WolframAlpha™ Webservice API Reference

API Version 2.0
Documentation Revision a

The Wolfram|Alpha Webservice API provides a web-based API allowing the computational and presentation capabilities of Wolfram|Alpha to be integrated into web, mobile, desktop, and enterprise applications.

The API allows clients to submit free-form queries similar to the queries one might enter at the Wolfram|Alpha website, and for the computed results to be returned in a variety of formats. The API is implemented in a standard REST protocol using HTTP GET requests. Each result is returned as a descriptive XML structure wrapping the requested content format.

Use of the Wolfram|Alpha Webservice API is subject to the API Terms of Use, which can be found at products.wolframalpha.com/api/termsofuse.html.

Obtaining an AppID

Visit the Wolfram|Alpha Webservice API website (products.wolframalpha.com/api) to register for a Wolfram|Alpha API account. An AppID is provided on completion of API account registration. An AppID is a string that identifies your application, and it must be supplied in all calls to the Wolfram|Alpha API. If you have multiple applications using the Wolfram|Alpha API, each must have its own AppID.

Basics of Wolfram|Alpha Output

To understand the workings of the API, it is necessary to know some details about Wolfram|Alpha output. Figure 1 is a screen shot of an example showing partial output for a user's entry of "tides Seattle", as seen on the Wolfram|Alpha site. The output is divided into rectangular regions called pods, each of which corresponds roughly to one category of result. The output pictured below has four pods. Each pod has a title ("Input interpretation" is the title of the first pod) and content, which is a GIF image. Pods may also have additional features such as a copyable plaintext representation that appears in a popup when you mouse over the image, and JavaScript buttons that replace the pod with different information in an AJAX-style operation (like the elements labeled "Pod States" in the output below).

The box at the top that starts with "Using nearest primary station" is not a pod, but rather a means to interact with Wolfram|Alpha's Assumptions facility. Assumptions are described in more detail later. The labels used in this picture are referred to later in the text, when these sections are introduced.

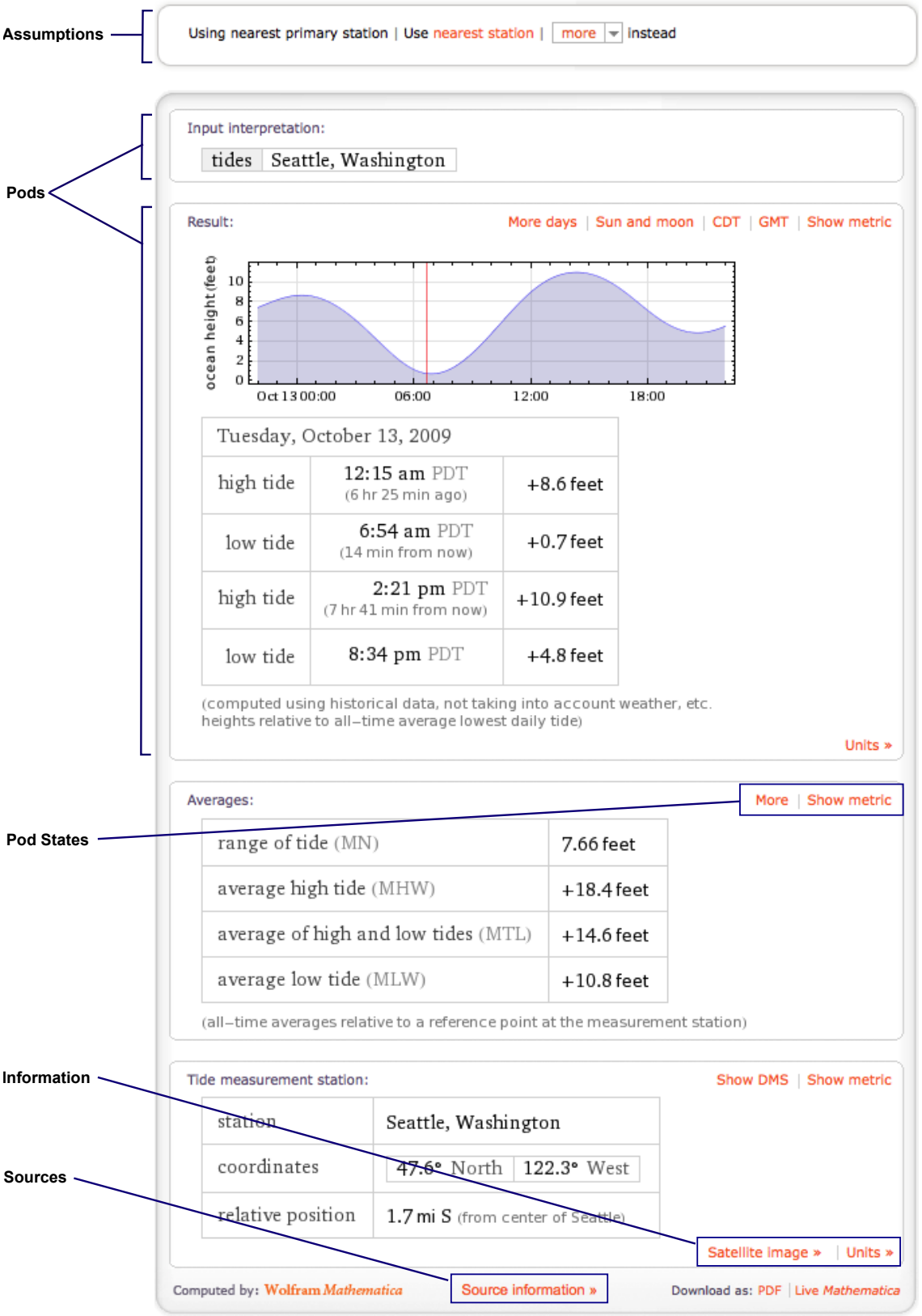

Figure 1. Annotated output from the Wolfram|Alpha website

Pods have subpods that enclose the actual content. Each of the pods in Figure 1 has only one subpod. Each subpod is a separate result and a separate image on the page. By convention, every pod has at least one subpod, so pods that appear to show only one result have that result in a subpod.

On the Wolfram|Alpha site, the content of each subpod is an image (even simple textual pods are GIF images, not text). Most results have alternative formats, such as various forms of textual representation. Users of the API can request any combination of these different types of representations.

Summary of Output Formats

The Wolfram|Alpha API provides a number of formats in which results can be returned. You can request any combination of formats. The result of a call to the API is always an XML document, with each pod and/or subpod represented in one or more of the following formats. More details on the format of the result XML are provided later; this is just a summary of the available format types.

Visual Representations

On the Wolfram|Alpha interactive website, results are displayed in the form of GIF images. This allows mathematical formulas, tables, and graphics to be formatted in a meaningful and attractive way. When using the Wolfram|Alpha API, you have two choices if you want such "pictures" of the output.

Image

The `image` format gives you the same types of GIF images as seen on the Wolfram|Alpha site. Each subpod is returned as an HTML `<img>` tag ready for direct inclusion in a web page. In some cases, JPEG images are returned instead of GIF. The filename in the `<img>` URL will tell you whether it is GIF or JPEG.

Mathematica Cell

The `cell` format type returns *Mathematica* Cell expressions. *Mathematica* is the computational engine that powers Wolfram|Alpha, and these cells are precisely what is rendered by Wolfram|Alpha to produce the pod images you see on the website or get using the `image` format type. The *Mathematica* environment has rich, direct integration with Wolfram|Alpha (using the API internally), but if for any reason you want to obtain these Cell expressions outside of *Mathematica*, you can request the `cell` format type.

Textual Representations

In some cases, you might not be satisfied with pictures of output, but will instead want some sort of structured textual representation of each subpod. You can then format it in a custom way, or pick it apart to extract only the desired piece. Not all results are available in all the formats listed below. For example, a plot of a mathematical function will have no plaintext representation, although it will have a *Mathematica* input representation.

Plaintext

This is the text format that you see in the "Copyable plaintext" popup that appears when you click results on the Wolfram|Alpha site. It represents a simple "best guess" for a meaningful readable textual form of a given result.

You will find it difficult to write general-purpose code to analyze text in this format, so use it only if you want simple text to display to your users, or if you know the structure of the text in advance (e.g., it will be a number, or a latitude-longitude pair, a table, etc.)

MathML

Some Wolfram|Alpha results are mathematical expressions or formulas that require traditional math notation to look good (superscripts, fractions, integral signs, etc.) Presentation MathML (www.w3.org/Math) is a W3C standard XML format for mathematics. Many browsers can render MathML, perhaps with the assistance of a plugin. MathML output is provided when you request the `mathml` format type.

Mathematica Input

This is the text format that you see in the "*Mathematica* plaintext input" popup that appears when you click some results on the Wolfram|Alpha site. Some results can be generated directly by single *Mathematica* input expressions. For example, the "Continued fraction" pod in the Wolfram|Alpha result for the query "pi" has a *Mathematica* input representation of `ContinuedFraction[Pi, 25]`. Use this form if you want to feed the input into *Mathematica* or use *Mathematica* as an environment for manipulating results. Use the `minput` format type to get results in this form.

Mathematica Output

This is the text format that you see in the "*Mathematica* plaintext output" popup that appears when you click some results on the Wolfram|Alpha site. This format is not available for all results, and it will sometimes be large (e.g., for mathematical plots), or not very useful (e.g., when the original source data is only available to *Mathematica* as a raster image, such as a country's flag). The first formula in the "Continued fraction" pod in the Wolfram|Alpha output for the query "pi" has a *Mathematica* output representation of {3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2, 1, 1}. Use this form if you want to feed the output into *Mathematica* or use *Mathematica* as an environment for manipulating results. Use the `moutput` format type to get results in this form.

Audio Representations

Some pods have sounds associated with them. Use the `sound` or `wav` format types to get the audio data.

Sample Query

A simple API call to retrieve output for the query "pi" would look like this:

```
http://api.wolframalpha.com/v2/query?input=pi&appid=XXXX
```

This query did not specify a desired output format, and the default is to retrieve the plaintext and image representations of each subpod. The image is returned as an `<img>` tag suitable for direct inclusion in a web page. Here is the output:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true' error='false' numpods='8'
    datatypes='MathematicalFunctionIdentity'' timedout='' timing='1.736'
```

```
    parsetiming='0.054' parsetimedout='false' recalculate='' version='2.0'>
  <pod title='Input' scanner='Identity' id='Input'
       position='100' error='false' numsubpods='1'>
    <subpod title=''>
      <plaintext>pi</plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP2311?MSPStoreType=image/gif'
          alt='pi' title='pi' width='9' height='20' />
    </subpod>
  </pod>
  <pod title='Decimal approximation' scanner='Numeric' id='DecimalApproximation'
       position='200' error='false' numsubpods='1'>
    <subpod title=''>
      <plaintext>3.14159265358979323846264338327950288419716939937510582097494...</plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP2321eh?MStoreType=image/gif'
           alt='3.14159265358979323846264338327950288419716939937510582097494...'
           title='3.14159265358979323846264338327950288419716939937510582097494...'
           width='491' height='20' />
    </subpod>
    <states count='1'>
      <state name='More digits' input='DecimalApproximation__More digits' />
    </states>
  </pod>
  <pod title='Property' scanner='Numeric' id='Property'
       position='300' error='false' numsubpods='1'>
    <subpod title=''>
      <plaintext>pi is a transcendental number</plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP233?MSPStoreType=image/gif'
           alt='pi is a transcendental number' title='pi is a transcendental number'
           width='194' height='20' />
    </subpod>
  </pod>
  <pod title='Number line' scanner='NumberLine' id='NumberLine'
       position='400' error='false' numsubpods='1'>
    <subpod title=''>
      <plaintext></plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP2b7?MSPStoreType=image/gif'
           alt='' title='' width='300' height='58' />
    </subpod>
  </pod>
  <pod title='Continued fraction' scanner='ContinuedFraction' id='ContinuedFraction'
       position='500' error='false' numsubpods='1'>
    <subpod title=''>
      <plaintext>[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2,
1, 1 ...]</plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP27h?MSPStoreType=image/gif'
           alt='[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2,
1, 1, ...]'
           title='[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84, 2,
1, 1, ...]'
           width='498' height='20' />
    </subpod>
    <states count='2'>
      <state name='Fraction form' input='ContinuedFraction__Fraction form' />
      <state name='More terms' input='ContinuedFraction__More terms' />
    </states>
  </pod>
  <pod title='Alternative representations' scanner='MathematicalFunctionData'
       id='AlternativeRepresentations:MathematicalFunctionIdentityData'
       position='600' error='false' numsubpods='3'>
    <subpod title=''>
      <plaintext>pi = 180 °</plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP2361?MSPStoreType=image/gif'
           alt='pi = 180 °' title='pi = 180 °' width='61' height='30' />
    </subpod>
    <subpod title=''>
      <plaintext>pi = -i log(-1)</plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP23b2?MSPStoreType=image/gif'
           alt='pi = -i log(-1)' title='pi = -i log(-1)' width='97' height='30' />
    </subpod>
    <subpod title=''>
```

```
      <plaintext>pi = cos^(-1)(-1)</plaintext>
      <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP2381?MSPStoreType=image/gif'
           alt='pi = cos^(-1)(-1)' title='pi = cos^(-1)(-1)' width='93' height='30' />
    </subpod>
    <states count='1'>
      <state name='More'
             input='AlternativeRepresentations:MathematicalFunctionIdentityData__More' />
    </states>
    <infos count='4'>
      <info text='log(x) is the natural logarithm'>
        <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP237?MSPStoreType=image/gif'
             alt='log(x) is the natural logarithm' title='log(x) is the natural logarithm'
             width='192' height='18' />
        <link url='http://reference.wolfram.com/mathematica/ref/Log.html'
              text='Documentation' title='Mathematica' />
        <link url='http://functions.wolfram.com/ElementaryFunctions/Log'
              text='Properties' title='Wolfram Functions Site' />
        <link url='http://mathworld.wolfram.com/NaturalLogarithm.html'
              text='Definition' title='MathWorld' />
      </info>
      <info text='i is the imaginary unit'>
        <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP26cf?MSPStoreType=image/gif'
             alt='i is the imaginary unit' title='i is the imaginary unit'
             width='137' height='18' />
        <link url='http://reference.wolfram.com/mathematica/ref/I.html'
              text='Documentation' title='Documentation' />
        <link url='http://mathworld.wolfram.com/i.html'
              text='Definition' title='MathWorld' />
      </info>
      <info text='cos^(-1)(x) is the inverse cosine function'>
        <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP24361?MSPStoreType=image/gif'
             alt='cos^(-1)(x) is the inverse cosine function'
             title='cos^(-1)(x) is the inverse cosine function'
             width='249' height='20' />
        <link url='http://reference.wolfram.com/mathematica/ref/ArcCos.html'
              text='Documentation' title='Mathematica' />
        <link url='http://functions.wolfram.com/ElementaryFunctions/ArcCos'
             text='Properties' title='Wolfram Functions Site' />
        <link url='http://mathworld.wolfram.com/InverseCosine.html'
              text='Definition' title='MathWorld' />
      </info>
      <info>
        <link url='http://functions.wolfram.com/Constants/Pi/27/ShowAll.html'
              text='More information' />
      </info>
    </infos>
  </pod>


  ... remaining pods deleted for brevity ...


