SRP_DateTime Parse

Parses a string into a datetime.

```
Datetime = SRP_DateTime("Parse", Text, Format = "Short", Locale = "")
```

Returns

A datetime in OI internal format.

Parameters

Parameter	Description
Text	A string containing a date and time in human readable format. **(REQUIRED)**
Format	A custom or predefined format. *(OPTIONAL)*
Locale	A locale for culture specific names and formatting. *(OPTIONAL)*

Remarks

The "Parse" service converts human readable text into an OI datetime, much like IConv. IConv is rather limited on the kinds of strings you can parse. This service provides a very optimistic parser that can successfully parse any well formed date and time, even those in other languages.

Guiding the Parser

The challenge with dates is that there is no universal order. Take the following as an example:

```
02/03/2004
```

If you are from the United States, this February 3, 2004, but in Europe, this is March 2, 2004. Therefore, we need to provide the parser a hint as to the expected order. This is done with the *Format* parameter.

You can set *Format* to any value described in the Format service. The Parser will use it only to identify the order in which to find date and time components. The parser is smart enough to find months whether they are names or numbers, so long as they are in the right place in the order you specify.

If you set *Format* to "Short" or "Long", then the order is determined by *Locale*. Locale can be set to Windows Locale Name to target a specific language or left blank to use the user's current locale settings. Using the above example date, if the locale is "en-US", then the parser would interpret it to be February 3, 2004. If the locale is "es-ES" (Spanish - Spain), then it would be interpreted as March 2, 2004.

Suppose you want a very specific order. In that case, you can set *Format* to something like "MMMM D, YYYY". Remember, only order matters. If you used this format in the Format service, it would product "February 3, 2004." When you use this format in the Parse service, it can handle "February 3, 2004", "2/3/04", and anything else so long as the order is Month, Day, Year. Note, however, that locale still matters. If you set *Format* to "MMMM D, YYYY" and *Locale* to "es-ES", then "February 3, 2004" will fail to parse but "febrero 3, 2004" and "2/3/04" will succeed.

2-Digit Year

The parser uses a unique rule to handle 2-digit years. The assumption is that, most of the time, the parser is used to parse a user's inputted date. Typically, people prefer to use shorthand. Here is the rule this parser uses to interpret a 2-digit year.

1. Any 2-digit year that is 20 or less years after the current 2-digit year is in the current or next century.
2. Any 2-digit year that is more than 20 years after the current year is in the previous century.

It is 2020 at the time of this writing. Therefore, any 2-digit year between 0 and 40 is considered to be in the 2000s, but 41 and greater would be in the 1900s. Thus 31 would be 2031, not 1931, but 41 is 1941, not 2041. This algorithm also handles the century transition, so if the current year is 2090, and the 2-digit year is 07, then 07 is considered to be 2107, not 2007, but 27 is considered to be 2027, not 2127.

Here is a cross reference chart showing the parser's output given the current year (left column) and a user's input (top row):

	"7"	"17"	"27"	"37"	"47"	"57"	"67"	"77"	"87"	"97"
2020	2007	2017	2027	2037	1947	1957	1967	1977	1987	1997
2030	2007	2017	2027	2037	2047	1957	1967	1977	1987	1997
2040	2007	2017	2027	2037	2047	2057	1967	1977	1987	1997
2050	2007	2017	2027	2037	2047	2057	2067	1977	1987	1997
2060	2007	2017	2027	2037	2047	2057	2067	2077	1987	1997
2070	2007	2017	2027	2037	2047	2057	2067	2077	2087	1997

2080	2007	2017	2027	2037	2047	2057	2067	2077	2087	2097
2090	2107	2017	2027	2037	2047	2057	2067	2077	2087	2097
2100	2107	2117	2027	2037	2047	2057	2067	2077	2087	2097

Default Order

The parser will attempt to succeed in parsing a datetime even if you don't supply the entire order. The default order is always Year, Month, Day, Hour, Minute, and Second. If your format is just "D/M", then it will look for the day and month first, then it will look for the remaining components in the default order: Year, Hour, Minute, Second.

Default Components

The parser is very optimistic, so if the incoming text is missing some components, it will attempt to reasonably fill the gaps. For example, let's say the format is "M/D/YY h:mm:ss", but the user types, "3/15", the parser will succeed, producing a datetime of March 15, 2020 at Midnight (assuming 2020 is the current year). Here are the defaults used when a component is omitted.

Component	Default	Reason
Year	Current Year	Defaulting to the current year allows parsing of month/day.
Month	*n/a*	If the month is missing, the parse will fail.
Day	1	Defaulting to the first day of the month allows parsing of month/year.
Hour	0	Defaulting to the midnight hour allows parsing of just dates in a date time field.
Minute	0	Defaulting to the top of the hour allows parsing of just the hour, e.g., "3/15 at 12am"
Second	0	Defaulting to the top of the minute allows parsing of just the hour and minute, e.g., "3/15 7:30"

Examples

```
// Parse a datetime using the default format and the current locale as a guide
Datetime = SRP_DateTime("Parse", "1/14/2020 9:58 AM")

// Parse a datetime using the long format and the current locale as a guide
Datetime = SRP_DateTime("Parse", "Tuesday, January 14, 2020 9:58:22 AM", "Long")

// Parse a datetime using the long format and the Spanish language as a guide
Datetime = SRP_DateTime("Parse", "martes, 14 de enero de 2020 15:17:43", "Long", "es")

// Parse a datetime using a custom format and the Spanish-MEXICAN language as a guide
Datetime = SRP_DateTime("Parse", "enero 14, 2020 at 3:17:43 p. m.", "MMMM D, YYYY 'at' h:mm:ss tt", "es-MX")
```