

JWT calendar specification

On this page

[Time specifiers](#) | [Frame definition](#) | [Scope definition](#) | [Hierarchy levels](#) | [Absolute vs. relative time frames](#) | [Absolute](#) | [Comments](#) | [Examples and syntax details](#)

A calendar is a system on which JWT will be able to do time calculations. Each calendars should be composed of at least one **time specifier**.

Each calendar configuration is highly flexible but the underlying specification must follow a dedicated syntax.

Time specifiers

A simple time specifier is composed of a **time definition or frame** and a **scope definition** in the following format:

```
<frame> { <scope>; } #e.g. JAN-MAR {08:00 - 12:00;}
```

Frame definition

Defines the time frame to be included in the specification (e.g. JAN-MAR).

Scope definition

The scope definition **further refines** a time frame. **Multiple** time definitions and scopes can be specified and **nested** based on their **hierarchy level**.

Each scope needs to be ended with a ";".

Hierarchy levels

Time specifiers are ordered in five hierarchical levels:

1. Global
2. Year
3. Month
4. Week
5. Day

A time specifier of a particular level is always explicitly or implicitly **contained** in a time specifier of the immediately higher level (e.g. a **day** is contained in a **month**).

When the higher level time specifier is not configured (e.g. **no month** defined) , then it is implicitly contained in an unrestricted higher level time specifier (e.g. **year**).

Examples for different levels

Calendar	Description
----------	-------------

08:00-18:00;	Every day between 8AM and 6PM.
<pre> MON-FRI{ #WEEK 08:00-12:00, 13:00-18:00; #DAY } </pre>	Only during the week , since the week "level" is higher than the day level.
<pre> JAN{ MON-FRI { #WEEK 08:00-12:00, 13:00-18: 00; #DAY } } </pre>	Only in January , since the year "level" is higher than the week or day level..

Absolute vs. relative time fames

Time specifiers can be classified into two categories:

Absolute

Defines the unique and specific parts of the time continuum. This category of time specifiers represents the ones in the **global** level.

```
2020/03/25 #this is a unique and specific date
```

Relative

Define parts of the time in the context of other time specifiers. The definition depends on the time specifier where it is contained. This category of time specifiers is contained in the all levels besides the global.

```

FRI {
    08:00 - 15:00; #the time specifier depends on the "container". In this a day of the week, which is
    Friday.
}

```

Comments

JWT calendars supports single-line comments. Comments begin with a #. e.g.:

```
#Summer Calendar
```

Examples and syntax details

Overview of different levels and priorities

The following table represents all the available time specifiers ordered by **level** and **priority**.

Time specifiers each have a dedicated priority. In case that two or more time specifiers overlap in the same level, the one with the **highest** priority is applied.

Time specifiers with the **same priority** in the **same level** are **not allowed** to overlap, i.e., the intersection of their time definitions must be empty.

	High priority	Medium priority	Low priority
Level			
Global	Date list 2020/03/25, 2020/AUG/18 2017/01/01, 2017/05/01, 2017/12/25-2017/12/31 #can contain intervals	Date 2018/JUN/15-2018/SEP/15 2018/01/01-2018/01/07, 2018 /04/10-2018/04/20	Year list 2000, 2005, 2010 2000-2011 2010-2015, 2018, 2020-2025 #can contain intervals
Year	Month-Day list JAN/1, MAY/1, JUL/4, DEC/25	Month-day MAR/20-MAR/25 AUG/1-AUG/15, NOV/5-NOV-15	Month list JAN, MAR, MAY, DEC JAN-JUN, SEP-DEC #can contain intervals MAY, JUL, OCT-DEC
Month	Day of month list 1, 15, 30 1-5, 15, 25-3 1, 15-31 25-5 #can contain intervals		
Week	Day of week list MON-THU MON-WED, FRI MON, WED, FRI SAT-MON, WED #can contain intervals		
Day	Whole day 00:00-00:00;	Time 8:00-15:00; 8:00-15:00, 16:00-19:00; 21:00-3:00;	Empty ; #Can be used to exclude dates Use ; to specify holidays.

Example specifications

Get some inspiration by looking at the example specifications below.