<assumptions count='1'>
 <assumption type='Clash'
     word='pi'
     count='5'>
  <value name='NamedConstant'
      desc='a mathematical constant'
      input='*C.pi-_*NamedConstant-' />
  <value name='Character'
      desc='a character'
      input='*C.pi-_*Character-' />
  <value name='MathWorld'
      desc='referring to a definition'
      input='*C.pi-_*MathWorld-' />
  <value name='Movie'
      desc='a movie'
      input='*C.pi-_*Movie-' />
  <value name='Word'
      desc='a word'
```

```
        input='*C.pi-_*Word-' />
   </assumption>
  </assumptions>
 </queryresult>
```

Complete descriptions of the various query parameters and the result XML elements are given in later sections, but it can be seen that the output is a series of pods with subpods. The <assumptions> section at the end tells you what assumptions Wolfram|Alpha made about certain parts of the input, and what the alternative values are for each assumption. You could use this information to resubmit a query using a different assumption about the meaning of pi.

There are numerous parameters you can specify in the URL to control how the query is performed and what types of results it returns. These parameters and their uses are described in the relevant sections later in this document.

Special Types of Wolfram|Alpha Output

In addition to pods and their data, results from the Wolfram|Alpha API include several other types of information. Some of this extra information applies to the query as a whole, and is thus represented as top-level XML element in the body of the <queryresult> element. Other information is pod-specific and is therefore found as a subelement to the <pod> element.

Assumptions

Wolfram|Alpha makes numerous assumptions when analyzing a query and deciding how to present its results. A simple example is a word that can refer to multiple things, like "pi", which is a well-known mathematical constant but is also the name of a movie. Other classes of assumptions are the meaning of a unit abbreviation like "m", which could be meters or minutes, or the default value of a variable in a formula, or whether 12/13/2001 is a date or a computation. On the website, the space just above the first pod is used to describe assumptions used and give the user the option of changing them (see Figure 1 for an example). The API makes this same information available via the <assumptions> element, which, if present, is a top-level subelement of <queryresult>.

Here is the <assumptions> element in the "pi" query. Each assumption is represented as a separate <assumption> subelement. The <assumption> element has a type attribute that tells you what class of assumption it is. In this example, it is a "Clash" assumption, which is generated when a word in the query can refer to multiple different entities.

```
<?xml version="1.0" encoding="utf-8"?>
<queryresult success='true' error='false' numpods='7'
    datatypes='MathematicalFunctionIdentity' timedout='' timing='2.304'
    parsetiming='0.067' version='1.0.1'>

... pods deleted ...

    <assumptions count='1'>
        <assumption type='Clash' word='pi' count='4'>
            <value name='NamedConstant' desc='a mathematical constant'
                   input='*C.pi-_*NamedConstant-' />
            <value name='Character' desc='a character' input='*C.pi-_*Character-' />
            <value name='MathWorld' desc='referring to a definition'
                   input='*C.pi-_*MathWorld-' />
            <value name='Movie' desc='a movie' input='*C.pi-_*Movie-' />
            <value name='Word' desc='a word' input='*C.pi-_*Word-' />
        </assumption>
    </assumptions>
</queryresult>
```

All assumption types have the same basic structure: a sequence of <value> subelements, one for each possible value of the assumption. Each <value> element has three attributes: name, which is a unique internal identifier, but which will often have some descriptive value to the programmer; desc, which is a textual description suitable for displaying to users; and input, which gives the parameter value needed to invoke this assumption in a subsequent query (this is described in detail below). The first-listed <value> element always names the assumption value that was in effect for the current query.

Here is another example. The query is "12/5/1999", which Wolfram|Alpha interprets as a date, but it's ambiguous whether this is written in the order month/day/year or day/month/year, so an assumption is generated. Here is the relevant <assumption> element, which has type "DateOrder". The month/day/year value is listed first, which means that it is the value that was used:

```
  <assumption type='DateOrder' count='2'>
   <value name='MonthDayYear' desc='month/day/year' input='DateOrder_**Month.Day.Year--' />
   <value name='DayMonthYear' desc='day/month/year' input='DateOrder_**Day.Month.Year--' />
  </assumption>
```

The Reference Guide section at the end of this document includes a complete listing of all the assumption types that can be generated, with examples.

Applying an Assumption

To apply an assumption in a query, use the assumption parameter. The value you pass for this parameter is the string found in the input attribute of a <value> element returned from a previous query. Here is how to invoke the query "pi" but specify that you want pi treated as the name of a movie. The obscure-looking assumption string here was taken from the earlier <assumptions> output for this query.

```
http://api.wolframalpha.com/v2/query?input=pi&appid=XXXX&assumption=*C.pi-_*Movie-
```

Here is how you would modify the "12/5/1999" query to change the date order to day/month/year:

```
http://api.wolframalpha.com/v2/query?input=12%2F5%2F1999&appid=XXXX&assumption=DateOrder_**Da
y.Month.Year--
```

The values for the `assumption` parameter are complicated-looking strings, but you don't have to understand their syntax—they are just tokens that you are given in the `<assumptions>` output of a previous query.

You can apply more than one assumption in a given query by including multiple `assumption=value` specifications in the URL.

Formulas with Input Fields

Some Wolfram|Alpha inputs are interpreted as referring to mathematical formulas. In such cases, the Wolfram|Alpha website provides a user interface for controlling aspects of the formula, such as what variable is being solved for and what values the variables should be given. Formula solving is part of the Wolfram|Alpha Assumptions facility, and the API gives you complete control over all aspects of formula manipulation.

Figure 2 is a screen shot from the website for the query "Doppler shift." The orange box at the top is part of the standard Assumptions section, and this particular example shows all the formula-related assumptions that Wolfram|Alpha can produce.

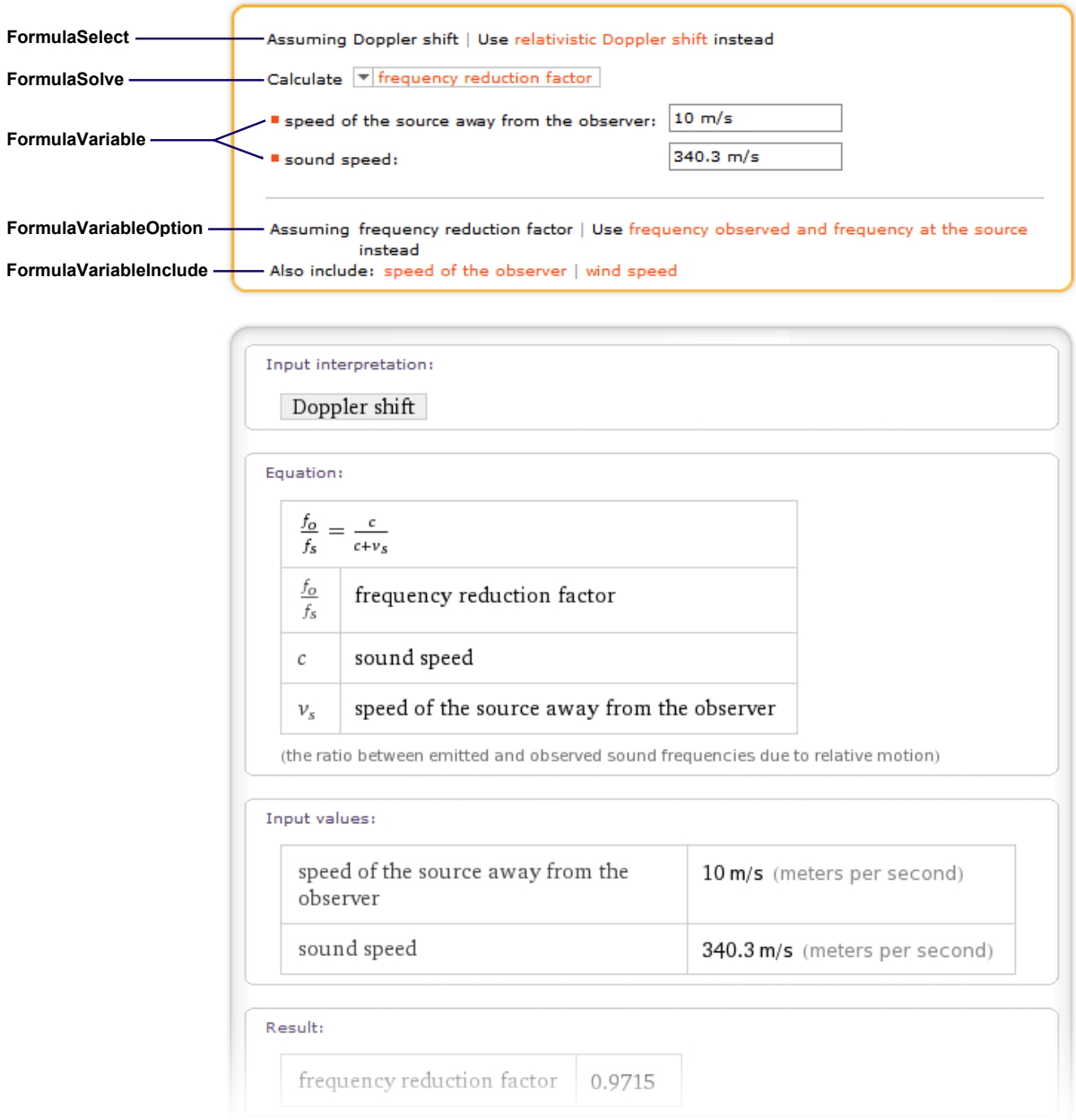

Figure 2. Interactive formula on the website

The labels on Figure 2 show the names the API uses for the types of assumptions involved in formulas. Here is a guide to the five different types. Not all formulas will have all of these assumptions available.

FormulaSelect

Some queries have more than one formula that applies. The `FormulaSelect` assumption allows you to choose the one you want. In this Doppler example, you can choose the classical Doppler shift formula (the default) or the relativistic one.

FormulaSolve

Formulas can be rearranged to solve for different variables. The `FormulaSelect` assumption lets you pick which one you want. In this example, the variables are the frequency reduction factor (f_o/f_s, treated as a single

entity), the speed of sound (c), and the speed of the source (v_s). Notice in the Result pod it shows a value for frequency reduction factor, which is the current choice for the variable to solve for. If you were to choose a different variable to solve for, it would show that value in this pod.

FormulaVariable

The `FormulaVariable` assumption lets you supply a value for a variable in a formula. It corresponds to an input field or pulldown menu of choices on the website.

FormulaVariableInclude

The `FormulaVariableInclude` assumption lets you add additional variables into a formula. For simplicity, Wolfram|Alpha presents the Doppler shift formula with a small number of variables, but it knows how to include two more: the speed of the observer and the wind speed. On the website, if you click to add one of these variables, the formula will change to include this variable, the tabular results will get an extra row for it, and you will get an extra input field to enter its value.

FormulaVariableOption

Wolfram|Alpha can sometimes present the same basic formula in terms of a different set of variables. In the Doppler example, you can choose to have the frequency reduction factor (f_o/f_s) broken up into two separate variables (f_o and f_s). You're not substituting a completely different formula (like `FormulaSelect`) or simply adding a new variable (like `FormulaVariableInclude`).

XML Format of Formula-Related Assumptions

Here is the `<assumptions>` section from the API output for the "Doppler shift" query:

```
<assumptions count='7'>
  <assumption type='Clash' word='doppler shift' count='2'>
    <value name='Formula' desc='a formula' input='*C.doppler+shift-_*Formula-' />
    <value name='Word' desc='a word' input='*C.doppler+shift-_*Word-' />
  </assumption>
  <assumption type='FormulaSolve' count='3'>
    <value name='DopplerShift.DopplerRatio'
           desc='frequency reduction factor'
           input='*FS-_**DopplerShift.DopplerRatio--' />
    <value name='DopplerShift.vs'
           desc='speed of the source away from the observer'
           input='*FS-_**DopplerShift.vs--' />
    <value name='DopplerShift.c'
           desc='sound speed'
           input='*FS-_**DopplerShift.c--' />
  </assumption>
  <assumption type='FormulaSelect' count='2'>
    <value name='DopplerShift'
           desc='Doppler shift'
           input='FSelect_**DopplerShift--' />
    <value name='RelativisticDopplerShift'
           desc='relativistic Doppler shift'
           input='FSelect_**RelativisticDopplerShift--' />
  </assumption>
  <assumption type='FormulaVariable'
           desc='speed of the source away from the observer'
           current='1' count='1'>
    <value name='DopplerShift.vs'
           desc='10 m/s' valid='true' input='*F.DopplerShift.vs-_ 10+m%2Fs' />
  </assumption>
  <assumption type='FormulaVariable'
           desc='sound speed' current='1' count='1'>
    <value name='DopplerShift.c'
           desc='340.3 m/s' valid='true' input='*F.DopplerShift.c-_ 340.3+m%2Fs' />
  </assumption>
  <assumption type='FormulaVariableOption' count='2'>
    <value name='DopplerShift.DopplerRatio'
           desc='frequency reduction factor'
           input='*FVarOpt-_**DopplerShift.DopplerRatio--' />
    <value name='DopplerShift.fo,DopplerShift.fs'
           desc='frequency observed and frequency at the source'
           input='*FVarOpt-_**DopplerShift.fo-.*DopplerShift.fs--' />
  </assumption>
  <assumption type='FormulaVariableInclude' count='2'>
    <value name='DopplerShift.vo'
           desc='speed of the observer'
           input='*FVarOpt-_**DopplerShift.vo-.*DopplerShift.DopplerRatio--' />
    <value name='DopplerShift.vw'
           desc='wind speed'
           input='*FVarOpt-_**DopplerShift.vw-.*DopplerShift.DopplerRatio--' />
  </assumption>
</assumptions>
```

Most of this output should be self-explanatory in the context of the earlier discussion of assumptions in general. Using the information in this output, you could build a user interface that allows your users to interact with the Doppler formula in exactly the same way as the Wolfram|Alpha website.

Applying Formula Assumptions

You apply formula-related assumptions just like other types of assumptions—by using the value of an `<assmuption>` element's input attribute in a subsequent query. For example, to perform the Doppler shift query with wind speed as an added variable, you would use this URL:

```
http://api.wolframalpha.com/v2/query?input=Doppler+shift&appid=XXXX&assumption=*FVarOpt-
_**DopplerShift.vw-.*DopplerShift.DopplerRatio--
```

You can specify as many assumptions as you want in a single query, which is often necessary when working with formulas (such as setting the value of multiple variables). To do this, include multiple `assumption=value` specifications in the URL.

Setting Values for Variables

You have to understand a few extra details when setting a value for a variable, which corresponds to applying a `FormulaVariable` assumption. Let's take a closer look at the `<assumption>` element that deals with the value of the speed of the source:

```
<assumption type='FormulaVariable'
       desc='speed of the source away from the observer' current='1' count='1'>
  <value name='DopplerShift.vs' desc='10 m/s' valid='true'
         input='*F.DopplerShift.vs-_10+m%2Fs' />
</assumption>
```

When you see such an assumption in output, you might choose to provide your users with an input field or other means to specify a value. The label for this input field would be the `desc` attribute of the `<assumptiontype>` element. The `count` attribute gives the number of `<value>` elements. For variables that take an arbitrary value, typically entered via an input field, the count will always be 1, but for variables that take one of a fixed set of values, typically represented as a pulldown menu of choices, the count will be the number of possible choices, with one `<value>` element for each possibility. For the moment, we restrict our attention to the common case of a variable that can take any user-specified value. The single `<value>` element for this assumption will have a `desc` attribute that gives the currently assumed value for the variable. If you were providing an input field for your users to specify a new value, you would probably want to "prime" the input field with this initial value, like the Wolfram|Alpha website does. To specify a different value, you need to work with the value of the `input` attribute. You can ignore everything that comes before the “-_” in the string. What comes after is the value—in this case, "10+m%2Fs". That value is of course the URL-encoded form of "10 m/s". To specify a different value for this variable, replace what comes after the “-_” character pair with the URL-encoded new value. Here is how to set the speed of the source to 6.5 m/s:

```
http://api.wolframalpha.com/v2/query?input=Doppler+shift&appid=XXXX&assumption=*F.DopplerShif
t.vs-_6.5+m%2Fs
```

Wolfram|Alpha intelligently parses the value, so it understands that "m/s" means "meters per second." Those are the default units for this value, so you could leave the units specification out entirely and just give a number. You could also write out "meters/second", "meters per second", etc. If you submit a value that cannot be understood by Wolfram|Alpha, then the corresponding `<value>` element in the result will have the `valid=false` attribute. For example, if you try to set the value to 6.5 f/s, then this is what the resulting `<assumption>` element looks like:

```
<assumption type='FormulaVariable'
       desc='speed of the source away from the observer' current='1' count='1'>
  <value name='DopplerShift.vs' desc='6.5 f/s' valid='false'
         input='*F.DopplerShift.vs-_ 6.5+f%2Fs' />
</assumption>
```

The problem is that Wolfram|Alpha does not understand the "f/s" units. To specify feet per second, you would need to use "ft/s", "feet/second", etc.

Now consider the case where a formula variable assumption can take on only a fixed set of values. These are not common in Wolfram|Alpha, but an example query that generates such assumptions is "heart disease", which on the website produces several pulldown menus allowing you to choose whether the person in question is a smoker, diabetic, etc. Here is the XML returned by the API for two of these assumptions:

```
<assumption type='FormulaVariable' desc='gender' current='1' count='2'>
  <value name='Gender:Male' desc='male'
      valid='true' input='*FP.HeartDisease.gender-_Gender%3AMale' />
  <value name='Gender:Female' desc='female'
      valid='true' input='*FP.HeartDisease.gender-_Gender%3AFemale' />
</assumption>
<assumption type='FormulaVariable' desc='smoker' current='2' count='2'>
  <value name='SmokingStatus:Yes' desc='yes'
      valid='true' input='*FP.HeartDisease.smoking-_SmokingStatus%3AYes' />
  <value name='SmokingStatus:No' desc='no'
      valid='true' input='*FP.HeartDisease.smoking-_SmokingStatus%3ANo' />
</assumption>
```

In all other assumption types, the first-listed <value> element names the currently active assumption, but in this one case that rule is violated. Instead, the current attribute gives the index of the <value> element that is currently active. In this way, the natural order of the different values is preserved, without artificially moving the current value to the top of the list. You would probably choose to preserve the order of the <value> elements in your pulldown menu. In the case of only a small number of choices, you might also choose to represent them as radio buttons.

Pod States

Many pods on the Wolfram|Alpha website have text buttons in their upper-right corners that substitute the contents of that pod with a modified version. In Figure 1, the Result pod has buttons titled "More days", "Sun and Moon", CDT", "GMT", and "Show metric". Clicking any of these buttons will recompute just that one pod to display different information. The "More days" button would, of course, regenerate the plot for a longer span of days. You can think of these buttons as switching the pod into a different state (like "metric units"). The API returns information about these pod states and allows you to programmatically invoke them.

A simple example is the query "pi", which returns a pod titled "Decimal approximation" with a button named "More digits". A website user can click this button to replace the pod with a new one showing more digits of pi. Here is what that pod looks like in the API result:

```
<pod title='Decimal approximation' scanner='Numeric' position='200'
      id='DecimalApproximation'
      error='false' numsubpods='1'>
   <subpod title=''>
      <plaintext>3.14159265358979323846264338327950288419716939937510582097494...</plaintext>
   </subpod>
   <states count='1'>
      <state name='More digits' input='Decimal Approximation__More digits'/>
   </states>
</pod>
```

The <states> element has all the alternative states available for that pod. The name of the state is the same as the text that appears on the website. You can perform the pi query and ask that the "More digits" state be invoked

automatically by using the podstate parameter, passing the value of the input attribute from the corresponding <state> element:

```
http://api.wolframalpha.com/v2/query?input=pi&appid=XXXX&podstate=DecimalApproximation__More+
digits
```

The result from the above query will be exactly as if a website user had clicked the "More digits" button.

In previous versions of the API, there was no input attribute in the <state> element, and the value passed for the podstate parameter was just the name of the state (e.g., "More digits"). This is still supported, but it has the drawback that state changes cannot be invoked on a per-pod basis, so any pod with a "More digits" state would have it invoked, not just the "Decimal approximation" pod. Using the value of the input attribute will restrict the state change to a specific pod, and is the preferred form.

Another detail to note is that the value of the input attribute in the <state> element is not URL-encoded, so you will have to perform this encoding yourself when you use it. This is in contrast to the input attribute in the <value> element for assumptions, which is URL-encoded before it is returned by the API.

State changes can be chained together to simulate any sequence of button clicks. After clicking the "More digits" button, the "Decimal approximation" pod comes back with another "More digits" button to show even more. You can simulate clicking this twice as follows:

```
http://api.wolframalpha.com/v2/query?input=pi&appid=XXXX&podstate=DecimalApproximation__More+
digits&podstate=DecimalApproximation__More+digits
```

When invoking the same podstate change multiple times, as in the above query, you can use a shortcut of specifying only one podstate parameter and indicating the multiplicity by prepending “n@”. For example, the following query invokes the “More digits” state twice, exactly like the previous example:

```
http://api.wolframalpha.com/v2/query?input=pi&appid=XXXX&podstate=2@DecimalApproximation__Mor
e+digits
```

Some states are logically grouped into sets, which are represented on the website as popup menus. For example, the "Weather history and forecast" pod from the query "weather" shows some charts along with a popup menu that controls the time period (it has values like "Current week", "Last month", etc.) This type of state control is represented with the <statelist> element. Here is the <states> element in the API result for that pod. Note that this pod has two other button-type states ("Show metric" and "More").

```
<pod title='Weather history &amp; forecast' scanner='Data'
    id='WeatherCharts:WeatherData'
    position='300' error='false' numsubpods='3'>

   ... pod content deleted ...

  <states count='3'>
    <state name='Show metric' input='WeatherCharts:WeatherData__Show metric' />
    <state name='More' input='WeatherCharts:WeatherData__More' />
    <statelist count='9' value='Current week'>
      <state name='Current week' input='WeatherCharts:WeatherData__Current week' />
      <state name='Current day' input='WeatherCharts:WeatherData__Current day' />
      <state name='Next week' input='WeatherCharts:WeatherData__Next week' />
      <state name='Past week' input='WeatherCharts:WeatherData__Past week' />
      <state name='Past month' input='WeatherCharts:WeatherData__Past month' />
      <state name='Past year' input='WeatherCharts:WeatherData__Past year' />
      <state name='Past 5 years' input='WeatherCharts:WeatherData__Past 5 years' />
      <state name='Past 10 years' input='WeatherCharts:WeatherData__Past 10 years' />
      <state name='All' input='WeatherCharts:WeatherData__All' />
    </statelist>
   </states>
 </pod>
```

The `value` attribute of the `<statelist>` element names the state that is currently in effect. To request that this pod show data for the last five years, you would use this URL:

```
http://api.wolframalpha.com/v2/query?input=weather&appid=XXXX&podstate=WeatherCharts:WeatherD
ata__Past+5+years
```

Subpod States

Most podstate changes are for entire pods, but a few Wolfram|Alpha queries have podstate changes at the subpod level. An example of one such query is “inductance of a circular coil”. On the website, the third pod is titled “Self-inductance of a single-layer circular coil”, and it has two subpods. The second one is called “Result”, and it has a podstate button called “Show formula”. You can see by the position and behavior of this button that it only modifies the Result subpod, not the entire pod.

In the API, such subpods have a `<states>` element of their own. Here is the XML for the third pod:

```
 <pod title='Self-inductance of a single-layer circular coil'
      scanner='Formula'
      id='Self-inductanceOfASingle-layerCircularCoil'
      position='300' error='false' numsubpods='2'>
   <subpod title='Input values'>
     <plaintext>coil turns | 10
coil radius | 1 cm  (centimeter)
coil length | 20 cm  (centimeters)
     </plaintext>
   </subpod>
   <subpod title='Result'>
     <plaintext>self-inductance | 189.3 nH  (nanohenries)
0.1893 muH  (microhenries)
1.893◊10^-7 H  (henries)
     </plaintext>
     <states count='1'>
       <state name='Show formula'
           input='Self-inductanceOfASingle-layerCircularCoil__Result _Show formula' />
     </states>
   </subpod>
 </pod>
```

You can see that the <states> element is a subelement of <subpod>, not <pod>. To invoke this podstate change, you use it like any other podstate:

```
http://api.wolframalpha.com/v2/query?input=weather&appid=XXXX&podstate=Self-
inductanceOfASingle-layerCircularCoil__Result _Show+formula
```

The result would have a third pod in which only the Result subpod was modified compared to the original query.

Information

Some pods on the Wolfram|Alpha website have text buttons in their lower-right corners that provide extra information about the contents of that pod. In Figure 1, the "Tide measurement" pod has buttons titled "Satellite image" and "Units". Clicking these types of buttons will cause a popup window to appear, or perhaps a new browser window pointing at a site with more information. The data for these "information" links is available in the API via the <infos> element, which appears inside any <pod> elements for which information links are available. Here is the <infos> element for the tide query from Figure 1:

```
<pod title='Tide measurement station' scanner='Tide' position='400'
     error='false' numsubpods='1'>

   ... other content deleted ...

  <infos count='2'>
   <info>
    <link url='http://maps.google.com/maps?t=h&amp;ie=UTF8&amp;ll=47.6017,-122.338&amp;z=17'
        text='Satellite image' />
   </info>
   <info>
    <units count='1'>
     <unit short='mi' long='miles' />
     <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP11d23?MSStoreType=image/gif' />
    </units>
   </info>
  </infos>
 </pod>
```

The content of the <infos> element is always one or more <info> elements. Each <info> element represents one piece of information about the contents of the pod. In some cases, these bits of information are formatted as separate lines on the website, and in other cases they are placed together on a single line separated by a vertical bar, as is the case in Figure 1.

Here is another example of an <infos> element. This is the "Alternative representations" pod for the "pi" query. If you do that query on the website, you will see four info lines for that pod, the first three of which are text followed by several links (on the website you have to roll the mouse over the text to see these links).

```
<infos count='4'>
  <info text='E (m) is the complete elliptic integral of the second kind'>
    <img
src='http://www1.wolframalpha.com/Calculate/MSP/MSP316197h7fe195?MSStoreType=image/gif'
        alt='E (m) is the complete elliptic integral of the second kind'
        title='E (m) is the complete elliptic integral of the second kind'
        width='350' height='18' />
    <link url='http://reference.wolfram.com/mathematica/ref/EllipticE.html'
        text='Documentation' title='Mathematica' />
    <link url='http://functions.wolfram.com/EllipticIntegrals/EllipticE'
        text='Properties' title='Wolfram Functions Site' />
    <link url='http://mathworld.wolfram.com/CompleteEllipticIntegraloftheSecondKind.html'
        text='Definition' title='MathWorld' />
  </info>
  <info text='i is the imaginary unit'>
    <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP3171977d?MSPStoreType=image/gif'
        alt='i is the imaginary unit'
        title='i is the imaginary unit'
        width='137' height='18' />
    <link url='http://reference.wolfram.com/mathematica/ref/I.html'
        text='Documentation' title='Documentation' />
    <link url='http://mathworld.wolfram.com/i.html'
        text='Definition' title='MathWorld' />
  </info>
  <info text='log (x) is the natural logarithm'>
    <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP3181ac6?MSPStoreType=image/gif'
        alt='log (x) is the natural logarithm'
        title='log (x) is the natural logarithm'
        width='192' height='18' />
    <link url='http://reference.wolfram.com/mathematica/ref/Log.html'
        text='Documentation' title='Mathematica' />
    <link url='http://functions.wolfram.com/ElementaryFunctions/Log'
        text='Properties' title='Wolfram Functions Site' />
    <link url='http://mathworld.wolfram.com/NaturalLogarithm.html'
        text='Definition' title='MathWorld' />
  </info>
  <info>
    <link url='http://functions.wolfram.com/Constants/Pi/27/ShowAll.html'
        text='More information' />
  </info>
</infos>
```

The content on the `<info>` element varies. In some cases, there is a `text` attribute that gives a string of textual information that is not intended to be a link in itself. The possible subelements are:

<link>

Used when the item is just a standard link of some text pointing to a URL, like in the "Satellite image" link in the result above, or the common "More information" link pointing to a *MathWorld* (mathworld.wolfram.com) page. The `<link>` element contains a `url` attribute giving the URL containing the referenced information, a `text` attribute giving a textual name for the link (the website uses this as the text of the link itself), and in some cases a `title` attribute (used by the website as a popup when the mouse hovers over the link).

<units>

Used when the link is a popup window showing a table of unit abbreviations used in the pod alongside their long names. The tides example above shows an unusual case where the table has only one row. Each row is represented as a `<unit>` element giving the short and long names of the unit. Wolfram|Alpha creates nicely rendered images of these units tables, so there is also an `<img>` element that points to the URL for the image of the entire table if you want to show your users a picture.

Some info lines contain descriptive text along with one or more links. The text is provided in the `text` attribute of the `<info>` element, but often the text contains a mathematical symbol or nomenclature that looks best when displayed as a typeset image. The `<img>` element contains a link to an image of the text. This is the same image that is displayed on the website.

Warnings

Wolfram|Alpha can return warnings for some circumstances. These generally correspond to situations where Wolfram|Alpha interprets your input as being different than what was actually entered. Warnings on the website appear at the top, above all other output. In the API, they come back as a `<warnings>` element. At the moment, there are only four possible warning types, listed below. Each one is represented by its own subelement of `<warnings>`. More warning subelements are likely to be added in the future, so make sure that your code will not break if new, unrecognized elements are added.

<spellcheck>

If you enter "chicag" as a query, Wolfram|Alpha assumes you meant "chicago." On the website, it says “Interpreting 'chicag' as 'chicago'". Here is the relevant part of the API result:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true' error='false' numpods='12'
    datatypes='City,MetropolitanArea,UrbanArea,USCounty,USState'
    timedout='' timing='6.464' parsetiming='0.141' parsetimedout='false'
    recalculate='' version='2.0'>

  .. pods and other content deleted ...

  <warnings count='1'>
    <spellcheck word='chicag'
        suggestion='chicago'
        text='Interpreting &amp;quot;chicag&amp;quot; as &amp;quot;chicago&amp;quot;' />
  </warnings>
</queryresult>
```

The `<spellcheck>` element has `word` and `suggestion` attributes, which give the actual word entered and the replacement word, respectively, and a `text` attribute, which gives the same string you see on the website.

