Wolfram|Alpha Full Results API *Reference*

The Wolfram|Alpha Full Results 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. It is implemented in a standard REST protocol using HTTP GET requests. Each result is returned as a descriptive XML or JSON structure wrapping the requested content format.

Although the majority of data available through the Wolfram|Alpha website is also available through this API, certain subjects may be restricted by default. To request access to additional topics, contact us. Use of the Full Results API is subject to the API Terms of Use.

Get Started | Contact Us | API Explorer

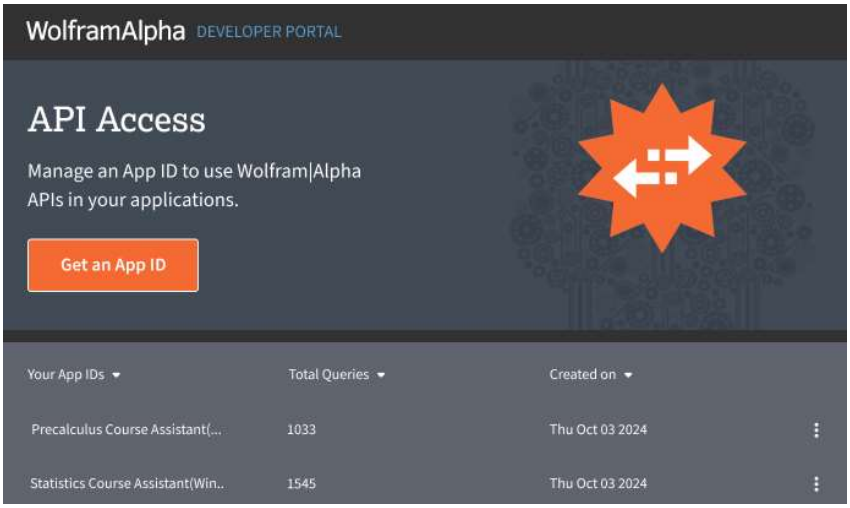
Language Libraries

Getting Started

Start using the Full Results API in minutes by following these basic steps.

Signup and Login

To get started, you must register a Wolfram ID and sign in at the Wolfram|Alpha Developer Portal.



Obtaining an AppID

Click the "Get an AppID" button to get your first AppID button to start the app creation process. Give your application a name, a simple description and select which app type to register an AppID. Each application must have its own unique AppID.

Sample Query

Now that you have an AppID, you can make your first query. The base URL for queries is:

```
http://api.wolframalpha.com/v2/query
```

Every query requires two pieces of information—an AppID and an input value—in order to be processed correctly. The appid parameter tells your query which AppID to use:

```
http://api.wolframalpha.com/v2/query?appid=DEMO
```

Next, use the input parameter to specify the URL-encoded input for your query. For instance, here is a query for “population of France”:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=population%20of%20france
```

When executed with a valid AppID, this URL will return an XML document with informational elements (referred to as pods relating to the input. Here is the XML output for the "population of France" query, with most elements collapsed for brevity:

```
<queryresult success="true" error="false" numpods="5"
datatypes="Country"
timedout="Data,Percent,Unit,AtmosphericProperties,Uni
tInformation,Music,Geometry" timedoutpods=""
timing="6.272" parsetiming="0.27"
parsetimedout="false" version="2.6">
    <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
    <pod title="Result" scanner="Data" id="Result"
position="200" error="false" numsubpods="1"
primary="true">
    <subpod title="">
        <plaintext>
        64.1 million people (world rank: 21st) (2014
estimate)
        </plaintext>
        <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP29
1g37h8915724h4b800004ec2h0de24da9sbp?
MSPStoreType=image/gif&s=12" alt="64.1 million people
(world rank: 21st) (2014 estimate)" title="64.1
million people (world rank: 21st) (2014 estimate)"
width="313" height="18"/>
    </subpod>
    </pod>
    <pod title="Recent population history"
scanner="Data"
```

```
id="RecentHistory:Population:CountryData"
position="300" error="false" numsubpods="1">...</pod>
    <pod title="Long-term population history"
scanner="Data"
id="LongTermHistory:Population:CountryData"
position="400" error="false" numsubpods="1">...</pod>
    <pod title="Demographics" scanner="Data"
id="DemographicProperties:CountryData" position="500"
error="false" numsubpods="1">...</pod>
    <warnings count="1">...</warnings>
    <sources count="1">...</sources>
    </queryresult><queryresult success="true"
error="false" numpods="5" datatypes="Country"
timedout="Data,Percent,Unit,AtmosphericProperties,Uni
tInformation,Music,Geometry" timedoutpods=""
timing="6.272" parsetiming="0.27"
parsetimedout="false" version="2.6">
    <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
    <pod title="Result" scanner="Data" id="Result"
position="200" error="false" numsubpods="1"
primary="true">
    <subpod title="">
        <plaintext>
        64.1 million people (world rank: 21st) (2014
estimate)
        </plaintext>
        <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP29
1g37h8915724h4b800004ec2h0de24da9ife?
MSPStoreType=image/gif&s=12" alt="64.1 million people
(world rank: 21st) (2014 estimate)" title="64.1
million people (world rank: 21st) (2014 estimate)"
width="313" height="18"/>
    </subpod>
    </pod>
    <pod title="Recent population history"
scanner="Data"
id="RecentHistory:Population:CountryData"
position="300" error="false" numsubpods="1">...</pod>
    <pod title="Long-term population history"
scanner="Data"
id="LongTermHistory:Population:CountryData"
position="400" error="false" numsubpods="1">...</pod>
    <pod title="Demographics" scanner="Data"
id="DemographicProperties:CountryData" position="500"
error="false" numsubpods="1">...</pod>
    <warnings count="1">...</warnings>
    <sources count="1">...</sources>
    </queryresult>
```

Formatting Input

All URLs used to make queries must be URL encoded (e.g. spaces represented as "%20" and backslashes represented as "%5c").

For mathematical queries, Wolfram|Alpha will also accept input formatted using presentation LaTeX or MathML. This can be useful when passing information back and forth between the API and a website or application using one of these formats.

Adding Parameters

You can add URL-encoded parameters to customize output. For instance, if you only wanted the "Result" pod from the above output, you could use the includepodid parameter:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=population%20france&includepodid=Res
ult
```

This way, only pods with that exact ID are returned in the XML output:

```
<queryresult success="true" error="false" numpods="1"
datatypes="" timedout="" timedoutpods=""
timing="0.895" parsetiming="0.277"
parsetimedout="false" recalculate=""
id="MSPa201c626hgh07fd2ee900003c1966c16708edi1"
host="http://www1.wolframalpha.com" server="13"
related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa211c626hgh07fd2ee9000368i61d9578b12h4&s=13"
version="2.6">
    <pod title="Result" scanner="Data" id="Result"
position="100" error="false" numsubpods="1"
primary="true">
        <subpod title="">
        <plaintext>
            64.1 million people (world rank: 21st)
(2014 estimate)
        </plaintext>
        <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP22
1c626hgh07fd2ee900004eagdagf39fchhhg?
MSPStoreType=image/gif&s=13"
            alt="64.1 million people (world rank:
21st) (2014 estimate)"
            title="64.1 million people (world rank:
```

```
21st) (2014 estimate)"
            width="313" height="18"/>
        </subpod>
    </pod>
   <sources count="1">...</sources>
    </queryresult>
```

Note that this result returns both a `<plaintext>` element and an `<img>` element. You can select which output type you prefer using the format parameter:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=population%20france&includepodid=Res
ult&format=plaintext
```

By using parameters in your queries, you can reduce the output to just the pieces you need. Notice how much shorter this output is than that of the original query:

```
<queryresult success="true" error="false" numpods="1"
datatypes="" timedout="" timedoutpods=""
timing="0.723" parsetiming="0.266"
parsetimedout="false" recalculate=""
id="MSPa1751g37h8915724h4b80000365icce2he8ca3g1"
host="http://www1.wolframalpha.com" server="12"
related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa1761g37h8915724h4b80000267ba77dbgig0651&s=12"
version="2.6">
    <pod title="Result" scanner="Data" id="Result"
position="100" error="false" numsubpods="1"
primary="true">
        <subpod title="">
        <plaintext>
            64.1 million people (world rank: 21st)
(2014 estimate)
        </plaintext>
        </subpod>
    </pod>    <sources count="1">...</sources>
    </queryresult>
```

A list of possible parameters is included below.

Parameter Reference

Basic Parameters

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
input	URL-encoded text specifying the input string	"5+largest+countries", "Doppler%20shift", "pascal%27s%20triangle"	N/A (C witho value
appid	An ID provided by Wolfram Research that identifies the application or organization making the request	X7WEHY-W45KYJL3C9	N/A (C witho will fa
format	The desired format for individual result pods	"image", "imagemap", "plaintext", "minput", "moutput", "cell", "mathml", "sound", "wav"	Retur and ir forma ("plai ")
output	The desired format for full results	"xml", "json"	Retur docur

Pod Selection

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
includepodid	Specifies a pod ID to include in the result	"Result", "BasicInformation:PeopleData", "DecimalApproximation"	All po
excludepodid	Specifies a pod ID to exclude from the result	"Result", "BasicInformation:PeopleData", "DecimalApproximation"	No po
podtitle	Specifies a pod title to include in the result	"Basic+Information", "Image", "Alternative%20representations"	All po

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
podindex	Specifies the index(es) of the pod(s) to return	"1", "7", "5,12,13"	All po
scanner	Specifies that only pods produced by the given scanner should be returned	"Numeric", "Data", "Traveling"	Pods f scann

Location

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
ip	Specifies a custom query location based on an IP address	"192.168.1.1", "127.0.0.1"	Use ca addre locati
latlong	Specifies a custom query location based on a latitude/longitude pair	"40.42,-3.71", "40.11, -88.24", "0,0"	Use ca addre locati
location	Specifies a custom query location based on a string	"Boston, MA", "The North Pole", "Beijing"	Use ca addre locati

Size

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
width	Specify an approximate width limit for text and tables	"200", "500"	Width pixels
maxwidth	Specify an extended maximum width for large objects	"200", "500"	Width pixels
plotwidth	Specify an approximate width limit for plots and graphics	"100", "200"	Plot w 200 pi
mag	Specify magnification of objects within a pod	"0.5", "1.0", "2.0"	Magni factor

Timeouts/Async

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
scantimeout	The number of seconds to allow Wolfram|Alpha to compute results in the "scan" stage of processing	"0.5", "5.0"	Scan s out af secon
podtimeout	The number of seconds to allow Wolfram|Alpha to spend in the	"0.5", "5.0"	Indivi time c secon

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
	"format" stage for any one pod		
formattimeout	The number of seconds to allow Wolfram|Alpha to spend in the "format" stage for the entire collection of pods	"0.5", "5.0"	Forma times secon
parsetimeout	The number of seconds to allow Wolfram|Alpha to spend in the "parsing" stage of processing	"0.5", "5.0"	Parsir times secon
totaltimeout	The total number of seconds to allow Wolfram|Alpha to spend on a query	"0.5", "5.0"	Queri after 2
async	Toggles asynchronous mode to allow partial results to return before all the pods are computed	"true", "false", "3.0"	Async mode ("false

Miscellaneous

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
reinterpret	Whether to allow Wolfram|Alpha to reinterpret	"true", "false"	Do no queri

PARAMETER NAME	FUNCTION	SAMPLE VALUES	DEFA
	queries that would otherwise not be understood		
translation	Whether to allow Wolfram|Alpha to try to translate simple queries into English	"true", "false"	Do no transl ("false
ignorecase	Whether to force Wolfram|Alpha to ignore case in queries	"true", "false"	Do no case (
sig	A special signature that can be applied to guard against misuse of your AppID	N/A	No sig applie
assumption	Specifies an assumption, such as the meaning of a word or the value of a formula variable	"*C.pi-_*Movie", "DateOrder_**Day.Month.Year--"	Assun made by the
podstate	Specifies a pod state change, which replaces a pod with a modified version, such as displaying more digits of a large decimal value	"WeatherCharts: WeatherData__Past+5+years", "2@DecimalApproximation__More+digits"	Pod st gener implic API
units	Lets you specify the preferred measurement system, either "metric" or "nonmetric" (US customary units)	"metric", "nonmetric"	Chose caller

List of XML Result Elements

This pseudo-XML shows the general hierarchy of XML results from the Full Results API. Note that some elements (e.g. <states>) can appear in multiple places. Additionally, not all these elements will be returned in one result (e.g. <tips> only appears for results not understood by the interpreter, so it would not be seen next to <pod> in a result).

