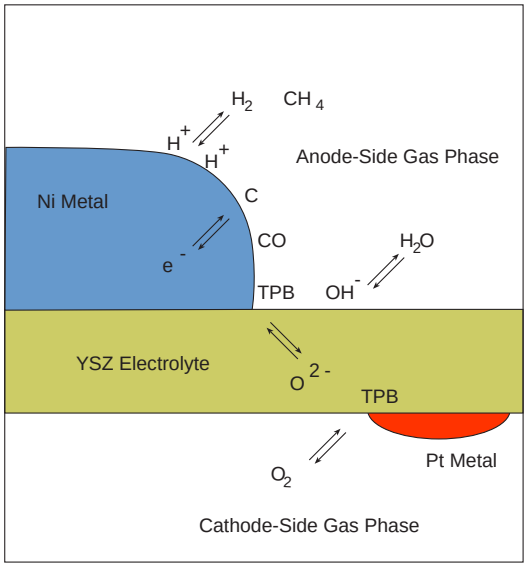


Figure 3.1: Phases entering into a hypothetical microkinetic simulation of an idealized solid-oxide fuel cell.

<phase_type>*(name, elements, species, reactions, kinetics, transport, initial_state, options)*

name
A string to identify the phase. Must be unique amomg the phase names within the file.

elements
The elements. A string of element symbols.

species
The species. A string or sequence of strings in the format described in Section 3.1.3.

reactions
The homogeneous reactions. If omitted, no reactions will be included. A string or sequence of strings in the format described in Section 3.1.4. This field is not allowed for **stoichiometric_solid** and **stoichiometric_liquid** entries.

kinetics
The kinetics model. Optional; if omitted, the default model for the phase type will be used.

transport
The transport property model. Optional. If omitted, transport property calculation will be disabled.

initial_state
Initial thermodynamic state, specified with an embedded **state** entry.

special
Special processing options. Optional.

Figure 3.2: Generic declaration for entries defining phases.

be used to specify the composition of charged species. Note that two-character symbols should have an uppercase first letter, and a lowercase second letter (e.g. Cu, not CU).

It should be noted that the order of the element symbols in the string determines the order in which they are stored internally by Cantera. For example, if a phase definition specifies the elements as

```
ideal_gas(name = "gasmix",
    elements = "H C O N Ar",
    ...
   )
```

then when this definition is imported by an application, element-specific properties will be ordered in the same way:

```
>>> gas = importPhase('phases.in','gasmix')
>>> for n in range(gas.nElements()):
...     print n, gas.elementSymbol(n)
0 H
1 C
2 O
3 N
4 Ar
```

For some calculations, such as multi-phase chemical equilibrium, it is important to synchronize the elements among multiple phases, so that each phase contains the same elements with the same ordering. In such cases, simply use the same string in the *elements* field for all phases.

3.1.3 Declaring the Species

The species in the phase are declared in the *species* field. They are not *defined* there, only declared. Species definitions may be imported from other files, or species may be defined locally using **species** entries elsewhere in the file.

If a single string of species symbols is given, then it is assumed that these are locally defined. For each one, a corresponding **species** entry must be present somewhere in the file, either preceding or following the **phase** entry. Note that the string may extend over multiple lines by delimiting it with triple quotes.

```
# commas are optional
species = 'AR SI Si2 SiH SiH2 SiH3 SiH4'

species = 'H, O, OH, H2O, HO2, H2O2, H2, O2'

# include all species defined in this file
species = 'all'

# a multi-line species declaration
species = """ H2  H  O  O2  OH  H2O  HO2  H2O2  C  CH
              CH2  CH2(S)  CH3  CH4  CO  CO2  HCO  CH2O  CH2OH  CH3O
              CH3OH  C2H  C2H2  C2H3  C2H4  C2H5  C2H6  HCCO  CH2CO  HCCOH
              N  NH  NH2  NH3  NNH  NO  NO2  N2O  HNO  CN
              HCN  H2CN  HCNN  HCNO  HOCN  HNCO  NCO  N2  AR  C3H7
              C3H8  CH2CHO  CH3CHO """
```

If the species are imported from another file, instead of being defined locally, then the string should begin with the file

name (without extension), followed by a colon:

```
# import selected species from silicon.xml
species = "silicon: SI SI2 SIH SIH2 SIH3 SIH4 SI2H6"

# import all species from silicon.xml
species = "silicon: all"
```

In this case, the species definitions will be taken from file 'silicon.xml', which must exist either in the local directory or somewhere on the Cantera search path.

It is also possible to import species from several sources, or mix local definitions with imported ones, by specifying a sequence of strings:

```
species = ["CL2 CL F F2 HF HCL",    # defined in this file
           "air: O2 N2 NO",         # imported from 'air.xml'
           "ions: CL- F-"]          # imported from 'ions.xml'
```

Note that the strings must be separated by commas, and enclosed in square brackets or parentheses.

3.1.4 Declaring the Reactions

The reactions among the species are declared in the *reactions* field. Just as with species, reactions may be defined locally in the file, or may be imported from one or more other files. All reactions must only involve species that have been declared for the phase.

Unlike species, reactions do not have a name, but do have an optional *id* field. If the *id* field is not assigned a value, then when the reaction entry is read it will be assigned a four-digit string encoding the reaction number, beginning with `'0001'` for the first reaction in the file, and incrementing by one for each new reaction.

If all reactions defined locally in the input file are to be included in the phase definition, then assign the *reactions* field the string `'all'`.

```
reactions = 'all'
```

If, on the other hand, only some of the reactions defined in the file are to be included, then a range can be specified using the reaction *id* fields.

```
reactions 'nox-12 to nox-24'
```

In determining which reactions to include, a *lexical* comparison of *id* strings is performed. This means, for example, that `'nox-8'` is greater than `'nox-24'`. (If it is rewritten `'nox-08'`, however, then it would be lexically less than `'nox-24'`.)

Just as described above for species, reactions can be imported from another file, and reactions may be imported from several sources.

Examples:

```
# import all reactions defined in this file
reactions = "all"

# import all reactions defined in rxns.xml
reactions = "rxns: all"

# import reactions 1-14 in rxns.xml
reactions = "rxns: 0001 to 0014"

# import reactions from several sources
reactions = ["all",                    # all local reactions
             "gas: all",               # all reactions in gas.xml
             "nox: n005 to n008"       # reactions 5 to 8 in nox.xml
            ]
```

3.1.5 The Kinetics Model

A *kinetics model* is a set of equations to use to compute reaction rates. In most cases, each type of phase has an associated kinetics model that is used by default, and so the *kinetics* field does not need to be assigned a value. For example, the **ideal_gas** entry has an associated kinetics model called `GasKinetics` that implements mass-action kinetics, requires integer stoichiometric coefficients, computes reverse rates from thermochemistry for reversible reactions, and provides pressure-independent, three-body, and falloff reactions. Other models could be implemented, and this field would then be used to select the desired model. For now, the *kinetics* field can be safely ignored.

3.1.6 The Transport Model

A *transport model* is a set of equations used to compute transport properties. For one phase type (**ideal_gas**), multiple transport models are available; the one desired can be selected by assiging a string to this field. See Section 3.1.9 for more details.

3.1.7 The Initial State

The phase may be assigned an initial state to which it will be set when the definition is imported into an application and an object created. This is done by assigning field *initial_state* an embedded entry of type **state**, described in Section 3.3.

Most of the attributes defined here are "immutable," meaning that once the definition has been imported into an application, they cannot be changed by the application. For example, it is not possible to change the elements or the species. The temperature, pressure, and composition, however, are "mutable" – they can be changed. This is why the field defining the state is called the *initial_state*; the object in the application will be initially set to this state, but it may be changed at any time.

3.1.8 Special Processing Options

The *options* field is used to indicate how certain conditions should be handled when importing the phase definition. The *options* field may be assigned a string or a sequence of strings from the table below.

Option String	Meaning
`no_validation`	Turn off all validation. Use when the definition has been previously validated to speed up importing the definition into an application. Use with caution!
`skip_undeclared_elements`	When importing species, skip any containing undeclared elements, rather than flagging them as an error.
`skip_undeclared_species`	When importing reactions, skip any containing undeclared species, rather than flagging them as an error.

Using the *options* field, it is possible to extract a sub-mechanism from a large reaction mechanism, as follows:

```
ideal_gas(name = 'hydrogen_mech',
          species = 'gri30: all',
          reactions = 'gri30:all',
          options = ('skip_undeclared_elements',
                     'skip_undeclared_species')
        )
```

If we import this into Matlab, for example, we get a gas mixture containing the 8 species (out of 53 total) that contain only H and O:

```
>> gas = importPhase('gas.in','hydrogen_mech')

       temperature                 300  K
       pressure                1237.28  Pa
       density                   0.001  kg/m^3
       mean mol. weight        2.01588  amu

                             X                 Y
                    -------------     ------------
                H2   1.000000e+00     1.000000e+00
                 H   0.000000e+00     0.000000e+00
                 O   0.000000e+00     0.000000e+00
                O2   0.000000e+00     0.000000e+00
                OH   0.000000e+00     0.000000e+00
               H2O   0.000000e+00     0.000000e+00
               HO2   0.000000e+00     0.000000e+00
              H2O2   0.000000e+00     0.000000e+00

>> eqs = reactionEqn(gas)

eqs =

    '2 O + M <=> O2 + M'
    'O + H + M <=> OH + M'
    'O + H2 <=> H + OH'
    'O + HO2 <=> OH + O2'
    'O + H2O2 <=> OH + HO2'
    'H + O2 + M <=> HO2 + M'
    'H + 2 O2 <=> HO2 + O2'
    'H + O2 + H2O <=> HO2 + H2O'
    'H + O2 <=> O + OH'
    '2 H + M <=> H2 + M'
    '2 H + H2 <=> 2 H2'
    '2 H + H2O <=> H2 + H2O'
```

ideal_gas*(name, elements, species, reactions, kinetics, transport, initial_state, options)*
A chemically-reacting ideal gas mixture of multiple species.

name
A string to identify the phase. Must be unique amomg the phase names within the file.

elements
The elements. A string of element symbols.

species
The species. A string or sequence of strings in the format described in Section 3.1.3.

reactions
The homogeneous reactions. If omitted, no reactions will be included. A string or sequence of strings in the format described in Section 3.1.4 below.

kinetics
The kinetics model. Usually this field is omitted, in which case kinetics model `GasKinetics`, appropriate for reactions in ideal gas mixtures, is used.

transport
The transport property model. One of the strings `'none'`, `'multi'`, or `'mix'`. Default = `'none'`

initial_state
Initial thermodynamic state, specified with an embedded **state** entry.

options
Special processing options. A string or sequence of strings in the format described in Section 3.1.8.Optional.