<delimiters>

If you enter a query with mismatched delimiters like "sin(x", Wolfram|Alpha attempts to fix the problem and reports this as a warning. On the website, it says something like "An attempt was made to fix mismatched delimiters". Here is the `<warnings>` element in the API result:

```
 <warnings count='1'>
  <delimiters text='An attempt was made to fix mismatched delimiters' />
 </warnings>
```

<translation>

Wolfram|Alpha will translate some queries from non-English languages into English. In some cases when it does this, you will get a `<translation>` element in the API result. This example is from the query “wetter heute”,

which is translated into “weather today”:

```
<warnings count='1'>
  <translation phrase='wetter heute'
      trans='weather today'
      lang='German'
      text='Translating from German to &quot;weather today&quot;' />
</warnings>
```

The <translation> element has phrase, trans, lang, and text attributes.

Whether Wolfram|Alpha automatically translates a phrase into English depends on some special URL parameters, and this behavior is likely to change over time. At present, the only way to see the <translation> warning is to turn on automatic translation with the translation=true URL parameter, as in:

```
http://api.wolframalpha.com/v2/query?input=wetter+heute&appid=XXXX&translation=true
```

<reinterpret>

Wolfram|Alpha can automatically try to reinterpret a query that it does not understand but that seems close to one that it can. This behavior drastically reduces the number of failed queries, at the cost of potentially giving the user a result that is far from the original intent. For example, if you try the query “bear shoe” on the website, it will reinterpret this nonsensical query as simply “bear”, and provide a message in the warnings section at the top of the page saying “Using closest Wolfram|Alpha interpretation: bear”.

On the website, this "reinterpretation" behavior is the default, but in the API it is not, for reasons of backward compatibility. If you want this behavior, and most clients probably do, then you must enable it using the reinterpret=true URL parameter:

```
http://api.wolframalpha.com/v2/query?input=bear+shoe&appid=XXXX&reinterpret=true
```

When Wolfram|Alpha performs a reinterpretation, it reports this using a <reinterpret> warning element. Here is an example:

```
<warnings count='1'>
  <reinterpret text='Using closest Wolfram|Alpha interpretation:' new='bear' />
</warnings>
```

This element alerts you that the original query has been modified, with the new interpretation being given in the new attribute. In some cases, additional suggested interpretations are provided as <alternative> subelements to the <reinterpret> element. Here is the <reinterpret> element from the query “blue mustang moon”:

```
<reinterpret text='Using closest Wolfram|Alpha interpretation:' new='blue moon'>
  <alternative>blue</alternative>
  <alternative>moon</alternative>
</reinterpret>
```

Sources

In Figure 1 you can see at the bottom of the output area a link titled "Source information". On the website this displays a popup window listing information about sources for the data used in computing this result. This information is provided in the API via the <sources> element. Here is an example for the tides query in Figure 1:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true' error='false' numpods='4' datatypes='City,Tide'
    timedout='' timing='1.003' parsetiming='0.148'>

.. pods and other content deleted ...

 <sources count='1'>
  <source url='http://www.wolframalpha.com/sources/CityDataSourceInformationNotes.html'
      text='City data' />
 </sources>
</queryresult>
```

The <sources> element contains a series of <source> subelements, each one defining a link to a web page of source information.

Source information is not always present, such as for a purely mathematical computation.

Sounds

Some queries generate sounds as part of their output. For example, if you try the query "C major" on the website you will find that the pod title "Music notation" has a “Play sound” button in the upper right. To get sounds in the API, include "sound" in the format specification:

```
http://api.wolframalpha.com/v2/query?input=C+major&appid=XXXX&podtitle=Music+notation&format=
image,sound
```

Note that for brevity we have restricted the output to the "Music notation" pod. Here is the result:

```
 <pod title='Music notation' scanner='Music'
    id='ScaleNotation’ position='200' error='false'
    numsubpods='1'>
  <sounds count='1'>
   <sound url='http://www1.wolframalpha.com/Calculate/MSP/MSP251c1a8SPStoreType=audio/midi'
      type='audio/midi' />
  </sounds>
  <subpod title=''>
   <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP271f0gha?MSPStoreType=image/gif'
      alt='' title='' width='336' height='53' />
  </subpod>
 </pod>
```

Any pods that have an associated sound will have a <sounds> element at the top level. The <sounds> element contains a sequence of <sound> elements, one for each sound present. The <sound> element includes a URL to retrieve the sound file, and the MIME type so you know what type of data it holds. The sound types that can be returned are "audio/midi" (MIDI format) and "audio/x-wav" (WAV format), depending on the query.

If your program can only handle WAV sounds, not MIDI, then instead of using the “sound” format specification, you can use “wav”, which will force all sounds to be converted into WAV format.

Generalizations

For some types of queries, Wolfram|Alpha decides that although it can provide some results for the precise query that was given, there is a "generalization" of the query for which more information can be provided. In these cases, it reports the suggested generalized query via the <generalization> element. Queries that produce this element are relatively rare. An example is "price of copernicium":

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true' error='false' numpods='2' datatypes='Element'
     timedout='' timing='1.103' parsetiming='0.607' parsetimedout='false'
     recalculate='' version='2.0'>
  <pod title='Input interpretation' scanner='Identity'
      id='Input' position='100' error='false' numsubpods='1'>

      ... pod content deleted for brevity

  </pod>
  <pod title='Commodity price' scanner='Data' id='Result'
      position='200' error='false' numsubpods='1' primary='true'>

      ... pod content deleted for brevity

  </pod>
  <generalization topic='copernicium'
      desc='General results for:'
      url='http://www1.wolframalpha.com/api/v1/query?id=MSP174219edd&amp;s=2' />
</queryresult>
```

The query returns two small pods, but suggests the generalization "copernicium", about which much more information is available. You might choose to display the extra results available for that query, or give users a chance to click something to request them. If you try this query on the website, you will see that it displays the two pods available for "price of copernicium", then a separator that reads "General results for: copernicium", followed by the pods generated by that second query. If you want to implement similar behavior, you can get the pods from the generalized query by calling the URL given in the `url` attribute. Note that you must append your own AppID to this URL. What comes back is a standard <queryresult> element, like from a normal query. Using the URL provided is faster than simply issuing a new query for "price of copernicium", since the work of parsing the query has already been performed.

Special Topics

Controlling Which Pods Are Returned

It is often the case that applications using the Wolfram|Alpha API are not interested in the complete set of pods returned for a given query. For example, if you are writing an application that wants to acquire weather information from Wolfram|Alpha, and you are only interested in the pod that displays the table of current weather conditions, you need a way to ask the API to return only that pod. Because Wolfram|Alpha has much less work to do to generate only that one pod, this query will execute much more quickly than getting all the pods and picking out the one you want yourself.

The API provides three parameters that you can use to control the set of pods returned for a given query. They are described in the following sections. These parameters can be used together in the same query, in which case

a pod will be included in the result if and only if it meets any of the included specifications (in other words, the set of pods returned is the union of those that meet any of the criteria, not the intersection).

Specifying Pods by Their ID

The most robust way to identify a pod is by its ID, which is given by its `id` attribute. To understand the usefulness of the ID, consider if it did not exist, and the only way to identify pods was by their titles. A problem with titles is that they can change in consecutive runs of the same query. This can happen if the title includes the current date or time, such as from a weather query. Titles can also contain some information from the original query, like the name of a city, so if you have an application that returns, say, tide information for user-specified locations, the pods that you want to display might have different titles for each query. Finally, pod titles are likely to undergo changes in future updates to Wolfram|Alpha, so applications that depend on the exact wording of titles will be fragile. What most programmers want is a way to identify a pod by its general content, not the precise wording of its title. This is what the ID provides.

You can choose to include or exclude pods based on their IDs. The URL parameters for these actions are `includepodid` and `excludepodid`. Let's say that you have a weather application and you want to display the pod that gives current weather information for a given location. Here is what that pod looks like for the query "weather chicago":

```
 <pod title='Latest recorded weather for Chicago, Illinois' scanner='Data'
      id='InstantaneousWeather:WeatherData'
      position='200' error='false' numsubpods='1'>
   <subpod title=''>
     <plaintext>temperature | 25 °F  (wind chill: 20 °F)
conditions | snow, fog, overcast
relative humidity | 86%  (dew point: 21 °F)
wind speed | 3 mph
(34 minutes ago)
     </plaintext>
  </subpod>

  ... elements deleted for brevity

 </pod>
```

The title includes the name of the city, but the ID is a logical description of the contents of the pod. If you wanted Wolfram|Alpha to return only this one pod, you would use the `includepodid` parameter like this:

```
http://api.wolframalpha.com/v2/query?input=weather+chicago&appid=XXXX&includepodid=Instantane
ousWeather:WeatherData
```

Similarly, if you wanted to include all the pods except for this one, you would use the `excludepodid` parameter. You can include or exclude multiple pods by specifying more than one instance of either parameter. However, you cannot mix `includepodid` and `excludepodid` in one query, as this is not a meaningful operation.

The Podtitle Parameter

You can also choose identify pods by their titles. The title is the text that appears in the upper left of the pod, followed by a colon. For example, in Figure 1 the titles of the pods are "Input interpretation", "Results", "Averages", and "Tide measurement station". Note that the trailing colon is not part of the title. The `podtitle` attribute allows you to specify one or more pod titles to include; any pods that do not have matching titles will be excluded. You can specify a specific title or match a class of titles by using "*" as a wildcard that matches zero or

more characters. To specify more than one title, use multiple `podtitle` parameters. For example, this query requests two pods: the one titled "Weather station information", and the one that begins with "Latest recorded weather" (we use a wildcard to match the rest of the title because it ends with the current date, which will change).

```
http://api.wolframalpha.com/v2/query?input=weather&appid=XXXX&podtitle=Weather+station+inform
ation&podtitle=Latest+recorded+weather*
```

The Podindex Parameter

Another way to specify pods is by their index. This is a 1-based count representing the order in which they were computed by Wolfram|Alpha, and it corresponds to the top-to-bottom ordering of pods on the website and in the XML returned by the API. You can specify a single pod or several pods. For example, `podindex=2` requests only the second pod, and `podindex=1,2,4` requests those three specific pods.

Pod indices are not a very stable way to identify specific pods, as ordering of pods in Wolfram|Alpha can change over time as new pod types are created or existing ones are promoted or demoted in importance. The first pod gives the input interpretation, but after that the output is likely to change over time. Using pod indices is good when you just want to say something like “give me whatever Wolfram|Alpha thinks are the five most important pods”.

If you are using the `podtimeout` parameter (described elsewhere), it is important to note that the index numbers are decided at the close of the "scan" stage. The culling of pods based on index is done before the later format stage, so pods that are dropped because they time out in formatting have no bearing on the index numbers.

The Scanner Parameter

Each pod is produced by a software component called a scanner. There are many scanners in Wolfram|Alpha, each one corresponding roughly to a subject area or data type. The <pod> element has a `scanner` attribute that shows the name of the scanner that produced it, and this name is a guide to the type of content that the pod contains. You can use the `scanner` parameter to include only pods generated by one or more specific scanners. To specify more than one scanner, you can either use commas to separate names, such as `scanner=Numeric,MathematicalFunctionData`, or multiple scanner specifications, like `scanner=Numeric&scanner=MathematicalFunctionData`.

Timeouts and Asynchronous Behavior

The API allows you to control how long queries are allowed to take by specifying timeout parameters for various stages of Wolfram|Alpha processing. You can also request that time-consuming pods be returned asynchronously, so that some results are returned before all pods have finished.

To understand the four timeout parameters, it is important to know a bit about the stages of a Wolfram|Alpha computation. In the first stage, called the "parse" stage, the input is interpreted into something that Wolfram|Alpha can understand. In the second, “scan” stage, it is handed off to a series of computational entities called scanners that correspond roughly to individual areas of knowledge (food, airports, chemistry, music, etc.) The scanners produce data structures that will correspond to pods in the final output. The third and final stage is the "format" stage, where these pod expressions are processed and rendered into various output formats. Despite its name, the format stage often involves significant computation that is deliberately deferred from the scan stage to support the asynchronous style of pod loading used on the website and also available via the API. For some types of

inputs, the scan stage dominates the overall timing, but for many others the format stage is the longest.

The format stage only occurs for pods that will be included in the output, so if you are limiting which pods are returned via the `includepodid`, `podtitle`, `podindex`, or related parameters, then the cost of formatting is only incurred for pods that are included.

The Parsetimeout Parameter

The `parsetimeout` parameter limits how long the parse stage is allowed to take. The default value is five seconds, and the vast majority of queries take considerably less time than that. In very rare cases, though, a query can time out in the parse stage. When this happens, you will get a `<queryresult>` element with `success=false` and `parsetimedout=true`. When this happens you can retry the query with a longer parse time using the `parsetimeout` parameter:

```
http://api.wolframalpha.com/v2/query?input=1+4-6+8-9+1+5+6-7-9-5+6+8+9-3-1-6-7+1-5-6-7+2+9+3+
4-9-7-5&appid=XXXX&parsetimeout=10
```

The Scantimeout Parameter

The `scantimeout` parameter limits how long the scan stage is allowed to take. You can think of the scan stage as the period where Wolfram|Alpha "thinks" about a query, what data might be available, and what pods to return. If you stop this process before it finishes, then you might get a smaller number of pods in the result. Any lost pods would typically be higher-numbered pods (ones that appear lower down on the page on the website).