```
<queryresult>
    <languagemsg>
    <tips>
        <tip>
    <futuretopic>
    <pod>
        <subpod>
            <img>
            <imagemap>
                <rect>
            <plaintext>
            <mathml>
                <math>
            <sound>
            <minput>
            <moutput>
            <cell>
                <![CDATA[...]]>
            <states>
                <state>
        <states>
            <state>
        <infos>
            <info>
        <error>
    <assumptions>
        <assumption>
    <examplepage>
    <warnings>
        <spellcheck>
        <delimiters>
        <translation>
        <reinterpret>
    <sources>
        <source>
    <generalization>
    <didyoumeans>
```

```
        <didyoumean>
    <error>
```

<queryresult>

<queryresult> contains the entire API result. The <queryresult> element is a superelement of all others listed. 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 (e.g. "Financial").
- **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 timeout query parameters).
- **parsetiming** — The time in seconds required by the parsing phase.
- **parsedtimeout** — 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 the main output of the Full Results API. Each pod contains a piece or category of information about the given query. It has the following attributes:

- **title** — The pod title, used to identify the pod and its contents.
- **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, and they form an increasing sequence from top to bottom.
- **scanner** — The name of the scanner that produced this pod. A general guide to the type of data it holds.
- **id** — A unique identifier for a pod, used for selecting specific pods to include or exclude.
- **numsubpods** — The number of subpod elements present.

<subpod>

Subelements of **<subpod>** that contain the results for a single subpod. <subpod> has a title attribute, which is usually an empty string because most subpods have no title.

HTML **<img>** elements suitable for direct inclusion in a webpage. 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. The **<img>** tag also contains the following attributes:

- **src** — The exact URL of the image being displayed, to be used for displaying the image.
- **alt** — Alternate text to display in case the image does not render correctly—usually the same as the `<plaintext>` representation of the image.
- **title** — Descriptive title for internal identification of an image—usually the same as the `<plaintext>` representation of the image.
- **width** — The width of the image in pixels; can be changed using the width control parameters.
- **height**: — The height of the image in pixels; scales depending on width setting.

<imagemap>

HTML image maps for visual elements that can be clicked to execute further queries. Contains `<rect>` subelements that specify the corners of clickable areas with coordinates (using the top-left corner as the origin).

<plaintext>

Textual representation of a single subpod. Only appears if the requested result formats include plain text. `<plaintext>` has no attributes or subelements.

<mathml>

Contains the MathML representation of a single subpod. MathML output opens with `<math>` and includes formatting subelements such as `<mfrac>` and `<msup>`. This element type will only appear if the requested result formats include mathml.

<sound>

HTML `<sound>` elements suitable for direct inclusion in a webpage. They point to stored sound files giving an audio representation of a single subpod. These elements only appear in pods if the requested result formats include sound or wav. The type attribute will tell whether the format is MIDI or WAV.

<minput>

Wolfram Language input that can be executed within a Wolfram Language environment to provide the result given in a single subpod.

<moutput>

Wolfram Language output representation of the result given in a single subpod.

<cell>

A Wolfram Language Cell expression that can be interpreted in a Wolfram Language environment to render exactly the same output

as the Wolfram|Alpha website for a single subpod. The expression is wrapped in a `<![CDATA[...]]>` element, and can sometimes span across multiple elements.

<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.

<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. It has the following attributes:

- **type** — Classification of an assumption that defines how it will function.
- **word** — The central word/phrase to which the assumption is applied.
- **template** — A statement outlining the way an assumption will be applied.
- **count** — Number of possible values available from an assumption.

<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.

<state>

The `<state>` element, always a subelement of `<states>`, contains a particular pod state for a single pod or subpod. It has a name attribute that describes the pod state and an input attribute that can be used to modify subsequent queries.,

<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.

<spellcheck>

The <spellcheck> element contains a warning about a word that was reinterpreted because Wolfram|Alpha concluded it was misspelled. It contains the following attributes:

- **word** — The exact word taken from the original query.
- **suggestion** — Wolfram|Alpha's suggested respelling of the word.
- **text** — The message normally displayed to the user when a word is reinterpreted because of a <spellcheck> warning.,

<delimiters>

The `<delimiters>` element contains a warning about mismatched parentheses or other bracketing elements that Wolfram|Alpha attempted to rectify. Currently, the only attribute is text, and its value is the message "An attempt was made to fix mismatched parentheses, brackets or braces."

<translation>

The `<translation>` element contains a warning about a phrase in a query that Wolfram|Alpha attempted to translate to English before computation. It has the following attributes:

- **phrase** — The exact phrase taken from the original query.
- **trans** — Wolfram|Alpha's suggested translation of the phrase.
- **lang** — The language from which Wolfram|Alpha attempted to translate.
- **text** — The message normally displayed on the Wolfram|Alpha website to inform the user of this warning (e.g. "Translating from German to 'weather today'").

<reinterpret>

The `<reinterpret>` element contains a warning about a query or part of a query that was reinterpreted by Wolfram|Alpha to provide more useful and/or relevant results. It has a text attribute

displaying the message "Using closest Wolfram|Alpha interpretation:", along with a list of `<alternative>` elements showing other possible interpretations.

<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 webpage of source information. See the "Sources" section for more details.

<infos>

The `<infos>` element contains `<info>` elements that contain pieces of information about the contents of a pod.

<languagemsg>

The `<languagemsg>` element appears if the input is recognized as a foreign language that is not supported. It includes messages in English and the unsupported language stating that the language is not yet supported.

<generalization>

The `<generalization>` element appears when Wolfram|Alpha recognizes that more information is available on a broader query than the one specified.

<tips>

The `<tips>` element contains various `<tip>` subelements with text offering pointers for better queries.

<didyoumeans>

The `<didyoumeans>` element contains various `<didyoumean>` elements with text suggesting alternative queries with similar spelling. Because it is difficult to verify the relevance of these suggestions, we recommend against using them in your implementations.

<futuretopic>

The `<futuretopic>` element appears when a query result refers to a topic still under development.

<examplepage>

The `<examplepage>` element appears when a query is recognized as a category for which a set of example queries has already been prepared.

Explanation of Pods

By default, the Wolfram|Alpha website returns a large set of categorized information related to a given query. Each category of output is contained in a rectangular pod, which in turn includes subpods with individual pieces of data. Every pod has a title (displayed in the top-left corner) and at least one subpod. Most information is displayed as a GIF image by default. Here is the main Wolfram|Alpha output for the "tides Seattle" query:

When you make a call to the Full Results API, it will return pod and subpod information as an XML document. Using only the required parameters, all pods from a particular query are returned in their default format and state. Here is the Full Results API output for the same query (with some XML elements truncated for brevity and readability):

```
<queryresult success="true" error="false" numpods="4"
datatypes="Tide" timedout="" timedoutpods=""
timing="2.041" parsetiming="0.159"
parsetimedout="false" recalculate=""
id="MSPa2841c605802big7c5h700001bi6h0g908bch9ei"
host="http://www1.wolframalpha.com" server="13"
related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa2851c605802big7c5h700003d89ihc926f55828&s=13"
version="2.6">
  <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
```

```
  <pod title="Result" scanner="Tide" id="Result"
position="200" error="false" numsubpods="1"
primary="true">
    <subpod title="">
      <plaintext>
        Tuesday, February 11, 2014 | | | |
        high tide | 4:06 am PST (4 h 10 min ago) |
+11 feet | | | | | |
        low tide | 9:32 am PST (1 h 16 min from now)
| +6.7 feet | | | | | |
        high tide | 2:10 pm PST (5 h 55 min from now)
| +9.9 feet | | | | | |
        low tide | 8:59 pm PST  | +0.5 feet | | | | |
|
        (computed using historical data, not taking
into account weather, etc. heights relative to all-
time average lowest daily tide)
      </plaintext>
      <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP28
71c605802big7c5h70000185h5d64e91gb027?
MSPStoreType=image/gif&s=13" alt=" ... " title=" ...
" width="400" height="326"/>
    </subpod>
    <states count="5">...</states>
    <infos count="1">...</infos>
  </pod>
  <pod title="Averages" scanner="Tide" id="Averages"
position="300" error="false" numsubpods="1">...</pod>
  <pod title="Tide measurement station"
scanner="Tide" id="TideMeasurementStation"
position="400" error="false" numsubpods="1">...</pod>
  <assumptions count="1">...</assumptions>
</queryresult>
```

Pod Selection

It is often the case that applications using the Full Results 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 following sections describe various parameters that you can use to control the set of pods returned for a given query. These parameters can be used together in the same query, in which case a pod will be included in the result if it meets any (not necessarily all) of the included specifications.

Selecting Pods by ID

The most robust way to identify an individual pod is by its ID, which is given by its **id** attribute. Because the title parameter can vary in its wording (which may include time, location or other query-specific information), it is not effective at identifying pods by general content. Similarly, the index parameter may change from query to query or as the API is updated, reducing its usefulness in broader contexts. Using the pod ID ensures consistent output for a category of queries.

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" (with some elements collapsed for brevity):

```
<pod title="Latest recorded weather for Chicago,
Illinois" scanner="Data"
id="InstantaneousWeather:WeatherData" position="200"
error="false" numsubpods="1" primary="true">
  <subpod title="">
    <plaintext>
      temperature | 73 \u005c[Degree]F (wind chill:
73 \u005c[Degree]F)
      conditions | partly cloudy
      relative humidity | 29% (dew point: 39
\u005c[Degree]F)
      wind speed | 17 mph (51 minutes ago)
    </plaintext>
    <img ... />
  </subpod>
  <states count="2">...</states>
  <infos count="1">...</infos>
</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?
appid=DEMO&input=weather+Chicago&includepodid=InstantaneousWeather: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.

Selecting Pods by Title

You can also choose to identify pods by their titles. The title is the text that appears in the upper-left corner of each pod. Note that the trailing colon is not part of the title. The **podtitle** parameter 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" (using a wildcard to match the rest of the title because it ends with the requested location, which may change from query to query).

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=weather+Chicago&podtitle=Weather+station+information&podtitle=Latest+recorded+weather*
```

Selecting Pods by Index

Another way to specify pods is by their indices (**podindex**). The index is an incremental count representing the order in which pods were computed by Wolfram|Alpha, starting at 1. 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. Additionally, pod indices are not explicitly stated in the XML output, making it difficult to track what pods should be returned for a given query. 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**, 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.

Selecting Pods by Scanner

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`.

Formatting Output

The Output Parameter

By default, the Full Results API returns an XML document with individual elements representing the different parts of a

Wolfram|Alpha result. Using the output parameter, you can tell the API to return a JSON result instead:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=tides%20seattle&output=json
```

Here is the "Result" pod from this query:

```
{
    "title" : "Result",
    "scanner" : "Tide",
    "id" : "Result",
    "position" : 200,
    "error" : false,
    "numsubpods" : 1,
    "primary" : true,
    "subpods" : [
        { ... }
    ]
 }
```

Note that this parameter only changes the web format of the result; all pods have the same output elements and values regardless of which setting you use.

