Proposed disposition of comments on date problems for Ecma 376

Antonis Christofides, National Technical University of Athens

19 February 2008

Introduction

This proposal deals with comments AU-0016, BR-0046, CA-0044, CA-0071, CH-0006, CH-0007, CH-0017, CL-0013, CL-0015, CL-0147, CL-00172, CO-0035, CO-0154, CO-0155, CO-0156, CZ-0009, DE-0030, DE-0031, DE-0032, DE-0072, DE-0073, DK-0033, DK-0136, DK-0137, DK-0153, FI-0013, FR-0182, FR-0183, FR-0351, FR-0352, GB-0300, GB-0301, GB-0304, GB-0305, GB-0363, GB-0364, GH-0002, GR-0003, GR-0004, GR-0005, GR-0006, GR-0007, GR-0008, IE-0002, IN-0007, IN-0057, IN-0058, IN-0061, IN-0062, IN-0080, IR-0001, IR-0002, KE-0054, KE-0055, MX-0005, PE-0002, PE-0003, PH-0005, PT-0085, SG-0002, US-0130, US-0131, US-0134, UY-0003, VE-0011, VE-0060, ZA-0014.

Regarding its treatment of dates, the Ecma 376 specification is inconsistent with ISO 8601, specifically with respect to dates before 1900, and with its treatment of 1900 as a leap year. Ecma's proposed disposition does address these problems, but it does not address comment GR-0008 and similar comments submitted by other countries, which mentions that “Having two different date systems with different base dates side-by-side in the same standard document format makes no sense. Rather, it is appropriate to fix a single base date. Applications which use a different base date can convert from the date representation used in the standard document format to the application's preferred date representation, and vice versa”. On the contrary, Ecma's proposed disposition has complicated the problem, because it has been proposed that dates be stored in one of four, rather than two, distinct formats, keeping the old formats for the purpose of compatibility with previous functionality.

We propose an alternative disposition which is much simpler, adequately resolves all related comments including GR-0008, and is also able to reproduce the 1900 bug whenever required, thus also preserving compatibility with previous functionality.

Problem description and overview of the solution

The entire problem stems from the fact that legacy applications, namely Microsoft Excel, which in turn reproduces behaviour from Lotus 1-2-3, have no notion of timestamps and durations as separate data types, and merely store them as real numbers. Excel has no underlying understanding of a date; to Excel, a date is merely a way of displaying a real number.

Despite the fact that, after several decades of usage, users now find it natural, it is important to understand that a timestamp is not a real number, and attempting to make operations such as adding 3.14 to the timestamp of 2008-02-03T12:31 makes no sense. You can add 3.14 days, or 3.14 minutes, or 3.14 years, or any kind of duration, but you can't add a pure real number to a timestamp any more than you can add an apple to an orange.

Therefore, the correct way to handle timestamps and durations is for the spreadsheet to inherently support them as distinct data types. However, converting legacy spreadsheets poses a problem: whenever the converter encounters a real number, it does not always know whether a real number per se is intended, or a date. The problem is illustrated in Figure 1.

Figure 1: The problem with migration of dates

The problem when migrating dates

Dates and numbers are different things, as are apples and oranges, and they should be stored as different data types. Let's suppose that the new format stores dates as dates, whereas the legacy format stores them as numbers. What happens when we try to migrate?

A legacy spreadsheet:

Stored content:

	A	B
1	61.5	=A1*8
2	=523.2*SIN(PI()/4)	=A2*13.7

Stored properties:

	A	B
1	Display as date	Display as number
2	Display as number	Display as date

How a legacy spreadsheet application displays it:

	A	B
1	1900-03-01 12:00	492
2	369.96	1913-11-15 10:16

What happens when a converter migrates to the new format:

In the legacy format, A1 contains a real number, displayed as a date. The converter decides that a date was meant, and stores a date in the new format.

In B1, the converter encounters a formula, and it stores it as is in the new format.

Stored content:

	A	B
1	1900-03-01T12:00	=A1*8
2	=523.2*SIN(PI()/4)	=A2*13.7

Stored properties:

	A	B
1	Display as date	Display as number
2	Display as number	Display as date

A2 contains a formula whose result is a real number. The converter stores it in the new format.

So does B2, but the legacy version requests that it be displayed as a date. The best thing the converter can do is leave everything as it is.

How the application displays the converted spreadsheet:

When the spreadsheet application attempts to display B1, it can't evaluate “=A1*8”, because A1 is a date.

	A	B
1	1900-03-01 12:00	#VALUE!
2	369.96	5068.43

When the spreadsheet application attempts to display B2, it needs to display a number as a date. But a number is a number and a date is a date, so how can you display the one as the other?

Ecma's conclusion:

Numbers and dates are fundamentally different; but since legacy spreadsheets store both of them as real numbers, there is nothing you can do to migrate them other than continue this false assumption into the future. The problem must be perpetuated and future spreadsheets must be burdened with the errors of the past.

Ecma has proposed that new spreadsheets also store timestamps as real numbers. Although this achieves compatible conversion from legacy spreadsheets, it perpetuates the problem. Rather than condemn us to carry the errors of the past for decades into the future, it is better to do a little more effort to correct them now. We proceed to propose a way, which actually greatly simplifies Ecma-376. The way is illustrated in Figure 2.

Figure 2: The solution to the date migration problem

The solution to the date migration problem

A legacy spreadsheet (same as above):

Stored content:

	A	B
1	61.5	=A1*8
2	=523.2*SIN(PI()/4)	=A2*13.7

Stored properties:

	A	B
1	Display as date	Display as number
2	Display as number	Display as date

How a legacy spreadsheet application displays it (same as above):

	A	B
1	1900-03-01 12:00	492
2	369.96	1913-11-15 10:16

What happens when the converter migrates to the new format:

Stored content:

	A	B
1	1900-03-01T12:00	=A1*8
2	=523.2*SIN(PI()/4)	=A2*13.7

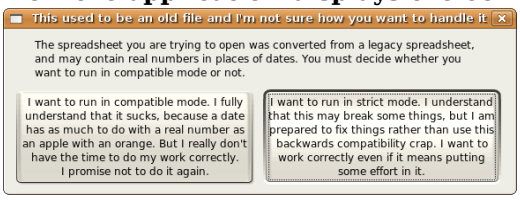
Stored properties:

	A	B
1	Display as date	Display as number
2	Display as number	Display as date

dateCompatibility="1900"

The converter does the same thing as in the example of Figure 1; however, it also sets the dateCompatibility attribute of the file to "1900", so that the application knows that it was converted from a legacy format.

How the application displays the converted spreadsheet:



In this example, the user chooses the backwards compatible option.

	A	B
1	1900-03-01 12:00	492
2	369.96	1913-11-15 10:16

When the application attempts to display B1, it needs to evaluate "=A1*8", where A1 is a date. Because it runs in compatible mode, it transparently converts the date to a number in order to perform the operation. The conversion is done by using the 1900 date base system.

When it attempts to display B2, it needs to display a number as a date, and it does it according to the 1900 date base system rules.

Our conclusion:

We have found a way to be backwards compatible and, at the same time, set the foundation for correcting the problem in the future.

The way to address the problem is similar to what has been done in OpenOffice and Open Document Format (ODF). ODF dictates that timestamps are stored as timestamps, leaving it to the application to handle legacy conversions. While OpenOffice Calc apparently treats timestamps in the same way as Microsoft Excel, in fact it includes underlying conversions so that it properly stores timestamps as required by ODF.