Figure 3.3: The declaration for the **ideal_gas** entry.

```
'H + OH + M <=> H2O + M'
'H + HO2 <=> O + H2O'
'H + HO2 <=> O2 + H2'
'H + HO2 <=> 2 OH'
'H + H2O2 <=> HO2 + H2'
'H + H2O2 <=> OH + H2O'
'OH + H2 <=> H + H2O'
'2 OH (+ M) <=> H2O2 (+ M)'
'2 OH <=> O + H2O'
'OH + HO2 <=> O2 + H2O'
'OH + H2O2 <=> HO2 + H2O'
'OH + H2O2 <=> HO2 + H2O'
'2 HO2 <=> O2 + H2O2'
'2 HO2 <=> O2 + H2O2'
'OH + HO2 <=> O2 + H2O'
```

3.1.9 Ideal Gas Mixtures

Now we turn to the specific entry types for phases, beginning with **ideal_gas**.

Many combustion and CVD simulations make use of reacting ideal gas mixtures. These can be defined using the **ideal_gas** entry. The Cantera ideal gas model allows any number of species, and any number of reactions among them. It supports all of the options in the widely-used model described by Kee et al. [1989], plus some additional options for species thermodynamic properties and reaction rate expressions.

An example of an **ideal_gas** entry is shown below.

```
ideal_gas(name = 'air8',
          elements = 'N O Ar',
          species = 'gri30: N2 O2 N O NO NO2 N2O AR',
          reactions = 'all',
          transport = 'mix',
          initial_state = state( temperature = 500.0,
                                 pressure = (1.0, 'atm'),
                                 mole_fractions = 'N2:0.78, O2:0.21, AR:0.01'))
```

This entry defines an ideal gas mixture that contains 8 species, the definitions of which are imported from dataset `gri30` (file 'gri30.xml'). All reactions defined in the file are to be included, transport properties are to be computed using mixture rules, and the state of the gas is to be set initially to 500 K, 1 atm, and a composition that corresponds to air.

Transport Models

Two transport models are available for use with ideal gas mixtures. The first is a multicomponent transport model that is based on the model described by Dixon-Lewis [1968] (see also Kee et al. [2003]). The second is a model that uses mixture rules. To select the multicomponent model, set the *transport* field to the string `'multi'`, and to select the mixture-averaged model, set it to the string `'mix'`.

```
ideal_gas(name = gas1,
          ...,
          transport = "multi",   # use multicomponent formulation
          ...)

ideal_gas(name = gas2,
          ...,
          transport = "mix",     # use mixture-averaged formulation
          ...)
```

3.1.10 Stoichiometric Solid

A stoichiometric solid is one that is modeled as having a precise, fixed composition, given by the composition of the one species present. A stoichiometric solid can be used to define a condensed phase that can participate in heterogeneous reactions. (Of course, there cannot be homogeneous reactions, since the composition is fixed.)

```
stoichiometric_solid(name = 'graphite',
              elements = 'C',
              species = 'C(gr)',
              density = (2.2, 'g/cm3'),
              initial_state = state( temperature = 300.0,
                                     pressure = (1.0, 'atm'))
```

stoichiometric_solid*(name, elements, species, transport, initial_state)*
A solid with fixed composition.

name
A string to identify the solid. Must be unique amomg the phase names within the file.

elements
The elements. A string of element symbols.

species
The species. A string containing exactly one species name.

transport
The transport property model to use to compute the thermal conductivity. Not yet implemented.

initial_state
Initial thermodynamic state, specified with an embedded **state** entry.

Figure 3.4: Declaration for the **stoichiometric_solid** entry.

stoichiometric_liquid*(name, elements, species, transport, initial_state)*
An incompressible liquid with fixed composition.

name
A string to identify the liquid. Must be unique amomg the phase names within the file.

elements
The elements. A string of element symbols.

species
The species. A string containing exactly one species name.

transport
Transport property model to use to compute the thermal conductivity and viscosity.

initial_state
Initial thermodynamic state, specified with an embedded **state** entry.

Figure 3.5: Declaration for the **stoichiometric_liquid** entry.

3.1.11 Stoichiometric Liquid

A stoichiometric liquid differs from a stoichiometric solid in only one respect: the transport manager computes the viscosity as well as the thermal conductivity.

3.1.12 Ideal Solutions

An ideal solution is one in which the activity of each species is equal to its mole fraction:

$$a_k = X_k, \tag{3.1}$$

and thus the chemical potential is given by

$$\mu_k = \mu_k^0(T, P) + \hat{R}T \log X_i. \tag{3.2}$$

It is straightforward to show that a consequence of this is that the volume of mixing and the enthalpy of mixing are both zero.

ideal_solution*(name, elements, species, reactions, kinetics, transport, initial_state, options)*

A chemically-reacting condensed-phase ideal solution of multiple species.

name
: A string to identify the phase. Must be unique amomg the phase names within the file.

elements
: The elements. A string of element symbols.

species
: The species. A string or sequence of strings in the format described in Section 3.1.3.

reactions
: The homogeneous reactions. If omitted, no reactions will be included. A string or sequence of strings in the format described in Section 3.1.4.

kinetics
: The kinetics model. Usually this field is omitted, in which case kinetics model `SolutionKinetics`, appropriate for reactions in ideal solutions, is used.

transport
: The transport property model. (Not yet implemented.)

initial_state
: Initial thermodynamic state, specified with an embedded **state** entry.

options
: Special processing options. A string or sequence of strings in the format described in Section 3.1.8.Optional.

Figure 3.6: Declaration for the **ideal_solution** entry.

```
ideal_solution(name = 'saline',
          elements = 'H O Cl Na',
          species = 'aqueous: H2O H+ OH- Na+ Cl-',
          initial_state = state( temperature = 300.0,
                                 mole_fractions = 'H2O:0.982, Na+:0.009, Cl-:0.009'))
```

Here an ideal solution is defined, with species imported from an external dataset `aqueous` (file `aqueous.xml`). Note that these species have thermodynamic properties appropriate for solutes in ionic solutions. For example, species `H+` is not to be confused with a gas-phase hydrogen ion, which would have much higher energy. By keeping aqueous species in a dataset separate from gas-phase ones, it is possible to avoid confusion about the identity of species like `H+`. (Alternatively, they could be assigned names like `H+(aq)`.)

3.2 Interfaces

Now that we have seen how to define bulk, three-dimensional phases, we can describe the procedure to define an interface between phases.

Cantera presently implements a simple model for an interface that treats is as a two-dimensional ideal solution of interfacial species. There is a fixed site density n^0, and each site may be occupied by one of several adsorbates, or may be empty. The chemical potential of each species is computed using the expression for an ideal solution:

$$\mu_k = \mu_k^0 + \hat{R}T \log \theta_k, \tag{3.3}$$

where θ_k is the coverage of species k on the surface. The coverage is related to the surface concentration C_k by

$$\theta_k = \frac{C_k n_k}{n^0}, \tag{3.4}$$

where n_k is the number of sites covered or blocked by species k.

The entry type for this interface model is **ideal_interface**. (Additional interface models will be added in future releases, that allow non-ideal, coverage-dependent properties.)

Defining an interface is much like defining a phase. There are two new fields: *phases* and *site_density*. The *phases* field specifies the bulk phases that participate in the heterogeneous reactions. Although in most cases this string will list one or two phases, no limit is placed on the number. This is particularly useful in some electrochemical problems, where reactions take place near the triple-phase bounday where a gas, an electrolyte, and a metal all meet.

The *site_density* field is the number of adsorption sites per unit area.

Another new aspect is in the embedded **state** entry in the *initial_state* field. When specifying the initial state of an interface, the **state** entry has a field *coverages*, which can be assigned a string specifying the initial surface species coverages.

```
ideal_interface( name = 'silicon_surface',
                 elements = 'Si H',
                 species = 's* s-SiH3 s-H',
                 reactions = 'all',
                 phases = 'gas bulk-Si',
                 site_density = (1.0e15, 'molec/cm2'),
                 innitial_state = state(temperature = 1200.0,
                                        coverages = 's-H:1')
```

ideal_interface*(name, elements, species, reactions, site_density, phases, initial_state, options)*
A chemically-reacting ideal surface solution of multiple species.

name
A string to identify the phase. Must be unique amomg the phase names within the file.

elements
The elements. A string of element symbols.

species
The species. A string or sequence of strings in the format described in Section 3.1.3.

reactions
The heterogeneous reactions at this interface. If omitted, no reactions will be included. A string or sequence of strings in the format described in Section 3.1.4 below.

site_density
The number of adsorption sites per unit area.

phases
A string listing the bulk phases that participate in reactions at this interface.

initial_state
Initial thermodynamic state, specified with an embedded **state** entry.

options
Special processing options, as described in Section 3.1.8.

Figure 3.7: Declaration for the **ideal_interface** entry.

3.3 The **state** entry

The initial state of either a phase or an interface may be set using an embedded **state** entry. Note that only one of *(pressure, density)* may be specified, and only one *(mole_fractions, mass_fractions, coverages)*.

state*(temperature, pressure, mole_fractions, mass_fractions, density, coverages)*
An embedded entry that specifies the thermodynamic state of a phase or interface.

temperature
The temperature.

pressure
The pressure.

density
The density. Cannot be specified if the phase is incompressible.

mole_fractions
A string specifying the species mole fractions. Unspecified species are set to zero.

mass_fractions
A string specifying the species mass fractions. Unspecified species are set to zero.

coverages
A string specifying the species coverages. Unspecified species are set to zero. Can only be specified for interfaces.

Figure 3.8: The declaration for the **state** entry.

CHAPTER

FOUR

Elements, Species, and Reactions

4.1 Elements

The **element** entry defines an element or an isotope of an element. Note that these entries are not often needed, since the the database file 'elements.xml' is searched for element definitions when importing phase and interface definitions. An explicit **element** entry is needed only if an isotope not in 'elements.xml' is required.

```
element(symbol = 'C-13',
        atomic_mass  = 13.003354826)

element("O-!8", 17.9991603)
```

4.2 Species

For each species, a **species** entry is required. Species are defined at the top-level of the input file – their definitions are not embedded in a phase or interface entry.

4.2.1 The Species Name

The *name* field may contain embedded parentheses, `'+'` or `'-'` signs to indicate the charge, or just about anything else that is printable and not a reserved character in XML.

Some example *name* specifications:

element*(symbol, atomic_mass)*
An atomic element or isotope.

symbol
The symbol for the element or isotope. A string of any length is allowed.

atomic_mass
The atomic mass in amu.