The Format Parameter

To change the output format of a pod or pods, use the **format** parameter in your URL. Not all results are available in all the formats listed below; the parameter will apply itself to any pods that allow the specified format. Multiple format options can be used in the same URL, separated by commas, and each applicable option will display in the result.

image

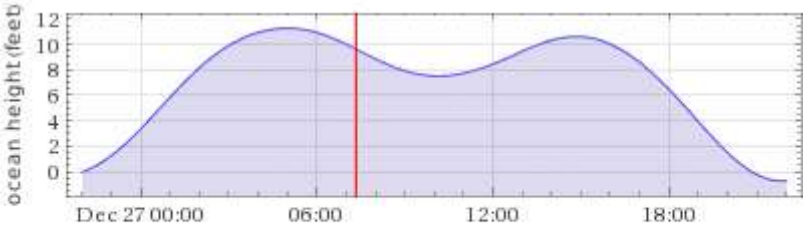
The **image** format option gives you the same types of GIF images as seen on the Wolfram|Alpha site. Each subpod is returned as an HTML <img> tag ready for direct inclusion in a webpage:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=tides%20seattle&&format=image
```

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. The image below shows the "Result" pod that is returned from the "tides Seattle" query, with the image URL highlighted:

```
<pod title="Result" scanner="Tide" id="Result"
position="200" error="false" numsubpods="1"
primary="true">
  <subpod title="">
    <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP22
51baf4ff97adai0b200001b128d6ace53249f?
MSPStoreType=image/gif&s=14"
         alt=" Tuesday, November 3, 2015 | | | | low
tide ... lowest daily tide)"
         width="400" height="366"/>
  </subpod>
  <states count="5">...</states>
  <infos count="1">...</infos>
</pod>
```

Loading this URL (or embedding the <img> tag in an HTML document) gives you back the exact image that would normally be displayed inside that pod:



Tuesday, December 27, 2016		
high tide	5:00 am PST (2 h 20 min ago)	+ 11.2 feet
low tide	10:08 am PST (2 h 48 min from now)	+ 7.5 feet
high tide	2:51 pm PST (7 h 31 min from now)	+ 10.6 feet
low tide	9:51 pm PST	− 0.8 feet

(computed using historical data, not taking into account weather, etc. heights relative to all-time average lowest daily tide)

The HTML alt text for the image is usually the same text returned by using the plaintext format option.

imagemap

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. This is useful for dynamic visual elements with different parts that 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 (in addition to image):

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

For reference, here is the "Name" pod on the website, with the links highlighted as they are when the mouse is hovering over the pod:

This 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/MSP93
ff?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.CountryProper
ty.FullName-'
         title='France full name' />
       <rect left='39' top='8' right='76' bottom='28'
         query='France+full+name'

assumptions='ClashPrefs_*Country.France.CountryProper
ty.FullName-'
         title='France full name' />
       <rect left='12' top='42' right='39'
bottom='62'
         query='France+full+native+name'

assumptions='ClashPrefs_*Country.France.CountryProper
ty.FullNativeNames-'
         title='France full native name' />
       <rect left='39' top='42' right='83'
bottom='62'
         query='France+full+native+name'

assumptions='ClashPrefs_*Country.France.CountryProper
ty.FullNativeNames-'
         title='France full native name' />
       <rect left='83' top='42' right='120'
bottom='62'
         query='France+full+native+name'

assumptions='ClashPrefs_*Country.France.CountryProper
ty.FullNativeNames-'
         title='France full native name' />
       <rect left='12' top='76' right='68'
bottom='96'
         query='France+internet+code'

assumptions='ClashPrefs_*Country.France.CountryProper
ty.InternetCode-'
         title='France internet code' />
       <rect left='68' top='76' right='98'
bottom='96'
         query='France+internet+code'

assumptions='ClashPrefs_*Country.France.CountryProper
ty.InternetCode-'
         title='France internet code' />
     </imagemap>
   </subpod>
   <states count='1'>
     <state name='More'
         input='Identifiers:CountryData__More' />
```

```
    </states>
  </pod>
```

The <imagemap> element contains 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. Notice that in this case, seven <rect> elements appear within the <imagemap> tag—each word in the boxes is considered 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 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.

The assumptions attribute gives a value you would include in the query to ensure that it is interpreted as desired. For example, to simulate 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?
appid=DEMO&input=France+full+name&assumption=ClashPre
fs_*Country.France.CountryProperty.FullName-
```

See the Assumptions section for more information about using assumptions to customize output.

plaintext

The **plaintext** format option produces the text format that you see when you highlight a pod and click the "Plaintext" button on the Wolfram|Alpha site. The result is contained in the <plaintext> tag and can be extracted as-is.

```
<pod title="Result" scanner="Tide" id="Result"
position="200" error="false" numsubpods="1"
primary="true">
  <subpod title="">
    <plaintext>
      Tuesday, November 3, 2015 | | | |
      low tide | 3:27 am PST (9 h 58 min ago) | +1.1
feet | | | | | |
      high tide | 11:02 am PST (2 h 23 min ago) |
+11.2 feet | | | | | |
      low tide | 5:42 pm PST (4 h 17 min
```

```
      from now) | +5.3 feet | | | | | |
      high tide | 10:20 pm PST (8 h 54 min from now)
| +7.9 feet | | | | | |
      (computed using historical data, not taking
into account weather, etc. heights relative to all-
time average lowest daily tide)
    </plaintext>
  </subpod>
  <states count="5">...</states>
  <infos count="1">...</infos>
</pod>
```

You may 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, a latitude-longitude pair, a table, etc.).

MathML

Some Wolfram|Alpha results are mathematical expressions or formulas that require traditional math notation to be readable (e.g. superscripts, fractions, integral signs). Presentation MathML is a W3C standard XML format for mathematics. Many browsers can render MathML (either natively or through a plugin), and it is widely used in scientific software and popular equation editors.

MathML output is provided when you request the **mathml** format type. The following is the "Statement" pod from the query "pythagorean theorem", using the mathml option:

```
<pod title="Statement" scanner="Data"
id="StatementPod:FamousMathProblem" position="200"
error="false" numsubpods="1">
  <subpod title="">
    <mathml>
      <math
xmlns="http://www.w3.org/1998/Math/MathML"
xmlns:mathematica="http://www.wolfram.com/XML/"
mathematica:form="StandardForm">
        <mrow>
          <mtext>For a right triangle with
legs</mtext>
          <mrow>
            <mi>a</mi>
          </mrow>
          <mtext>and</mtext>
          <mrow>
            <mi>b</mi>
```

```
            </mrow>
            <mtext>and hypotenuse</mtext>
            <mrow>
              <mrow>
                <mi>c</mi>
              </mrow>
              <mtext>,</mtext>
            </mrow>
            <mrow>
              <mrow>
                <mrow>
                  <mrow>
                    <msup>
                      <mi>a</mi>
                      <mn>2</mn>
                    </msup>
                    <mo>+</mo>
                    <msup>
                      <mi>b</mi>
                      <mn>2</mn>
                    </msup>
                  </mrow>
                  <mo>=</mo>
                  <msup>
                    <mi>c</mi>
                    <mn>2</mn>
                  </msup>
                </mrow>
              </mrow>
              <mo>.</mo>
            </mrow>
          </mrow>
        </math>
      </mathml>
    </subpod>
</pod>
```

When the MathML output (enclosed in the **<math>** tag) is parsed in a web document, it displays clean, formatted mathematical text:

For a right triangle with legs a and b and hypotenuse c, $a^2+b^2=(c^2)$.

Sound

Some pods have sounds associated with them. Using the **sound** format type will return audio data as either "audio/midi" (MIDI format) or "audio/x-wav" (WAV format), depending on the nature of the result.

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=C+major&podtitle=Music+notation&form
at=sound
```

```
<pod title="Music notation" scanner="Music"
id="MusicNotation" position="300" error="false"
numsubpods="0">
  <sounds count="1">
    <sound
url="http://www1.wolframalpha.com/Calculate/MSP/MSP38
31c605804acbi0aeb000016c4402g20b8c038?
MSPStoreType=audio/midi&s=13" type="audio/midi"/>
  </sounds>
</pod>
```

The MIME type is specified within the <sound> tag; in this case, the output is a MIDI object. As with the image format type, the highlighted link leads directly to an audio file that can be embedded into an HTML page using the <sound> tag. Note that the sound format will only provide output for queries that generate a sound as part of their natural output—it will not convert text or other visual formats to audio.

wav

For programs that can only handle WAV sounds, the **wav** format type returns audio data as an uncompressed audio file. Here is the result of using the wav option on the same query as above:

```
<pod title="Music notation" scanner="Music"
id="MusicNotation" position="300" error="false"
numsubpods="0">
  <sounds count="1">
    <sound
url="http://www1.wolframalpha.com/Calculate/MSP/MSP73
1g35g30d3e5h09e600005bhcec0af7cf0g5e?
MSPStoreType=audio/x-wav&s=12" type="audio/x-wav"/>
  </sounds>
</pod>
```

The pod is nearly identical, except that the audio type has changed to "audio/x-wav". This format type is a more robust way to retrieve sounds, since the WAV format is understood by most platforms. As

with the sound format, this will not convert text or other visual formats to audio.

Wolfram Language Input

This is the text format that you see in the "Wolfram Language plaintext input" popup that appears when you click some results on the Wolfram|Alpha site. Some results can be generated directly by single Wolfram Language input expressions. For example, the "Continued fraction" pod in the Wolfram|Alpha result for the query "pi" has a Wolfram Language input representation of ContinuedFraction[Pi, 27]. Use the **minput** option to get this type of result. This can be useful if you want to feed the input into Wolfram Language code for manipulating results:

```
<pod title="Continued fraction"
scanner="ContinuedFraction" id="ContinuedFraction"
position="500" error="false" numsubpods="1">
  <subpod title="">
    <minput>ContinuedFraction[Pi, 27]</minput>
  </subpod>
  <states count="2">...</states>
</pod>
```

Wolfram Language Output

Using the **moutput** option returns the text format that you see in the "Wolfram Language 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 the Wolfram Language 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 Wolfram Language 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, 15, 3}. This can be useful if you want to feed the output into a Wolfram Language environment:

```
<pod title="Continued fraction"
scanner="ContinuedFraction" id="ContinuedFraction"
position="100" error="false" numsubpods="1">
```

```
  <subpod title="">
    <moutput>
      {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, 15, 3}
    </moutput>
  </subpod>
  <states count="2">...</states>
</pod>
```

Note that the results from the minput type will evaluate directly to the results from the moutput type when entered in a Wolfram Language environment.

In[1]:= **ContinuedFraction[Pi, 27]**

Out[1]= {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, 15, 3}

Wolfram Language Cell Expression

The cell format type returns Wolfram Language Cell expressions. These are the same cells rendered by Wolfram|Alpha to produce the pod images you see on the website. Here is the **cell** output of the same pod from the "pi" query, with the Wolfram Language Cell expression highlighted:

```
<pod title="Continued fraction"
scanner="ContinuedFraction" id="ContinuedFraction"
position="500" error="false" numsubpods="1">
  <subpod title="">
    <cell compressed="false">
      <![CDATA[

Cell[BoxData[FormBox[TagBox[GridBox[List[List[Templat
eBox[List["\\"[\\"", "\\"
        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, 15,\\"", "\\" \\
[Ellipsis]]\\""],
        "RowDefault"]]], Rule[GridBoxAlignment,
List[Rule["Columns", List[List[Left]]]]],
        Rule[DefaultBaseStyle, "Column"],
Rule[GridBoxItemSize, List[Rule["Columns",
List[List[Scaled[1.003`]]]]]], "Column"],
        TraditionalForm]], "Output", List[],
Rule[PageWidth, 500], Rule[Magnification, 1],
Rule[CellMargins, List[List[0, 0], List[0, 0]]],
        Rule[ShowCellBracket, False],
Rule[FontFamily, "Bitstream Charter"], Rule[FontSize,
14], Rule[NumberSeparator, "\\[ThinSpace]"]]
      ]]>
    </cell>
```

```
  </subpod>
<states count="2">...</states>
</pod>
```

Although it can be cumbersome to look at, this cell statement is interpreted by the Wolfram Language to provide a parsed version of the pod content. In this particular case, the cell returned (when processed as a Wolfram Language statement) is essentially the plaintext result with added formatting:

[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, 15, 3, 13, ...]