The default value is three seconds. The units are in seconds, so to specify 500 milliseconds you would use `scantimeout=.5`. If you know that you are only interested in the first few pods for a given query, or speed is critical and you are content with whatever you can get in a short time interval, then you can set `scantimeout` to a lower value. It is also possible that for certain classes of queries, increasing it from the default value will allow more pods to be generated. Most queries complete scanning in considerably less than three seconds.

The `<queryresult>` element contains a `timedout` attribute that gives a comma-separated sequence of names of scanners that timed out (for example, `timedout="Earthquake,Elevation,Date"`). If this is not an empty string, then you might get more pods by increasing the `scantimeout` value, especially if the names of the timed out scanners represent categories in which you are interested.

The Podtimeout and Formattimeout Parameters

The two remaining timeout parameters control the format stage. The `podtimeout` parameter limits how long any one pod can spend in the format stage. If the pod is not completely formatted and ready within this time span, it will be dropped from the output. The `formattimeout` parameter limits how long the entire formatting stage can take, for the full collection of pods. If you specify `formattimeout=2.5`, then you cap the entire formatting stage at 2.5 seconds, and you have a reasonably good idea about the maximum time any query can take from start to finish (especially if you also use the `scantimeout` parameter). On the other hand, if the third pod takes 2.4 seconds to format, then later pods might get no chance at all to appear in the output. In contrast, if you specify `podtimeout=0.5` instead, then any early time-consuming pods will get dropped without preventing later pods from showing up. The maximum time for the formatting stage would then be (`numpods` x 0.5). You can specify `podtimeout` and `formattimeout` together in the same query if desired (and also `scantimeout`).

Note that if you are using either of these format timeouts, you should consider using the asynchronous mode of the API instead, discussed in the next section.

The Async Parameter

The Wolfram|Alpha website is designed to allow some pods to appear in the user's browser before all the pods are ready. For many queries ("weather" is a typical example) you will see one to several pods appear quickly, but pods lower down on the screen show up as progress bars that have their content spliced in when it becomes available. The Wolfram|Alpha server stores certain pod expressions as files before they are formatted, and then waits for the client (a web browser, because here we are describing the behavior of the website) to request the formatted versions, at which point the formatting stage of the computation is performed and the result for each pod is returned as a separate transaction. You can get the same behavior in the API using the `async` parameter.

By default, the API behaves synchronously, meaning that the entire XML document that represents the result of a query is returned as a single unit. The caller gets nothing back until the entire result is ready. By specifying `async=true`, you can tell Wolfram|Alpha to return an XML document in which some pods are represented as URLs that need to be requested in a second step to get their actual XML content. Do not confuse this with image URLs that are part of a normal result when the image format type is requested. Although the actual data in the images must be requested as a second step, the images themselves are already completely generated by the time the original XML result is returned.

Here is an example of an asynchronous query:

```
http://api.wolframalpha.com/v2/query?input=weather&appid=XXXX&async=true
```

This returns the following XML:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true' error='false' numpods='6' datatypes='City,Weather' timedout=''
timing='2.255' parsetiming='0.07'>
 <pod title='Input interpretation' scanner='Identity'
      id='{&quot;Input&quot;, {&quot;IdentityScanner&quot;, &quot;DataScanner&quot;}}'
      position='100' error='false' numsubpods='1'>
  <subpod title=''>
   <plaintext>weather</plaintext>
  </subpod>
 </pod>
 <pod title='Latest recorded weather for Champaign, Illinois' scanner='Data'
      id='InstantaneousWeather:WeatherData'
      position='200' error='false' numsubpods='1'>
  <subpod title=''>
   <plaintext>temperature | 64 deg F
conditions | overcast
relative humidity | 77%  (dew point: 57 deg F)
wind speed | 3 mph
(1 hour  1 minute ago)</plaintext>
  </subpod>
 </pod>
 <pod title='Weather history &amp; forecast' scanner='Data'
      id='WeatherCharts:WeatherData'
      position='300' error='false' numsubpods='0'
      async='http://www1.wolframalpha.com/api/v2/asyncPod.jsp?id=MSP8198aa84d09f0i4ibc' />
 <pod title='Historical temperatures for August 22' scanner='Data'
      id='HistoricalTemperature:WeatherData'
      position='400' error='false' numsubpods='0'
      async='http://www1.wolframalpha.com/api/v2/asyncPod.jsp?id=MSP91b98aa8g006ieb4d3' />

 .. some pods and other elements deleted for brevity ...

</queryresult>
```

You can see in this result that the first two pods are returned in standard format, but the next two have no content. Instead, these <pod> elements have an async attribute that gives a URL that you can use to request the XML content that corresponds to the pod, with all the originally requested properties intact (format types, width, etc.) The XML that gets returned from a request to this URL is a <pod> element and subelements that directly replace the "stub" <pod> element. In the above output, the second async pod (the title is "Historical temperatures for August 22") holds the following URL:

```
http://www1.wolframalpha.com/api/v2/asyncPod.jsp?id=MSP8198aa84fgd09f0i4ibc
```

A request to that URL returns the following XML:

```
<?xml version='1.0' encoding='UTF-8'?>
<pod title='Historical temperatures for August 22' scanner='Data'
     id='HistoricalTemperature:WeatherData'
     position='400' error='false' numsubpods='1'>
 <subpod title=''>
  <plaintext>
minimum: 48 deg F
Aug 1997 | average: 73 deg F
 | maximum: 94 deg F
Aug 1975
(daily ranges, not corrected for changes in local weather station environment)
  </plaintext>
 </subpod>
 <states count='2'>
  <state name='Show table' input='Historical temperatures for August 22_Show table' />
  <state name='Show metric' input='Historical temperatures for August 22_Show metric' />
 </states>
</pod>
```

This is exactly the <pod> element that would have been present in the original output if it had not been done asynchronously.

It is usually the case that the first few pods are not delivered asynchronously, and later pods are, but this is not guaranteed. Low-numbered pods typically format quickly and thus fall within the internal timeout values that control whether a pod is delivered synchronously or asynchronously. Like the URLs in <img> tags, async URLs refer to temporary files with lifetimes of about a half hour or so. They cannot be stored for retrieval at arbitrary times in the future.

When you specify async=true, the default behavior is to allow pods 0.4 seconds to format, and if they fail to complete within that interval they are handled asynchronously. If you want control over that time interval, you can specify a number of seconds as the value for the async parameter, instead of just true. For example, async=0.2 would only allow 0.2 seconds for any given pod to format before switching to asynchronous mode.

The async parameter is a powerful way for more advanced programmers to let their users see Wolfram|Alpha results quickly, at the cost of having to manage the extra requests for async pods. It is mainly of use when you are preparing output for users to look at and you want them to be able to get some results right away with the rest coming soon after. If your program can do nothing useful with partial results, then ignore asynchronicity altogether. If you are using podtitle, podindex, or related parameters to request only one or two pods from a given query, then asynchronicity will be of little use. Remember that asynchronous behavior is not enabled by default, so unless you add the async parameter to a query, the entire XML result will always come back from the initial request.

Recalculate

The async parameter described in the previous section provides a way to get some results back from Wolfram|Alpha quickly, deferring longer computations for later. A related feature is the ability to "recalculate" a query. To understand the recalculate feature, make sure you have read the preceding section on the stages of a Wolfram|Alpha computation and the scantimeout parameter. In the so-called "scan" stage of a computation, multiple scanners are at work to produce pods relating to various topics. If one or more of those scanners are not finished within the scantimeout period, they will be interrupted and will not have a chance to produce a pod. If this happens, the <queryresult> element will name the scanners that timed out in its timedout attribute, and the recalculate attribute will have a non-empty value giving a URL. You can call this URL to redo the query with a longer scantimeout, to give the scanners that timed out a chance to finish and give you some new

pods. The advantage of using the `recalculate` URL instead of simply redoing the original query yourself and specifying a longer `scantimeout` is that the recalculate operation is much faster because it is able to skip a lot of the work that was done in the original query. For example, pods that were already computed are not computed again.

With the default `scantimeout` interval of three seconds, not many queries will have scanners time out and thus have a `recalculate` URL. One way to use the recalculate feature is to specify a short `scantimeout` in the original query, say one second, to get the initial set of pods back quickly, then use the `recalculate` URL to get the rest of the pods in a second call. This is how the Wolfram|Alpha website operates, and also Wolfram Research's Wolfram|Alpha Apps for iPhone and Android. Here is an example of an initial query specifying a short `scantimeout`:

```
http://api.wolframalpha.com/v2/query?input=pi&appid=XXXX&scantimeout=1.0
```

Note the `recalculate` URL in the result:

```
<queryresult success='true' error='false'
    numpods='5'
    datatypes='MathematicalFunctionIdentity'
    timedout='MathematicalFunctionData,Recognize,NumberLine'
    timing='1.339'
    parsetiming='0.07'
    parsetimedout='false'
    recalculate='http://www1.wolframalpha.com/api/v1/recalc.jsp?id=MSP46019e0a0c5idd'
    version='2.0'>

    ... content deleted for brevity

</queryresult>
```

We see in the above result that only five pods were returned and several scanners timed out. Your client program could display the five pods that came back in the initial result and then immediately make a call to the recalculate URL in the background. Here is what that call returns:

```
<queryresult success='true' error='false'
    numpods='3'
    datatypes='MathematicalFunctionIdentity'
    timedout=''
    timing='1.899' parsetiming='0' parsetimedout='false'
    recalculate=''
    version='2.0'>
 <pod title='Number line' scanner='NumberLine' id='NumberLine'
     position='350'
     error='false' numsubpods='1'>

  ... pod content deleted for brevity

 </pod>
 <pod title='Series representations' scanner='MathematicalFunctionData'
     id='SeriesRepresentations:MathematicalFunctionIdentityData'
     position='600'
     error='false' numsubpods='3'>

  ... pod content deleted for brevity

 </pod>
 <pod title='Integral representations' scanner='MathematicalFunctionData'
     id='IntegralRepresentations:MathematicalFunctionIdentityData'
     position='700'
     error='false' numsubpods='3'>

  ... pod content deleted for brevity

 </pod>
</queryresult>
```

Note that a recalculate call returns a result that is similar to a normal query. One difference is that it will not have <warnings> or <assumptions> elements, as those would have come in the original query. It might have a <sources> element, as new pods can specify new sources for their data. Most programmers will only be concerned with the <pod> elements from a recalculate result.

To use the recalculate result, you splice the new pods into the set returned by the original query. To know where the pods belong, look at their position attributes.The position attributes use the same numbering scheme as the original query, so that the first pod above, with a position of 350, would be inserted between the existing pods with positions 300 and 400 (or whatever are the nearest numbers above and below 350). The recalculate pods often have higher positions than the pods from the original query, and thus belong at the end, but as the above example demonstrates, sometimes they are intended to be spliced in between existing pods. It is possible for a recalculate pod to have the same position number as a pod from the original query, in which case it is a new version of that pod, and you should replace the old one with the new one.

If the original query used the async parameter, then it is possible for recalculate pods to be asynchronous as well. You handle these in the same way you would handle asynchronous pods in the original query.

Specifying Your Location

Many queries return results that depend on your current location. A query about weather conditions, for example, needs to know the caller's location, and a mortgage computation wants to present results in the local currency. By default, Wolfram|Alpha attempts to determine the caller's location from the IP address, but you can override this by specifying location information in one of three forms. The ip parameter lets you set the IP address of the caller, so if you are forwarding calls from your own web visitors to the Wolfram|Alpha API, you can propagate their IP addresses. The location parameter lets you specify a string like "Los Angeles, CA", or "Madrid", and the

latlong parameter lets you specify a latitude/longitude pair like "40.42,-3.71". Negative latitude values are South, and negative longitude values are West.

Controlling the Width of Results

Wolfram|Alpha formats pod images to a default maximum width of 500 pixels. If you want to change this width, such as for a small mobile device screen, the API provides three parameters you can use. The first two, width and maxwidth, apply to images of text and tables, which are the majority of Wolfram|Alpha output. The third parameter, plotwidth, applies to plots and other graphics.

An easy way to see the effect of the width parameter is to look at the "Decimal approximation" pod from the "pi" query. Here is that pod when the width parameter is not specified:

```
<pod title='Decimal approximation'  ... other attributes deleted >
 <subpod title=''>
  <plaintext>3.14159265358979323846264338327950288419716939937510582097494...</plaintext>
  <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP6119badi?MSPStoreType=image/gif'
      alt='3.14159265358979323846264338327950288419716939937510582097494...'
      title='3.14159265358979323846264338327950288419716939937510582097494...'
      width='491'
      height='20' />
 </subpod>
 ... other elements deleted
</pod>
```

The number of digits shown in the plaintext representation is chosen so as to fit within a width of 500 pixels, and you can see that the width attribute of the <img> element shows that the rendered image is 491 pixels wide. If we try this query again adding the parameter width=300, here is the new pod:

```
<pod title='Decimal approximation'  ... other attributes deleted >
 <subpod title=''>
  <plaintext>3.14159265358979323846264338327950288419716939937510582097494...</plaintext>
  <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP11979hc06?MSPStoreType=image/gif'
      alt='3.14159265358979323846264338327950288419716939937510582097494...'
      title='3.14159265358979323846264338327950288419716939937510582097494...'
      width='291'
      height='20' />
   </subpod>
 ... other elements deleted
</pod>
```

With this smaller width requested, Wolfram|Alpha displays fewer digits, and the new image width is 291 pixels.

In most cases, unlike this example with digits of pi, Wolfram|Alpha cannot choose to change the amount of output when a smaller width is requested, so instead it will insert line breaks to wrap lines to the desired width. One problem with forcing lines to be wrapped at a certain width is that it can introduce ugly line breaks. If you perform the query "mortgage" on the website, you will see a "Payments table" pod that includes a wide table of values. If you force this table to be wrapped at, say, 300 pixels, then every table cell becomes squashed, leading to line-wrapping within individual numbers. The maxwidth parameter allows users of the API to adopt a two-part strategy for controlling width. You can specify a desired width using the width parameter, but if you also specify a higher maxwidth value, then when Wolfram|Alpha formats results to the width value, it will attempt to detect if undesirable line breaks were forced to be used, and if so it will automatically re-format to your larger maxwidth. In other words, using width and maxwidth together lets you say, in effect, "Please format text and tables to a certain width, but if this is going to look really ugly, I'd rather have it formatted at a higher width

instead".

The plotwidth parameter controls the width at which plots and graphics are rendered. The default value is 200 pixels. There are many graphics in Wolfram|Alpha that are deliberately rendered at larger sizes to accommodate their content. Specifying plotwidth is currently an experimental feature that does not yet affect many type of graphics.

The mag parameter controls the magnification of pod images. The default value is 1.0, meaning no magnification. Magnification does not affect the pixel width of images, so if you specify a width parameter, or accept the default of 500 pixels, images will still come back at that size if you specify a magnification value. What happens is that if you specify, say, mag=2, then the pod image is formatted to a width of half the requested width (say, 250 pixels) and then blown up by a factor of 2 during rendering, to 500 pixels. You might wonder what the difference is between formatting an image at 250 pixels and magnifying it by 2 during rendering versus formatting it at 500 pixels and not magnifying. The answer is that image elements like font sizes, tick marks, line-wrapping, and spacing, are very sensitive to the formatting width. Let's say your program will be displaying its results on a very high-density screen like a tablet, with a screen density of 240 pixels per inch and a screen width of 960 pixels (4 inches). If you specify width=900, you will get images that fit the screen, but fonts will be tiny—the images will look shrunken by the high number of pixels per inch. If you specify, say, mag=2, then your images will still be wrapped to fit the 900 pixel screen, but they will be blown up by a factor of 2 and therefore much more readable. If you are displaying images on a standard computer monitor, you will probably not need to deal with the mag parameter. The images generated by the Wolfram|Alpha website do not use this parameter.

Classifying Queries

If you are using the Wolfram|Alpha API in a chemistry application, you probably want to know if Wolfram|Alpha interprets one of your user's queries as referring to music before you splash the results onto the user's screen. The API gives access to several ways to understand the subject areas and "data types" that Wolfram|Alpha thinks a query represents. A more formal characterization of queries is under development, but for now you can use the following main pieces of information.

Clash Assumptions

The <assumptions> element in output has already been described in detail. One type of assumption, the "Clash" assumption, provides especially useful information about the meaning of a word or words in a query.

This type of <assumptions> element is not generated unless Wolfram|Alpha detects a clash in meanings of a word, so you cannot use it in general to determine the categories of arbitrary words in a query, but it is useful when it appears.

The datatypes Attribute

The datatypes attribute of the <queryresult> tag is the most useful way to obtain a list of subject categories for a query. This attribute gives a comma-separated sequence of subject areas and types of data that Wolfram|Alpha used in generating the set of results. As an example, consider an application that allows its users to enter DNA sequences and uses Wolfram|Alpha as part of its analysis. The query GATTACACCAGGATAAC results in the following XML:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true' error='false' numpods='5'
    datatypes='DNAString' timedout='' timing='0.914' parsetiming='0.167'>
 <pod title='Input interpretation' scanner='Identity' position='100'
      error='false' numsubpods='1'>
  <subpod title=''>
   <plaintext>GATTACACCAGGATAAC  (genome sequence)</plaintext>
  </subpod>
 </pod>

... pods deleted for brevity ...

</queryresult>
```

The value of the `datatypes` attribute is "DNAString", which tells your application that Wolfram|Alpha recognized this query as a DNA sequence. For such a long string of characteristic letters, there is little chance that Wolfram|Alpha will not recognize it as a DNA sequence. In contrast, here is what is returned from the query GATA:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='true' error='false' numpods='5'
    datatypes='Country,Language,WritingSystem' timedout='
    timing='1.205' parsetiming='0.202'>

 .. pods describing Gata as a language in India deleted ...

 <assumptions count='1'>
  <assumption type='Clash' word='gata' count='2'>
   <value name='Language' desc='a language' input='*C.gata-_*Language-' />
   <value name='DNAString' desc='a DNA string' input='*C.gata-_*DNAString-' />
  </assumption>
 </assumptions>
</queryresult>
```

Now the `datatypes` attribute is "Country,Language,WritingSystem", which is a clear indication that Wolfram|Alpha did not interpret this query in the way that you wanted. Note also that there is an `<assumption>` element of the Clash type that clearly shows the ambiguous meaning of "gata", as well as the currently chosen value (always the first-listed one). You could use the assumptions mechanism to force short, ambiguous strings to be interpreted as DNA strings. (In this example, however, simply adding the word DNA to the query is enough to get Wolfram|Alpha to interpret GATA as a DNA sequence.)

Because many Wolfram|Alpha results are tailored to the user's present location, the "City" data type appears frequently. This does not mean that the query was interpreted as directly referring to a city. For example, the query "12/5/2007 moon" returns "Astronomical,City", for datatypes. If you try this query on the website you will find that some pods, such as "Sky position", refer to the caller's city.

There is not a complete listing of the types that can appear in the datatypes list at the present time.

Scanner Names

As described earlier, pods are produced by computational units called "scanners", which correspond roughly to single subject areas. For example, there are scanners called Tide, Statistics, Species, Physiology, NumberSystems, and many more. Each `<pod>` element has a `scanner` attribute that gives the name of the scanner that produced it. This is a good guide to the contents of the pod. The complete set of scanner names for all pods returned can be used as a guide to the overall characterization of the query.

Image Maps

Many pods on the Wolfram|Alpha website have HTML image maps associated with them, so that you can click parts of the pod image to execute queries. Most table-style pods have this property, so that each element in the table or list can be clicked to trigger a query based on the content of that item. For example, in the query “France”, virtually every pod is a table or list of textual items, and each item can be clicked to trigger another query. The API provides information that you can use to replicate this “image map” functionality in your own programs.

We will use the “France” query as an example. To get image map data in the result, you need to include `imagemap` as a format type:

```
http://api.wolframalpha.com/v2/query?input=france&appid=XXXX&format=image,imagemap
```

For reference, here is a screenshot of the “Name” pod on the website, with the links highlighted as they are when the mouse is hovering over the pod:

Here is what the “Name” pod looks like in the API result from the above query:

```
 <pod title='Name'
      scanner='Data' id='Identifiers:CountryData' position='200'
      error='false' numsubpods='1'>
   <subpod title=''>
     <img src='http://www1.wolframalpha.com/Calculate/MSP/MSP93ff?MSStoreType=image/gif'
       alt='full name | French Republic
full native name | République française
internet code | .fr'
       title='full name | French Republic
full native name | République française
internet code | .fr'
       width='294' height='106' />
     <imagemap>
       <rect left='12' top='8' right='39' bottom='28'
         query='France+full+name'
         assumptions='ClashPrefs_*Country.France.CountryProperty.FullName-'
         title='France full name' />
       <rect left='39' top='8' right='76' bottom='28'
         query='France+full+name'
         assumptions='ClashPrefs_*Country.France.CountryProperty.FullName-'
         title='France full name' />
       <rect left='12' top='42' right='39' bottom='62'
         query='France+full+native+name'
         assumptions='ClashPrefs_*Country.France.CountryProperty.FullNativeNames-'
         title='France full native name' />
       <rect left='39' top='42' right='83' bottom='62'
         query='France+full+native+name'
         assumptions='ClashPrefs_*Country.France.CountryProperty.FullNativeNames-'
         title='France full native name' />
       <rect left='83' top='42' right='120' bottom='62'
         query='France+full+native+name'
         assumptions='ClashPrefs_*Country.France.CountryProperty.FullNativeNames-'
         title='France full native name' />
       <rect left='12' top='76' right='68' bottom='96'
         query='France+internet+code'
         assumptions='ClashPrefs_*Country.France.CountryProperty.InternetCode-'
         title='France internet code' />
       <rect left='68' top='76' right='98' bottom='96'
         query='France+internet+code'
         assumptions='ClashPrefs_*Country.France.CountryProperty.InternetCode-'
         title='France internet code' />
     </imagemap>
   </subpod>
   <states count='1'>
     <state name='More'
         input='Identifiers:CountryData__More' />
   </states>
 </pod>
```

The <imagemap> element returns information that you can use to identify clickable areas within the pod and the queries that should be issued if those areas are clicked. Each <rect> subelement identifies a separate region. The coordinates are based on (0, 0) being the top-left corner of the pod. The query attribute gives the query input, and it is already URL-encoded for direct use in a subsequent query URL. The assumptions attribute gives an assumption value you would include in the query to ensure that it is interpreted as desired. The title attribute is a text string that shows the query in a readable form, which you might want to display as a tooltip when the mouse hovers over the region.

As an example, if you detected a user click in the region identified by the first <rect> element in the above pod, you might execute the following query:

```
http://api.wolframalpha.com/v2/query?input=France+full+name&appid=XXXX&assumption=ClashPrefs_
*Country.France.CountryProperty.FullName-
```

You might notice that although there appears to be only three clickable boxes in the table in the screen shot, there are seven <rect> elements in the <imagemap>. This is because each word in the boxes generates a separate region.

Primary Result Tagging

Although Wolfram|Alpha returns many pods for most queries, there is sometimes the notion of a "primary result" for a given query. This is especially true for queries that correspond to *Mathematica* computations ("2+2") or simple data lookups ("France GDP"). If you are looking to display the closest thing to a simple "answer" that Wolfram|Alpha can provide, you can look for pods tagged as primary results via the primary=true attribute. Here is an example, the "Result" pod from the query "France GDP":

```
 <pod title='Result' scanner='Data'
      id='Result' position='200' error='false' numsubpods='1'
      primary='true'>
   <subpod title=''>
     <plaintext>$2 .669 trillion per year  (US dollars per year)  (world rank: 5th)  (Q2
2010 estimate)
     </plaintext>
   </subpod>
 </pod>
```

Primary result tagging is a relatively new feature in Wolfram|Alpha, and many queries do not have a primary result, often because it is not meaningful for that query.

Some Miscellaneous URL Parameters

This section treats a few URL parameters that do not fit neatly in any other sections. Of the parameters listed here, only reinterpret is likely to be of widespread usefulness.

reinterpret

Wolfram|Alpha has the ability to "reinterpret" queries that it does not understand, meaning that it can switch to a related query for which some results are available. The idea is to reduce the number of queries that fall through and produce no results, at the cost of giving the user results for something that might not be what they intended. When Wolfram|Alpha reinterprets a query, it produces a <reinterpret> warning, as described in the section on Warnings. If you want to enable this behavior, you must specify reinterpret=true in your query URL. This is the default behavior on the website, and many clients will likely want to turn it on as well.

signature

Because the AppID is provided in plaintext form in the query URL, some clients might be concerned that a third party could steal and misuse their AppID. There is a special URL parameter called sig that can be used to supply a specially computed signature in each of your query URLs that prevents anyone else from using your AppID. Contact Wolfram|Alpha for more information about how to use this feature.

ignorecase

By default, Wolfram|Alpha treats uppercase as significant in queries. It is generally quite flexible about case, and callers rarely need to be concerned with case in queries, but there are rare examples where it is useful to force

Wolfram|Alpha to ignore case. This is done by setting `ignorecase=true`. An example of this is the query “mpg”, which is interpreted as “miles per gallon”, but MPG is also an airport code. The query “MPG” is also interpreted as “miles per gallon”, but an assumption is generated to let users specify that they want the airport code instead. This assumption is not generated for the query “mpg”, because the mismatched case makes it a poor fit. If you specify `ignorecase=true`, however, then the airport code assumption is generated, making the query “mpg” essentially equivalent to “MPG”.

translation

Wolfram|Alpha is able to translate some simple non-English queries into English. For example, the query “uno dos tres” is automatically translated into “one two three” and handled as such. A `<translation>` warning (described in the section on Warnings) is generated in such cases to inform the caller that a translation has occurred. If you want to turn automatic translation off, specify `translation=false` in your query URL.

Queries That Are Not Understood

If a query string cannot be understood by Wolfram|Alpha, the `<queryresult>` tag will have its `success` attribute set to false and the `error` attribute also false. For example, the query "fogasdgrd masdustasn":

```
http://api.wolframalpha.com/v2/query?input=fogasdgrd+masdustasn&appid=XXXX
```

returns the following:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='false' error='false'
     numpods='0' datatypes='' timedout=''
     timing='0.68' parsetiming='0.169' parsetimedout='false'
     recalculate='' version='2.0'>
  <tips count='1'>
    <tip text='Check your spelling, and use English' />
  </tips>
</queryresult>
```

On the website, this type of query returns a page that says "Wolfram|Alpha doesn’t know how to interpret your input." We call these types of results "fallthrough results".

Because Wolfram|Alpha could not understand "fogasdgrd masdustasn", there is no pod content in the output, but for such queries there can be various other types of elements in the API result. The following sections describe the possible element types.

<tips>

The previous example shows the `<tips>` element. Each tip gives a line of text that you might choose to display to users, like the website does.

<didyoumeans>

The Wolfram|Alpha website sometimes has a "Did you mean" prompt that suggests a different query close in spelling to the one you entered. This alternative suggestion is available in the API via the `<didyoumeans>` element. Here is an example for the query "Francee splat", which suggests "Frances split" as an alternative:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='false' error='false'
     numpods='0' datatypes=''
     timedout='' timing='0.475' parsetiming='0.0745' parsetimedout='false'
     recalculate='' version='2.0'>
  <warnings count='1'>
    <spellcheck word='Francee' suggestion='frances' />
  </warnings>
  <didyoumeans count='1'>
    <didyoumean>frances split</didyoumean>
  </didyoumeans>
</queryresult>
```

<languagemsg>

If Wolfram|Alpha cannot understand your query, but recognizes it as a foreign language, it will generate a `<languagemsg>` element. Here is an example for the query "wo noch nie":

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='false' error='false'
     numpods='0' datatypes=''
     timedout='' timing='3.724' parsetiming='1.032' parsetimedout='false'
     recalculate='' version='2.0'>
  <languagemsg english='Wolfram|Alpha does not yet support German.'
      other='Wolfram|Alpha versteht noch kein Deutsch.' />
  <tips count='1'>
    <tip text='Try spelling out abbreviations' />
  </tips>
</queryresult>
```

The `<languagemsg>` element has an `english` attribute giving the textual message in English and an `other` attribute giving the textual message in whatever language the query appears to be.

<futuretopic>

Queries that refer to topics that are under development generate a `<futuretopic>` element. This example is for the query "microsoft windows":

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='false' error='false'
    numpods='0' datatypes='FutureTopic'
    timedout='' timing='0.194' parsetiming='0.124' parsetimedout='false'
    recalculate='' version='2.0'>
  <futuretopic topic='Operating Systems'
     msg='Development of this topic is under investigation...' />
</queryresult>
```

<relatedexamples>

Although Wolfram|Alpha cannot understand the query "bear shoe", it sees the word "bear" and provides users with information about related examples via the `<relatedexamples>` element. Here is the output of the "bear shoe" query:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='false' error='false'
     numpods='0' datatypes=''
     timedout='' timing='0.542' parsetiming='0.117' parsetimedout='false'
     recalculate='' version='2.0'>
  <relatedexamples count='1'>
    <relatedexample input='bear' desc='information about an animal' category='Animals'
        categorythumb='http://www.wolframalpha.com/thumbnails/splat/AnimalsPlants.gif'
        categorypage='http://www.wolframalpha.com/examples/AnimalsPlants.html' />
  </relatedexamples>
</queryresult>
```

If you perform this query on the website you will see how the various attributes of the <relatedexample> element can be used.