Figure 4.1: The declaration for the **element** entry.

species*(name, atoms, thermo, transport, size, charge])*
A constituent of a phase or interface.

name
The species name (or formula). The name may be arbitrarily long, although usually a relatively short, abbreviated name is most convenient. Required parameter.

atoms
The atomic composition, specified by a string containing space-delimited <*element*>:<*atoms*> pairs. The number of atoms may be either an integer or a floating-point number.

thermo
The parameterization to use to compute the reference-state thermodynamic properties. This must be one of the entry types described in Section 4.2.3. To specify multiple parameterizations, each for a different temperature range, group them in parentheses.

transport
An entry specifying parameters to compute this species' contribution to the transport properties. This must be one of the entry types described in Section 4.2.4, and must be consistent with the transport model of the phase into which the species is imported. To specify parameters for multiple transport models, group the entries in parentheses.

size
The species "size." Currently used only for surface species, where it represents the number of sites occupied.

charge
The charge, in multiples of $|e|$. If not specified, the charge will be calculated from the number of "atoms" of element E, which represents an electron.

Figure 4.2: Declaration of the **species** entry.

```
name = CH4
name = methane
name = argon_2+
name = CH2(singlet)
```

4.2.2 The Elemental Composition

The elemental composition is specified in the *atoms* entry, as follows.

```
atoms = "C:1 O:2"              # CO2
atoms = "C:1, O:2"             # CO2 with optional comma
atoms = "Y:1 Ba:2 Cu:3 O:6.5"  # stoichiometric YBCO
atoms = ""                     # a surface species representing an empty site
atoms = "Ar:1 E:-2"            # Ar++
```

For gaseous species, the elemental composition is well-defined, since the species represent distinct molecules. For species in solid or liquid solutions, or on surfaces, there may be several possible ways of defining the species. For example, an aqueous species might be defined with or without including the water molecules in the solvation cage surrounding it.

For surface species, it is possible to omit the *atoms* field entirely, in which case it is composed of nothing, and represents an empty surface site. This can also be done to represent vacancies in solids. A charged vacancy can be defined to be composed solely of electrons:

```
species(name = 'ysz-oxygen-vacancy',
        atoms = 'O:0, E:2',
        ...
       )
```

Note that an atom number of zero may be given if desired, but is completely equivalent to omitting that element.

The number of atoms of an element must be non-negative, except for the special "element" E that represents an electron.

4.2.3 Species Thermodynamic Properties

The **phase** and **interface** entries discussed in the last chapter implement specific models for the thermodynamic properties appropriate for the type of phase or interface they represent. Although each one may use different expressions to compute the properties, they all require thermodynamic property information for the individual species. For the phase types implemented at present, the properties needed are

1. the molar heat capacity at constant pressure $\hat{c}_p^0(T)$ for a range of temperatures and a reference pressure P_0;
2. the molar enthalpy $\hat{h}(T_0, P_0)$ at P_0 and a reference temperature T_0;
3. the absolute molar entropy $\hat{s}(T_0, P_0)$ at (T_0, P_0).

The entry types described in this section can be used to provide these data. Each implements a different *parameterization* (functional form) for the heat capacity. Note that there is no requirement that all species in a phase use the

NASA*(range, coeffs, p0)*
The NASA polynomial parameterization.

coeffs
Array of seven coefficients $(a_0, \ldots, a_6)$

p0
The reference-state pressure, usually 1 atm or 1 bar. If omitted, the default value is used, which is set by the **standard_pressure** directive.

range
The temperature range over which the parameterization is valid. This must be entered as a sequence of two temperature values. Default: none; required input.

Figure 4.3: Declaration for the **NASA** entry.

same parameterization; each species can use the one most appropriate to represent how the heat capacity depends on temperature.

Currently, three entry types are implemented, all of which provide species properties appropriate for models of ideal gas mixtures, ideal solutions, and pure compounds. Non-ideal phase models are not yet implemented, but may be in future releases. When they are, additional entry types may also be added that provide species-specific coefficients required by specific non-ideal equations of state.

The NASA Polynomial Parameterization

The NASA polynomial parameterization is used to compute the species reference-state thermodynamic properties $\hat{c}_p^0(T)$, $\hat{h}^0(T)$, and $\hat{s}^0(T)$.

The NASA parameterization represents $c_p^0(T)$ with a fourth-order polynomial.

$$\frac{c_p^0(T)}{R} = a_0 + a_1 T + a_2 T^2 + a_3 T^3 + a_4 T^4, \tag{4.1}$$

$$\frac{h^0(T)}{RT} = a_0 + \frac{a_1}{2} T + \frac{a_2}{3} T^2 \frac{a_3}{4} T^3 + \frac{a_4}{5} T^4 + a_5, \tag{4.2}$$

$$\frac{s^0(T)}{R} = a_0 \ln T + a_1 T + \frac{a_2}{2} T^2 + \frac{a_3}{3} T^3 + \frac{a_4}{4} T^4 + a_6. \tag{4.3}$$

Note that this is the "old" NASA polynomial form, used in the original NASA equilibrium program and in CHEMKIN™. It is not compatible with the form used in the most recent version of the NASA equilibrium program, which uses 9 coefficients, not 7. (For more on the new form, see `http://cea.grc.nasa.gov`.)

A NASA parameterization is defined by an embedded NASA entry. Very often, two NASA parameterizations are used for two contiguous temperature ranges. This can be specified by assigning the *thermo* field of the **species** entry a sequence of two **NASA** entries.

```
# use one NASA parameterization for T < 1000 K, and another for
# T > 1000 K.
species(name = "O2",
    atoms = " O:2 ",
    thermo = (
       NASA( [  200.00,  1000.00], [  3.782456360E+00,  -2.996734160E-03,
                9.847302010E-06,  -9.681295090E-09,   3.243728370E-12,
               -1.063943560E+03,   3.657675730E+00] ),
       NASA( [ 1000.00,  3500.00], [  3.282537840E+00,   1.483087540E-03,
```

Shomate*(range, coeffs, p0)*

coeffs
Sequence of seven coefficients $(A, \ldots, G)$

p0
The reference-state pressure, usually 1 atm or 1 bar. If omitted, the default value set by the **standard_pressure** directive is used.

range
The temperature range over which the parameterization is valid. This must be entered as a sequence of two temperature values. Default: none; required input.

Figure 4.4: The declaration for the **Shomate** entry.

```
  -7.579666690E-07,   2.094705550E-10,  -2.167177940E-14,
  -1.088457720E+03,   5.453231290E+00] )
) )
```

The Shomate Parameterization

The Shomate parameterization is

$$\hat{c}_p^0(T) = A + Bt + Ct^2 + Dt^3 + \frac{E}{t^2}, \tag{4.4}$$

$$\hat{h}^0(T) = At + \frac{Bt^2}{2} + \frac{Ct^3}{3} + \frac{Dt^4}{4} - \frac{E}{t} + F, \tag{4.5}$$

$$\hat{s}^0(T) = A\ln t + Bt + \frac{Ct^2}{2} + \frac{Dt^3}{3} - \frac{E}{2t^2} + G, \tag{4.6}$$

where $t = T/1000$. It requires 7 coefficients A, B, C, D, E, F, and G. This parameterization is used to represent reference-state properties in the NIST Chemistry WebBook (`http://webbook.nist.gov/chemistry`). **The values of the coefficients A through G should be entered precisely as shown there, with no units attached.** Unit conversions to SI will be handled internally.

```
# use s single Shomate parameterization.
species(name = "O2",
    atoms = " O:2 ",
    thermo = Shomate( [298.0, 6000.0],
                      [29.659,  6.137261, -1.186521, 0.09578, -0.219663,
                       -9.861391, 237.948]
                    ) )
```

Constant Heat Capacity

In some cases, species properties may only be required at a single temperature or over a narrow temperature range. In such cases, the heat capacity can be approximated as constant, and simpler expressions used for the thermodynamic

const_cp *(t0, h0, s0, cp0)*

t0
Temperature parameter T_0. Default: 298.15 K.

h0
Reference-state molar enthalpy at temperature T_0. Default: 0.0.

s0
Reference-state molar entropy at temperature T_0. Default: 0.0.

cp0
Reference-state molar heat capacity (constant). Default: 0.0.

Figure 4.5: The declaration for the **const_cp** entry.

properties. The **const_cp** parameterization computes the properties as follows:

$$\hat{c}_p^0(T) = \hat{c}_p^0(T_0), \tag{4.7}$$

$$\hat{h}^0(T) = \hat{h}^0(T_0) + \hat{c}_p^0(T - T_0), \tag{4.8}$$

$$\hat{s}^0(T) = \hat{s}^0(T_0) + \hat{c}_p^0 \ln(T/T_0) \tag{4.9}$$

The parameterization uses four constants: T_0, $(\hat{c}_p^0(T_0), \hat{h}^0(T_0), \hat{s}^0(T_0))$.

```
thermo = const_cp( t0 = 1200.0,
                   h0 = (-5.0, 'kcal/mol') )
```

See Chapter 5.2 for more examples of use of this parameterization.

4.2.4 Species Transport Coefficients

Transport property models in general require coefficients that express the effect of each species on the transport properties of the phase. The *transport* field may be assigned an embedded entry that provides species-specific coefficients.

The **gas_transport** Entry

Currently, the only entry type is **gas_transport**, which supplies parameters needed by the ideal-gas transport property models. The field values and their units of the **gas_transport** entry are compatible with the transport database parameters described by Kee et al. [1986]. Entries in transport databases in the format described in their report can be used directly in the fields of the **gas_transport** entry, without requiring any unit conversion. The numeric field values should all be entered as pure numbers, with no attached units string.

4.3 Reactions

Cantera supports a number of different types of reactions, including several types of homogeneous reactions, surface reactions, and electrochemical reactions. For each, there is a corresponding entry type. The simplest entry type is **reaction**, which can be used for any homogeneous reaction that has a rate expression that obeys the law of mass action, with a rate coefficient that depends only on temperature.

gas_transport*(geom, diam, well_depth, dipole, polar, rot_relax)*
Provides species-specific coefficients for ideal-gas transport property models.

geom
A string specifying the molecular geometry. One of `'atom'`, `'linear'`, or `'nonlin'`. Required.

diam
The Lennard-Jones collision diameter in Angstroms. Required.

well_depth
The Lennard-Jones well depth in Kelvin. Required.

dipole
The permanent dipole moment in Debye. Default: 0.0

polar
The polarizability in A^3. Default: 0.0

rot_relax
The rotational relaxation collision number at 298 K. Dimensionless. Default: 0.0

Figure 4.6: The declaration for the **gas_transport** entry.

All of the entry types that define reactions share some common features. These are described first, followed by descriptions of the individual reaction types in the following sections.

4.3.1 The Reaction Equation

The reqction equation determines the reactant and product stoichiometry. A relatively simple parsing strategy is currently used, which assumes that all coefficient and species symbols on either side of the equation are delimited by spaces.

```
2 CH2 <=> CH + CH3           # OK
2 CH2<=>CH + CH3             # OK
2CH2 <=> CH + CH3            # error
CH2 + CH2 <=> CH + CH3       # OK
2 CH2 <=> CH+CH3             # error
```

The incorrect versions here would generate "undeclared species" errors and would halt processing of the input file. In the first case, the error would be that the species `'2CH2'` is undeclared, and in the second case it would be species `'CH+CH3'`.