Similarly, our proposal is that Ecma-376 store timestamps and durations in ISO 8601 format, including recommendations that applications automatically treat them as numbers whenever required for compatibility reasons. Applications may work in compatibility mode, where such automatic treating of a date as a real number is possible, but should warn users against compatibility mode and prefer strict mode, where an error will occur in such cases.

Specific changes to Ecma-376

Part 4, §3.17.2.1, page 2,509, line 21:

3.17.2.1 Constants

A *constant* is a predefined value that is not calculated, and, therefore, does not change. A constant has the following form:

constant:
error-constant
logical-constant
numerical-constant
string-constant
array-constant
datetime-constant
duration-constant

error-constant:
#DIV/0! | #N/A | #NAME? | #NULL! | #NUM! | #REF! | #VALUE!

logical-constant:
FALSE
TRUE

numerical-constant:
whole-number-part [.] [exponent-part]
. fractional-part [exponent-part]
whole-number-part . fractional-part [exponent-part]

whole-number-part:
digit-sequence

fractional-part:
digit-sequence

exponent-part:
e *[sign] digit-sequence*
E *[sign] digit-sequence*

sign:
+
-

digit-sequence:
a series of one or more decimal digits

string-constant:
" [string-chars] "

string-chars
string-char
string-chars string-char

string-char
""
any character except "

datetime-constant
any XSD date, time, or dateTime format, where the year is positive

duration-constant
any XSD duration format

To include a double-quote character (") in *string-chars*, precede it with another double-quote character. [*Example*: `"ab""cd"` contains the characters `ab"cd`, and `"""abcd"""` contains the characters `"abcd"`. *end example*]

An *array constant* is a list of one or more constants organized in one or two dimensions, and delimited by braces. An array constant has the following form:

array-constant:
{ constant-list-rows }

constant-list-rows:
constant-list-row
constant-list-rows ; constant-list-row

constant-list-row:
constant
constant-list-row , constant

An *array-constant* shall not contain

- An *array-constant.*
- Columns or rows of unequal length.

Any *numerical-constant* in an *array-constant* can be preceded immediately by a *prefix-operator*.

The *constant*s in an *array-constant* can have different types.

[G*uidance* An implementation is encouraged to not unnecessarily limit the number of rows and columns in an *array-constant*. *end guidance*]

[*Example*: `{1,3.5,TRUE,"Hello"}` is a 1x4 array of constants.

To represent the values 10, 20, 30, and 40, as a 1x4 array, use `{10,20,30,40}`.

To represent the values 10, 20, 30, and 40 in the first row, and 50, 60, 70, and 80 in the second row, use the following 2x4 array constant: `{10,20,30,40;50,60,70,80}`. *end example*]

error-constant is described in §3.17.2.7.

Each *constant* has a corresponding type (§3.17.2.6), as follows:

Constant Form	Type
array-constant	array
error-constant	error
logical-constant	logical
numerical-constant	number
string-constant	text
datetime-constant	datetime
duration-constant	duration

In the context of cell formulas and values in SpreadsheetML, the following definition of precision shall apply:

By default, default representation of precision shall be as defined by the XML schema `double` type http://www.w3.org/TR/xmlschema-2/#double. The default is therefore 53-bits of mantissa precision.

An application that uses XML schema `double` can optionally state the precision in the Additional Characteristics part by writing out the number of bits in the mantissa and exponent.

A compliant consumer shall parse numbers of arbitrary precision without error.

Part 4, §3.17.2.6, page 2,519, line 13:

3.17.2.6 Types and Values

Each *expression* has a type. SpreadsheetML formulas support the following types: array, error, logical, number, datetime, duration, and text.

An array value or constant represents a collection of one or more elements, whose values can have any type (i.e., the elements of an array need not all have the same type).

An error value (§3.17.2.7) or constant represents an error, and can have any value defined for *error-constant* (§3.17.2.1).

A logical value or constant represents a truth value, and can have any value defined for *logical-constant* (§3.17.2.1).

A numeric value or constant represents a real number, and can have any value defined for *numeric-constant* (§3.17.2.1). The term "number" is used as a generic name for any expression of type *number*.

A datetime value or constant represents a date, a time, or a date and time, and can have any value defined for *datetime-constant* (§3.17.2.1).

A duration value or constant represents a time duration, and can have any value defined for *duration-constant* (§3.17.2.1).

A text value or constant represents arbitrary text, and can have any value defined for *string-constant* (§3.17.2.1). The term "string" is used as a generic name for any expression of type *text*.

Some legacy spreadsheet applications have been storing dates/times and durations as real numbers. Microsoft Excel and Lotus 1-2-3, in particular, have been storing dates and times as the number of days since the beginning of 1900 or 1904, and durations as a number of days. In order to be compatible with legacy spreadsheets or familiar to users of legacy applications, an application may work in a compatibility mode that transparently converts datetimes and durations to numbers when asked to add them to real numbers or perform other operations that make sense with real number operands but not with datetimes and durations. Likewise, numbers may also be transparently converted to datetimes or durations as needed. In addition, when working in compatibility mode, the nonexistent date of 29 February 1900 is allowed.However, applications should also have a strict mode of operation that prohibits such legacy operations, and should discourage users from working in compatibility mode. The *dateCompatibility* attribute (§3.2.28) describes how datetimes are converted into numbers when applications work in compatibility mode.

An implementation is permitted to provide an implicit conversion from *string-constant* to number. However, the rules by which such conversions take place are implementation-defined. [*Example*: An implementation might choose to accept "123"+10 by converting the string "123" to the number 123. Such conversions might be locale-specific in that a *string-constant* such as "10,56" might be converted to 10.56 in some locales, but not in others, depending on the radix point character. *end example*]

[*Guidance* An implementation is encouraged to support strings at least as long as 32,767 characters. *end guidance*]

A complex number is represented as a string in one of two equivalent text formats: *x* + *y*`i` or *x* + *y*`j`, where *x* is the real part, and *y* is the imaginary part. [*Example*: `"3+4i"` and `"-2.5-34.6j"` *end example*]

Part 4, §3.17.4, page 2,522, line 5:

~~**3.17.4 Dates and Times**~~

~~Each unique instant in SpreadsheetML time is represented as a distinct non-negative numeric serial value, which is made up of an integer date component and a fractional time component. As dates and times are numeric values, they can take part in arithmetic operations.~~

~~Numerous functions take as arguments one or more serial values or strings representing dates and/or times. Functions that care only about the date shall ignore any time information that is provided. Functions that care only about the time shall ignore any date information that is provided.~~

~~**3.17.4.1 Date Representation**~~

~~Going forward in time, the date component of a serial value increases by 1 each day.~~

~~There are two different bases for serial values:~~

- ~~In the 1900 date base system, the lower limit is January 1, 1900, which has serial value 1. The upper-limit is December 31, 9999, which has serial value 2,958,465.~~
- ~~In the 1904 date base system, the lower limit is January 1, 1904, which has serial value 0. The upper-limit is December 31, 9999, which has serial value 2,957,003.~~

~~A serial value outside of the range for its date base system is ill-formed.~~

~~As to which date base system an implementation uses by default or whether it allows its users to switch between date base systems, is unspecified. See §3.17.6.7 for XML.~~

related details. [Note: As the XML allows either date base system to be used, an implementation must be able to deal with both systems. end note]

For legacy reasons, an implementation using the 1900 date base system shall treat 1900 as though it was a leap year. [Note: That is, serial value 59 corresponds to February 28, and serial value 61 corresponds to March 1, the next day, allowing the (non-existent) date February 29 to have the serial value 60. end note] A consequence of this is that for dates between January 1 and February 28, WEEKDAY shall return a value for the day immediately prior to the correct day, so that the (non-existent) date February 29 has a day-of-the-week that immediately follows that of February 28, and immediately precedes that of March 1.

