Date and Time API Preliminary Documentation

For Mac OS 9

Preliminary Draft

Technical Publications

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Date & Time for Mac OS 9

The new Date & Time functionality for Mac OS 9 includes changes to the Date & Time Control Panel and the addition of functions to handle dates larger than 2040 and for conversion between UTC and Macintosh local time.

Important

This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple Computer, Inc. is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation.

You can check <http://developer.apple.com/techpubs/macos8/SiteInfo/whatsnew.html> for information about updates to this and other developer documents. To receive notification of documentation updates, you can sign up for ADC's free Online Program and receive their weekly Apple Developer Connection News e-mail newsletter. (See <http://developer.apple.com/membership/index.html> for more details about the Online Program.)

User Issues

Most of the enhancements to Date & Time for Mac OS 9 deal with year 2000/2040 issues. On the user side, with the Date & Time control panel, there have been two issues. First of all, as the year 2000 approaches, we want to have an unambiguous way of entering dates. Traditionally, dates have been entered using two digits for day, month, and year. Changing the way years are entered to four digits by default will avoid ambiguity. There will be no wondering if the user meant 1919 or 2019 when they entered “19”, for example.

The second control panel issue has been that the range of dates has been restricted to 1920 to 2019. Up to now this restriction/range was needed since we were displaying only two digits for years, thus requiring a 100-year range to be defined. However, when users enter four digits for the year, this restriction is unnecessary.

Developer Issues

On the developer side, there have been three issues: breaking the 2040 barrier, returning a higher precision by the system clock, and being able to use UTC ("Coordinated Universal Time") instead of local time.

Breaking the 2040 Barrier

The 2040 limitation is due to the clock hardware which is limited to 32 bits worth of seconds and an epoch defined as beginning on January 1, 1904. We can partially address this problem by introducing a 64-bit API. However, having APIs that break the 2040 barrier aren't of great value if we can't have the clock support those values.

The key to solving this problem is to work under the assumption that, in the future, a user's machine is more likely to run with a date set past 2040 than it is to run with a date set in the far past. Therefore, the epoch used when interpreting the clock chip value has been changed from 1904–2040 to 1972–2108. This maintains compatibility with current software, as dates can be set all the way back to 1972 (well beyond the date the Mac was born), while providing breathing room into the future.

Please note that this only affects the way the actual hardware clock value is interpreted. This in no way affects the way dates are manipulated or the formats that developers have been using. Also note that this forces limitations on the date element used by the control panel to reflect the dates that the clock can be set at, thus changing its range from 1904–2040 to 1972–2108, as well.

Clock Value in Hex	Actual Date
0x80000000	Wed, Jan 19, 1972 03:14:08 AM
0	Mon, Feb 06, 2040 06:28:16 AM
0x7FFFFFFF	Sat, Feb 25, 2108 09:42:23 AM

Returning a Higher Clock Precision

The issue of higher precision is one that involves both hardware and software. Traditionally, the clock chip used by the Macintosh only returned up to the second precision. If users wished to get higher precision, they would have to write code that uses other timing services to achieve this goal. However, these mechanisms have been imprecise and costly. With the new ROM-in-RAM ("New World") approach, however, there is now support for greater clock precision.

On the software end, Date& Time for Mac OS 9 includes higher precision in data types and functions so that developers may access these features when available on a given machine.

Obtaining UTC

The last request by developers regards the interpretation and storage of the Macintosh clock as UTC. To address this request we are providing routines to quickly obtain time as UTC, as well as routines to convert between Macintosh local time and UTC.

Reference

Data Types and Constants

```
enum{
      kUTCUnderflowErr = -8850,
      kUTCOverflowErr = -8851,
      kIllegalClockValueErr = -8852
};
```

kUTCUnderflowErr
Returned by UTC to Local time conversion routines when the conversion yields a value that is too low to represent.

kUTCOverflowErr
Returned by UTC to Local time conversion routines when the conversion yields a value that is too high to represent.