Whether the reaction is reversible or not is determined by the form of the equality sign in the reaction equation. If either `<=>` or `=` is found, then the reaction is regarded as reversible, and the reverse rate will be computed from detailed balance. If, on the other hand, `=>` is found, the reaction will be treated as irreversible.

4.3.2 The Rate Coefficient

The rate coefficient is specified with an embedded entry corresponding to the rate coefficient type. At present, the only implemented type is the modified Arrhenius function

$$k_f(T) = AT^n \exp(-E/\hat{R}T), \tag{4.10}$$

which is defined with an **Arrhenius** entry:

<reaction_type>*(equation, rate_coeff, id, options)*

equation
A string specifying the chemical equation.

rate_coeff
The rate coefficient for the forward direction. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

id
An optional identification string. If omitted, it defaults to a four-digit numeric string beginning with `0001` for the first reaction in the file.

options
Processing options, as described in Section 4.3.4

Figure 4.7: The declaration for a generic reaction entry.

Arrhenius*(A, n, E)*

A
The pre-exponential coefficient. Required input. If entered without units, the units will be computed considering all factors that affect the units. The resulting units string is written to the CTML file individually for each reaction pre-exponential coefficient.

n
The temperature exponent. Dimensionless. Default: 0.0.

E
Activation energy. Default: 0.0.

Figure 4.8: The declaration for the **Arrhenius** entry.

```
rate_coeff = Arrhenius( A = 1.0e13, n = 0, E = (7.3, 'kcal/mol'))
rate_coeff = Arrhenius(1.0e13, 0, (7.3, 'kcal/mol'))
```

As a shorthand, if the *rate_coeff* field is assigned a sequence of three numbers, these are assumed to be (A, n, E) in the modified Arrhenius function.

```
rate_coeff = [1.0e13, 0, (7.3, 'kcal/mol')]  # equivalent to above
```

The units of the pre-exponential factor A can be specified explicitly if desired. If not specified, they will be constructed using the *quantity*, *length*, and *time* units specified in the **units** directive. Since the units of A depend on the reaction order, the units of each reactant concentration (different for bulk species in solution, surface species, and pure condensed-phase species), and the units of the rate of progress (different for homogeneous and heterogeneous reactions), it is usually best *not* to specify units for A, in which case they will be computed taking all of these factors into account.

Note: if $n \neq 0$, then the term T^n should have units of K^n, which would change the units of A. This is *not* done, however, so the units associated with A are really the units for k_f. One way to formally express this is to replace T^n

by the non-dimensional quantity $[T/(1\text{ K})]^n$.

4.3.3 The ID String

An optional identifying string can be entered in the *id* field, which can then be used in the *reactions* field of a phase or interface entry to identify this reaction. If omitted, the reactions are assigned *id* strings as they are read in, beginning with `'0001'`, `'0002'`, ...

Note that the *id* string is only used when selectively importing reactions. If all reactions in the local file or in an external one are imported into a phase or interface, then the reaction *id* field is not used.

4.3.4 Options

Certain conditions are normally flagged as errors by Cantera. In some cases, theey may not be errors, and the *options* field can be used to specify how they should be handled.

skip The `skip` option can be used to temporarily remove this reaction from the phase or interface that imports it, just as if the **reaction** entry were commented out. The advantage of using `skip` instead of commenting it out is that a warning message is printed each time a phase or interface definition tries to import it. This serves as a reminder that this reaction is not included, which can easily be forgotten when a reaction is "temporarily" commented out of an input file.

duplicate Normally, when a reaction is imported into a phase, it is checked to see that it is not a duplicate of another reaction already present in the phase, and an error results if a duplicate is found. But in some cases, it may be appropriate to include duplicate reactions, for example if a reaction can proceed through two distinctly different pathways, each with its own rate expression.

Another case where duplicate reactions can be used is if it is desired to implement a reaction rate coefficient of the form

$$k_f(T) = \sum_{n=1}^{N} A_n T^{b_n} \exp(-E_n/\hat{R}T). \tag{4.11}$$

While Cantera does not provide such a form for reaction rates, it can be implemented by defining N duplicate reactions, and assigning one rate coefficient in the sum to each reaction.

If the `duplicate` option is specified, then the reaction not only *may* have a duplicate, it *must*. Any reaction that specifies that it is a duplicate, but cannot be paired with another reaction in the phase that qualifies as its duplicate generates an error.

negative_A. If some of the terms in the above sum have negative A_n, this scheme fails, since Cantera normally does not allow negative pre-exponential factors. But if there are duplicate reactions such that the total rate is positive, then negative A parameters are acceptable, as long as the `negative_A` option is specified.

4.3.5 Reactions with Pressure-Independent Rate

The **reaction** entry is used to represent homogeneous reactions with pressure-independent rate coefficients and mass action kinetics.

Examples of **reaction** entries that implement some reactions in the GRI-Mech 3.0 natural gas combustion mechanism [Smith et al., 1997] are shown below.

```
units(length = 'cm', quantity = 'mol', act_energy = 'cal/mol')
...
reaction( "O + H2 <=> H + OH",               [3.87000E+04,  2.7,     6260])
```

reaction*(equation, rate_coeff, id, options)*

A homogeneous chemical reaction with pressure-independent rate coefficient and mass-action kinetics.

equation
: A string specifying the chemical equation.

rate_coeff
: The rate coefficient for the forward direction. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

id
: An optional identification string. If omitted, it defaults to a four-digit numeric string beginning with `0001` for the first reaction in the file.

options
: Processing options, as described in Section 4.3.4

Figure 4.9: Declaration for the **reaction** entry.

```
reaction( "O + HO2 <=> OH + O2",              [2.00000E+13,  0.0,        0])
reaction( "O + H2O2 <=> OH + HO2",            [9.63000E+06,  2.0,     4000])
reaction( "O + HCCO <=> H + 2 CO",            [1.00000E+14,  0.0,        0])
reaction( "H + O2 + AR <=> HO2 + AR",         [7.00000E+17, -0.8,        0])
reaction( "HO2 + C3H7 <=> O2 + C3H8",         [2.55000E+10,  0.255,   -943])
reaction( "HO2 + C3H7 => OH + C2H5 + CH2O",   [2.41000E+13,  0.0,        0])
```

4.3.6 Three-Body Reactions

A three-body reaction is a gas-phase reaction of the form

$$\mathrm{A} + \mathrm{B} + \mathrm{M} \rightleftharpoons \mathrm{AB} + \mathrm{M}. \tag{4.12}$$

Here M is an unspecified collision partner that carries away excess energy to stabilize the AB molecule (forward direction) or supplies energy to break the AB bond (reverse direction).

Different species may be more or less effective in acting as the collision partner. A species that is much lighter than A and B may not be able to transfer much of its kinetic energy, and so would be inefficient as a collision partner. On the other hand, a species with a transition from its ground state that is nearly resonant with one in the $\mathrm{AB}^{\ddagger}$ activated complex may be much more effective at exchanging energy than would otherwise be expected.

These effects can be accounted for by defining a collision efficiency ϵ for each species, defined such that the forward reaction rate is

$$k_f(T)[\mathrm{A}][\mathrm{B}][\mathrm{M}], \tag{4.13}$$

where

$$[\mathrm{M}] = \sum_k \epsilon_k C_k, \tag{4.14}$$

where C_k is the concentration of species k. Since any constant collision efficiency can be absorbed into the rate coefficient $k_f(T)$, the default collision efficiency is 1.0.

A three-body reaction may be defined using the **three_body_reaction** entry. The equation string for a three-body reaction must contain an `'M'` or `'m'` on both the reactant and product sides of the equation.

Some examples from GRI-Mech 3.0 are shown below.

three_body_reaction*(equation, rate_coeff, efficiencies, id, options)*

A three-body reaction.

equation
: A string specifying the chemical equation. The reaction can be written in either the association or dissociation directions, and may be reversible or irreversible.

rate_coeff
: The rate coefficient for the forward direction. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

efficiencies
: A string specifying the third-body collision efficiencies. The efficiencies for unspecified species are set to 1.0.

id
: An optional identification string. If omitted, it defaults to a four-digit numeric string beginning with `0001` for the first reaction in the file.

options
: Processing options, as described in Section 4.3.4

Figure 4.10: The declaration for the **three_body_reaction** entry.

```
three_body_reaction( "2 O + M <=> O2 + M",   [1.20000E+17, -1, 0],
         " AR:0.83  C2H6:3  CH4:2  CO:1.75  CO2:3.6  H2:2.4  H2O:15.4 ")

three_body_reaction( "O + H + M <=> OH + M",   [5.00000E+17, -1, 0],
         efficiencies = " AR:0.7  C2H6:3  CH4:2  CO:1.5  CO2:2  H2:2  H2O:6 ")

three_body_reaction(
         equation = "H + OH + M <=> H2O + M",
         rate_coeff = [2.20000E+22, -2, 0],
         efficiencies = " AR:0.38  C2H6:3  CH4:2  H2:0.73  H2O:3.65 "
       )
```

As always, the field names are optional *if* the field values are entered in the declaration order.

4.3.7 Falloff Reactions

A *falloff reaction* is one that has a rate that is first-order in [M] at low pressure, like a three-body reaction, but becomes zero-order in [M] as [M] increases. Dissociation / association reactions of polyatomic molecules often exhibit this behavior.

The simplest expression for the rate coefficient for a falloff reaction is the Lindemann form [Lindemann, 1922]

$$k_f(T, [\mathrm{M}]) = \frac{k_0[\mathrm{M}]}{1 + \frac{k_0[\mathrm{M}]}{k_\infty}} \tag{4.15}$$

In the low-pressure limit, this approaches $k_0[\mathrm{M}]$, and in the high-pressure limit it approaches k_∞.

Defining the non-dimensional *reduced pressure*

$$P_r = \frac{k_0[\mathrm{M}]}{k_\infty}, \tag{4.16}$$

falloff_reaction*(equation, rate_coeff_inf, rate_coeff_0, efficiencies, falloff, id, options)*
A gas-phase falloff reaction.

equation
A string specifying the chemical equation.

rate_coeff_inf
The rate coefficient for the forward direction in the high-pressure limit. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

rate_coeff_0
The rate coefficient for the forward direction in the low-pressure limit. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

efficiencies
A string specifying the third-body collision efficiencies. The efficiency for unspecified species is set to 1.0.

falloff
An embedded entry specifying a falloff function. If omitted, a unity falloff function (Lindemann form) will be used.

id
An optional identification string. If omitted, it defaults to a four-digit numeric string beginning with `0001` for the first reaction in the file.

options
Processing options, as described in Section 4.3.4

Figure 4.11: The declaration for the **falloff_reaction** entry.