[Example: For the 1900 date base system:

DATEVALUE("01-Jan-1900") results in the serial value 1.0000000…
DATEVALUE("03-Feb-1910") results in the serial value 3687.0000000…
DATEVALUE("01-Feb-2006") results in the serial value 38749.0000000…
DATEVALUE("31-Dec-9999") results in the serial value 2958465.0000000…

For the 1904 date base system:

DATEVALUE("01-Jan-1904") results in the serial value 0.0000000…
DATEVALUE("03-Feb-1910") results in the serial value 2225.0000000…
DATEVALUE("01-Feb-2006") results in the serial value 37287.0000000…
DATEVALUE("31-Dec-9999") results in the serial value 2957003.0000000…

end example]

3.17.4.2 Time Representation

The time component of a serial value ranges in value from 0–0.99999999, and represents times from 0:00:00 (12:00:00 AM) to 23:59:59 (11:59:59 P.M.), respectively.

Going forward in time, the time component of a serial value increases by 1/86,400 each second. [Note: As such, the time 12:00 has a serial value time component of 0.5. end note]

[Example:

TIMEVALUE("00:00:00") results in the serial value 0.0000000…
TIMEVALUE("00:00:01") results in the serial value 0.0000115…
TIMEVALUE("10:05:54") results in the serial value 0.4207639…
TIMEVALUE("12:00:00") results in the serial value 0.5000000…
TIMEVALUE("23:59:59") results in the serial value 0.9999884…

end example]

3.17.4.3 Combined Date and Time Representation

Any date component can be added to any time component to produce a serial value for that date/time combination.

[Example: For the 1900 date base system:

DATE(1910,2,3)+TIME(10,5,54) results in the serial value 3687.4207639…
DATE(1900,1,1)+TIME(12,0,0) results in the serial value 1.5000000…
DATE(9999,12,31)+TIME(23,59,59) results in the serial value 2958465.9999884…

~~For the 1904 date base system:~~

~~DATE(1910,2,3)+TIME(10,5,54) results in the serial value 2225.4207639...~~
~~DATE(1904,1,1)+TIME(12,0,0) results in the serial value 0.5000000...~~
~~DATE(9999,12,31)+TIME(23,59,59) results in the serial value 2957003.9999884...~~

~~end example]~~

Part 4, §3.2.27, page 1,908, line 26:

- Properties: the workbook has several property collection that store basic workbook settings, such as the ~~date system to use,~~ file protection settings, calculation settings, and smart tag behaviors.
- Names: represent descriptive that represent cells, ranges of cells, formulas, or constant values.

[*Example*:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<workbook
xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/5/main"
mlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships
">
  <fileVersion lastEdited="4" lowestEdited="4" rupBuild="4017"/>
  <workbookPr ~~date1904="1"~~ vbName="ThisWorkbook"
defaultThemeVersion="123820"/>
```

Part 4, §3.2.28, page 1,911, line 1:

[*Example*:

```
  <workbookPr ~~date1904="1"~~ showObjects="none"
  saveExternalLinkValues="0"
    defaultThemeVersion="123820"/>
```

end example]

Part 4, §3.3, page 1,926, line 22:

Worksheet cells can contain text, numbers, dates/times/durations, and formulas.

Part 4, §3.2.28, page 1,912:

Attributes	Description
~~date1904 (Date 1904)~~	~~Specifies a boolean value that indicates whether the date systems used in the workbook starts in 1904.~~ ~~A value of on, 1, or true indicates the date system starts in 1904.~~ ~~A value of off, 0, or false indicates the workbook uses the 1900 date system, where 1/1/1900 is the first day in the system.~~ ~~The default value for this attribute is false.~~ ~~The possible values for this attribute are defined by the XML Schema boolean datatype.~~
dateCompatibility (Date operations in compatibility	Specifies how datetimes and durations are converted to real numbers when working in date compatibility mode. A value of “1900” indicates that if the consumer has a date

mode)	compatibility mode, it should enable it, and that when asked to perform operations with datetimes and/or durations that do not make sense unless the datetimes and/or durations are converted to numbers, then the consumer should attempt to convert durations to number of days and datetimes to number of days since 1899-12-31, making, in addition, the assumption that 1900 was a leap year. Conversely, the consumer should attempt to convert numbers to datetimes or durations as needed; for example, when a number is used as an argument to a spreadsheet function accepting a date. A value of “1904” indicates that if the consumer has a date compatibility mode, it should enable it, and that when asked to perform operations with datetimes and/or durations that do not make sense unless the datetimes and/or durations are converted to numbers, then the consumer should attempt to convert durations to number of days and datetimes to number of days since 1904-01-01. Conversely, the consumer should attempt to convert numbers to datetimes or durations as needed. If the attribute is missing or is an empty string, the consumer should disable the date compatibility mode. Consumers that do not have a date compatibility mode should warn the user if the attribute is present and non-empty, but shall otherwise ignore it.

Part 4, §3.2.28, page 1,915, line 3:

The following XML Schema fragment defines the contents of this element:

```
<complexType name="CT_WorkbookPr">
    <attribute name="date1904" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="dateCompatibility" type="xsd:string" use="optional" default=""/>
    <attribute name="showObjects" type="ST_Objects" use="optional" default="all"/>
    <attribute name="showBorderUnselectedTables" type="xsd:boolean" use="optional"
      default="true"/>
    <attribute name="filterPrivacy" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="promptedSolutions" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="showInkAnnotation" type="xsd:boolean" use="optional" default="true"/>
    <attribute name="backupFile" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="saveExternalLinkValues" type="xsd:boolean" use="optional"
      default="true"/>
    <attribute name="updateLinks" type="ST_UpdateLinks" use="optional" default="userSet"/>
    <attribute name="codeName" type="xsd:string" use="optional"/>
    <attribute name="hidePivotFieldList" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="showPivotChartFilter" type="xsd:boolean" default="false"/>
    <attribute name="allowRefreshQuery" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="publishItems" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="checkCompatibility" type="xsd:boolean" use="optional" default="false"/>
    <attribute name="autoCompressPictures" type="xsd:boolean" use="optional" default="true"/>
    <attribute name="refreshAllConnections" type="xsd:boolean" use="optional"
      default="false"/>
    <attribute name="defaultThemeVersion" type="xsd:unsignedInt" use="optional"/>
</complexType>
```

Part 4, §3.8.31, page 2,140, line 10:

To display	As	Use this code
Years	19000001-9999	yyyy

Part 4, §3.8.31, page 2,140, line 12:

~~See §3.17.4.1 for special handling of certain days in the year 1900.~~

Part 4, §3.17.6.7, page 2,529, line 27:

~~**3.17.6.7 Dates and Times**~~

~~As a date and/or time is represented by a number, a date/time serial value shall be stored in XML as the unformatted text form of that number, as accurately as possible.~~

~~The date base system is recorded in the Workbook part's XML by the presence or absence of the date1904 attribute of the workbookPr element. A value of 1 for this attribute indicates 1904. [*Example*:~~