The underflow and overflow errors occur because there are only certain numbers that may be represented for UTC for any particular location. Local time for any particular location is from 0 to 4294967296 seconds. Hence, if a location is at +1 h GMT, like say Paris, a local time of 0 cannot be represented in GMT because it would be -3600 which in unsigned would mean 4294963696. This would be indeed a quite high UTC value. Hence, a per location restriction of +-GMT offset is applied to conversions. Values that fall in this range will generate any of the corresponding errors above.

kIllegalClockValueErr
Returned by clock setting APIs when attempting to set the clock to a value that is too large or small.

```
enum{
      kUTCDefaultOptions = 0
};
```

kUTCDefaultOptions

Default options to be used in any of the APIs that use the OptionsBit parameter. In the future there may be other options that will either enhance or modify the behavior of the APIs.

```
typedef struct UTCDateTime{
      UInt16            highSeconds;
      UInt32            lowSeconds;
      UInt16            fraction;
}UTCDateTime, *UTCDateTimePtr;


typedef struct LocalDateTime{
      UInt16            highSeconds;
      UInt32            lowSeconds;
      UInt16            fraction;
}LocalDateTime, *LocalDateTimePtr;
```

UTCDateTime and LocalDateTime are both 64 bits wide. The first 48 bits represent the number of seconds since 1904. The remaining 16 bits are used to indicate a fractional seconds value, which has no inherent precision. Each unit of this 16-bit value represents 1/65535 of a second. Developers may apply the appropriate arithmetic to derive milliseconds or microseconds.

Note that the decision to have the lowSeconds field divided between the high and low 32 bits of the 64 bit structure was intentional. The structure above is perfect for performing 64 bit math and logical comparisons. Having the lowSeconds field in the low or high 32 bits would have been easier for the compilers to handle and probably execute faster, however it would have render the structure unusable for 64 bit math and logical comparisons.

Functions

```
OSStatus ConvertLocalTimeToUTC(UInt32           localSeconds,
                               UInt32*          utcSeconds);
```

Given a local time in `localSeconds` the function will place the corresponding UTC value in `utcSeconds`. This routine returns `noErr` if the conversion is successful. Otherwise, it may return `kUTCUnderflowErr` or `kUTCOverflowErr`.

```
OSStatus ConvertUTCToLocalTime(UInt32           utcSeconds,
                               UInt32*          localSeconds);
```

Given a UTC time in `utcSeconds` the function will place the corresponding

local value in localSeconds. This routine returns noErr if the conversion is successful. Otherwise, it may return kUTCUnderflowErr or kUTCOverflowErr.

```
OSStatus GetUTCDateTime(        UTCDateTimePtr          utcTime,
                                OptionBits              options);
```

This API will return the current time as UTC in utcTime. Otherwise, it is set to 0. Use kUTCDefaultOptions in the options for default behavior. Different behavior may be specified through this parameter in the future. If the operation is successful noErr will be returned. If a NULL pointer is passed in utcTime, paramErr will be returned.

```
OSStatus SetUTCDateTime( const UTCDateTimePtr          utcTime,
                                OptionBits              options);
```

Use this call to set the Macintosh clock to the time passed in utcTime. Use kUTCDefaultOptions in the options for default behavior. Different behavior may be specified through this parameter in the future. If successful noErr is returned. Other errors include kIllegalClockValueErr, kUTCUnderflowErr, kUTCOverflowErr, and paramErr if NULL is passed for utcTime. It may also return clkWrErr due to a failed attempt to write the value to the clock chip.

```
OSStatus GetLocalDateTime(      LocalDateTimePtr        localTime,
                                OptionBits              options);
```

This API will return the current time in localTime. Otherwise, it is set to 0. Use kUTCDefaultOptions in the options for default behavior. Different behavior may be specified through this parameter in the future. If the operation is successful noErr will be returned. If a NULL pointer is passed in localTime, paramErr will be returned.

```
OSStatus SetLocalDateTime(      const       LocalDateTimePtr    localTime,
                                OptionBits                      options);
```

Use this call to set the Macintosh clock to the time passed in localTime. Use kUTCDefaultOptions in the options for default behavior. Different behavior may be specified through this parameter in the future. If successful noErr is returned. Other errors include kIllegalClockValueErr, paramErr if localTime is NULL, or clkWrErr due to a failed attempt to write the value to the clock chip.

```
OSStatus ConvertLocalToUTCDateTime( const LocalDateTimePtr      localTime,
                                    UTCDateTimePtr              utcTime);
```

Given a local time in `localTime` the function will place the corresponding UTC value in `utcTime`. This routine returns `noErr` if the conversion is successful. Otherwise, it may return `kUTCUnderflowErr, kUTCOverflowErr, or paramErr if utcTime is NULL.`

```
OSStatus ConvertUTCToLocalDateTime( const UTCDateTimePtr    utcTime,
                                    LocalDateTimePtr  localTime);
```

Given a UTC time in `utcTime` the function will place the corresponding local value in `localTime`. This routine returns `noErr` if the conversion is successful. Otherwise, it may return `kUTCUnderflowErr, kUTCOverflowErr, or or paramErr if localTime is NULL.`