Eq. (4.15) may be written as

$$k_f(T, P_r) = k_\infty \left(\frac{P_r}{1 + P_r} \right). \tag{4.17}$$

More accurate models for unimolecular processes lead to other, more complex, forms for the dependence on reduced pressure. These can be accounted for by multiplying the Lindemann expression by a function $F(T, P_r)$:

$$k_f(T, P_r) = k_\infty \left(\frac{P_r}{1 + P_r} \right) F(T, P_r). \tag{4.18}$$

This expression is used to compute the rate coefficient for falloff reactions. The function $F(T, P_r)$ is the *falloff function*, and is specified by assigning an embedded entry to the *falloff* field.

The Troe Falloff Function

A widely-used falloff function is the one proposed by Gilbert et al. [1983].

Troe*(A, T3, T1, T2)*
The Troe falloff function.

A, T3, T1, T2
Numerical parameters. These must be entered as pure numbers with no attached dimensions.

$$\log_{10} F(T, P_r) = \frac{\log_{10} F_{cent}(T)}{1/(1 + f_1^2)}, \tag{4.19}$$

where

$$\begin{aligned} F_{cent}(T) &= (1 - A)\exp(-T/T_3) + A\exp(-T/T_1) + \exp(-T_2/T) && (4.20)\\ f_1 &= (\log_{10} P_r + C)/\left(N - 0.14(\log_{10} P_r + C)\right) && (4.21)\\ C &= -0.4 - 0.67\log_{10} F_{cent} && (4.22)\\ N &= 0.75 - 1.27\log_{10} F_{cent} && (4.23) \end{aligned}$$

This function requires specifying the four parameters (A, T_3, T_1, T_2).

The SRI Falloff Function

This falloff function is based on the one originally due to Stewart et al. [1989], which required three parameters (a, b, c). Kee et al. [1989] generalized this function slightly by adding two more parameters (d, e). (The original form corresponds to $d = 1$, $e = 0$.) Cantera supports the extended 5-parameter form, given by

$$F(T, P_r) = d\left[a\exp(-b/T) + \exp(-T/c)\right]^{1/(1+\log_{10}^2 P_r)} T^e. \tag{4.24}$$

In keeping with the nomenclature of [Kee et al., 1989], we will refer to this as the "SRI" falloff function.

SRI*(a, b, c, d, e)*
The SRI falloff function.

a, b, c, d, e
Numerical parameters. These must be entered as pure numbers without attached dimensions.

The Wang-Frenklach Falloff Function

This falloff function is described in Wang and Frenklach [1993].

WangFrenklach*(A, T3, T1, T2, a0, a1, a2, s0, s1, s2)*
The Wang-Frenklach falloff function.[Wang and Frenklach, 1993]

A, T3, T1, T2
Numerical parameters with the same meaning as those for the Troe falloff function.

a0, a1, a2
The parameters α_0, α_1, α_2.

s0, s1, s2
The parameters σ_0, σ_1, σ_2.

$$\log_{10} F(T, P_r) = \frac{\log_{10} F_{cent}}{\exp[(\log_{10} P_r - \alpha)^2/\sigma^2]} \tag{4.25}$$

$$\alpha = \alpha_0 + \alpha_1 T + \alpha_2 T^2 \tag{4.26}$$

$$\sigma = \sigma_0 + \sigma_1 T + \sigma_2 T^2 \tag{4.27}$$

$$F_{cent}(T) = (1 - A)\exp(-T/T_3) + A\exp(-T/T_1) + \exp(-T_2/T) \tag{4.28}$$

$$\tag{4.29}$$

4.3.8 Surface Reactions

The reaction types described above can only be imported into bulk phases; if one of them is imported into an interface definition, an error results. Conversely, the reaction types described in this section and the next one model heterogeneous reactions and can only be imported into interface definitions.

For the present purposes, a "surface reaction" is any reaction that takes place on a surface, including both reactions solely among surface species and gas-surface reactions. A **surface_reaction** entry specifies a surface reaction.

A surface reaction can involve any species that belong to the interface it is imported into, or one of the bulk phases specified in the *phases* field of the interface. Species are identified by name, and so the species names in all participating phases should be unique among all the phases.

surface_reaction*(equation, rate_coeff, id, options)*

A heterogeneous chemical reaction with pressure-independent rate coefficient and mass-action kinetics.

equation

A string specifying the chemical equation.

rate_coeff

The rate coefficient for the forward direction. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

sticking_prob

The reactive sticking probability for the forward direction. This can only be specified if there is only one bulk-phase reactant and it belongs to an ideal gas phase. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

id

An optional identification string. If omitted, it defaults to a four-digit numeric string beginning with `0001` for the first reaction in the file.

options

Processing options, as described in Section 4.3.4

4.3.9 Charge-Transfer Reactions

A charge-transfer reaction is an electrochemical oxidation/reduction reaction that transfers charge between phases at different electric potentials. The electrical potential energy for the charged species participating in the reaction must be added to their internal energies; therefore the presence of the potential difference between the phases alters the reaction thermochemistry, which must be taken into account in computing reverse reaction rates for reversible reactions.

It also affects the activation energy, as shown in Fig. 4.12. Here the simplest case is shown in which a negatively-charged species "r^-" in solution gives up an electron to an immersed electrode and becomes the neutral species "o":

$$\mathrm{r}^- \rightleftharpoons \mathrm{o} + \mathrm{e}^-. \tag{4.30}$$

A hypothetical potential energy curve for this reaction is shown for the case of zero electrical bias between the electrode and the electrolytic solution, and for positive and negative bias.

The shift in the product state energy is simply $-F\Delta V$, the change in electrical potential energy of the electron in the electrode. Here F is Faraday's constant.

The energy surface in the neighborhood of the transition state is also affected by the bias, but the magnitude of the change with bias depends on the nature of the transition state, and is affected by the double-layer structure at the interface. The two extremes are

1. the forward barrier is unaffected, and thus the reverse barrier shows the full dependence on the bias:

$$E_{rev}(\Delta V) = E_{rev}(0) + F\Delta V; \tag{4.31}$$

   and

2. the reverse barrier is unaffected, and thus

$$E_{fwd}(\Delta V) = E_{fwd}(0) - F\Delta V; \tag{4.32}$$

For most electron-transfer reactions, the dependence of the activation energy on bias is between these two extremes:

$$E_{fwd}(\Delta V) = E_{fwd}(0) - \beta F\Delta V, \tag{4.33}$$
$$E_{rev}(\Delta V) = E_{rev}(0) + (1-\beta)F\Delta V \tag{4.34}$$

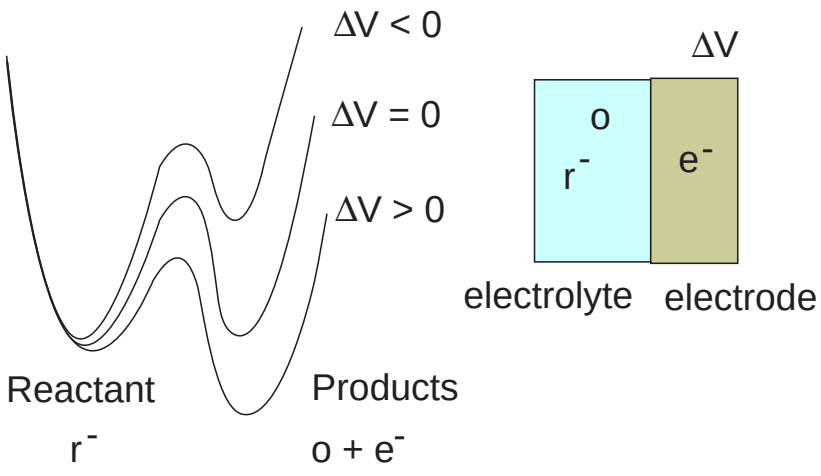


Figure 4.12: Potential energy surface along the reaction coordinate for an electron-transfer reaction.

with $0 < \beta < 1$.

charge_transfer_reaction*(equation, rate_coeff, id, options)*
A heterogenous chemical reaction that transfers charge between phases.

equation
A string specifying the chemical equation.

rate_coeff
The rate coefficient for the forward direction. If a sequence of three numbers is given, these will be interpreted as $[A, n, E]$ in the modified Arrhenius function $AT^n \exp(-E/\hat{R}T)$.

j0
The exchange current density. Instead of specifying the rate coefficient, an exchange current density may be specified. This value is for unit activity of all reactants, and therefore may differ from tabulated exchange current densities.

beta
The β parameter.

id
An optional identification string. If omitted, it defaults to a four-digit numeric string beginning with `0001` for the first reaction in the file.

options
Processing options, as described in Section 4.3.4

The results for a simple electron-transfer reaction can be generalized to any charge-transfer reaction. In the general case, the net change in electrical potential energy in the reaction is

$$\Delta E_{\mathrm{pot},i} = F \sum_k \nu_{k,i} z_k V_k \tag{4.35}$$

where F is Faraday's constant, $\nu_{k,i}$ is the net stoichiometric coefficient for reaction i, z_k is the charge of species k, and V_k is the electric potential of the *phase* in which species k resides.

Therefore,

$$E_{f,i} = E^0_{f,i} + \beta \Delta E_{\mathrm{pot},i}, \tag{4.36}$$

$$E_{r,i} = E^0_{r,i} - (1-\beta) \Delta E_{\mathrm{pot},i}, \tag{4.37}$$

$$\tag{4.38}$$

If $\beta = 0.5$, the potential difference acts symmetrically on the forward and reverse reactions.

Note: although this is formulated for a general charge transfer reaction, in reality charge transfers typically occur one electron (or proton) at a time, and so $\Delta E_{\mathrm{pot},i} = \pm F \Delta V$. Care should be taken in writing an elementary electrochemical reaction mechanism to avoid "elementary" reactions that transfer multiple electrons at once, or else the effect of the bias on the reaction rate may be overstated.

The Exchange Current Density

Rate coefficients of charge transfer reactions are often expressed in terms of the *exchange current density* $j0$, defined for an elementary reaction as

$$j_0 = F|n|k_f \prod C_{k,0}^{\nu_k^{(r)}}, \tag{4.39}$$

where n is the number of units of charge transferred (usually 1), and the concentrations are for a specified reference state (e.g., the values in the stirred electrolytic solution far from the electrode). Here $C_{k,0}$ is taken to be the reference concentration that results in unit activity for species k. This may be different than the concentrations used to generate tabulated values of j_0, and so care should be taken to insure that the proper value for j_0 is entered.