```
~~1900: <workbookPr show0bjects="all"/>~~
~~1904: <workbookPr date1904="1" show0bjects="all"/>~~
```

~~*end example*~~]

Part 4, §3.17.7.2, page 2,534, line 6:

- ~~*issue*, *first interest*, or *settlement* is out of range for the current date base value, #NUM! is returned~~

Part 4, §3.17.7.3, page 2,535, line 16:

- ~~*issue* or *settlement* is out of range for the current date base value, #NUM! is returned~~

Part 4, §3.17.7.7, page 2,539, line 5:

- ~~*date purchased* or *first period* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.8, page 2,541, line 1:

- ~~*date purchased* or *first period* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.57, page 2,583, line 6:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.58, page 2,584, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.59, page 2,586, line 6:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.60, page 2,586, line 6:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.61, page 2,587, line 7:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.62, page 2,588, line 6:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.74, page 2,600, line 11:

3.17.7.74 DATE

Syntax:

DATE (year , month , day [, add1900])

Description: ~~Computes the serial value for the given date~~ Constructs a datetime.

Arguments:

Name	Type	Description
year	number	A year, truncated to integer, that together with month number and day specifies the date value ~~whose serial value is to be computed~~ to be constructed. ~~For the 1900 date base system:~~ • ~~If *year* is in the range 0–1899, inclusive, the year shall be interpreted as *year* + 1900.~~ • ~~If *year* is in the range 1900–9999, inclusive, the year shall be interpreted as *year*.~~ ~~For the 1904 date base system:~~ • ~~If *year* is in the range 4–1899, inclusive, the year shall be interpreted as *year* + 1900.~~ • ~~If *year* is in the range 1904–9999, inclusive, the year shall be interpreted as *year*.~~
month	number	A month, truncated to integer, that together with year number and day specifies the date ~~whose serial value is to be computed~~ to be constructed. *month* shall be interpreted as the number of months relative to the final month of the year prior to the specified year. ~~If *month* is in the range 1–12, the month shall be interpreted as *month*. If *month* is less than 1 or greater than 12, the month shall be interpreted as the normalized value (see below) of *month*, and the year shall be adjusted accordingly.~~

Name	Type	Description
day	number	A day, truncated to integer, that together with month and number year specifies the date ~~whose serial value is to be computed~~ to be constructed. *day* shall be interpreted as the number of days relative to the last day of the month (and its associated year) prior to the month (and its associated year) as determined from *month* and *year* (see below). ~~If *day* is in the allowable range of days for the month, the day shall be interpreted as *day*. If *day* is less than 1 or greater than the number of days in the given month, the day shall be interpreted as the normalized value (see below) of *day*, and the year and month shall be adjusted~~
add1900	logical	If true, it specifies that the year is to be interpreted as *year*+1900 if *year* is in less than 1900, and as *year* if it is greater than or equal to 1900. The default value for this parameter is false.

The value of *month* or *day* in a *year-month-day* argument triplet can be out of range. *month* is simply an instance of counting a given number of months, minus one, relative to January of the year specified, using the Gregorian calendar [ISO 8601]. This calendar defines that there are 12 months in a year, and that when counting forward, the month following December of one year is January of the following year, and when counting backward, the month preceding January of one year is December of the previous year. Likewise, *day* is simply an instance of counting a given number of days, minus one, relative to the first day of the adjusted month, using the Gregorian calendar. This calendar defines the number of days in each month, and that when counting forward, the day following the final day of one month is the first day of the following month, and when counting backward, the day preceding the first day of one month is the final day of the previous month. [*Example*: The *year-month-day* argument triplets (2007,12,32), (2007,13,1), and (2008,1,1) all result in the same date. *end example*]

Return Type and Value: ~~number – The serial value for the given date.~~ datetime – The datetime that corresponds to the specified parameters.

If *year* is greater than 9999, or less than 1 with *add1900*=false, or less than 0 with *add1900*=true, or if the arguments would result in a date greater than 9999-12-31, #NUM! is returned.

When the dateCompatibility attribute has a value of “1900”, a consumer may consider 1900 to be a leap year. In such a case, DATE(1900, 2, 29) will return 1900-02-29, rather than the correct value of 1900-03-01.

~~However, if~~

- ~~year is less than 0 or is greater than or equal to 10000, and the 1900 date base system is being used, #NUM! is returned.~~
- ~~year is less than 4, is greater than or equal to 10000, is in the range 1900–1903, inclusive, and the 1904 date base system is being used, #NUM! is returned.~~

~~[*Example*: For the 1900 date base system:~~

~~DATE(0,1,1) results in a serial value of 1~~
~~DATE(1899,1,1) results in a serial value of 693598~~
~~DATE(1900,1,1) results in a serial value of 1~~
~~DATE(9999,12,31) results in a serial value of 2958465~~

~~For the 1904 date base system:~~

~~DATE(4,1,1) results in a serial value of 0~~
~~DATE(1899,1,1) results in a serial value of 692136~~
~~DATE(1904,1,1) results in a serial value of 0~~
~~DATE(9999,12,31) results in a serial value of 2957003~~

~~*end example*~~]

Part 4, §3.17.7.75, page 2,601, line 29:

3.17.7.75 DATEDIF

Syntax:

`DATEDIF` (start-date , end-date [, unit])

Description: Calculates the ~~number of days, months, or years~~ difference between two dates.

Arguments:

Name	Type	Description
start-date	datetime ~~number~~	The first date in the period~~, truncated to integer~~.
end-date	datetime ~~number~~	The last date in the period~~, truncated to integer~~.
unit	text	The ~~count~~ type of result to be returned. If omitted, returns the difference as a duration. Otherwise, returns a number that depends on *unit*, as follows: **Value** / **Day Count Basis** `"Y"` — The number of complete years in the period. `"M"` — The number of complete months in the period. `"D"` — The number of days in the period. `"MD"` — The difference between the days in *start-date* and *end-date*. The months and years of the dates are ignored. `"YM"` — The difference between the months in *start-date* and *end-date*. The days and years of the dates are

Name	Type	Description
		<table><tr><td></td><td>ignored.</td></tr><tr><td>"YD"</td><td>The difference between the days of *start-date* and *end-date*. The years of the dates are ignored.</td></tr></table>

Return Type and Value: number or duration -- The number of days, months, or years between two dates, depending on the value of *unit*, or the duration between the two dates, if *unit* is omitted.

However, if

- ~~start-date or end-date is out of range for the current date base value, #NUM! is returned.~~
- *start-date* ≥ *end-date* `#NUM!` is returned.
- *unit* is any value other than those shown in the table above, `#NUM!` is returned.

[*Example*:

`DATEDIF(DATE(2001,1,1),DATE(2003,1,1),"Y")` results in `2` complete years
`DATEDIF(DATE(2001,6,1),DATE(2002,8,15),"D")` results in `440` days
`DATEDIF(DATE(2001,6,1),DATE(2002,8,15),"YD")` results in `75` days
`DATEDIF(DATE(2001,6,1),DATE(2002,8,15),"MD")` results in `14` days

end example]

Part 4, §3.17.7.75, page 2,601, line 29:

3.17.7.76 DATEVALUE

Syntax:

`DATEVALUE` `(` date-time-string `)`

Description: Determines ~~Computes the serial value of~~ the date ~~and/or time~~ represented by a string~~the string date-time-string, taking into account the current date base value~~.

Arguments:

Name	Type	Description
date-time-string	text	The ~~date and/or time whose serial value is~~ string representing the date to be computed. *date-time-string* can have any valid date ~~and/or time~~ format. If the year portion of *date-time-string* is omitted, the current year is used. Any time information in *date-time-string* shall be ignored.

Return Type and Value: ~~number~~ datetime -- The ~~serial value of the~~ date ~~and/or time~~ represented by the string *date-time-string*.

However, if

- ~~date-time-string is out of range for the current date base value, #VALUE! is returned.~~
- *date-time-string* does not represent a date, `#VALUE!` is returned.

[*Example*: When the current year is 2006,