<examplepage>

When a query cannot be meaningfully computed, but is recognized by Wolfram|Alpha as a category for which a set of example queries has already been prepared, it generates an <examplepage> element. Conceptually, this is a bit like the <relatedexamples> element, but here the query is more of a "direct hit" on a specific topic. Here is the output generated for the query "chemical" (you can ignore the <assumptions> element that is also generated):

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='false' error='false'
     numpods='0' datatypes=''
     timedout='' timing='0.395' parsetiming='0.108' parsetimedout='false'
     recalculate='' version='2.0'>
  <assumptions count='1'>
    <assumption type='Clash' word='chemical' count='3'>
      <value name='ExamplePage' desc='a general topic'
             input='*C.chemical-_*ExamplePage-' />
      <value name='Financial' desc='a financial entity' input='*C.chemical-_*Financial-' />
      <value name='Word' desc='a word' input='*C.chemical-_*Word-' />
    </assumption>
  </assumptions>
  <examplepage category='ChemicalCompounds'
      url='http://www.wolframalpha.com/examples/ChemicalCompounds-content.html' />
</queryresult>
```

The url attribute gives a link to an HTML page of sample queries in the topic.

<warnings> and <assumptions>

The <warnings> and <assumptions> elements are described in earlier sections. They can sometimes appear in a fallthrough result. The above example for the didyoumeans section shows a <warnings> element, and the examplepage section shows an <assumptions> element.

Errors

There are several circumstances that can trigger errors from the API. Examples of such error conditions are when the input parameters are not legal in some way, the AppID is incorrect, or an internal error occurs in Wolfram|Alpha. Errors are indicated by the error=true attribute in the <queryresult> tag. Error results will have an <error> element that gives a code and short description of the error. For example, this query:

```
http://api.wolframalpha.com/v2/query?input=mortgage
```

returns the following:

```
<?xml version='1.0' encoding='UTF-8'?>
<queryresult success='false' error='true' numpods='0'
     datatypes='' timedout='' timing='0.066' parsetiming='0'>
 <error>
  <code>2</code>
  <msg>Appid missing</msg>
 </error>
</queryresult>
```

Errors are distinct from queries that fail because Wolfram|Alpha cannot make sense of the input. As discussed in the previous section, those results have the `success=false` attribute, but not `error=true`.

It is also possible that the processing of individual pods can fail in some way. In such cases, the query as a whole succeeds, but a specific pod will have the `error=true` attribute. The body of the <pod> element might then contain an <error> element that describes the error condition.

Handling Future Enhancements

The Wolfram|Alpha API will continue to evolve and grow over time, as new capabilities are added to Wolfram|Alpha. There are a few steps you can take to help ensure that your programs continue to work in the future. You will notice that query URLs include the version number “v2” at the moment, so changes to the API that are likely to break a significant fraction of existing programs can be handled at our end by changing the query URL to “v3” and keeping the old behavior in “v2” URLs. Most future changes, however, are additions that should not break existing clients, as long as they are written with future flexibility in mind.

The most important thing you can do to avoid problems with future updates is to be tolerant of new element types and attributes showing up. For example, top-level elements (that is, direct subelements of <queryresult>) include <pod>, <warnings>, <assumptions>, <sources>, etc., and it is almost certain that new elements will be added. This is true at all levels of the XML hierarchy. The basic principle is to handle elements you know about, but ignore ones that you don’t. Similarly for attributes—elements may acquire new attributes over time, although this is generally quite easy for client programs to accommodate by simply ignoring them.

Be aware that Wolfram|Alpha output is constantly changing. Queries that were interpreted in one way today might be interpreted differently (more accurately, presumably) tomorrow. The number and types of pods produced by some queries will change over time. This documentation uses “pi” as an example query in many places, and the example outputs had to be redone recently because a new type of pod was introduced in the output for that query. If you are trying to restrict Wolfram|Alpha output to a subset of pods for a given query, using the `excludepodid` and `includepodid` parameters is generally preferable to relying on `podtitle` or `podindex` parameters, as titles and positions are more likely to drift over time than IDs.

Pods have an `error` attribute that indicates whether the content was generated correctly or not. Currently, this attribute is set to `true` only for asynchronous pods; synchronous pods are simply not present in the output if there was an error generating or formatting them. In the future, this is likely to change, so that the initial query result might come back with some pods having `error=true`. Write your code so as to detect error pods and not to try to display them. Error pods will possibly have an <error> subelement that describes the problem. This error information is intended for programmers, not for display to your users.

The validatequery Function

So far we have been dealing with the `query` function. There is another function in the API called `validatequery`. This is a specialized function that performs only the initial parsing phase of Wolfram|Alpha processing. Its purpose is to quickly determine whether Wolfram|Alpha can make sense of a given input, bypassing the more time-consuming stages of fully analyzing the input and preparing results. Here is a sample URL that calls the `validatequery` function:

```
http://api.wolframalpha.com/v2/validatequery?input=pi&appid=xxxxx
```

Here is the result:

```
<?xml version='1.0' encoding='UTF-8'?>
<validatequeryresult success='true' error='false' timing='0.097' parsetiming='0.089'
version='2.0'>
 <assumptions count='1'>
  <assumption type='Clash' word='pi' count='4'>
   <value name='NamedConstant' desc='a mathematical constant'
          input='*C.pi-_*NamedConstant-' />
   <value name='Character' desc='a character' input='*C.pi-_*Character-' />
   <value name='Movie' desc='a movie' input='*C.pi-_*Movie-' />
   <value name='Word' desc='a word' input='*C.pi-_*Word-' />
  </assumption>
 </assumptions>
</validatequeryresult>
```

The `success` attribute of the `<validatequeryresult>` element tells you whether Wolfram|Alpha could successfully parse the input string. If its value is true then it is highly likely that Wolfram|Alpha can return sensible results for this input using the `query` function.

The `<validatequeryresult>` element has no pod content, but it can contain `<assumptions>`, `<warnings>`, and `<error>` elements. You can use `<assumptions>` information in the same way as you would from the result of the `query` function, such as to perform a subsequent `query` call with a modified assumption. Not all of the assumptions that can be generated by a `query` call appear in the results of `validatequery`. Some assumptions are generated in later stages of processing that do not occur in `validatequery`, and these assumptions will not be present in the output.

Reference Appendix

This section includes short descriptions of all the input parameters and the main XML elements in results. It also has a complete listing of all the assumption types that can be generated, with examples.

Query Parameters

There are numerous parameters you can specify in the URL to control how the query is performed and what types of results it returns. Here is an example request URL that uses several of these parameters:

```
http://api.wolframalpha.com/v2/query?input=5+largest+countries&appid=xxxx&podtitle=Result+by+
area&format=plaintext&scantimeout=3
```

All queries require the `input` and `appid` parameters.

input

Specifies the input string, such as "5 largest countries". The text must be URL-encoded, making this example string "5%20largest%20countries" or "5+largest+countries".

Required.

appid

An ID provided by Wolfram Research that identifies the application or organization making the request.

Required.

format

The desired result format(s). Possible values are `image`, `plaintext`, `minput`, `moutput`, `cell`, `mathml`, `imagemap`, `sound`, `wav`. To request more than one format type, separate values with a comma: "`plaintext,minput,image`".

Optional; defaults to "plaintext,image".

includepodid

Specifies a pod ID to include. You can specify more than one of these elements in the query. Only pods with the given IDs will be returned.

Optional; default is all pods.

excludepodid

Specifies a pod ID to exclude. You can specify more than one of these elements in the query. Pods with the given IDs will be excluded from the result.

Optional; default is to exclude no pods.

podtitle

Specifies a pod title. You can specify more than one of these elements in the query. Only pods with the given titles will be returned. You can use * as a wildcard to match zero or more characters in pod titles.

Optional; default is all pods.

podindex

Specifies the index of the pod(s) to return. This is an alternative to specifying pods by title or ID. You can give a single number or a sequence like "2,3,5".

Optional; default is all pods.

scanner

Specifies that only pods produced by the given scanner should be returned. You can specify more than one of these elements in the query.

Optional; default is all pods.

async

Wolfram|Alpha can use an asynchronous mode that allows partial results to come back before all the pods are computed. See the section "Timeouts and Asynchronous Behavior" for more details.

Optional; defaults to false.

location specifications (ip, latlong, location)

Some Wolfram|Alpha computations take into account the caller's current location. By default, Wolfram|Alpha attempts to determine the caller's location from the IP address, but you can override this by specifying location information in one of three forms. See the "Specifying Your Location" section for more details.

Optional; defaults to determining location via the IP address of the caller.

assumption

Specifies an assumption, such as the meaning of a word or the value of a formula variable. See the "Assumptions" section for more details.

Optional.

podstate

Specifies a pod state change, which replaces a pod with a modified version, such as a switch from Imperial to metric units. See the "Pod States" section for more details.

Optional.

units

Lets you specify the preferred measurement system, either "metric" or "nonmetric" (U.S. customary units).

Optional; defaults to making a decision based on the caller's geographic location.

width specifications (width, maxwidth, plotwidth, mag)

These specifications control the page width in pixels for which the output should be formatted. See the section "Controlling the Width of Results" for more details.

Optional. Default width and maxwidth are 500; default plotwidth is 200; default mag is 1.0.

scantimeout

The number of seconds to allow Wolfram|Alpha to compute results in the "scan" stage of processing. See the section "Timeouts and Asynchronous Behavior" for more details.

Optional; defaults to 3.0.

podtimeout

The number of seconds to allow Wolfram|Alpha to spend in the "format" stage for any one pod. See the section "Timeouts and Asynchronous Behavior" for more details.

Optional; defaults to 4.0.

formattimeout

The number of seconds to allow Wolfram|Alpha to spend in the "format" stage for the entire collection of pods. See the section "Timeouts and Asynchronous Behavior" for more details.

Optional; defaults to 8.0.

parsetimeout

The number of seconds to allow Wolfram|Alpha to spend in the "parsing" stage of processing. See the section "Timeouts and Asynchronous Behavior" for more details.

Optional; defaults to 5.0.

reinterpret

Whether to allow Wolfram|Alpha to reinterpret queries that would otherwise not be understood. See the section "Some Miscellaneous URL Parameters" for more details.

Optional; defaults to false.

translation

Whether to allow Wolfram|Alpha to try to translate simple queries into English. See the section "Some Miscellaneous URL Parameters" for more details.

Optional; defaults to true.

ignorecase

Whether to force Wolfram|Alpha to ignore case in queries. See the section "Some Miscellaneous URL Parameters" for more details.

Optional; defaults to false.

sig

A special signature that can be applied to guard against misuse of your AppID.

Optional.

Result XML Description

Results come back in XML format. Here is a listing of the more important result elements and their attributes.

<queryresult>

`<queryresult>` is the outer wrapper for all results from the `query` function. It has the following attributes:

`success`	true or false depending on whether the input could be successfully understood. If false there will be no <pod> subelements.
`error`	true or false depending on whether a serious processing error occurred, such as a missing required parameter. If true there will be no pod content, just an <error> subelement.
`numpods`	The number of pods.
`version`	The version specification of the API on the server that produced this result.
`datatypes`	Categories and types of data represented in the results.
`timing`	The wall-clock time in seconds required to generate the output.
`timedout`	The number of pods that are missing because they timed out (see the scantimeout query parameter).
`parsetiming`	The time in seconds required by the parsing phase.
`parsetimedout`	Whether the parsing stage timed out (try a longer parsetimeout parameter if true)
`recalculate`	A URL to use to recalculate the query and get more pods.

<pod>

`<pod>` elements are subelements of `<queryresult>`. Each contains the results for a single pod. `<pod>` has the following attributes:

`title`	The pod title, used to identify the pod.
`error`	true or false depending on whether a serious processing error occurred with this specific pod. If true, there will be an <error> subelement.
`position`	A number indicating the intended position of the pod in a visual display. These numbers are typically multiples of 100, but it is only relevant that they form an increasing sequence from top to bottom.
`scanner`	The name of the scanner that produced this pod. A guide to the type of data it holds.
`id`	This attribute is undocumented at this time.

numsubpods	The number of subpod elements present.

<subpod>

<subpod> elements are subelements of <pod>. Each contains the results for a single subpod. <subpod> has a title attribute, which is usually an empty string, because most subpods have no title.

<plaintext>

<plaintext> elements are the textual representation of a single subpod. They only appear if the requested result formats include plaintext. <plaintext> has no attributes or subelements.

<img> elements are HTML img elements suitable for direct inclusion in a web page. They point to stored image files giving a formatted visual representation of a single subpod. They only appear in pods if the requested result formats include img. In most cases, the image will be in GIF format, although in a few cases it will be in JPEG format. The filename in the <img> URL will tell you whether it is GIF or JPEG.

<mathml>

<mathml> elements enclose the Presentation MathML representation of a single subpod. They only appear if the requested result formats include mathml. <mathml> has no attributes.

<assumptions>

The <assumptions> element is a subelement of <queryresult>. Its content is a series of <assumption> elements. It has a count attribute, giving the number of <assumption> subelements. See the “Assumptions” section of the main text for more details.

<assumption>

The <assumption> element is a subelement of <assumptions>. It defines a single assumption, typically about the meaning of a word or phrase, and a series of possible other values. See the “Assumptions” section of the main text for more details.

<states>

The <states> element is a subelement of <pod> or <subpod>. It has a count attribute and it contains a series of <state> and/or <statelist> elements. See the “Pod States” section of the main text for more details.

<warnings>

The <warnings> element occurs as a subelement of <queryresult>. It contains <warning> subelements, each of which describes a particular warning generated during the query.

<error>

The <error> element occurs as either a subelement of <queryresult>, if there was a failure that prevented any result from being returned, or as a subelement of <pod>, if there was an error that only prevented the result from a given pod from being returned. <error> has the following attributes:

code	The error code, an integer.
msg	A short message describing the error.

<sources>

The <sources> element is a subelement of <queryresult>. The <sources> element contains a series of <source> subelements, each one defining a link to a web page of source information. See the “Sources” section of the main text for more details.

Listing of Assumption Types

Here is a complete listing of all the assumption types that can currently be returned by the API. This list will grow in the future as new types are added to Wolfram|Alpha itself.

Clash

The Clash assumption is generated when a word can represent different categories of things, such as "pi" being treated as a mathematical constant, a movie, a character, or simply as a word. Here is a typical example of a Clash <assumption> element, this generated from the query "pi":