CHAPTER

FIVE

Examples

Now that we have covered how to write Cantera input files, in the next few sections we'll look at examples where they are used for real applications. [1]

5.1 Example 1: Hydrogen / Oxygen Combustion

In the first example, we'll define a reacting ideal-gas mixture that can be used to simulate hydrogen combustion in oxygen. We first create input file 'h2mech.in', containing the text shown below. This file includes species transport data, although transport properties will not be used in this example. The species and reaction data are taken from GRI-Mech 3.0 [Smith et al., 1997].

5.1.1 The Input File

```
units(length = "cm", time = "s", quantity = "mol", act_energy = "cal/mol")

ideal_gas(name = "hydrogen_oxygen",
          elements = " O  H  Ar ",
          species =  " H2  H  O  O2  OH  H2O  HO2  H2O2  AR",
          reactions = "all"
         )

#-------------------------------------------------------------------------------
#  Species data
#-------------------------------------------------------------------------------

species(name = "H2",
    atoms = " H:2 ",
    thermo = (
       NASA( [  200.00,  1000.00], [  2.344331120E+00,   7.980520750E-03,
              -1.947815100E-05,   2.015720940E-08,  -7.376117610E-12,
              -9.179351730E+02,   6.830102380E-01] ),
       NASA( [ 1000.00,  3500.00], [  3.337279200E+00,  -4.940247310E-05,
               4.994567780E-07,  -1.795663940E-10,   2.002553760E-14,
              -9.501589220E+02,  -3.205023310E+00] )
             ),
    transport = gas_transport(
```

[1] These examples, in some cases, use parts of Cantera that have not been fully documented yet, such as heterogeneouus chemistry. For these, documentation is in the works.

```
                      geom = "linear",
                      diam =     2.92,
                      well_depth =    38.00,
                      polar =     0.79,
                      rot_relax =   280.00)
       )

  species(name = "H",
      atoms = " H:1 ",
      thermo = (
         NASA( [  200.00,  1000.00], [  2.500000000E+00,   7.053328190E-13,
                 -1.995919640E-15,   2.300816320E-18,  -9.277323320E-22,
                  2.547365990E+04,  -4.466828530E-01] ),
         NASA( [ 1000.00,  3500.00], [  2.500000010E+00,  -2.308429730E-11,
                  1.615619480E-14,  -4.735152350E-18,   4.981973570E-22,
                  2.547365990E+04,  -4.466829140E-01] )
               ),
      transport = gas_transport(
                      geom = "atom",
                      diam =     2.05,
                      well_depth =   145.00)
         )

  species(name = "O",
      atoms = " O:1 ",
      thermo = (
         NASA( [  200.00,  1000.00], [  3.168267100E+00,  -3.279318840E-03,
                  6.643063960E-06,  -6.128066240E-09,   2.112659710E-12,
                  2.912225920E+04,   2.051933460E+00] ),
         NASA( [ 1000.00,  3500.00], [  2.569420780E+00,  -8.597411370E-05,
                  4.194845890E-08,  -1.001777990E-11,   1.228336910E-15,
                  2.921757910E+04,   4.784338640E+00] )
               ),
      transport = gas_transport(
                      geom = "atom",
                      diam =     2.75,
                      well_depth =    80.00)
         )

  species(name = "O2",
      atoms = " O:2 ",
      thermo = (
         NASA( [  200.00,  1000.00], [  3.782456360E+00,  -2.996734160E-03,
                  9.847302010E-06,  -9.681295090E-09,   3.243728370E-12,
                 -1.063943560E+03,   3.657675730E+00] ),
         NASA( [ 1000.00,  3500.00], [  3.282537840E+00,   1.483087540E-03,
                 -7.579666690E-07,   2.094705550E-10,  -2.167177940E-14,
                 -1.088457720E+03,   5.453231290E+00] )
               ),
      transport = gas_transport(
                      geom = "linear",
                      diam =     3.46,
                      well_depth =   107.40,
                      polar =     1.60,
                      rot_relax =     3.80)
         )

  species(name = "OH",
      atoms = " O:1  H:1 ",
```

```
    thermo = (
       NASA( [  200.00,  1000.00], [  3.992015430E+00,  -2.401317520E-03,
                4.617938410E-06,  -3.881133330E-09,   1.364114700E-12,
                3.615080560E+03,  -1.039254580E-01] ),
       NASA( [ 1000.00,  3500.00], [  3.092887670E+00,   5.484297160E-04,
                1.265052280E-07,  -8.794615560E-11,   1.174123760E-14,
                3.858657000E+03,   4.476696100E+00] )
             ),
    transport = gas_transport(
                     geom = "linear",
                     diam =     2.75,
                     well_depth =    80.00)
       )

species(name = "H2O",
    atoms = " H:2  O:1 ",
    thermo = (
       NASA( [  200.00,  1000.00], [  4.198640560E+00,  -2.036434100E-03,
                6.520402110E-06,  -5.487970620E-09,   1.771978170E-12,
               -3.029372670E+04,  -8.490322080E-01] ),
       NASA( [ 1000.00,  3500.00], [  3.033992490E+00,   2.176918040E-03,
               -1.640725180E-07,  -9.704198700E-11,   1.682009920E-14,
               -3.000429710E+04,   4.966770100E+00] )
             ),
    transport = gas_transport(
                     geom = "nonlinear",
                     diam =     2.60,
                     well_depth =   572.40,
                     dipole =     1.84,
                     rot_relax =     4.00)
       )

species(name = "HO2",
    atoms = " H:1  O:2 ",
    thermo = (
       NASA( [  200.00,  1000.00], [  4.301798010E+00,  -4.749120510E-03,
                2.115828910E-05,  -2.427638940E-08,   9.292251240E-12,
                2.948080400E+02,   3.716662450E+00] ),
       NASA( [ 1000.00,  3500.00], [  4.017210900E+00,   2.239820130E-03,
               -6.336581500E-07,   1.142463700E-10,  -1.079085350E-14,
                1.118567130E+02,   3.785102150E+00] )
             ),
    transport = gas_transport(
                     geom = "nonlinear",
                     diam =     3.46,
                     well_depth =   107.40,
                     rot_relax =     1.00)
       )

species(name = "H2O2",
    atoms = " H:2  O:2 ",
    thermo = (
       NASA( [  200.00,  1000.00], [  4.276112690E+00,  -5.428224170E-04,
                1.673357010E-05,  -2.157708130E-08,   8.624543630E-12,
               -1.770258210E+04,   3.435050740E+00] ),
       NASA( [ 1000.00,  3500.00], [  4.165002850E+00,   4.908316940E-03,
               -1.901392250E-06,   3.711859860E-10,  -2.879083050E-14,
               -1.786178770E+04,   2.916156620E+00] )
             ),
```

```
    transport = gas_transport(
                    geom = "nonlinear",
                    diam =     3.46,
                    well_depth =   107.40,
                    rot_relax =     3.80)
       )

species(name = "AR",
    atoms = " Ar:1 ",
    thermo = (
       NASA( [  300.00,  1000.00], [  2.500000000E+00,   0.000000000E+00,
                0.000000000E+00,   0.000000000E+00,   0.000000000E+00,
               -7.453750000E+02,   4.366000000E+00] ),
       NASA( [ 1000.00,  5000.00], [  2.500000000E+00,   0.000000000E+00,
                0.000000000E+00,   0.000000000E+00,   0.000000000E+00,
               -7.453750000E+02,   4.366000000E+00] )
             ),
    transport = gas_transport(
                    geom = "atom",
                    diam =     3.33,
                    well_depth =   136.50)
       )

#-------------------------------------------------------------------------------
#  Reaction data
#-------------------------------------------------------------------------------

#  Reaction 1
three_body_reaction( "2 O + M <=> O2 + M",   [1.20000E+17, -1, 0],
         efficiencies = " AR:0.83  H2:2.4  H2O:15.4 ")

#  Reaction 2
three_body_reaction( "O + H + M <=> OH + M",   [5.00000E+17, -1, 0],
         efficiencies = " AR:0.7  H2:2  H2O:6 ")

#  Reaction 3
reaction( "O + H2 <=> H + OH",   [3.87000E+04, 2.7, 6260])

#  Reaction 4
reaction( "O + HO2 <=> OH + O2",   [2.00000E+13, 0, 0])

#  Reaction 5
reaction( "O + H2O2 <=> OH + HO2",   [9.63000E+06, 2, 4000])

#  Reaction 6
reaction( "H + 2 O2 <=> HO2 + O2",   [2.08000E+19, -1.24, 0])

#  Reaction 7
reaction( "H + O2 + H2O <=> HO2 + H2O",   [1.12600E+19, -0.76, 0])

#  Reaction 8
reaction( "H + O2 + AR <=> HO2 + AR",   [7.00000E+17, -0.8, 0])

#  Reaction 9
reaction( "H + O2 <=> O + OH",   [2.65000E+16, -0.6707, 17041])

#  Reaction 10
three_body_reaction( "2 H + M <=> H2 + M",   [1.00000E+18, -1, 0],
         efficiencies = " AR:0.63  H2:0  H2O:0 ")
```

```
#  Reaction 11
reaction( "2 H + H2 <=> 2 H2",   [9.00000E+16, -0.6, 0])

#  Reaction 12
reaction( "2 H + H2O <=> H2 + H2O",   [6.00000E+19, -1.25, 0])

#  Reaction 13
three_body_reaction( "H + OH + M <=> H2O + M",   [2.20000E+22, -2, 0],
         efficiencies = " AR:0.38  H2:0.73  H2O:3.65 ")

#  Reaction 14
reaction( "H + HO2 <=> O + H2O",   [3.97000E+12, 0, 671])

#  Reaction 15
reaction( "H + HO2 <=> O2 + H2",   [4.48000E+13, 0, 1068])

#  Reaction 16
reaction( "H + HO2 <=> 2 OH",   [8.40000E+13, 0, 635])

#  Reaction 17
reaction( "H + H2O2 <=> HO2 + H2",   [1.21000E+07, 2, 5200])

#  Reaction 18
reaction( "H + H2O2 <=> OH + H2O",   [1.00000E+13, 0, 3600])

#  Reaction 19
reaction( "OH + H2 <=> H + H2O",   [2.16000E+08, 1.51, 3430])

#  Reaction 20
falloff_reaction( "2 OH (+ M) <=> H2O2 (+ M)",
         rate_coeff_inf = [7.40000E+13, -0.37, 0],
         rate_coeff_0   = [2.30000E+18, -0.9, -1700],
         falloff = Troe(A = 0.7346, T3 = 94, T1 = 1756, T2 = 5182),
         efficiencies = " AR:0.7  H2:2  H2O:6 ")

#  Reaction 21
reaction( "2 OH <=> O + H2O",   [3.57000E+04, 2.4, -2110])

#  Reaction 22
reaction( "OH + HO2 <=> O2 + H2O",   [1.45000E+13, 0, -500])

#  Reaction 23
reaction( "OH + H2O2 <=> HO2 + H2O",   [2.00000E+12, 0, 427])

#  Reaction 24
reaction( "OH + H2O2 <=> HO2 + H2O",   [1.70000E+18, 0, 29410])

#  Reaction 25
reaction( "2 HO2 <=> O2 + H2O2",   [1.30000E+11, 0, -1630])

#  Reaction 26
reaction( "2 HO2 <=> O2 + H2O2",   [4.20000E+14, 0, 12000])

#  Reaction 27
reaction( "OH + HO2 <=> O2 + H2O",   [5.00000E+15, 0, 17330])
```