```
DATEVALUE("2/1/2006")
DATEVALUE("01-Feb-2006 10:06 AM")
DATEVALUE("2006/2/1")
DATEVALUE("2006-2-1")
DATEVALUE("1-Feb")
```

all result in 2006-02-01~~38749 for the 1900 date base system, or 37287 for the 1904 date base system~~. *end example*]

Part 4, §3.17.7.78, page 2,605, line 11:

3.17.7.78 DAY

Syntax:

`DAY ( ` date-value ` )`

Description: Computes the numeric Gregorian day for the date and/or time having the given *date-value*~~, taking into account the current date base value~~.

Arguments:

Name	Type	Description
date-value	~~number~~ datetime, text	The date ~~and/or time~~ whose day is to be computed. That date and/or time shall be expressed either as a ~~serial~~ datetime value~~, in which case, its fractional part is ignored,~~ or as a *string-constant* having any valid date and/or time format.~~, in which case, a~~Any time information shall be ignored.

Return Type and Value: number -- The Gregorian day for the date and/or time having the given *date-value*. The returned value shall be in the range 1--31.

~~However, if date-value is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`DAY(DATE(2006,1,2))` results in `2`
`DAY(DATE(2006,0,2))` results in `31`
`DAY("2006/1/2 10:45 AM")` results in `2`
~~DAY(30000) results in 18 for the 1900 date base system, or 19 for the 1904 date base system~~

end example]

Part 4, §3.17.7.79, page 2,606, line 16:

3.17.7.79 DAYS360

Syntax:

`DAYS360 ( start-date , end-date [ , method-flag ] )`

Description: Computes the signed number of days between two dates based on a 360-day year (twelve 30-day months).

Arguments:

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>start-date</td><td>datetime ~~number~~</td><td>start-date and end-date are the dates for which the difference is to be computed. start-date can be earlier than, the same as, or later than end-date.</td></tr>
<tr><td>start-date</td><td>datetime ~~number~~</td><td></td></tr>
<tr><td>method-flag</td><td>logical</td><td>Specifies whether to use the U.S. or European method in the calculation, as follows:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td><code>FALSE</code> or omitted</td><td>U.S. (NASD) method: If the start-date is the 31st day of a month, it is changed to the 30th day of that same month. If the end-date is the 31st day of a month and the start-date is earlier than the 30th day of a month, the end-date is changed to the 1st day of the following month; otherwise the end-date is changed to the 30th day of the same month.</td></tr>
<tr><td><code>TRUE</code></td><td>European method: start-dates and end-dates that occur on the 31st day of a month are changed to the 30th day of the same month.</td></tr>
</table></td></tr>
</table>

Return Type and Value: number -- The signed number of days between two dates based on a 360-day year (12 30-day months). If *start-date* is later than *end-date*, the return value shall be negative, and the magnitude shall be the difference in days.

~~However, if start-date or end-date is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`DAYS360(DATE(2002,2,3),DATE(2005,5,31))` results in `1198`
`DAYS360(DATE(2005,5,31),DATE(2002,2,3))` results in `-1197`
`DAYS360(DATE(2002,2,3),DATE(2005,5,31),FALSE)` results in `1198`
`DAYS360(DATE(2002,2,3),DATE(2005,5,31),TRUE)` results in `1197`

end example]

Part 4, §3.17.7.91, page 2,617, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.101, page 2,624, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.104, page 2,626, line 7:

3.17.7.104 EDATE

Syntax:

`EDATE ( start-date , month-offset )`

Description: ~~Computes the serial value of~~ Determines the date that is *month-offset* months from the date specified by the date ~~*date-string*~~ *start-date*~~, taking into account the current date base value~~.

Arguments:

Name	Type	Description
start-date	~~number~~ datetime	The start date.
month-offset	number	The number of months before or after *start-date*, truncated to integer. A positive value yields a ~~future~~ date after *start-date*; a negative value yields a ~~past~~ date before *start-date*; a zero value yields the date *start-date*.

Return Type and Value: datetime -- The ~~serial value of the~~ date that is *month-offset* months from the date specified by the date ~~*date-string*~~ *start-date*~~, as a whole number~~.

~~However, if~~

- ~~start-value is out of range for the current date base value, #NUM! is returned.~~
- ~~*start-value* plus *month-offset* is out of range for the current date base value, #NUM! is returned.~~

[Example: ~~For the 1900 date base system:~~

EDATE(DATE(2006,1,31),5) results in ~~a serial value of 38898~~ 2006-06-30
EDATE(DATE(2004,2,29),12) results in ~~a serial value of 38411~~ 2005-02-28
EDATE(DATE(2004,2,28),12) results in ~~a serial value of 38411~~ 2005-02-28
EDATE(DATE(2004,1,15),-23) results in ~~a serial value of 37302~~ 2002-02-15

~~For the 1904 date base system:~~

~~EDATE(DATE(2006,1,31),5) results in a serial value of 37436~~
~~EDATE(DATE(2004,2,29),12) results in a serial value of 36949~~
~~EDATE(DATE(2004,2,28),12) results in a serial value of 36949~~
~~EDATE(DATE(2004,1,15),-23) results in a serial value of 35840~~

end example]

Part 4, §3.17.7.106, page 2,627, line 22:

3.17.7.106 EOMONTH

Syntax:

`EOMONTH` (start-date , month-offset)

Description: ~~Computes the serial value~~ Determines the date of the last day of the month for the date that is *month-offset* months from ~~the date specified by the date~~ *start-date*~~, taking into account the current date base value~~.

Arguments:

Name	Type	Description
start-date	~~number~~ datetime	The start date.
month-offset	number	The number of months before or after *start-date*, truncated to integer. A positive value yields a ~~future~~ date after *start-date*; a negative value yields a ~~past~~ date before *start-date*; a zero value yields the date *start-date*.

Return Type and Value: ~~number -- The serial value~~ datetime – The date of the last day of the month ~~for the date~~ that is *month-offset* months from the date specified by ~~the date~~ *start-date*~~, as a whole number~~.

~~However, if~~

- ~~start-date is not a valid date, #NUM! is returned.~~
- ~~start-date plus month-offset yields an invalid date, #NUM! is returned.~~

[*Example*: ~~For the 1900 date base system:~~