The Wolfram Language has rich, direct integration with Wolfram|Alpha, using the API internally to process these cell expressions. If for any reason you want to obtain them outside of the Wolfram Language, you can request the cell format type.

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 four parameters you can use. The **width** and **maxwidth** parameters apply to images of text and tables, which are the majority of Wolfram|Alpha output, whereas **plotwidth** only applies to plots and other graphics. You can use the **mag** parameter to change the size of a pod's content without changing the size of the pod. It's important to note that these values are approximate, and the result of using these parameters may vary for different query types.

Width

The **width** parameter specifies a "hard limit" on the width of text and table elements. 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" scanner="Numeric"
id="DecimalApproximation" position="100"
error="false" numsubpods="1" primary="true">
  <subpod title="">
    <plaintext>
```

```
3.14159265358979323846264338327950288419716939937510
8...
    </plaintext>
    <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP29
1g37i75c7bh16e7b0000698324f506ghc9c5?
MSPStoreType=image/gif&s=12"

alt="3.14159265358979323846264338327950288419716939937
51058..."

title="3.14159265358979323846264338327950288419716939
93751058..."
         width="443" height="20"/>
  </subpod>
  <states count="1">...</states>
</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 443 pixels wide. If we try this query again and add the parameter `width=250`, here is the new pod:

```
<pod title="Decimal approximation" scanner="Numeric"
id="DecimalApproximation" position="100"
error="false" numsubpods="1" primary="true">
  <subpod title="">
    <plaintext>

3.14159265358979323846264338327950288419716939937510
8...
    </plaintext>
    <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP14
1g37i758a591379d00006ac21dae92c98172?
MSPStoreType=image/gif&s=12"
         alt="3.14159265358979323846264 3..."
         title="3.14159265358979323846264 3..."
         width="219" height="20"/>
  </subpod>
  <states count="1">...</states>
</pod>
```

With this smaller width requested, Wolfram|Alpha displays fewer digits, and the new image width is 219 pixels. This option works well for single-element or plain text results, since Wolfram|Alpha can choose to truncate the output. However, for more complex output, requesting a specific width can force elements to be

displayed in an undesirable way (e.g. mid-expression line breaks, awkward vertical spacing, etc.).

Max Width

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 undesirable line breaks and automatically reformat to your larger maxwidth. In other words, using width and maxwidth together lets you tell the API, "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."

Consider the effect of using this strategy on the "Doppler shift" query. If you wanted most output to be limited to 100 pixels but you realized that some elements would display poorly this way, you might set a maxwidth of 200.

Plot Width

The **plotwidth** parameter controls the width at which plots and graphics are rendered. The default value is 200 pixels. Many graphics in Wolfram|Alpha 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.

Magnification

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. For instance, if you specify `mag=2.0`, 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. This option is useful for displaying image elements like font sizes, tick marks, line-wrapping and spacing on high-density screens (e.g. tablets or mobile

phones). Since such elements can be very sensitive to formatting width, they may fit on a screen but be too small to read or interpret; magnification can "blow up" an image without changing its absolute size.

For instance, suppose you wanted to display the "Averages" pod from the "tides Seattle" query on a small vertical screen. Without any size parameters, this table is wider than it is tall:

range of tide (MN)	7.66 feet
average high tide (MHW)	+ 18.4 feet
average of high and low tides (MTL)	+ 14.6 feet
average low tide (MLW)	+ 10.8 feet

(all–time averages relative to a reference point at the measurement station)

This would be okay for a normal wide screen, but shrinking it to fit on a vertical screen would make the text too small to read. Combining the mag and width parameters (in this case, `"mag=2"` and `"width=400"`), you can increase the size of the text without changing the absolute width of the table:

range of tide (MN)	7.66 feet
average high tide (MHW)	+18.4 feet
average of high and low tides (MTL)	+14.6 feet
average low tide (MLW)	+10.8 feet

(all–time averages relative to a reference point at the measurement station)

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 the three forms listed.

Note that the units used in query results are altered by these location parameters. See the **Unit Assumption** subsection in the **Applying Assumptions** section for more information on changing units.

Semantic Location

The **location** parameter lets you specify a string like "Los Angeles, CA" or "Madrid" to be semantically interpreted as a location. This is essentially the same as calling the Wolfram Language's Interpreter function and using the first result as the location for the query.

Latitude and Longitude

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. Although this is the most exact way to specify a location, it will fail if no valid data is available nearby for the query.

IP Address

The **ip** parameter lets you set the IP address of the caller, which will be used to determine a location. With this parameter, if you are forwarding calls from your own web visitors to the Full Results API, you can propagate their IP addresses.

Combining Location Parameters

You can include multiple location parameters in your query, and they will be evaluated in order of precedence (location → latlong → ip) until a valid location is found. In this way, you can use extra location parameters as "backups" in case the initial specification fails. For instance, if you were executing a "weather" query using a location string, you might also want to include a default latitude/longitude location in case the given string is invalid:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=weather&format=image&location=the+mo
on&latlong=51.5,-0.1167
```

In this case, Wolfram|Alpha will attempt to parse the location string first. Since "the Moon" is not a valid location for a weather query, the given latitude/longitude will be used instead.

Informational Elements

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 the results from the "tides Seattle" query, the "Tide measurement" pod has buttons titled "Satellite image" and "Units". Clicking these types of buttons will cause either a popup window or a new browser window to appear with more information about the request:

The data for these informational 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>` XML element from this query:

```
<pod title="Tide measurement station" scanner="Tide"
id="TideMeasurementStation" position="400"
error="false" numsubpods="1">
  <subpod title="">...</subpod>
  <states count="2">...</states>
  <infos count="2">
    <info>
      <units count="1">
        <unit short="mi" long="miles"/>
        <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP12
31c61gh8510686 7i900006848e4e79bchdi50?
MSPStoreType=image/gif&s=13" width="88" height="26"/>
      </units>
    </info>
      <info>
        <link url="http://maps.google.com?
ie=UTF8&z=17&t=k&ll=47.6017%2C-
```

```
122.338&q=47.6017%20N%2C%20122.338%20W"
text="Satellite image"/>
      </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 (as shown above), and in other cases they are placed together on a single line separated by a vertical bar.

Here is another example of an <infos> element. This is the "Alternative representations" pod for the "pi" query:

```
<infos count="4">
  <info text="log(x) is the natural logarithm">
    <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP11
71bagf712c7721c3i000014840g8d3116fgd4?
MSPStoreType=image/gif&s=14"
         alt="log(x) is the natural logarithm"
         title="log(x) is the natural logarithm"
         width="198" 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.ht
ml" text="Definition" title="MathWorld"/>
  </info>
  <info text="i is the imaginary unit">
    <img
src="http://www1.wolframalpha.com/Calculate/MSP/MSP11
81bagf712c7721c3i00005a6fh9g2b203976h?
MSPStoreType=image/gif&s=14"
         alt="i is the imaginary unit"
         title="i is the imaginary unit"
         width="146" height="18"/>
    <link
url="http://reference.wolfram.com/mathematica/ref/I.h
tml" 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/MSP11
91bagf712c7721c3i0000395b11i0377aigfh?
MSPStoreType=image/gif&s=14"
         alt="cos^(-1)(x) is the inverse cosine
function"
         title="cos^(-1)(x) is the inverse cosine
function"
         width="275" height="18"/>
    <link
url="http://reference.wolfram.com/mathematica/ref/Arc
Cos.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/Sho
wAll.html" text="More information"/>
  </info>
</infos>
```

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 mouseover links:

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 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).

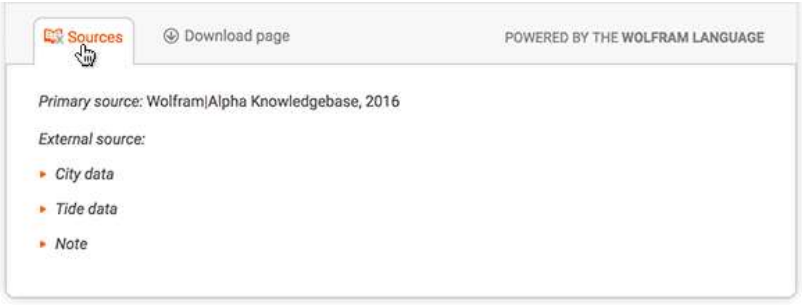
<units>

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.

Sources

In the bottom section of many Wolfram|Alpha results is a link titled "Sources". On the website this displays a popup window listing information about data sources that were either used to compute the result or could be referenced for more information. Here is the expanded "Sources" element for the "tides Seattle" query:



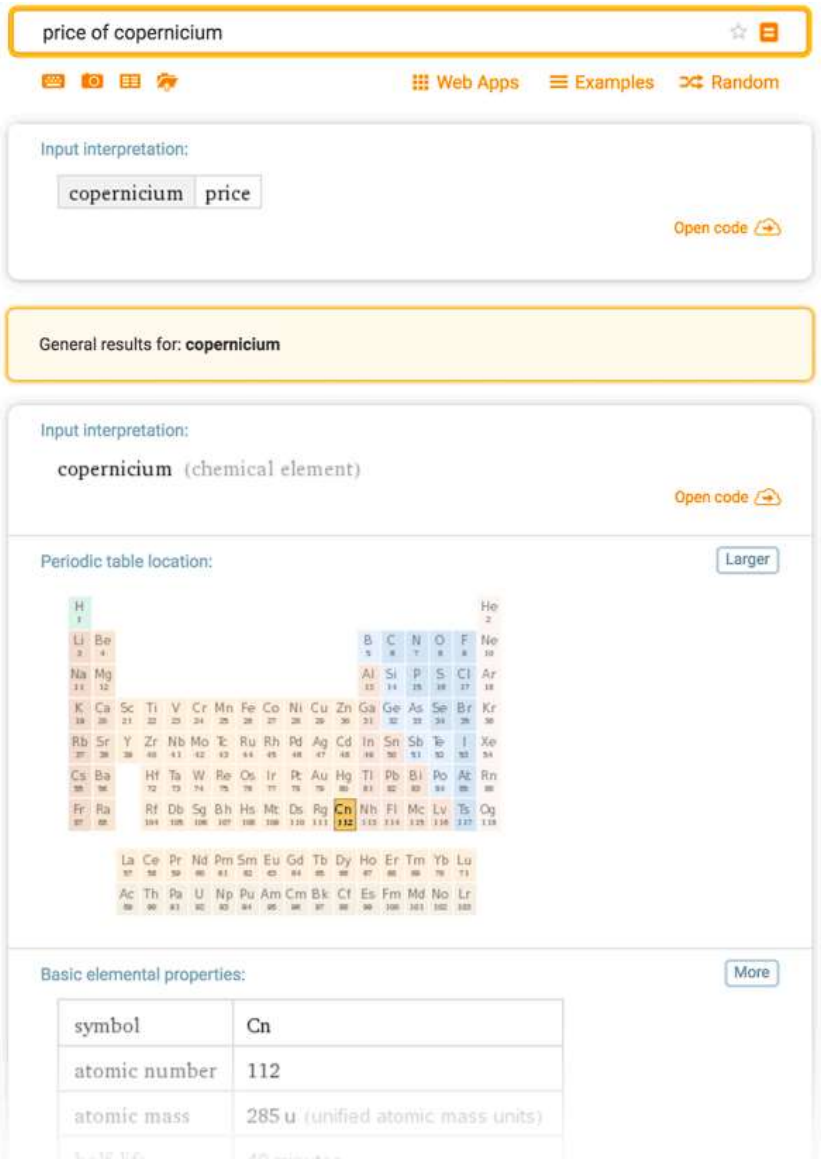
The API returns this information inside the `<sources>` element. The `<sources>` element contains a series of `<source>` subelements, each one defining a link to a webpage of source information:

```
<sources count="2">
  <source
url="http://www.wolframalpha.com/sources/CityDataSour
ceInformationNotes.html" text="City data"/>
  <source
url="http://www.wolframalpha.com/sources/TideDataSour
ceInformationNotes.html" text="Tide data"/>
</sources>
```

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

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": 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 the "copernicium" query:



The query returns two small pods, but suggests the generalization "copernicium", about which much more information is available. Here is the XML result generated by sending this query to the Full Results API:

```
<queryresult success="true" error="false" numpods="1"
datatypes="Element" timedout="" timedoutpods=""
timing="1.313" parsetiming="0.259"
parsetimedout="false" recalculate=""
id="MSPa641c6749c3894adgef000045hg0hc62e590bf6"
host="http://www1.wolframalpha.com" server="13"
version="2.6">
  <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
  <sources count="1">...</sources>
  <generalization topic="copernicium" desc="General
results for:"
url="http://www1.wolframalpha.com/api/v2/query?
```

```
id=MSPa671c6749c3894adgef0000469f1ii4h3ca513c&s=13"/>
</queryresult>
```

If you want to display the extra results available for that query or give users a chance to click something to request them, you can get the pods from the generalized query by calling the URL given in the url attribute (highlighted above). Note that you must append your own AppID to this URL:

```
http://www1.wolframalpha.com/api/v2/query?
id=MSPa1431c67498dg069cdf700004e52c8i1e6fbe3b0&s=13&a
ppid=DEMO
```

What comes back is a standard `<queryresult>` element with results from the "copernicium" query. Using the URL provided is faster than simply issuing a new query for "copernicium", since the work of parsing the query has already been performed.