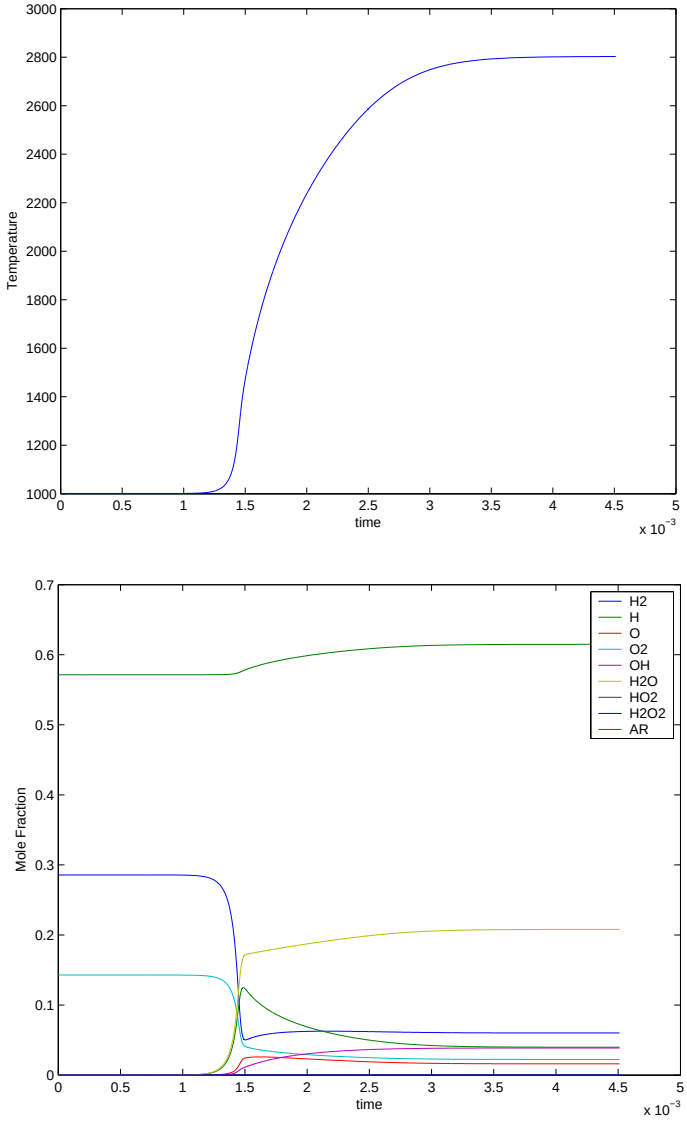
5.1.2 The Matlab m-File

Here this input file is used in a Matlab application to compute adiabatic, constant-volume combustion of a hydrogen/oxygen mixture. The Matlab m-file is shown here.

```
gas = importPhase('h2mech.cti','hydrogen_oxygen');
nsp = nSpecies(gas);

% set the initial conditions
set(gas,'T',1001.0,'P',0.1*oneatm,'X','H2:2, O2:1, AR:4');

% create a reactor, and insert the gas
r = Reactor(gas);

t = 0;
dt = 3.0e-4;
tnow = 0;
m = 0;
for n = 1:15
    t = t + dt;
    while tnow < t
      tnow = step(r, t);  % take one internal timestep
      m = m + 1;
      tim(m) = tnow;
      temp(m) = temperature(r);
      xm(:,m) = moleFractions(gas);
    end
end

% make plots

figure(1);
plot(tim, temp);
xlabel('time');
ylabel('Temperature');

figure(2);
plot(tim, xm);
xlabel('time');
ylabel('Mole Fraction');
legend(speciesNames(gas));
```

5.1.3 Results

The plots produced by running the Matlab m-file are shown below.



5.2 Example 2: Chemical Vapor Deposition

Here we'll simulate a problem with heterogeneous chemistry. (The heterogeneous chemistry capabilities of Cantera are not yet fully documented. This example shows a bit of what can be done; more complete documentation will be forthcoming.)

This example implements a subset of the diamond chemical vapor deposition mechanism of Harris and Goodwin [1993]. Specifically, it implements the "trough" part of the mechanism, which describes addition of a carbon that bridges between two dimer rows on the (2x1) reconstructed diamond (100) surface. (The rest of the mechanism describes insertion of a carbon into the dimer 5-member rings; this process is faster than bridging across the trough, and is therefore not rate-limiting.)

The input file is shown below.

```
# Trough mechanism from 'S. J. Harris and D. G. Goodwin, 'Growth on
# the Reconstructed Diamond (100) Surface,' J. Phys. Chem. vol. 97,
# 23-28 (1993). Reactions a - t are taken directly from Table II, with
# thermochemistry from Table IV. Reaction u is added here.

units(length = 'cm', quantity = 'mol', act_energy = 'kcal/mol')

# ------- the gas -------------------------

ideal_gas(name = 'gas',
          elements = 'H C',
          species = 'gri30: H H2 CH3 CH4',
          initial_state = state(temperature = 1200.0,
                      pressure = 20.0*OneAtm/760.0,
                      mole_fractions = 'H:0.002, H2:1, CH4:0.01, CH3:0.0002'))


#------- bulk diamond --------------------

stoichiometric_solid(name = 'diamond',
                     elements = 'C',
                     density = (3.52, 'g/cm3'),
                     species = 'C(d)')

species(name = 'C(d)', atoms = 'C:1' )  # no thermo needed (rxn 'u' is irrev.)




#------- the diamond surface -------------

ideal_interface(name = 'diamond_100',
                elements = 'H C',
                species = 'c6HH c6H* c6*H c6** c6HM c6HM* c6*M c6B',
                reactions = 'all',
                phases = 'gas diamond',
                site_density = (3.0e-9, 'mol/cm2'),
                initial_state = state(temperature = 1200.0,
                                      coverages = 'c6H*:0.1, c6HH:0.9'))


species(name = 'c6H*',
        atoms = 'H:1',
        thermo = const_cp(h0 = (51.7, 'kcal/mol'),
                          s0 = (19.5, 'cal/mol/K') ) )

species(name = 'c6*H',
        atoms = 'H:1',
        thermo = const_cp(h0 = (46.1, 'kcal/mol'),
                          s0 = (19.9, 'cal/mol/K') ) )

species(name = 'c6HH',
        atoms = 'H:2',
        thermo = const_cp(h0 = (11.4, 'kcal/mol'),
                          s0 = (21.0, 'cal/mol/K')) )

species(name = 'c6HM',
        atoms = 'C:1 H:4',
        thermo = const_cp(h0 = (26.9, 'kcal/mol'),
```

```
                        s0 = (40.3, 'cal/mol/K') ) )

species(name = 'c6HM*',
        atoms = 'C:1 H:3',
        thermo = const_cp(h0 = (65.8, 'kcal/mol'),
                        s0 = (40.1, 'cal/mol/K') ) )

species(name = 'c6*M',
        atoms = 'C:1 H:3',
        thermo = const_cp(h0 = (53.3, 'kcal/mol'),
                        s0 = (38.9, 'cal/mol/K') ) )

species(name = 'c6**',
        atoms = 'C:0',
        thermo = const_cp(h0 = (90.0, 'kcal/mol'),
                        s0 = (18.4, 'cal/mol/K') ) )

species(name = 'c6B',
        atoms = 'H:2 C:1',
        thermo = const_cp(h0 = (40.9, 'kcal/mol'),
                        s0 = (26.9, 'cal/mol/K') ) )

surface_reaction( 'c6HH + H <=> c6H* + H2',        [1.3e14,   0.0,  7.3]) # a
surface_reaction( 'c6H* + H <=> c6HH',             [1.0e13,   0.0,  0.0]) # b
surface_reaction( 'c6H* + CH3 <=> c6HM',           [5.0e12,   0.0,  0.0]) # c
surface_reaction( 'c6HM + H <=> c6*M + H2',        [1.3e14,   0.0,  7.3]) # d
surface_reaction( 'c6*M + H <=> c6HM',             [1.0e13,   0.0,  0.0]) # e
surface_reaction( 'c6HM + H <=> c6HM* + H2',       [2.8e7,    2.0,  7.7]) # f
surface_reaction( 'c6HM* + H <=> c6HM',            [1.0e13,   0.0,  0.0]) # g
surface_reaction( 'c6HM* <=> c6*M',                [1.0e8,    0.0,  0.0]) # h
surface_reaction( 'c6HM* + H <=> c6H* + CH3',      [3.0e13,   0.0,  0.0]) # i
surface_reaction( 'c6HM* + H <=> c6B + H2',        [1.3e14,   0.0,  7.3]) # k
surface_reaction( 'c6*M + H <=> c6B + H2',         [2.8e7,    2.0,  7.7]) # l
surface_reaction( 'c6HH + H <=> c6*H + H2',        [1.3e14,   0.0,  7.3]) # m
surface_reaction( 'c6*H + H <=> c6HH',             [1.0e13,   0.0,  0.0]) # n
surface_reaction( 'c6H* + H <=> c6** + H2',        [1.3e14,   0.0,  7.3]) # o
surface_reaction( 'c6** + H <=> c6*H',             [1.0e13,   0.0,  0.0]) # p
surface_reaction( 'c6*H + H <=> c6** + H2',        [4.5e6,    2.0,  5.0]) # q
surface_reaction( 'c6** + H <=> c6*H',             [1.0e13,   0.0,  0.0]) # r
surface_reaction( 'c6** + CH3 <=> c6*M',           [5.0e12,   0.0,  0.0]) # s
surface_reaction( 'c6H* <=> c6*H',                 [1.0e8,    0.0,  0.0]) # t

# reaction added here to add new carbon atom to bulk, and regenerate
# initial site
surface_reaction('c6B  => c6HH + C(d)',          [1.0e9,    0.0,  0.0]) # u
```

5.2.1 The Python Script

A Python script to use this mechanism to compute the growth rate and surface coverages as a function of the atomic hydrogen mole fraction at the surface is shown below. (Of course, this problem could have been simulated just as easily in Matlab; we show a Python script here for the sake of variety.)

```
from Cantera import *

# import the bulk phases
```

```
g, dbulk = importPhases('diamond.cti',['gas','diamond'])

# import the interface
d = importInterface('diamond.cti','diamond_100',phases = [g, dbulk])

mw = dbulk.molarMasses()[0]  # mol. wt. of carbon

t = g.temperature()
p = g.pressure()
x = g.moleFractions()
ih = g.speciesIndex('H')

f = open('d.csv','w')
for n in range(20):
    x[ih] /= 1.4
    g.setState_TPX(t, p, x)
    # integratae the coverage equations to steady state
    d.advanceCoverages(100.0)
    cdot = d.netProductionRates(phase = dbulk)[0]  # net rate of C(d) production / m^2
    mdot = mw*cdot
    linear_rate = mdot/dbulk.density()
    writeCSV(f,[x[ih],rate]+list(d.coverages()))
f.close()
```

Function `importPhases` can be used to import multiple phases from one input file. The syntax is as shown here. Function `importInterface` imports a single interface definition. Note that a list of objects must be supplied to this function that implement the bulk phases. These are checked to verify that there is one in the list for each phase specified in the *phases* field of the interface definition. Therefore, when constructing an object representing an interface, it is necessary to build the bulk-phase objects first.