`EOMONTH(DATE(2006,1,31),5)` results in ~~a serial value of 38898~~ 2006-06-30
`EOMONTH(DATE(2004,2,29),12)` results in ~~a serial value of 38411~~ 2005-02-28

`EOMONTH(DATE(2004,2,28),12)` results in ~~a serial value of 38411~~ 2005-02-28
`EOMONTH(DATE(2004,1,15),-23)` results in ~~a serial value of 37315~~ 2002-02-28

~~For the 1904 date base system:~~

~~EOMONTH(DATE(2006,1,31),5) results in a serial value of 37436~~
~~EOMONTH(DATE(2004,2,29),12) results in a serial value of 36949~~
~~EOMONTH(DATE(2004,2,28),12) results in a serial value of 36949~~
~~EOMONTH(DATE(2004,1,15),-23) results in a serial value of 35853~~

end example]

Part 4, §3.17.7.136, page 2,650, line 10:

Field names and names for items other than dates/times ~~(which shall be expressed as numbers)~~ and numbers shall be enclosed in quotation marks.

Part 4, §3.17.7.143, page 2,658, line 4:

3.17.7.143 HOUR

Syntax:

`HOUR ( ` time-value `)`

Description: ~~Computes~~ Determines the hour for the date and/or time ~~having the~~ given ~~time value~~.

Arguments:

Name	Type	Description
time-value	~~number~~ datetime, string	The date and/or time whose hour is to be ~~computed~~determined. That date and/or time shall be expressed either as a ~~serial value~~datetime, ~~in which case, its integer part is ignored,~~ or as a *string-constant* having any valid date and/or time format~~, in which case~~In both cases, any date information shall be ignored.

Return Type and Value: number -- The hour for the date and/or time ~~having the~~ given ~~*time-value*~~. The returned value shall be in the range 0--~~59~~24.

~~However, if time value is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`HOUR(DATE(2006,2,26)+TIME(2,10,20))` results in `2`
`HOUR(TIME(22,56,34))` results in `22`
~~HOUR(0) results in 0, since serial value 0 represents 00:00:00~~
~~HOUR(10.5) results in 12, since serial value .5 represents 12:00:00~~
`HOUR("22-Oct-2001 10:53:12")` results in `10`
`HOUR("10:53:12 pm")` results in `22`
`HOUR("22:53:12")` results in `22`

end example]

Part 4, §3.17.7.170, page 2,681, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.206, page 2,709, line 4:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.212, page 2,713, line 15:

3.17.7.212 MINUTE

Syntax:

`MINUTE ( time-value )`

Description: Computes the minute for the date and/or time ~~having the~~ given ~~*time-value*~~.

Arguments:

Name	Type	Description
time-value	number	The date and/or time whose minute is to be computed. That date and/or time shall be expressed either as a ~~serial value, in which case, its integer part is ignored~~ datetime, or as a *string-constant* having any valid date and/or time format~~, in which case,~~. In both cases, any date information shall be ignored.

Return Type and Value: number -- The minute for the date and/or time ~~having the~~ given ~~*time-value*~~. The returned value shall be in the range 0--59.

~~However, if time-value is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`MINUTE(DATE(2006,2,26)+TIME(2,10,20))` results in `10`
`MINUTE(TIME(22,56,34))` results in `56`
~~MINUTE(0) results in 0, since serial value 0 represents 00:00:00~~
~~MINUTE(10.5) results in 0, since serial value .5 represents 12:00:00~~
`MINUTE("22-Oct-2001 10:53:12")` results in `53`
`MINUTE("10:53:12 pm")` results in `53`
`MINUTE("22:53:12")` results in `53`

end example]

Part 4, §3.17.7.218, page 2,717, line 23:

3.17.7.218 MONTH

Syntax:

`MONTH` (date-value)

Description: ~~Computes~~ Determines the numeric Gregorian month for the date and/or time ~~having the~~ given ~~*date-value,* taking into account the current date base value~~. That date and/or time shall be expressed either as a ~~serial value, in which case, its fractional part is ignored~~ datetime, or as a *string-constant* having any valid date and/or time format~~, in which case,~~In both cases, any time information shall be ignored.

Arguments:

Name	Type	Description
date-value	~~number~~ dateTime, text	The date and/or time whose month is to be ~~computed~~ determined. That date and/or time shall be expressed either as a ~~serial value, in which case, its fractional part is ignored~~ datetime, or as a *string-constant* having any valid date and/or time format~~, in which case~~. In both cases, any time information shall be ignored.

Return Type and Value: number -- The Gregorian month for the date and/or time ~~having the~~ given ~~*date-value*~~, in the range ~~1900-9999~~1-12.

~~However, if date-value is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`MONTH(DATE(2006,1,2))` results in `1`
`MONTH(DATE(2006,0,2))` results in `12`
`MONTH("2006/1/2 10:45 AM")` results in `1`
~~MONTH(30000) results in 2 for both the 1900 and 1904 date base systems~~

end example]

Part 4, §3.17.7.224, page 2,722, line 9:

3.17.7.224 NETWORKDAYS

Syntax:

`NETWORKDAYS` (start-date , end-date [, holidays])

Description: Computes the number of whole working days between *start-date* and *end-date*. Weekend days and any holidays specified by *holidays* are not considered as working days.

Arguments:

Name	Type	Description
start-date	~~number~~ datetime	The dates for which the difference is to be computed. *start-date* can be earlier than, the same as, or later than *end-date*.
end-date	~~number~~ datetime	
holidays	reference, array	An optional set of one or more dates that are to be excluded from the working day calendar. *holidays* shall be a range of cells that contain the dates, or an array constant of ~~the serial values that represent~~ those dates. The ordering of dates ~~or serial values~~ in *holidays* can be arbitrary.

Return Type and Value: number -- The number of whole working days between *start-date* and *end-date*, excluding the specified holidays. If *start-date* is later than *end-date*, the return value shall be negative, and the magnitude shall be the number of whole working days.

~~However, if~~

- ~~start-date is out of range for the current date base value, #NUM! is returned.~~
- ~~end-date is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`NETWORKDAYS(DATE(2006,1,1),DATE(2006,1,31))` results in 23
`NETWORKDAYS(DATE(2006,1,31),DATE(2006,1,1))` results in `-23`
`NETWORKDAYS(DATE(2006,1,1),DATE(2006,2,1),{"2006/1/2","2006/1/16"})` results in `21`

end example]

Part 4, §3.17.7.231, page 2,727, line 9:

3.17.7.231 NOW

Syntax:

```
NOW ( )
```

Description: ~~Computes~~ Returns the ~~serial value of the~~ current date and time~~, taking into account the current date base value~~.

Arguments: None.

Return Type and Value: ~~number~~datetime -- The ~~serial value of the~~ current date and time.

~~[Example: On February 26, 2006, between 23:01 and 23:02, NOW() resulted in 38774.95958611110 for the 1900 date base system. On February 26, 2006, between 23:02 and 23:03, NOW() resulted in 37312.95982569440 for the 1904 date base system. end example]~~

Part 4, §3.17.7.238, page 2,735, line 6:

- ~~*settlement, maturity, issue,* or *first-coupon* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.240, page 2,738, line 4:

- ~~*settlement, maturity,* or *last-interest* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.241, page 2,736, line 10:

- ~~*settlement, maturity, issue,* or *first-coupon* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.239, page 2,739, line 9:

- ~~*settlement, maturity,* or *last-interest* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.254, page 2,750, line 8:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.255, page 2,751, line 17:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.256, page 2,753, line 5:

- ~~*settlement, maturity,* or *issue* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.268, page 2,762, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.284, page 2,775, line 10:

3.17.7.284 SECOND

Syntax:

`SECOND` (time-value)

Description: ~~Computes~~ Determines the second for the date and/or time ~~having the~~ given ~~time-value~~.

Arguments:

Name	Type	Description

time-value	~~number~~ datetime, string	The date and/or time whose second is to be ~~computed~~determined. That date and/or time shall be expressed either as a ~~serial value, in which case, its integer part is ignored~~ datetime, or as a *string-constant* having any valid date and/or time format~~, in which case~~. In both cases, any date information shall be ignored.

Return Type and Value: number -- The second for the date and/or time ~~having the~~ given ~~time-value~~.

~~However, if time-value is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`SECOND(DATE(2006,2,26)+TIME(2,10,20))` results in `20`
`SECOND(TIME(22,56,34))` results in `34`
~~SECOND(0) results in 0, since serial value 0 represents 00:00:00~~
~~SECOND(10.5) results in 0, since serial value .5 represents 12:00:00~~
`SECOND("22-Oct-2001 10:53:12")` results in `12`
`SECOND("10:53:12 pm")` results in `12`
`SECOND("22:53:12")` results in `12`

end example]

Part 4, §3.17.7.315, page 2,799, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.316, page 2,800, line 4:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.317, page 2,801, line 3:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.319, page 2,802, line 19:

~~TEXT(1234.567,"YYYY-MM-DD HH:MM:SS") results in 1903-05-18 13:36:29 in the 1900 date base system.~~

TEXT(1964-02-29T13:46, “MM/DD/YYYY HH:MM”) results in 29/02/1964 13:46)

Part 4, §3.17.7.320, page 2,802, line 23:

3.17.7.320 TIME

Syntax:

`TIME` (hour , minute , second)

Description: ~~Computes the serial value for the given~~ Constructs a time.

Arguments:

Name	Type	Description
hour	number	A number in the range 0--32767, inclusive, truncated to integer, that represents the hour. Any value greater than 23 shall be divided by 24 and the remainder shall be treated as the hour value.
minute	number	A number in the range 0--32767, inclusive, truncated to integer, that represents the minute. Any value greater than 59 shall be converted to the corresponding number of hours and minutes.
second	number	A number in the range 0--32767, inclusive, truncated to integer, that represents the second. Any value greater than 59 shall be converted to the corresponding number of hours, minutes, and seconds.

Return Type and Value: ~~number~~ datetime -- The ~~serial~~ time value for the given arguments~~time, as a value greater than or equal to 0 and less than or equal to 1~~.

However, if *hour*, *minute*, or *second* are out of range, `#NUM!` is returned.

~~[Example: The following serial values are displayed with 16 decimal places.~~

~~TIME(0,0,0) results in a serial value of 0.0000000000000000~~
~~TIME(0,0,1) results in a serial value of 0.0000115740740741~~
~~TIME(0,0,2) results in a serial value of 0.0000231481481481~~
~~TIME(0,0,20) results in a serial value of 0.0002314814814815~~
~~TIME(2,3,20) results in a serial value of 0.0856481481481481~~
~~TIME(12,0,0) results in a serial value of 0.5000000000000000~~
~~TIME(23,59,59) results in a serial value of 0.9999884259259260~~
~~TIME(26,120,240) results in a serial value of 0.1694444444444450~~

~~end example]~~

Part 4, §3.17.7.321, page 2,803, line 17:

3.17.7.321 TIMEVALUE

Syntax:

`TIMEVALUE` (date-time-string)

Description: ~~Computes the serial value of~~ Determines the ~~date and/or~~ time represented by the string *date-time-string*.

Arguments:

Name	Type	Description
date-time-string	text	~~The date and/or time whose serial value is to be computed.~~ *date-time-string* can have any valid date and/or time format. Any date information in *date-time-string* shall be ignored.

Return Type and Value: ~~number~~ datetime -- The ~~serial value of the date and/or~~ time represented by the string *date-time-string*.

However, if *date-time-string* is ill-formed, `#VALUE!` is returned.

[*Example*: ~~The following serial values are displayed with 16 decimal places.~~

`TIMEVALUE("10:02:34 ")` results in ~~0.4184490740740740~~10:02:34
`TIMEVALUE("01-Feb-2006 10:15:29 AM")` results in ~~0.42741898148233330~~10:15:29
`TIMEVALUE("22:02")` results in ~~0.9180555555555560~~22:02:00

end example]

Part 4, §3.17.7.323, page 2,805, line 3:

3.17.7.323 TODAY

Syntax:

`TODAY ( )`

Description: ~~Computes the serial value of~~ Returns the current date~~, taking into account the current date base value~~.

Arguments: None.

Return Type and Value: ~~number~~ datetime -- The ~~serial value of the~~ current date.

[*Example*:

On February 25, 2006, `TODAY()` results in ~~38773 for the 1900 date base system, or 37311 for the 1904 date base system~~ 2006-02-25.

end example]

Part 4, §3.17.7.331, page 2,810, line 5:

3.17.7.331 TYPE

Syntax:

`TYPE ( value )`

Description: Computes the type of *value* or, if *value* is a reference to a single cell, the type of the value in that cell.

Arguments:

Name	Type	Description
value	any	The value whose type is to be determined. No conversion shall take place on an argument passed to this function.

Return Type and Value: number -- An integer that indicates the type of *value* or, if *value* is a reference to a single cell, the type of the value in that cell, as follows:

Type of *value*	Value Returned
number	1
text	2
logical	4
datetime	8
error value	16
duration	32
array of any kind	64

When running in date compatibility mode (§3.2.28), an application may choose to return 1 instead of 8 or 32 when the argument is a datetime or duration.

[*Example*:

`TYPE(10.5)` results in `1`
`TYPE(A10)` results in `1`, when A10 contains a number
`TYPE("ABC")` results in `2`
`TYPE(A10)` results in `2`, when A10 contains a string
`TYPE(TRUE)` results in `4`
`TYPE(A10)` results in `4`, when A10 contains a logical value

`TYPE(5/0)` results in `16`
`TYPE(A10)` results in `16`, when A10 contains any error value
`TYPE({1,2,3})` results in `64`
`TYPE({TRUE,2.5,#N/A})` results in `64`
`TYPE(IF(10>5,"Yes",20))` results in `2`
`TYPE(IF(10<5,"Yes",20))` results in `1`

end example]

Part 4, §3.17.7.334, page 2,812, line 11:

3.17.7.334 VALUE

Syntax:

`VALUE ( string )`

Description: Converts *string* to a number.

Arguments:

Name	Type	Description
string	text	Designates a string that contains a number formatted using any number~~,~~or currency format, or a date and/or time in any ~~date, or time~~ format. (See §3.8.31 for the set of formats.) ~~Date and time strings are converted to their equivalent serial value.~~

Return Type and Value: number, datetime -- The number or date/time represented by *string*.

[*Example*:

`VALUE("123.456")` results in `123.456`
`VALUE("$1,000")` results in `1000`
`VALUE("23-Mar-2002")` results in 2002-03-23~~the corresponding serial value~~
`VALUE("16:48:00")-VALUE("12:17:12")` results in ~~0.188056~~ P4H30M48S

end example]

Part 4, §3.17.7.341, page 2,818, line 16:

3.17.7.341 WEEKDAY

Syntax:

`WEEKDAY ( ~~serial value~~date [ , weekday-start-flag ] )`