Calendar	Description
----------	-------------

08:00-15:00, 16:00-20:00;	Any date-time with time part between 08:00 and 15:00, or 16:00 and 20:00. By convention 15:00 and 20:00 (the outer limits of the time frame) are outside of the calendar definition.
MON-FRI{08:00-15:00, 16:00-20:00;}	Mondays to Fridays from 08:00 to 15:00 and from 16:00 to 20:00.
MON-THU{08:00-15:00, 16:00-20:00;} FRI{08:00-15:00;}	• Mondays to Thursdays from 08:00 to 15:00 and from 16:00 to 20:00. • Fridays from 08:00 to 15:00.
<pre># Winter Calendar MON - THU { 08:00 - 15:00, 16:00 - 20:00; } FRI { 08:00 - 15:00; } # Summer Calendar JUN/15 - SEP/15 { MON - FRI { 08:00 - 14:30; } }</pre>	• From June 15th to September 15th: Mondays to Fridays from 8:00 to 14:30. • For the rest of the year, Mondays to Thursdays from 08:00 to 15:00 and from 16:00 to 20:00. Fridays from 08:00 to 15:00.
<pre># Winter Calendar MON - THU { 08:00 - 15:00, 16:00 - 20:00; } FRI { 08:00 - 15:00; } # Summer Calendar JUN/15 - SEP/15 { MON - FRI { 08:00 - 14:30; } } # Annual Holidays JAN/1, MAY/1, NOV/1, DEC/25 {;} # 2021 Holidays 2021/JAN/12, 2021/APR/13, 2021/APR/14, 2021/NOV/23 {;}</pre>	This calendar additionally contains a specification for: • annual holidays, i.e., holidays that have the same day every year • particular holidays per year (2021 in the example).

```

# NEW Calendar effective 1st
December 2020

# Winter Calendar
MON-THU {
    08:30 - 15:30,
    16:00 - 19:30;
}

FRI {
    08:00 - 15:00;
}

# Summer Calendar
JUN/15 - SEP/15 {
    MON - FRI {
        08:00 - 14:30;
    }
}

# Annual Holidays
JAN/1, MAY/1, JUL/4, NOV/1, DEC/25
{;}

# 2021 Holidays
2021/03/27 - 2021/03/30, 2021/10/22
{;}

# Current calendar valid up to 30th
November 2020

2000/01/01 - 2020/11/30 {

    # Winter Calendar
    MON-THU {
        08:00 - 15:00,
        16:00 - 20:00;
    }

    FRI {
        08:00 - 15:00;
    }

    # Summer Calendar
    JUN/15 - SEP/15 {
        MON - FRI {
            08:00 - 14:30;
        }
    }

    # Annual Holidays
    JAN/1, MAY/1, NOV/1, DEC/25 {;}
}

# 2020 Holidays
2020/JAN/12, 2020/APR/13, 2020/APR
/14, 2020/NOV/23 {;}

```

This example combines **two** calendars.

A **future** calendar coming into effect on December 1st of 2021 and the **currently valid** calendar.

Since the 2nd calendar is wrapped in a **global** time specifier with **higher priority** it is the one that is **currently valid**.

```

2000/01/01 - 2020/11/30 { #global specifier
...
}

```

Backus–Naur Syntax Specification

The syntax of the calendar specification follows the Backus-Naur form. [Read more about the details.](#)

<schedule> ::= <global_level_specifier> +

<global_level_specifier> ::= <date_interval_list> { <year_level_specifier>* } | <date_list> { <year_level_specifier>* } | <year_list> { <year_level_specifier>* } | <year_level_specifier>+

<year_level_specifier> ::= <month_day_list> { <month_level_specifier>* } | <month_day_interval_list> { <month_level_specifier>* } | <month_list> { <month_level_specifier>* } | <month_level_specifier>+

<month_level_specifier> ::= <day_of_month_list> { <week_level_specifier>* } | <week_level_specifier>+

<week_level_specifier> ::= <day_of_week_list> { <day_level_specifier>* } | <day_level_specifier>+

<day_level_specifier> ::= <time_interval_list>

<time_interval_list> ::= <time_interval> (, <time_interval>)* ; ;

<time_interval> ::= <time_literal> - <time_literal>

<time_literal> examples: 00:00, 6:30, 09:55, 13:01 and 23:59.

<date_interval_list> ::= <date_interval> (, <date_interval>)*

<date_list> ::= (<date_literal> | <date_interval>) (, (<date_literal> | <date_interval>))*

<date_interval> ::= <date_literal> - <date_literal>

<date_literal> examples: 2017/06/27, 2017/JUN/27, 2018/01/01 and 2020/MAR/2.

<year_list> ::= (<year> | <year_interval>) (, (<year> | <year_interval>))*

<year_interval> ::= <year> - <year>

<year> examples: 1991, 2000 and 2017.

<month_day_list> ::= <month_day> (, <month_day>)*

<month_day_interval_list> ::= <month_day_interval> (, <month_day_interval>)*

<month_day_interval> ::= <month_day> - <month_day>

<month_day> ::= <month> / <day_of_month>
examples: JAN/1, MAR/02, AUG/18 and DEC/25.

<month_list> ::= (<month> | <month_interval>) (, (<month> | <month_interval>))*

<month> ::= JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC

<day_of_month_list> ::= (<day_of_month> | <day_of_month_interval>) (, (<day_of_month> | <day_of_month_interval>))*

<day_of_month_interval> ::= <day_of_month> - <day_of_month>

<day_of_week_list> ::= (<day_of_week> | <day_of_week_interval>) (, (<day_of_week> | <day_of_week_interval>))*

<day_of_week> ::= MON | TUE | WED | THU | FRI | SAT | SUN

If you still have questions, feel free to refer to our [support](#) team.