```
<assumption type='Clash' word='pi' count='4'>
   <value name='NamedConstant' desc='a mathematical constant' input='*C.pi-_*NamedConstant-'
/>
   <value name='Character' desc='a character' input='*C.pi-_*Character-' />
   <value name='Movie' desc='a movie' input='*C.pi-_*Movie-' />
   <value name='Word' desc='a word' input='*C.pi-_*Word-' />
</assumption>
```

Unit

The Unit assumption is generated when a word is interpreted as a unit abbreviation, but it is ambiguous as to what unit it represents. An example is "m", meaning either meters or minutes. Here is a typical example of a Unit <assumption> element, this generated from the query "10 m":

```
<assumption type='Unit' word='m' count='3'>
   <value name='Meters' desc='meters' input='UnitClash_*m.*Meters--' />
   <value name='MinimsUS' desc='U.S. minims' input='UnitClash_*m.*MinimsUS--' />
   <value name='Minutes' desc='minutes of time' input='UnitClash_*m.*Minutes--' />
</assumption>
```

AngleUnit

The AngleUnit assumption is generated when a number is interpreted as a unit of angle, but it is ambiguous whether it should be interpreted as degrees or radians. This assumption type always has two <value> elements, one for the assumption of degrees and the other for the assumption of radians. Here is the AngleUnit <assumption> element generated from the query "sin(30)":

```
<assumption type='AngleUnit' count='2'>
   <value name='D' desc='degrees' input='TrigRD_D' />
   <value name='R' desc='radians' input='TrigRD_R' />
</assumption>
```

Function

The Function assumption is generated when a word is interpreted as referring to a mathematical function, but it is ambiguous which function is meant. An example is "log" meaning either log base *e* or log base 10. Here is a typical example of a Function <assumption> element, this generated from the query "log 20":

```
<assumption type='Function' word='log' count='2'>
   <value name='Log' desc='the natural logarithm' input='*FunClash.log-_*Log.Log10-' />
   <value name='Log10' desc='the base 10 logarithm' input='*FunClash.log-_*Log10.Log-' />
</assumption>
```

MultiClash

The MultiClash assumption is a type of clash where multiple overlapping strings can have different interpretations. An example is the query "log 0.5", where the whole phrase can be interpreted as the mathematical object "log(0.5)", or the word "log" can be interpreted as a probability distribution or a plotting function:

```
<assumption type='MultiClash' word='' count='3'>
   <value name='Math' word='log 0.5' desc='a mathematical object' input='*MC.log+0%21.5-
_*Math-' />
   <value name='Distribution' word='log' desc='a probability distribution'
         input='*MC.log+0%21.5-_*DistributionNoFluff-' />
   <value name='PlotsWord' word='log' desc='a plotting function' input='*MC.log+0%21.5-
_*PlotsWord-' />
</assumption>
```

SubCategory

The SubCategory assumption is similar to the Clash type in that a word can refer to multiple types of entities, but for SubCategory all the interpretations are within the same overall category. An example is the query "hamburger", which generates a SubCategory assumption for different types of hamburger (basic hamburger, McDonald's hamburger, Burger King hamburger, etc.) The hamburger query also generates a Clash assumption over whether hamburger should be treated as a type of food or a simple word, but given that Wolfram|Alpha is treating hamburger as a type of food in this query, it also can be resolved into subcategories of hamburger. Here is the SubCategory <assumption> element from the "hamburger" query:

```
<assumption type='SubCategory' word='hamburger' count='3'>
   <value name='Hamburger' desc='hamburger' input='*DPClash.ExpandedFoodE.hamburger-
_*Hamburger-' />
   <value name='BurgerKingHamburger' desc='Burger King hamburger'
          input='*DPClash.ExpandedFoodE.hamburger-_*BurgerKingHamburger-' />
   <value name='McDonaldsHamburger' desc='McDonald&amp;apos;s hamburger'
          input='*DPClash.ExpandedFoodE.hamburger-_*McDonaldsHamburger-' />
</assumption>
```

Attribute

You can think of the Attribute assumption as the next step down in the sequence of Clash and SubCategory. Wolfram|Alpha emits an Attribute assumption to allow you to modify an attribute of an already well-characterized entity. In the query "hamburger", Wolfram|Alpha assumes you mean that hamburger is a food item (although it

gives you a Clash assumption to modify this) and that you mean a "basic" hamburger (and it gives you a SubCategory assumption to make this, say, a McDonald's hamburger). It also gives you an Attribute assumption to modify details like patty size and whether it has condiments. Here is the Attribute <assumption> element from the "hamburger" query:

```
<assumption type='Attribute' word='Hamburger' count='9'>
   <value name='{Food:FoodSize -&gt; Food:LargePatty, Food:PattyCount -&gt; Food:Single,
Food:Variety -&gt; Food:FastFood, Food:Variety -&gt; Food:Plain}'
       desc='hamburger, fast food, large patty, plain, single'
       input='*EAC.ExpandedFood.Hamburger-
_**Hamburger.*Food%3AFoodSize_Food%3ALargePatty.Food%3APattyCount_Food%3ASingle.Food%3AVariet
y_Food%3AFastFood.Food%3AVariety_Food%3APlain---' />
   <value name='{Food:FoodSize -&gt; Food:RegularPatty, Food:PattyCount -&gt; Food:Single,
Food:Variety -&gt; Food:FastFood, Food:Variety -&gt; Food:Plain}'
       desc='hamburger, fast food, plain, regular patty, single'
       input='*EAC.ExpandedFood.Hamburger-
_**Hamburger.*Food%3AFoodSize_Food%3ARegularPatty.Food%3APattyCount_Food%3ASingle.Food%3AVari
ety_Food%3AFastFood.Food%3AVariety_Food%3APlain---' />
   <value name='{Food:AddedFoods -&gt; Food:WithCondiments, Food:FoodSize -&gt;
Food:LargePatty, Food:PattyCount -&gt; Food:Single, Food:Variety -&gt; Food:FastFood}'
       desc='hamburger, fast food, large patty, single, with condiments'
       input='*EAC.ExpandedFood.Hamburger-
_**Hamburger.*Food%3AAddedFoods_Food%3AWithCondiments.Food%3AFoodSize_Food%3ALargePatty.Food%
3APattyCount_Food%3ASingle.Food%3AVariety_Food%3AFastFood---' />
   <value name='{Food:AddedFoods -&gt; Food:WithCondiments, Food:FoodSize -&gt;
Food:RegularPatty, Food:PattyCount -&gt; Food:Single, Food:Variety -&gt; Food:FastFood}'
       desc='hamburger, fast food, regular patty, single, with condiments'
       input='*EAC.ExpandedFood.Hamburger-
_**Hamburger.*Food%3AAddedFoods_Food%3AWithCondiments.Food%3AFoodSize_Food%3ARegularPatty.Foo
d%3APattyCount_Food%3ASingle.Food%3AVariety_Food%3AFastFood---' />

   ... several more <value> elements deleted for brevity ...

</assumption>
```

The name attributes for Attribute assumptions can become rather cryptic, but the desc attributes are much clearer to understand.

TimeAMOrPM

When Wolfram|Alpha recognizes a string in a query as referring to a time, and it is ambiguous as to whether it represents AM or PM, a TimeAMOrPM assumption is generated. Here is the <assumption> element from the query "3:00":

```
<assumption type='TimeAMOrPM' count='2'>
   <value name='pm' desc='PM' input='TimeAMOrPM_pm' />
   <value name='am' desc='AM' input='TimeAMOrPM_am' />
</assumption>
```

There are always two <value> elements in this assumption: one for AM and one for PM. As always, the first-listed one is the current value for the assumption, and this will depend on what time of day the query is executed.

DateOrder

When Wolfram|Alpha recognizes a string in a query as referring to a date in numerical format, and it is ambiguous as to the order of the day, month, and year elements (such as 12/11/1996), a DateOrder assumption is generated. Here is the <assumption> element from the query "12/11/1996":

```
<assumption type='DateOrder' count='2'>
  <value name='MonthDayYear' desc='month/day/year' input='DateOrder_**Month.Day.Year--' />
  <value name='DayMonthYear' desc='day/month/year' input='DateOrder_**Day.Month.Year--' />
</assumption>
```

The number and order of <value> elements depends on specifics of the date string in the query, and also on the locale of the caller. The name attributes will be a combination of Day, Month, and Year in the corresponding order.

ListOrTimes

The ListOrTimes assumption is generated when a query contains elements separated by spaces and it is unclear whether this is to be interpreted as multiplication or a list of values. For example, the query "3 x" is interpreted as 3*x, but it could also be the list {3, x}. Here is the ListOrTimes <assumption> element from that query:

```
<assumption type='ListOrTimes' count='2'>
   <value name='Times' desc='multiplication' input='ListOrTimes_Times' />
   <value name='List' desc='a list' input='ListOrTimes_List' />
</assumption>
```

The ListOrTimes assumption always has two <value> elements: one named "List" and one named "Times." There is no word attribute in the <assumption> element for this type.

ListOrNumber

The ListOrNumber assumption is generated when a query contains a string that could be either a number with a comma as a thousands separator, or a list of two separate numbers, such as the query "1,234.5." Here is the ListOrNumber <assumption> element from that query:

```
<assumption type='ListOrNumber' word='1,234.5' count='2'>
  <value name='Number' desc='number' input='ListOrNumber_*Number.1%2C234%21.5-' />
  <value name='List' desc='list' input='ListOrNumber_*List.1%2C234%21.5-' />
</assumption>
```

The ListOrNumber assumption always has two <value> elements: one named "List" and one named "Number."

CoordinateSystem

The CoordinateSystem assumption is generated when it is ambiguous which coordinate system a query refers to. For example, the query "div(x rho,y z,z x)" mixes elements from standard notation for 3D Cartesian coordinates and cylindrical coordinates. Here is the CoordinateSystem <assumption> element from that query:

```
<assumption type='CoordinateSystem' count='2'>
  <value name='Cartesian3D' desc='3D Cartesian coordinates'
input='CoordinateSystem_*Cartesian3D-' />
  <value name='Cylindrical3D' desc='cylindrical coordinates'
input='CoordinateSystem_*Cylindrical3D-' />
</assumption>
```

The possible values for the name attribute are Cartesian2D, Cartesian3D, Polar2D, Cylindrical3D, Spherical3D, General2D, and General3D. There is no word attribute in the <assumption> element for this type.

I

The I assumption is generated when a query uses "i" in a way that could refer to a simple variable name (similar to, say, "x") or the mathematical constant equal to the square root of -1. Here is what this assumption looks like:

```
<assumption type='I' count='2'>
  <value name='ImaginaryI' desc='i is the imaginary unit' input='i_ImaginaryI' />
  <value name='Variable' desc='i is a variable' input='i_Variable' />
</assumption>
```

The I assumption always has two <value> elements: one named "ImaginaryI" and one named "Variable". There is no word attribute in the <assumption> element for this type, as it always refers to the letter "i".

NumberBase

The NumberBase assumption is generated when a number could be interpreted as being written in more than one base, such as "100110101", which looks like a binary number (base 2) but could also be base 10 (it could be other bases as well, but those are rarely used and thus do not occur as assumption values). Here is the NumberBase <assumption> element from that query:

```
<assumption type='NumberBase' word='100110101' count='2'>
  <value name='Decimal' desc='decimal' input='NumberBase_*Decimal.100110101-' />
  <value name='Binary' desc='binary' input='NumberBase_*Binary.100110101-' />
</assumption>
```

At the present time, the only possible <value> elements for this assumption are "Decimal" and "Binary".

MixedFraction

The MixedFraction assumption is generated when a string could be interpreted as either a mixed fraction or a multiplication, such as "3 1/2". Here is the MixedFraction <assumption> element from that query:

```
<assumption type='MixedFraction' word='3 1/2' count='2'>
  <value name='Mix' desc='mixed fraction' input='MixFrac_*Mix.3+1%2F2-' />
  <value name='Mult' desc='product' input='MixFrac_*Mult.3+1%2F2-' />
</assumption>
```

The MixedFraction assumption always has two <value> elements: one named "Mix" and one named "Mult."

MortalityYearDOB

The MortalityYearDOB assumption is a very specialized type generated in some mortality-related queries, such as "life expectancy France 1910". The year 1910 could refer to the year of the data (that is, life expectancy data from France in the year 1910), or the year of birth ("life expectancy data in France for people born in 1910"). The MortalityYearDOB assumption distinguishes between those two meanings. Here is the <assumption> element from that query:

```
<assumption type='MortalityYearDOB' word='1910' count='2'>
  <value name='Year' desc='the year of the data' input='MortYrDOB_*Yr.1910-' />
  <value name='DateOfBirth' desc='the year of birth' input='MortYrDOB_*DOB.1910-' />
</assumption>
```

The MortalityYearDOB assumption always has two <value> elements: one named "Year" and one named "DateOfBirth."

DNAOrString

The DNAOrString assumption is generated when a query could be interpreted as a sequence of DNA bases or just a string of characters, such as "AGGGAAAA". Here is the DNAOrString <assumption> element from that query:

```
<assumption type='DNAOrString' count='2'>
  <value name='DNA' desc='genome sequence' input='DNAString_DNA' />
  <value name='String' desc='text string' input='DNAString_String' />
</assumption>
```

The DNAOrString assumption always has two <value> elements: one named "DNA" and one named "String." There is no word attribute in the <assumption> element for this type.

TideStation

The TideStation assumption is generated in tide-related queries. It distinguishes between different tide stations. Here is an example from the query "tides Seattle", which is the query that was used to generate Figure 1 at the beginning of this document:

```
<assumption type='TideStation' count='5'>
  <value name='PrimaryStation'
       desc='nearest primary station'
       input='TideStation_PrimaryStation' />
  <value name='NearestStation'
       desc='nearest station'
       input='TideStation_NearestStation' />
  <value name='Seattle, Washington (1.7 mi)'
       desc='Seattle, Washington (1.7 mi)'
       input='TideStation_*UnitedStates .9447130.PrimaryStation-' />
  <value name='Bangor, Washington (20.2 mi)'
       desc='Bangor, Washington (20.2 mi)'
       input='TideStation_*UnitedStates .9445133.PrimaryStation-' />
  <value name='Tacoma, Washington (25.2 mi)'
       desc='Tacoma, Washington (25.2 mi)'
       input='TideStation_*UnitedStates .9446484.PrimaryStation-' />
</assumption>
```

Formula-Related Types (FormulaSelect, FormulaSolve, FormulaVariable, FormulaVariableOption, FormulaVariableInclude)

These assumption types are described in the earlier section "Formulas with Input Fields".