Description: Computes the weekday number for the date ~~having the~~ given ~~serial value~~, taking into account ~~the current date base value and~~ *weekday-start-flag*, if present. ~~See §3.17.4.1 for special handling of certain days in 1900.~~

Arguments:

Name	Type	Description
~~*serial-value*~~ *date*	~~number~~ datetime	The date whose weekday number is to be computed. ~~The value of serial-value is truncated to an integer.~~ Any time information is ignored.
weekday-	number	When truncated to integer, indicates the weekday

Name	Type	Description
start-flag		numbering convention to be used, as follows: ***Value*** / **Meaning** 1 or omitted — 1 (Sunday) through 7 (Saturday) 2 — 1 (Monday) through 7 (Sunday) 3 — 0 (Monday) through 6 (Sunday)

Return Type and Value: number -- The weekday number for the date ~~having the~~ given ~~serial value~~.

However, if

- ~~*serial-value* is out of range for the current date base value, #NUM! is returned.~~
- *weekday-start-flag* is out of the range specified in the table above, `#NUM!` is returned.

[*Example*:

`WEEKDAY(DATE(2006,2,1))` results in `4` (Wednesday)
`WEEKDAY(DATE(2006,2,1),1)` results in `4` (Wednesday)
`WEEKDAY(DATE(2006,2,1),2)` results in `3` (Wednesday)
`WEEKDAY(DATE(2006,2,1),3)` results in `2` (Wednesday)

end example]

Part 4, §3.17.7.342, page 2,819, line 19:

3.17.7.342 WEEKNUM

Syntax:

`WEEKNUM ( ~~serial-value~~date [ , weekday-start-flag ] )`

Description: Computes the week number of the date given~~corresponding to serial-value~~. The week containing January 1 is the first week of the year, and is numbered week 1.

Arguments:

Name	Type	Description
~~serial-value~~ date	~~number~~ datetime	The date whose week number is to be computed. ~~The value of serial-value is truncated to an integer.~~ Any time information is ignored.

Name	Type	Description
weekday-start-flag	number	When truncated to integer, indicates the weekday on which the week begins, as follows: **weekday-start-flag** / **Meaning** 1 or omitted — Week begins on Sunday. 2 — Week begins on Monday.

Return Type and Value: number -- The week number of the date given ~~corresponding to serial value~~.

However, if

- ~~serial value is out of range for the current date base value, #NUM! is returned.~~
- *weekday-start-flag* is out of the range specified in the table above, `#NUM!` is returned.

[*Example*:

`WEEKNUM(DATE(2006,1,1)` results in `1`
`WEEKNUM(DATE(2006,1,1),1)` results in `1`
`WEEKNUM(DATE(2006,2,1),1)` results in `5`
`WEEKNUM(DATE(2006,2,1),2)` results in `6`

end example]

Part 4, §3.17.7.344, page 2,821, line 17:

3.17.7.344 WORKDAY

Syntax:

`WORKDAY ( start-date , day-offset [ , holidays ] )`

Description: Computes the ~~serial value of the~~ date that is *day-offset* working days offset from *start-date*. Weekend days and any holidays specified by *holidays* are not considered as working days.

Arguments:

Name	Type	Description
start-date	~~number~~ datetime	The start date~~, truncated to integer~~. Any time information is ignored.
day-offset	number	The number of working days before or after *start-date*. A positive value yields a future date; a negative value yields a

Name	Type	Description
		past date; a zero value yields the date *start-date*. *day-offset* is truncated to an integer.
holidays	reference, array	An optional set of one or more dates that are to be excluded from the working day calendar. *holidays* shall be a range of cells that contain the dates, or an array constant of ~~the serial values that represent~~ those dates. The ordering of dates ~~or serial values~~ in *holidays* can be arbitrary.

Return Type and Value: ~~number~~ datetime -- The ~~serial value of the~~ date that is *day-offset* working days offset from *start-date*, excluding the specified holidays.

However, if

- ~~start-date is out of range for the current date base value, #NUM! is returned.~~
- ~~Any date in holidays is out of range for the current date base value, #NUM! is returned.~~
- *start-date* plus *day-offset* yields an invalid date, `#NUM!` is returned.

[*Example*:

`WORKDAY(DATE(2006,1,1),0)` results in ~~a serial value corresponding to 1-Jan-2006~~ 2006-01-01
`WORKDAY(DATE(2006,1,1),10)` results in ~~a serial value corresponding to 13-Jan-2006~~ 2006-01-13
`WORKDAY(DATE(2006,1,1),-10)` results in ~~a serial value corresponding to 19-Dec-2005~~ 2005-12-19
`WORKDAY(DATE(2006,1,1),20,{"2006/1/2","2006/1/16"})` results in ~~a serial value corresponding to 31-Jan-2006~~ 2006-01-31

end example]

Part 4, §3.17.7.345, page 2,823, line 11:

- ~~Any date in *dates* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.346, page 2,824, line 17:

- ~~Any date in *dates* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.347, page 2,825, line 8:

3.17.7.347 YEAR

Syntax:

`YEAR (` date-value `)`

Description: Computes the numeric Gregorian year for the date and/or time ~~having the~~ given ~~*date-value*, taking into account the current date base value~~. That date and/or time shall be expressed either as a datetime~~serial value, in which case, its fractional part is ignored,~~ or as a *string-constant* having any valid date and/or time format~~, in which case, a~~Any time information shall be ignored.

Arguments:

Name	Type	Description
date-value	number, text	The date and/or time whose year is to be computed. That date and/or time shall be expressed either as a datetime~~serial value, in which case, its fractional part is ignored,~~ or as a *string-constant* having any valid date and/or time format~~, in which case, a~~Any time information shall be ignored.

Return Type and Value: number -- The Gregorian year for the date and/or time ~~having the~~ given ~~*date-value*~~. ~~For the 1900 date base system, the returned value shall be in the range 1900--9999. For the 1904 date base system, the returned value shall be in the range 1904--9999.~~

~~However, if date-value is out of range for the current date base value, #NUM! is returned.~~

[*Example*:

`YEAR(DATE(2006,1,2))` results in `2006`
`YEAR(DATE(2006,0,2))` results in `2005`
`YEAR("2006/1/2 10:45 AM")` results in `2006`
~~YEAR(30000) results in 1982 for the 1900 date base system, or 1986 for the 1904 date base system~~

end example]

Part 4, §3.17.7.349, page 2,829, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.350, page 2,829, line 5:

- ~~*settlement* or *maturity* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.17.7.351, page 2,830, line 5:

- ~~*settlement, maturity* or *issue* is out of range for the current date base value, #NUM! is returned.~~

Part 4, §3.18.13, page 2,842:

`timePeriod` (Time	This conditional formatting rule highlights cells containing dates in the specified time period. The underlying value of the cell is

Period)	evaluated, therefore the cell does not need to be formatted as a date to be evaluated. For example, with a cell containing the value 38913 when dateCompatibility="1900" (§3.2.28), the conditional format shall be applied if the rule requires a value of ~~7/14/2006~~2006-07-14.

Copyright notice

The original text quoted in Section 3 of this document is from the Ecma 376 specification, and is presumably copyright by Ecma and/or Microsoft. The alterations for the DATE function (§3.17.7.74) that have to do with clarifying day and month normalization are from the Ecma Disposition of Comments for Greece and are presumably copyright by Ecma (this includes most of the alterations in the description of the *month* and *day* arguments, and the paragraph after the table of arguments).

The rest of the text, including Sections 1 and 2, Figures 1 and 2, and the rest of the alterations, is

Copyright ©2008 National Technical University of Athens

To the extent that it concerns the National Technical University of Athens, permission is granted to reproduce and modify this document under the terms of the Creative Commons Attribution-ShareAlike Greece license (http://creativecommons.org/licenses/by-sa/3.0/gr/). This means that you may reproduce and modify Sections 1 and 2, and Figures 1 and 2, under the terms of the aforementioned license, but you may probably reproduce Section 3 only under terms of fair use.

If you only want to reproduce the document verbatim, it is suggested (but not required) to point, instead, to the original location, http://elot.ece.ntua.gr/ooxml/disposition-on-dates, so that readers automatically get new versions.

The author thanks Alexios Zavras, Iván Stalyn Muela Flor, and Georg C. F. Greve for useful comments on earlier drafts.