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 Wolfram Language computations (e.g. "2+2") or simple data lookups (e.g. "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>
      $2.829 trillion per year (world rank: 6th)
(2014 estimate)
    </plaintext>
    <img src= ... />
</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.

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 (a different state). Clicking any of these buttons will recompute just that one pod to display different information. The API returns information about these pod states and allows you to programmatically invoke them (similar to **applying assumptions**, described later).

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"
id="DecimalApproximation" position="200"
error="false" numsubpods="1" primary="true">
  <subpod title="">
    <plaintext>

3.1415926535897932384626433832795028841971693993751058
8...
    </plaintext>
    <img ... />
  </subpod>
  <states count="1">
    <state name="More digits"
input="DecimalApproximation__More digits"/>
  </states>
</pod>
```

The <states> element contains all the alternative states available for that pod. The name of each state is the same as the text that appears on the Wolfram|Alpha 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?
appid=DEMO&input=pi&podstate=DecimalApproximation__Mo
re+digits
```

Note 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. The result from the above query will be exactly as if a website user had clicked the "More digits" button (notice the difference in the plaintext output):

```
<pod title="Decimal approximation" scanner="Numeric"
id="DecimalApproximation" position="200"
error="false" numsubpods="1" primary="true">
  <subpod title="">

<plaintext>3.14159265358979323846264338327950288419716
93993751058...</plaintext>
    <img ... />
  </subpod>
  <states count="2">
    <state name="Fewer digits"
input="DecimalApproximation__Fewer digits"/>
    <state name="More digits"
input="DecimalApproximation__More digits"/>
  </states>
</pod>
```

Changing the current state of a pod may also invoke more possible states—in this case, a "Fewer digits" state is now available, along with an extended "More digits" state. State changes can be chained together to simulate any sequence of button clicks. You can simulate clicking the "More digits" button twice as follows:

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

When invoking the same pod state 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?
appid=DEMO&input=pi&podstate=2@DecimalApproximation__
More+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"):

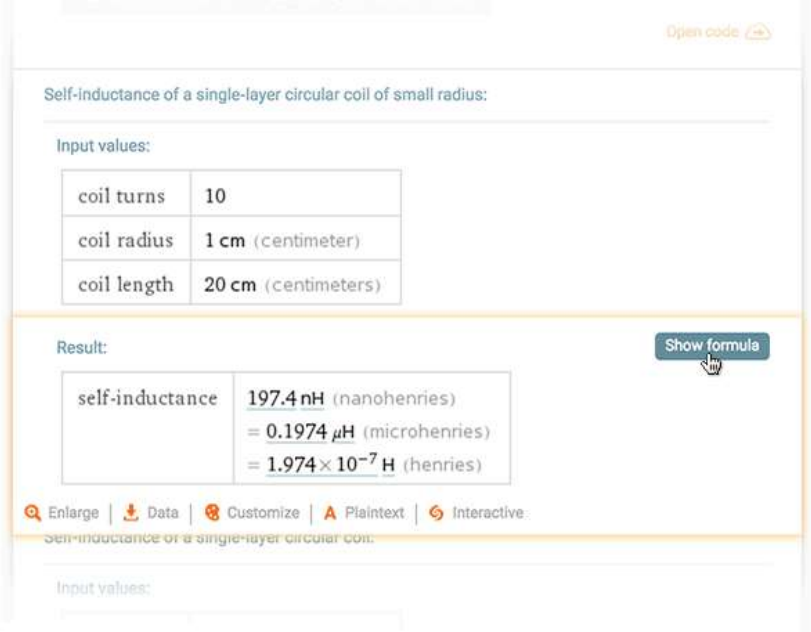
```
<states count="3">
  <statelist count="9" value="Current week"
delimiters="">
    <state name="Current week"
input="WeatherCharts:WeatherData__Current week"/>
    <state name="Current day"
input="WeatherCharts:WeatherData__Current day"/>
    <state name="Next week"
input="WeatherCharts:WeatherData__Next 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>
<state name="Show metric"
input="WeatherCharts:WeatherData__Show metric"/>
<state name="More"
input="WeatherCharts:WeatherData__More"/>
</states>
```

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=DEMO&podstate=WeatherCharts:WeatherData__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 of small radius", 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, each such subpod has its own `<states>` element. Here is the XML for that pod:

```
<pod title="Self-inductance of a single-layer
circular coil" scanner="Formula"
id="SelfInductanceSingleLayerCircularCoil"
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>
    <img src= ... />
  </subpod>
  <subpod title="Result">
    <plaintext>
      self-inductance | 189.3 nH (nanohenries)
      = 0.1893 muH (microhenries)
      = 1.893*10^-7 H (henries)
    </plaintext>
    <img src= ... />
    <states count="1">
      <state name="Show formula"
input="SelfInductanceSingleLayerCircularCoil__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?
appid=DEMO&input=weather&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.

Note: 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.

Using 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. Using the API, you can programmatically invoke these assumption values to alter the output of a query (similar to **pod states**, described above).

Listing of Assumption Types

This is a complete listing of possible assumptions, categorized by functionality. Each assumption name links to a more in-depth explanation of the parameter, its usage and available options.

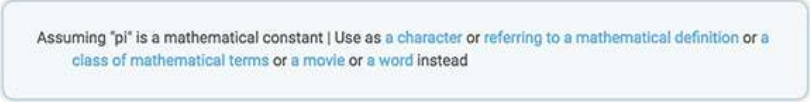
- Categorical assumptions
  - **Clash** — choose among possible interpretation categories for a query (e.g. "pi" could be a named constant or a movie)
  - **MultiClash** — choose among possible interpretation categories for in Structure of Assumptions dividual words in a query (e.g. in "log 0.5", "log" could be a PDF or a plotting function)
  - **SubCategory** — choose among similar results within the same interpretation category (e.g. "hamburger" could be a basic hamburger, McDonald's hamburger, etc.)
  - **Attribute** — choose specific traits of results within a SubCategory (e.g. for a "basic hamburger" you can specify patty size, condiments, etc.)
- Mathematical assumptions
  - **Unit** — choose among possible interpretations of a query as a unit of measure (e.g. "m" could be meters or minutes)
  - **AngleUnit** — choose whether to interpret angles in degrees or radians (e.g. "sin(30)" could be 30 degrees or 30 radians)
  - **Function** — choose among possible interpretations of a query as a mathematical function (e.g. "log" could be the natural logarithm or the base 10 logarithm)
  - **ListOrTimes** — choose whether to interpret space-separated elements as a list of items or as multiplication (e.g. "3 x" could be {3, x} or 3×x)
  - **ListOrNumber** — choose whether to treat commas between numbers as thousands separators or list delimiters (e.g. "1,234.5" could be 1234.5 or {1, 234.5})
  - **CoordinateSystem** — choose which coordinate system to use, given an ambiguous query (e.g. "div(x rho, y z, z x)" could refer to Cartesian3D or Cylindrical3D)

- **I** — choose whether to interpret "i" as a variable or as the imaginary unit (e.g. "5x+3i" could be a complex number with one variable or an expression with two variables)
- **NumberBase** — choose which number base to use, given an ambiguous query (e.g. "100110101" could be binary or decimal)
- **MixedFraction** — choose whether to interpret input as a mixed fraction or as multiplication (e.g. "3 1/2" could be 3½ or 3×½)

- Date and time assumptions
  - **TimeAMOrPM** — choose whether a time is in the morning or the afternoon (e.g. "3:00" could be 3am or 3pm)
  - **DateOrder** — choose the order of day, year and month elements in a date (e.g. "12/11/1996" could be Dec. 11, 1996, or Nov. 12, 1996)
  - **MortalityYearDOB** — choose whether a mortality rate refers to a data year or a birth year (e.g. "life expectancy France 1910" could refer to statistics from 1910 or those for people born in 1910)
- Scientific assumptions
  - **TideStation** — choose among tide stations in a tide query (e.g. "tides Seattle" could use Seattle, Bangor or Tacoma tide stations as a source)
- Formula input assumptions
  - **FormulaSelect** — choose among possible interpretations of a query as a mathematical formula (e.g. "Doppler shift" can refer to either the classical or relativistic Doppler shift)
  - **FormulaSolve** — choose which value to solve for in a formula (e.g. solving for sound speed in the Doppler shift formula)
  - **FormulaVariable** — supply a non-default value to a specific variable in a formula (e.g. give the "Doppler shift" query a source speed of 30 m/s)
  - **FormulaVariableOption** — choose among different representations of a variable or variables in a formula (e.g. the Doppler shift formula can split the frequency reduction factor into source and observed frequencies)
  - **FormulaVariableInclude** — choose extra variables to include in a formula (e.g. "Doppler shift" can also include the observer's speed and wind speed)

Structure of Assumptions

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. Clicking any of the links in this area will prompt Wolfram|Alpha to send a new query based on the given information. Here is the assumptions area from the result for "pi":



The API makes this same information available via the `<assumptions>` element, which, if present, is a top-level subelement of `<queryresult>`. Each assumption is represented as a separate `<assumption>` subelement containing a sequence of `<value>` subelements, one for each possible value of the assumption. 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. Here is the `<assumptions>` element in the "pi" query:

```
<assumptions count="1">
  <assumption type="Clash" word="pi"
template="Assuming "${word}" is ${desc1}. Use as
${desc2} instead" count="6">
    <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
mathematical definition" input="*C.pi-_*MathWorld-"/>
    <value name="MathWorldClass" desc="a class of
mathematical terms" input="*C.pi-_*MathWorldClass-"/>
    <value name="Movie" desc="a movie" input="*C.pi-
_*Movie-"/>
    <value name="Word" desc="a word" input="*C.pi-
_*Word-"/>
  </assumption>
</assumptions>
```

Each `<value>` element has three attributes: **name**, which is a unique internal identifier (often with some descriptive value to the programmer); **desc**, which is a textual description suitable for displaying to users; and **input**, which gives the exact parameter value needed to invoke this assumption in a subsequent query. The **template** attribute shows the natural language statement normally displayed by the Wolfram|Alpha website to allow users to alter the current query. In this example, ${word} refers to the input, "pi", ${desc1} refers to the current interpretation of "a mathematical constant" and ${desc2} refers to a new interpretation (e.g. "a character" or "a movie"). These values are populated from information within the XML, with each ${wordX} drawing from the "word" attribute of the `<assumption>` element and each ${descX} drawing from the "desc" attributes of individual `<value>`

elements. The first-listed <value> element always names the assumption value that was in effect for the current query.

Many other assumption types exist. For instance, Wolfram|Alpha interprets the query "12/5/1999" as a date; however, 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 to get the current result:

```
<assumption type="DateOrder" template="Assuming
${desc1}. Use ${desc2} instead" 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 next section includes a listing of the main assumption types that can be generated, with examples. Note that formula input assumptions are covered in the **Formulas with Input Fields** section.