Function `importInterface` returns an object belonging to Python class `Interface`. One of its methods is `advanceCoverages`, which advances the surface species coverages in time by integrating the coverage rate equations, holding the bulk-phase concentrations fixed. This is used here to determine the steady-state coverages, by integrating for 100 s.

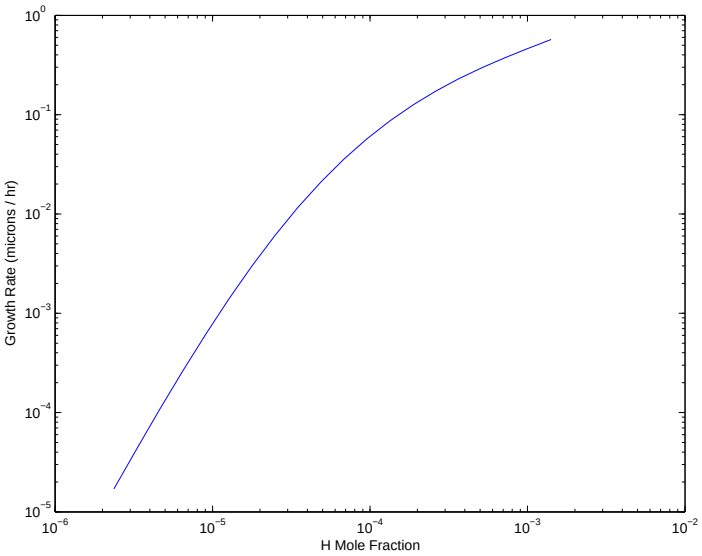
At steady state, the net production rate for each surface species is zero, but this is not true for the bulk-phase species. In this example, at steady state H and CH_3 are being consumed, and H_2 and bulk diamond are being produced. The steady-state deposition rate can be determined from the rate at which bulk diamond is being produced. Method `netProductionRates` returns an array of the net production rates (kmol/m^2/s) of the species of the specified phase. For bulk diamond, there is only one species, so the net carbon deposition rate is given by

```
sdot = d.netProductionRates(phase = dbulk)
cdot = sdot[0]
```

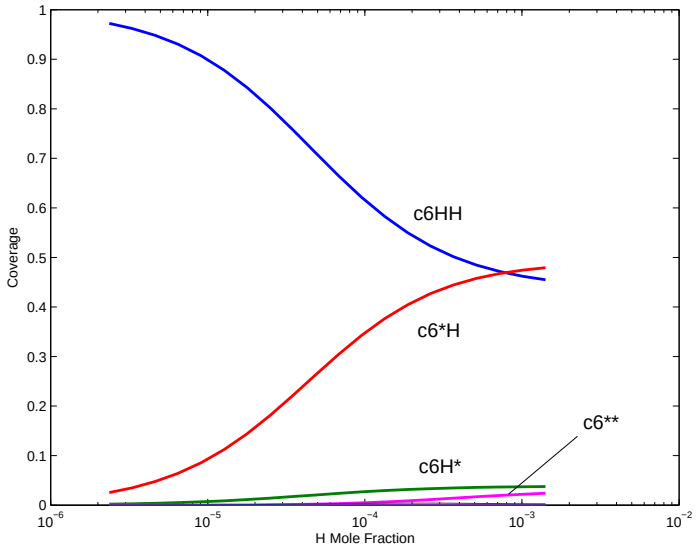
which may be combined into one statement, as is done in the script. This can easily be converted to a mass deposition rate by multiplying by the molecular weight of cargon, and then the linear growth rate can be determined by dividing this by the density of bulk diamond.

5.2.2 Results

The computed linear growth rate converted to microns per hour is shown below, as a function of the atomic hydrogen mole fraction holding the CH_3 mole fraction fixed. The strong dependence on the H mole fraction results since it is atomic hydrogen that creates the open surface sites where CH_3 may attach.



The computed steady-state surface coverages are shown below.



APPENDIX

A

Glossary

bulk
: Not pertaining to, or affected by, an interface. A bulk phase is a macroscopic three-dimensional phase, and may be a solid, liquid, or gas.

extensive
: A thermodynamic property that scales with system size. The volume, mass, internal energy, and entropy are all extensive.

intensive
: A thermodynamic property that is independent of the system size. Temperature, pressure, density, and chemical potential are intensive, as is any extensive property expressed per mole or per unit mass, such as the molar volume, the molar entropy, the specific enthalpy, and so on.

interface
: The boundary between two phases. An interface is itself a type of phase, and may have its own set of species (adsorbates) that exist only at the interface.

phase
: Matter with a homogeneous chemical composition and physical structure. A phase may be crystalline, amorphous, liquid, or gaseous; it may be a pure element, a chemical compound, or a solution of many components. A phase need not be thermodynamically stable — graphite, diamond, and solid C_{60} are all phases of carbon, and may co-exist indefinitely.

mixture
: Any material sample containing multiple consituents.

manager
: A Cantera C++ class that is responsible for implementing a model. Managers are grouped by function into families — there is a family of "kinetics managers," a family of "transport managers," etc. All managers of given family share a common interface, and are interchangeable (within limits).

model
: A set of equations to evaluate a property or multiple properties, complete will all boundary conditions and/or coefficient values.

solution
: A single-phase mixture of multiple species. A solution is fully mixed on all scales, from molecular to macroscopic.

species
: A constituent of a phase or interface. A species has a fixed elemental composition, and in a gas, corresponds to a distinct molecule. This may not be the case in a liquid or solid, or at an interface. In every case, however, the phase composition is fixed by specifying the relative proportions of the species.

stoichiometric
: A phase with fixed composition, or a reaction among reactants present in specified proportions.

surface
: An interface between a gaseous phase and a condensed phase.

transport model
: A model to compute transport properties.

APPENDIX

B

The Elements Database

The element atomic masses contained in file ‘elements.xml’ is listed in the table below.

Element	Atomic Mass (amu)
H	1.00794
D	2.0147
Tr	3.016327
He	4.002602
Li	6.941
Be	9.012182
B	10.811
C	12.011
N	14.00674
O	15.9994
F	18.9984032
Ne	20.1797
Na	22.98977
Mg	24.3050
Al	26.98154
Si	28.0855
P	30.97376
S	32.066
Cl	35.4527
Ar	39.948
K	39.0983
Ca	40.078
Sc	44.95591
Ti	47.88
V	50.9415
Cr	51.9961
Mn	54.9381
Fe	55.847
Co	58.9332
Ni	58.69
Cu	63.546
Zn	65.39
Ga	69.723
Ge	72.61

Element	Atomic Mass (amu)
As	74.92159
Se	78.96
Br	79.904
Kr	83.80
Rb	85.4678
Sr	87.62
Y	88.90585
Zr	91.224
Nb	92.90638
Mo	95.94
Tc	97.9072
Ru	101.07
Rh	102.9055
Pd	106.42
Ag	107.8682
Cd	112.411
In	114.82
Sn	118.710
Sb	121.75
Te	127.6
I	126.90447
Xe	131.29
Cs	132.90543
Ba	137.327
La	138.9055
Ce	140.115
Pr	140.90765
Nd	144.24
Pm	144.9127
Sm	150.36
Eu	151.965
Gd	157.25
Tb	158.92534
Dy	162.50

Element	Atomic Mass (amu)
Ho	164.93032
Er	167.26
Tm	168.93421
Yb	173.04
Lu	174.967
Hf	178.49
Ta	180.9479
W	183.85
Re	186.207
Os	190.2
Ir	192.22
Pt	195.08
Au	196.96654
Hg	200.59
Ti	204.3833
Pb	207.2
Bi	208.98037
Po	208.9824
At	209.9871
Rn	222.0176
Fr	223.0197
Ra	226.0254
Ac	227.0279
Th	232.0381
Pa	231.03588
U	238.0508
Np	237.0482
Pu	244.0482
E	0.000545

BIBLIOGRAPHY

G. Dixon-Lewis. Flame structure and flame reaction kinetics, II: Transport phenomena in multicomponent systems. *Proc. Roy. Soc. A*, 307:111–135, 1968.

R. G. Gilbert, K. Luther, and J. Troe. *Ber. Bunsenges. Phys. Chem.*, 87:169, 1983.

S. J. Harris and D. G. Goodwin. Growth on the reconstructed diamond (100) surface. *J. Phys. Chem.*, 97:23–28, 1993.

R. J. Kee, M. E. Coltrin, and P. Glarborg. *Chemically Reacting Flow: Theory and Practice*. John Wiley and Sons, 2003.

R. J. Kee, G. Dixon-Lewis, J. Warnatz, , M. E. Coltrin, and J. A. Miller. A Fortran computer code package for the evaluation of gas-phase multicomponent transport properties. Technical Report SAND86-8246, Sandia National Laboratories, 1986.

R. J. Kee, F. M. Rupley, and J. A. Miller. Chemkin-II: A Fortran chemical kinetics package for the analysis of gas-phase chemical kinetics. Technical Report SAND89-8009, Sandia National Laboratories, 1989.

F. Lindemann. *Trans. Faraday Soc.*, 17:598, 1922.

Gregory P. Smith, David M. Golden, Michael Frenklach, Nigel W. Moriarty, Boris Eiteneer, Mikhail Goldenberg, C. Thomas Bowman, Ronald K. Hanson, Soonho Song, William C. Gardiner, Jr., Vitali V. Lissianski, , and Zhiwei Qin. GRI-Mech version 3.0, 1997. see `http://www.me.berkeley.edu/gri_mech.`

P. H. Stewart, C. W. Larson, and D. Golden. *Combustion and Flame*, 75:25, 1989.

H. Wang and M. Frenklach. *Chem. Phys. Lett.*, 205:271, 1993.

INDEX