Applying Assumptions

To apply an assumption in a query, use the **assumption** parameter. The value you pass for this parameter is the same as 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 assumption string here was taken from the earlier <assumptions> output for this query:

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

The values for the assumption parameter are complicated-looking strings, but you don't have to understand their syntax—just use the 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.

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, generated from the query "pi":

```
<assumptions count="1">
  <assumption type="Clash" word="pi"
template="Assuming "${word}" is ${desc1}. Use as
${desc2} instead" count="6">
    <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
mathematical definition" input="*C.pi-_*MathWorld-"/>
    <value name="MathWorldClass" desc="a class of
mathematical terms" input="*C.pi-_*MathWorldClass-"/>
    <value name="Movie" desc="a movie" input="*C.pi-
_*Movie-"/>
    <value name="Word" desc="a word" input="*C.pi-
_*Word-"/>
  </assumption>
</assumptions>
```

MultiClash

The **MultiClash** assumption is a type of clash where multiple overlapping strings can have different interpretations. An example is the query "delta sigma", where the whole phrase can be interpreted as a formula, or the word "delta" can be interpreted as a financial entity, a variable or an administrative division:

```
<assumptions count="1">
  <assumption type="MultiClash" word=""
template="Assuming ${word1} is referring to ${desc1}.
```

```
Use "${word2}" as ${desc2}. Use "${word3}" as
${desc3}. Use "${word4}" as ${desc4}." count="4">
    <value name="Financial" word="delta" desc="a
financial entity" input="*MC.%7E-_*Financial-"/>
    <value name="Variable" word="delta" desc="a
variable" input="*MC.%7E-_*Variable-"/>
    <value name="AdministrativeDivision" word="delta"
desc="an administrative division" input="*MC.%7E-
_*AdministrativeDivision-"/>
    <value name="Formula" word="" desc="a formula"
input="*MC.%7E-_*Formula-"/>
  </assumption>
</assumptions>
```

In contrast to Clash, the MultiClash assumption can interpret phrases in addition to individual words.

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"
template="Assuming ${desc1}. Use ${desc2} instead"
count="8">
  <value name="Hamburger" desc="hamburger"
input="*DPClash.ExpandedFoodE.hamburger-_*Hamburger-
"/>
  <value name="GroundBeefPatty" desc="ground beef
patty" input="*DPClash.ExpandedFoodE.hamburger-
_*GroundBeefPatty-"/>
  <value name="GroundBeef" desc="ground beef"
input="*DPClash.ExpandedFoodE.hamburger-_*GroundBeef-
"/>
   ... Remaining elements deleted for brevity ...
</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"
template="Assuming ${desc1}. Use ${desc2} instead"
count="9">
  <value name="Hamburger"
         desc="any type of hamburger"
         input="*EAC.ExpandedFood.Hamburger-
_**a.Hamburger--"/>
  <value name="{Food:FoodSize -> Food:LargePatty,
Food:PattyCount -> Food:Single, Food:Variety ->
Food:FastFood, Food:Variety -> Food:Plain}"
         desc="hamburger, fast food, large patty,
plain, single"
         input="*EAC.ExpandedFood.Hamburger-
_**Hamburger.*Food%3AFoodSize_Food%3ALargePatty.Food%
3APattyCount_Food%3ASingle.Food%3AVariety_Food%3AFast
Food.Food%3AVariety_Food%3APlain---"/>
  ... Remaining elements deleted for brevity ...
</assumption>
```

The names and input values for Attribute assumptions can become rather cryptic, but the desc attributes are always given in natural language for clarity.

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" template="Assuming
${desc1} for "${word}". Use ${desc2} instead"
count="3">
  <value name="Meters" desc="meters"
input="UnitClash_*m.*Meters--"/>
  <value name="MinimsUS" desc="US minims of volume"
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" template="Assuming
trigonometric arguments in ${desc1}. Use ${desc2}
instead" 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"
template="Assuming "${word}" is ${desc1}. Use
${desc2} instead" 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>
```

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" template="Assuming
${desc1}. Use ${desc2} instead" count="2">
  <value name="pm" desc="3:00 PM"
input="*TimeAMOrPM.*%7FAutomatic.%7FAutomatic.%7FAuto
matic.%7F3.%7F0.%7FAutomatic--_pm"/>
  <value name="am" desc="AM"
input="*TimeAMOrPM.*%7FAutomatic.%7FAutomatic.%7FAuto
matic.%7F3.%7F0.%7FAutomatic--_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 `<assumption>` assumption is generated. Here is the `<assumption>` element from the query "12/11/1996":

```
<assumption type="DateOrder" template="Assuming
${desc1}. Use ${desc2} instead" 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.

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"
template="Assuming ${word} is ${desc1}. Use as
${desc2} instead" 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."

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" template="Assuming
${desc1}. Use ${desc2} instead" 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"
template="Assuming ${word} is a ${desc1}. Use ${word}
as a ${desc2} instead" 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" template="Using
${desc1}. Use ${desc2} instead" 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" template="Assuming ${desc1}. Use
${desc2} instead" 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 "1011001110", 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="1011001110"
template="Assuming ${word} is ${desc1}. Use ${desc2}
instead" count="2">
  <value name="Binary" desc="binary"
input="NumberBase_*Binary.1011001110-"/>
  <value name="Decimal" desc="decimal"
input="NumberBase_*Decimal.1011001110-"/>
</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 as multiplication, such as

"3 1/2". Here is the MixedFraction `<assumption>` element from that query:

```
<assumption type="MixedFraction" word="3 1/2"
template="Assuming ${word} is a ${desc1}. Use ${word}
as a ${desc2} instead" 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".

TideStation

The **TideStation** assumption is generated in tide-related queries. It distinguishes between different tide stations. Here is an example from the "tides Seattle" query:

```
<assumption type="TideStation" template="Using
${desc1}. Use ${desc2} instead" 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 (1.4 mi)"
desc="Seattle, Washington (1.4 mi)"
input="TideStation_*UnitedStates.9447130.PrimaryStati
on-"/>
  <value name="Bangor, Washington (19.6 mi)"
desc="Bangor, Washington (19.6 mi)"
input="TideStation_*UnitedStates.9445133.PrimaryStati
on-"/>
  <value name="Tacoma, Washington (24.6 mi)"
desc="Tacoma, Washington (24.6 mi)"
input="TideStation_*UnitedStates.9446484.PrimaryStati
on-"/>
</assumption>
```

Note that the default station for these queries is the nearest station to the caller's location.

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.

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

```
<assumptions count="7">
  <assumption type="Clash" word="Doppler shift"
template="Assuming "${word}" is ${desc1}. Use as
${desc2} instead" count="3">...</assumption>
  <assumption type="FormulaSolve" template="Calculate
${desc1}" 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" template="Assuming
${desc1}. Use ${desc2} instead" 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"
template="Assuming ${desc1}. Use ${desc2} instead"
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"
template="Also include ${desc1}" 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

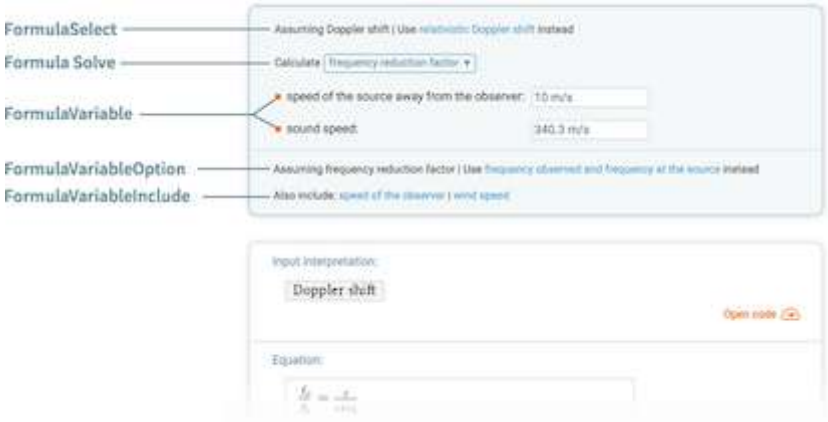
You apply formula-related assumptions just like other types of assumptions—by using the value of an `<assumption>` element's input attribute in a subsequent query. For example, to perform the Doppler shift query with wind speed as an added variable, you would use this URL:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=Doppler+shift&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.

Formula Assumption Types

The following is a screenshot from the website for the query "Doppler shift":



The box at the top is part of the standard Assumptions section, with labels showing the names the API uses for these assumption types. This particular example shows all the formula-related assumptions that Wolfram|Alpha can produce; not all formulas will have all of these assumptions available. Here is a guide to the five different formula assumption types:

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:

```
<assumption type="FormulaSelect" template="Assuming
${desc1}. Use ${desc2} instead" count="2">
  <value name="DopplerShift" desc="Doppler shift"
input="FSelect_*DopplerShift-"/>
  <value name="RelativisticDopplerShift"
desc="relativistic Doppler shift"
input="FSelect_**RelativisticDopplerShift--"/>
</assumption>
```

FormulaSolve

Formulas can be rearranged to solve for different variables. The **FormulaSolve** assumption lets you pick which one you want. In this example, the variables are the frequency reduction factor (f_0/f_n, 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:

```
<assumption type="FormulaSolve" template="Calculate
${desc1}" 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>
```

FormulaVariable

The **FormulaVariable** assumption lets you supply a value for a variable in a formula. It corresponds to an input field or pull-down menu of choices on the website. You have to understand a few extra details when applying a FormulaVariable assumption. Let's take a closer look at the `<assumption>` element from the "Doppler shift" query 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 `<assumption>` 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 pull-down 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", 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?
appid=DEMO&input=Doppler+shift&assumption=*F.DopplerS
hift.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 a

fillable form with several pull-down menus allowing you to choose whether the person in question is a smoker, diabetic, etc.:

Note that the fillable fields in this formula work as described in the previous example. However, the pull-down menus are derived from assumptions with only a few specific options available. Here is the XML returned by the API for one 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>
```

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 of `<assumption>` gives the index of the `<value>` element that is currently active (in this case, "1" for "male" and "2" for "female"). In this way, the natural order of the different values is preserved, without artificially moving the current value to the top of the list. Whether representing these options in a pull-down menu, as radio buttons, or using some other representation, it will probably make sense to preserve the order of the `<value>` elements. The process for applying a fixed FormulaVariable assumption is the same as for other types described in **the Using Assumptions section**.

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.

```
<assumption type="FormulaVariableInclude"
template="Also include ${desc1}" 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>
```

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_0/f_n) broken up into two separate variables (f_0 and f_n). You're not substituting a completely different formula (like FormulaSelect) or simply adding a new variable (like FormulaVariableInclude).

```
<assumption type="FormulaVariableOption"
template="Assuming ${desc1}. Use ${desc2} instead"
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>
```

Advanced Topics

Classifying Queries

If you are using the Full Results 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

Assumptions and their applications are described in detail above. The Clash assumption provides especially useful information about the meaning of a word or words in a query. Although this element does not provide generalized disambiguation, it can be helpful in discovering possible points of confusion in typical queries for your application.

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 uses 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 queryresult element:

```
<queryresult success="true" error="false" numpods="6"
datatypes="DNAString" timedout="" timedoutpods=""
timing="1.981" parsetiming="0.672"
parsetimedout="false" recalculate=""
  id="MSPa71g3d0a8ddia7d88i00001e5413i1ich9b439"
```

```
host="http://www1.wolframalpha.com" server="12"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa71g3d0a8ddia7d88i00001e5413i1ich9b439&s=12"
version="2.6">
```

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":

```
<queryresult success="true" error="false" numpods="4"
datatypes="Country,Language,WritingScript"
             timedout="" timedoutpods=""
timing="4.133" parsetiming="0.239"
parsetimedout="false" recalculate=""

id="MSPa591c678icaa2gbgchh00000i9d13be67bcb57b"
host="http://www1.wolframalpha.com" server="13"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa601c678icaa2gbgchh00004ib18g06hebf1706&s=13"
version="2.6">

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

    <assumption type="Clash" word="GATA"
template="Assuming "${word}" is ${desc1}. Use as
${desc2} instead" count="2">
      <value name="Language" desc="a language"
input="*C.GATA-_*Language-"/>
      <value name="DNAString" desc="a genome
sequence" input="*C.GATA-_*DNAString-"/>
    </assumption>
  </assumptions>
<sources count="2">...</sources>
</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.

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.

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 (e.g. food, airports, chemistry, music). The scanners produce data structures that will correspond to pods in the final output. For purely computational queries, the bulk of processing is

done during the scan stage. 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. You can use any combination of the following parameters to limit the processing time for your queries. All timeout values are in seconds.

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.

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. Most queries complete scanning in considerably less than the default of three seconds. If you stop the scanning 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 `<queryresult>` element contains a timedout attribute that gives a comma-separated sequence of names of scanners that timed out. 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.

Increasing the scantimeout parameter can allow more processing time for large computations. The query "10,000,000th Fibonacci number" times out during calculation with the default setting (notice the value of the timedout attribute):

```
<queryresult success="true" error="false" numpods="0"
datatypes=""

timedout="PowerTower,Numeric,List,Factorial"
timedoutpods="" timing="4.785" parsetiming="0.513"
parsetimedout="false"

recalculate="http://www1.wolframalpha.com/api/v2/reca
lc.jsp?
id=MSPa261g3d0ef867ddhf940000522c808eic67ab0d&s=12"

id="MSPa271g3d0ef867ddhf94000061a1ffic9c0h723a"
host="http://www1.wolframalpha.com" server="12"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa281g3d0ef867ddhf9400025i9chiafeb84fbb&s=12"
version="2.6">
<sources count="0"/>
</queryresult>
```

But adding `"scantimeout=8"` allows the computation to finish so that a result is obtained:

```
<queryresult success="true" error="false" numpods="2"
datatypes=""
            timedout="" timedoutpods=""
timing="7.741" parsetiming="0.572"
parsetimedout="false"

recalculate="http://www1.wolframalpha.com/api/v2/reca
lc.jsp?
id=MSPa621g3d0eeb27a6af7e00003f471cbb4a9h1081&s=12"

id="MSPa631g3d0eeb27a6af7e00005e4b5c3ea4ae97g8"
host="http://www1.wolframalpha.com" server="12"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa641g3d0eeb27a6af7e00004009i7df09d2egh3&s=12"
version="2.6">
  <pod title="Input" scanner="Identity" id="Input"
position="100" error="false" numsubpods="1">...</pod>
```

```
  <pod title="Decimal approximation"
scanner="Numeric" id="Result" position="200"
error="false" numsubpods="1" primary="true">
    <subpod title="">
      <plaintext>

1.12983437822539976031706363774586637294483 71... *
10^2089876
      </plaintext>
    </subpod>
    <states count="1">...</states>
  </pod>
  <sources count="0"/>
</queryresult>
```

For certain classes of queries, increasing the scantimeout parameter from the default value may also allow more pods to be generated. For example, a very general query such as "Maine" will generate a long list of information from many scanners that may take more than three seconds to process. By default, this query returns 8 pods with information about the US state of Maine. Increasing the scantimeout value by one second (`scantimeout=4`) nearly doubles the number of pods returned (additional pods are highlighted):

```
<queryresult success="true" error="false" numpods="8"
datatypes="City,USState"
             timedout="Data,Character"
timedoutpods="" timing="6.251" parsetiming="0.167"
parsetimedout="false"

recalculate="http://www1.wolframalpha.com/api/v2/reca
lc.jsp?
id=MSPa371c67962ihf94cb1h000011e033cac3f8316e&s=13"
id="MSPa381c67962ihf94cb1h00005944g42975582hg3"
             host="http://www1.wolframalpha.com"
server="13"
related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa391c67962ihf94cb1h00004igb226h2i54dacb&s=13"
version="2.6">

  <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
  <pod title="Basic information" scanner="Data"
id="BasicInformation:USStateData" position="200"
error="false" numsubpods="1" primary="true">...</pod>
  <pod title="Location" scanner="Data"
id="Location:USStateData" position="300"
```

```
error="false" numsubpods="1">...</pod>
  <pod title="Population" scanner="Data"
id="Demographics:USStateData" position="400"
error="false" numsubpods="1">...</pod>
  <pod title="Population history" scanner="Data"
id="PopulationHistory:USStateData" position="500"
error="false" numsubpods="1">...</pod>
  <pod title="Demographics" scanner="Data"
id="ACSPercentageEntrainments:ACSData" position="600"
error="false" numsubpods="1">...</pod>
  <pod title="Educational attainment" scanner="Data"
id="ACSEducationEntrainments:ACSData" position="700"
error="false" numsubpods="1">...</pod>
  <pod title="Geographic properties" scanner="Data"
id="GeographicProperties:USStateData" position="800"
error="false" numsubpods="1">...</pod>
  <pod title="Housing" scanner="Data"
id="Housing:USStateData" position="900" error="false"
numsubpods="1">...</pod>
  <pod title="Economic properties" scanner="Data"
id="Economy:USStateData" position="1000"
error="false" numsubpods="1">...</pod>
  <pod title="Crime statistics" scanner="Data"
id="StateCrimeInformation:CrimeData" position="1100"
error="false" numsubpods="1">...</pod>
  <pod title="Flag" scanner="Data"
id="Flag:USStateData" position="1200" error="false"
numsubpods="1">...</pod>
  <pod title="Voting and registration rates"
scanner="Data"
id="VotingAndRegistrationRates:USStateData"
position="1300" error="false" numsubpods="1">...
</pod>
  <pod title="Public school finance information"
scanner="Data" id="EducationFunding:USStateData"
position="1400" error="false" numsubpods="1">...
</pod>
  <pod title="Wikipedia page hits history"
scanner="Data" id="PopularityPod:WikipediaStatsData"
position="1500" error="false" numsubpods="1">...
</pod>
  <assumptions count="1">...</assumptions>
  <sources count="1">...</sources>
</queryresult>
```

The non-empty value of "Data,Character" in the timedout attribute implies that even more pods could be generated for this query.

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. You might use this as a way to prevent a single pod from dominating too much processing time, or to return only the "quick" information in your result. However, keep in mind that the most important pods in many queries will not be the quickest to format. You can use the timedoutpods attribute in the `<queryresult>` element to get a list of pods that timed out during the format stage, giving you an idea of what kind of information might be missing. Setting `podtimeout=0.3` for the "weather" query results in a number of quickly formatting pods, but it leaves out some pods that could be important, like "Latest recorded weather for Champaign, IL":

```
<queryresult success="true" error="false" numpods="2"
datatypes="City,Weather"
             timedout="Data,Character"
timedoutpods="Latest recorded weather for
Champaign%2C Illinois,Weather history &
forecast,Historical temperatures for December
1,Weather station information"
             timing="5.077" parsetiming="0.09"
parsetimedout="false"

recalculate="http://www1.wolframalpha.com/api/v2/reca
lc.jsp?
id=MSPa81bb382f264efhba900003deii48a033ggi4d&s=14"
id="MSPa81bb382f264efhba900003deii48a033ggi4d"
             host="http://www1.wolframalpha.com"
server="14"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa101bb382f264efhba900001a5146b20150hah0&s=14"
version="2.6">
  <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
  <pod title="Weather forecast for Champaign,
Illinois" scanner="Data"
id="WeatherForecast:WeatherData" position="300"
error="false" numsubpods="2" primary="true">...</pod>
  <pod title="Weather station information"
scanner="Data"
id="WeatherStationInformation:WeatherData"
position="600" error="false" numsubpods="1">...</pod>
  <assumptions count="1">...</assumptions>
```

```
  <sources count="4">...</sources>
</queryresult>
```

In contrast, the **formattimeout** parameter limits how long the entire format stage can take, for the full collection of pods. If you specify `formattimeout=2`, then you cap the entire formatting stage at 2 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). However, if the third pod takes 1.9 seconds to format, then later pods might get no chance at all to appear in the output. Setting `formattimeout=2` for the "weather" query yields a slightly different result, displaying the "most relevant" pods (including the previously dropped "Latest recorded weather for Champaign, IL") until the timeout limit is reached:

```
<queryresult success="true" error="false" numpods="3"
datatypes="City,Weather"
            timedout="Data,Character"
timedoutpods="Weather history & forecast,Historical
temperatures for December 1,Weather station
information" timing="4.573" parsetiming="0.09"
parsetimedout="false"

recalculate="http://www1.wolframalpha.com/api/v2/reca
lc.jsp?
id=MSPa241bb3824736e9f4h600005cb57b31ibcb9f61&s=14"
id="MSPa251bb3824736e9f4h6000066b50g24ci9gf998"
            host="http://www1.wolframalpha.com"
server="14"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa261bb3824736e9f4h600003b3bdig33a7ac85h&s=14"
version="2.6">
  <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
  <pod title="Latest recorded weather for Champaign,
Illinois" scanner="Data"
id="InstantaneousWeather:WeatherData" position="200"
error="false" numsubpods="1" primary="true">...</pod>
  <pod title="Weather forecast for Champaign,
Illinois" scanner="Data"
id="WeatherForecast:WeatherData" position="300"
error="false" numsubpods="2" primary="true">...</pod>
  <assumptions count="1">...</assumptions>
```

```
  <sources count="4">...</sources>
</queryresult>
```

Note that if you are using either of these format timeouts, you should consider using the **asynchronous mode** of the API, discussed below.

The totaltimeout Parameter

If you want to limit the total time for a query to process, use the **totaltimeout** parameter. It will set an absolute limit to the processing time for a query, regardless of what stage has been reached. Because it offers no indication of what processing stage was interrupted, this parameter should rarely be used by itself unless you've already explored the other options. When combined with other timeout parameters, totaltimeout will define a last-resort time limit for queries. Its default value is 20.0, which is the sum of the default values for all other timeout values.

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 using the "weather" query:

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

This returns the following XML:

```
<queryresult success="true" error="false" numpods="7"
datatypes="City,Weather"
             timedout="" timedoutpods=""
timing="4.274" parsetiming="0.11"
parsetimedout="false" recalculate=""

id="MSPa991c679e3ea1d0h94h00004c75c0230bf09ci5"
host="http://www1.wolframalpha.com" server="13"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa1001c679e3ea1d0h94h00002i36e8ebag05c7df&s=13"
version="2.6">
  <pod title="Input interpretation"
scanner="Identity" id="Input" position="100"
error="false" numsubpods="1">...</pod>
  <pod title="Latest recorded weather for Champaign,
Illinois" scanner="Data"
id="InstantaneousWeather:WeatherData" position="200"
error="false" numsubpods="1" primary="true">...</pod>
  <pod title="Weather forecast for Champaign,
Illinois" scanner="Data"
id="WeatherForecast:WeatherData" position="300"
error="false" numsubpods="2" primary="true">...</pod>
  <pod title="Weather history & forecast"
scanner="Data" id="WeatherCharts:WeatherData"
position="400" error="false" numsubpods="0"

async="http://www1.wolframalpha.com/api/v2/asyncPod.j
sp?
id=MSPa1061c679e3ea1d0h94h00005cfi845a23f3if9a&s=13"/
>
  <pod title="Historical temperatures for December 1"
scanner="Data" id="HistoricalTemperature:WeatherData"
position="500" error="false" numsubpods="0"

async="http://www1.wolframalpha.com/api/v2/asyncPod.j
sp?
```

```
id=MSPa1071c679e3ea1d0h94h00002h3ggc950ehaibf9&s=13"/
>
<pod title="Weather station information"
scanner="Data"
id="WeatherStationInformation:WeatherData"
position="600" error="false" numsubpods="1">...</pod>
<pod title="Weather station comparisons"
scanner="Data" id="LocalTemperature:WeatherData"
position="700" error="false" numsubpods="1">...</pod>
<assumptions count="1">...</assumptions>
<sources count="4">...</sources>
</queryresult>
```

You can see in this result that the first three 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 (titled "Historical temperatures for December 1") contains the following URL:

```
http://www1.wolframalpha.com/api/v2/asyncPod.jsp?
id=MSPa1071c679e3ea1d0h94h00002h3ggc950ehaibf9&s=13
```

A request to that URL returns the following XML:

```
<pod title="Historical temperatures for December 1"
scanner="Data" id="HistoricalTemperature:WeatherData"
position="500" error="false" numsubpods="1">
  <subpod title="">
    <plaintext>
      low: 8 \\[Degree]F Dec 2002 | average high: |
43 \\[Degree]F average low: | 28 \\[Degree]F | high:
63 \\[Degree]F Dec 2012 (daily ranges, not corrected
for changes in local weather station environment)
    </plaintext>
    <img src= ... />
  </subpod>
  <states count="2">...</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. Asynchronous pods have an error attribute that tells whether the pod content was generated correctly; this can be useful for optimizing your async time.

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. Any scanner that does not finish within the scantimeout period is interrupted before it produces 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/queryappid=DEMO&?
input=pi&scantimeout=1.0
```

Note the recalculate URL in the result:

```
<queryresult success="true" error="false" numpods="6"
datatypes="MathematicalFunctionIdentity"

timedout="Numeric,MathematicalFunctionData,Recognize"
timedoutpods="" timing="1.96" parsetiming="0.258"
parsetimedout="false"

recalculate="http://www1.wolframalpha.com/api/v2/reca
lc.jsp?
id=MSPa741bb3843hh8i9h3ad00001fh9feb817ad9e83&s=14"

id="MSPa751bb3843hh8i9h3ad0000630fabg4408a7cib"
```

```
host="http://www1.wolframalpha.com" server="14"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa761bb3843hh8i9h3ad00003echac6dch1h90a1&s=14"
version="2.6">
<pod title="Input" scanner="Identity" id="Input"
position="100" error="false" numsubpods="1">...</pod>
<pod title="Decimal approximation" scanner="Numeric"
id="DecimalApproximation" position="200"
error="false" numsubpods="1" primary="true">...</pod>
<pod title="Property" scanner="Numeric" id="Property"
position="300" error="false" numsubpods="1">...</pod>
<pod title="Number line" scanner="NumberLine"
id="NumberLine" position="400" error="false"
numsubpods="1">...</pod>
<pod title="Continued fraction"
scanner="ContinuedFraction" id="ContinuedFraction"
position="500" error="false" numsubpods="1">...</pod>
<pod title="Alternative representations"
scanner="MathematicalFunctionData"
id="AlternativeRepresentations:MathematicalFunctionId
entityData" position="600" error="false"
numsubpods="3">...</pod>
<assumptions count="1">...</assumptions>
<sources count="0"/>
</queryresult>
```

We see in the above result that only six 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="2"
datatypes="MathematicalFunctionIdentity" timedout=""
timedoutpods="" timing="2.855" parsetiming="0."
parsetimedout="false" recalculate="" id=""
host="http://www1.wolframalpha.com" server="14"
related="" version="2.6">
  <pod title="Series representations"
scanner="MathematicalFunctionData"
id="SeriesRepresentations:MathematicalFunctionIdentit
yData" position="700" error="false"
numsubpods="3">...</pod>
  <pod title="Integral representations"
scanner="MathematicalFunctionData"
id="IntegralRepresentations:MathematicalFunctionIdent
ityData" position="800" error="false"
numsubpods="3">...</pod>
```

```
<sources count="0"/>
</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.

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:

```
http://api.wolframalpha.com/v2/query?
appid=DEMO&input=kitty+danger&reinterpret=true
```

This will allow Wolfram|Alpha to reinterpret the query "kitty danger" to get a more useful result. This is the default behavior on the website, and many clients will likely want to turn it on as well.

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, among other things. 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.

```
<assumption type="Clash" word="MPG"
template="Assuming "${word}" is ${desc1}. Use as
${desc2} instead" count="6">
  <value name="Unit" desc="a unit" input="*C.MPG-
_*Unit-"/>
  <value name="AcronymClass" desc="an acronym"
input="*C.MPG-_*AcronymClass-"/>
  <value name="Airport" desc="an airport"
input="*C.MPG-_*Airport-"/>
  <value name="Financial" desc="a financial entity"
input="*C.MPG-_*Financial-"/>
  <value name="FileFormat" desc="a file format"
input="*C.MPG-_*FileFormat-"/>
  <value name="Protein" desc="a protein"
input="*C.MPG-_*Protein-"/>
</assumption>
```

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". Be aware that the ignorecase parameter will often reinterpret short words such as "the" and "for" as acronyms, file extensions, airports or any number of other uppercase entities.

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 on, specify translation=true in your query URL.

signature

Because the AppID is provided in plain text 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.

units

By default, measurement systems are selected automatically based on location. The units parameter allows you to override this and pick your preferred measurement system—either "metric" or "nonmetric" (US customary units).

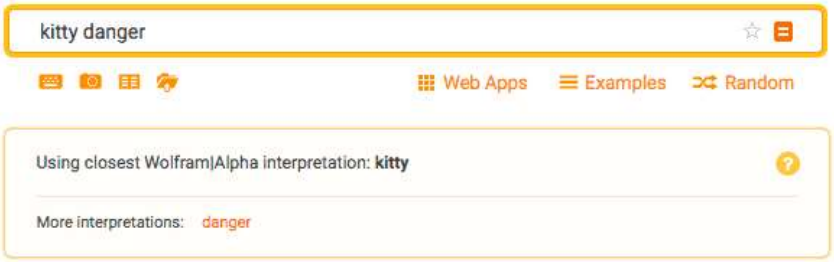
Warnings

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 located near the end of the XML result (just before `<sources>`). 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.

<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 "kitty danger" on the website, it will reinterpret this nonsensical query as simply "kitty", and provide a message in the warnings section at the top of the page:



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 (as described above). When Wolfram|Alpha performs a reinterpretation, it reports this using a `<reinterpret>` warning element:

```
<reinterpret text="Using closest Wolfram|Alpha
interpretation:" new="kitty" score="0.416667"
level="medium">
  <alternative score="0.385429"
level="medium">danger</alternative>
</reinterpret>
```

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>
```

<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 XML result:

```
<warnings count="1">
  <spellcheck word="Chicag" suggestion="Chicago"
text="&quot;chicag&quot; as &quot;chicago&quot;"/>
</warnings>
```

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 displays a message stating, "An attempt was made to fix mismatched parentheses, brackets or braces." Here is the <warnings> element in the API result:

```
<warnings count="1">
  <delimiters text="An attempt was made to fix
mismatched parentheses, brackets, or braces."/>
</warnings>
```

<translation>

When using the translation parameter, Wolfram|Alpha will attempt to translate simple queries from non-English languages into

English. 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
&amp;quot;weather today&amp;quot;/>
</warnings>
```

The `<translation>` element has **phrase**, **trans**, **lang** and **text** attributes. At present, the only way to see the `<translation>` warning is to turn on automatic translation with the translation=true URL parameter. This behavior is likely to change over time.

Queries That Are Not Understood

If, after attempting the remedies from the **Warnings** section, a query still 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" returns the following `<queryresult>` element:

```
<queryresult success="false" error="false"
numpods="0" datatypes=""
            timedout="" timedoutpods=""
timing="1.018" parsetiming="0.068"
parsetimedout="false"
            recalculate="" id=""
host="http://www1.wolframalpha.com" server="12"
related="" version="2.6">
```

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 this query, 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.

<didyoumeans>

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

```
<queryresult success="false" error="false"
numpods="0" datatypes="" timedout="" timedoutpods=""
timing="1.355" parsetiming="0.329"
parsetimedout="false" recalculate="" id=""
host="http://www1.wolframalpha.com" server="14"
related="" version="2.6">
  <didyoumeans count="2">
    <didyoumean score="0.416667"
level="medium">kitty</didyoumean>
    <didyoumean score="0.385429"
level="medium">danger</didyoumean>
  </didyoumeans>
</queryresult>
```

The `<didyoumeans>` element's count attribute is the number of `<didyoumean>` subelements generated for the query. Each `<didyoumean>` subelement includes two attributes, a score between 0 and 1 and a level (low, medium or high), both of which describe the likelihood that each `<didyoumean>` suggestion is closer to the original intent of the query. This element is useful in testing. Because it is difficult to verify the relevance of these suggestions, we recommend against using them in your implementations.

<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":

```
<queryresult success="false" error="false"
numpods="0" datatypes="" timedout="" timedoutpods=""
timing="2.387" parsetiming="0.08"
parsetimedout="false" recalculate="" id=""
host="http://www1.wolframalpha.com" server="12"
related="" version="2.6">
  <languagemsg english="Wolfram|Alpha does not yet
support German." other="Wolfram|Alpha versteht noch
kein Deutsch."/>
```

```
  <tips count="1">...</tips>
</queryresult>
```

The `<languagemsg>` element has an English attribute giving the textual message in English, and another attribute giving the textual message in whatever language the query appears to be. Note that setting `translation=true` in your query (as described under Miscellaneous URL Parameters) will prompt Wolfram|Alpha to translate certain simple queries, in some cases giving a result for the English-language equivalent (along with a `<translation>` warning) instead of a `<languagemsg>` element.

<futuretopic>

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

```
<queryresult success="false" error="false"
numpods="0" datatypes="FutureTopic"
            timedout="" timedoutpods=""
timing="0.325" parsetiming="0.142"
parsetimedout="false"

recalculate="http://www1.wolframalpha.com/api/v2/reca
lc.jsp?
id=MSPa131bb386a30ae881cb0000356575b5c6eda0g3&s=14"
id="MSPa141bb386a30ae881cb00000fhbgiaigf0hi0g7"
            host="http://www1.wolframalpha.com"
server="14"

related="http://www1.wolframalpha.com/api/v2/relatedQ
ueries.jsp?
id=MSPa151bb386a30ae881cb00002f7da6d462h97776&s=14"
version="2.6">
<sources count="0"/>
<futuretopic topic="Operating Systems"
msg="Development of this topic is under
investigation..."/>
</queryresult>
```

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

<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. Here is the output generated for the query "chemical" (you can ignore the `<assumption>` element that is also generated):

```
<queryresult success="false" error="false"
numpods="0" datatypes="" timedout="" timedoutpods=""
timing="0.638" parsetiming="0.229"
parsetimedout="false" recalculate=""
id="MSPa421c679i70ha8g50ff00003370ce1317h24399"
host="http://www1.wolframalpha.com" server="13"
related="" version="2.6">
  <assumptions count="1">...</assumptions>
  <examplepage category="ChemicalCompounds"
url="http://www.wolframalpha.com/examples/ChemicalCom
pounds-content.html"/>
</queryresult>
```

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

<tips>

The `<tips>` element is generated to give helpful feedback when Wolfram|Alpha is unable to formulate any of the other elements in this section from your query. Here is the full output from the query "fogasdgrd masdustasn":

```
<queryresult success="false" error="false"
numpods="0" datatypes="" timedout="" timedoutpods=""
timing="1.198" parsetiming="0.086"
parsetimedout="false" recalculate="" id=""
host="http://www1.wolframalpha.com" server="13"
related="" version="2.6">
  <tips count="1">
    <tip text="Check your spelling, and use
English"/>
  </tips>
</queryresult>
```

Each `<tips>` subelement contains a text attribute with a message that you might choose to display to users (like the website does) so that they may refine their queries for meaningful results.

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:

```
<queryresult success="false" error="true" numpods="0"
datatypes="" timedout="" timedoutpods=""
timing="0.02" parsetiming="0." parsetimedout="false"
recalculate="" id=""
host="http://www1.wolframalpha.com" server="13"
related="" version="2.6">
  <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`. Rather, errors are generated when Wolfram|Alpha experiences a problem that causes it to stop short of the scan stage described earlier.

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.

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