Schedules Definition Grammar

On this page

Schedules Definition Grammar (SDG)

Schedules Definition Grammar is a language designed for specifying *schedules*, i.e., custom subsets of the time continuum where a certain activity is or can be done.

Schedules are defined in Jira at **`Administration > Add-ons > JIRA WORKFLOW TOOLBOX > Schedules`**.

There is a set of parser functions available to be used in any feature of the plugin (conditions, validations, post-functions and calculated fields) to do time calculations on your custom schedules defined using SDG. The basic operations that can be done are:

- **Belonging**: asks for whether a certain instant of the time belongs or not to a particular schedule.
- **Time subtraction**: calculates the difference between 2 particular natural time instants within the time subset defined by a particular schedule.
- **Time addition**: calculates the instant of time resulting of adding a certain time duration to a base instant within the time subset defined by a particular schedule.

SDG Syntax

Main Concepts

- **Schedule**: formal definition of a part of the natural time (i.e., time continuum) on which we will be able to do time calculations. Schedules are composed of at least one **time specifier**.
- **Time Specifier**: are composed of a **time definition** and a **block**, and are written like this: <time_definition> { <block_content> }.
  - **Time definition**: is a definition of a part of the time continuum.
  - **Block**: is written between curly brackets (i.e., {}), and defines a scope where lower level **time specifiers** can be written.
- **Time Specifier's Level**: time specifiers are arranged in a 5 level hierarchy: **global** level, **year** level, **month** level, **week** level and **day** level. The top level is the **global** one. A time specifier of a particular level is always explicitly or implicitly contained in a time specifier of the immediately higher level. When the higher level time specifier is not written, then it's implicitly contained in an unrestricted higher level time specifier.
- **Time Specifier's Priority**: time specifiers have also a **priority** associated, so that when 2 time specifiers overlap in a same level, the one with the higher priority is applied. Time specifiers with same priority in a same level are not allowed to overlap, i.e., the intersection of their *time definitions* must be empty.
- **Time Specifier's Category**: we can classify time specifiers in 2 categories.
  - **Absolute**: define unique and specific parts of the time continuum. This category of time specifiers represents the ones in the **global** level.
  - **Relative**: define parts of the time in the context of other time specifiers, i.e., the actual part of the time continuum defined depends on the time specifier where they are contained. This category of time specifiers contains the ones in the rest of levels: **year** level, **month** level, **week** level and **day** level.
- **Comment**: SDG supports single-line comments. Comments begin with #. Example: **`#Summer Schedule`**

Time Specifiers

The following table shows all the time specifiers available, ordered by level an priority, and some examples of each of them.

Level / Priority	Highest Priority	...	Lowest Priority

GLOBAL LEVEL	**Date List** **2018/03/25, 2018/AUG/18** **2017/01/01, 2017/05/01, 2017/12/25-2017/12/31** Can contain intervals.	**Date Interval List** **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 LEVEL	**Month-Day List** **JAN/1, MAY/1, JUL/4, DEC/25**	**Month-Day Interval List** **MAR/20-MAR/25** **AUG/1-AUG/15, NOV/5-NOV-15**	**Month List** **JAN, MAR, MAY, DEC** **JAN-JUN, SEP-DEC** **JAN-MAR, MAY, JUL, OCT-DEC** Can contain intervals.
MONTH LEVEL	**Day of Month List** **1, 15, 30** **1-5, 15, 25-3** **1, 15-31** **25-5** Can contain intervals.	...	...
WEEK LEVEL	**Day of Week List** **MON-THU** **MON-WED, FRI** **MON, WED, FRI** **SAT-MON, WED** Can contain intervals.	...	...
DAY LEVEL	**Whole Day** **00:00-00:00;**	**Time Interval List** **8:00-15:00;** **8:00-15:00, 16:00-19:00;** **21:00-3:00;**	**Empty** **;** Used for defining holidays.

Examples

Schedule	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 time instants are out of the schedule.
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.

`# Winter Schedule`
`MON - THU {`
`  08:00 - 15:00,`
`  16:00 - 20:00;`
`}`
 	
`FRI {`
`  08:00 - 15:00;`
`}`
 	
`# Summer Schedule`
`JUN/15 - SEP/15 {`
`  MON - FRI {`
`  08:00 - 14:30;`
`  }`
`}`
	From 15 June to 15 September, 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.

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

FRI {
    08:00 - 15:00;
}

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

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

# 2017 Holidays
2017/JAN/12, 2017/APR/13,
2017/APR/14, 2017/NOV/23
{;}
```

From 15 June to 15 September, 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.

We added also a specification for **annual holidays**, i.e., holidays the happen the same day every year, and for **particular year holidays** (2017 in the example).

```
# Schedule from 1st
December 2017 on

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

FRI {
   08:00 - 15:00;
}

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

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

# 2018 Holidays
2018/03/27 - 2018/03/30,
2018/10/22 {;}

# Schedule up to 30th
November 2017
2000/01/01 - 2017/11/30 {

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

   FRI {
      08:00 - 15:00;
   }

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

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

# 2017 Holidays
2017/JAN/12, 2017/APR/13,
2017/APR/14, 2017/NOV/23
{;}
```

In this example we show how to introduce some modifications in the previous schedule that will be applied since 1st December 2017 on, keeping the old schedule valid only up to 30th November 2017.

Functions for Time Calculations on Schedules

The following functions are available in most features of the add-on (conditions, validations and post-functions) for doing time calculations on your custom schedules:

Function	Returned value
inSchedule(number **time_instant**, string **schedule_name**, timeZone **time_zone**) : boolean	Returns `true` if the time instant **time_instant** belongs to the schedule with name **schedule_name** for **time_zone** timezone. Example: **`inSchedule(2017/12/01 7:30, "my_schedule", LOCAL)`** returns **`false`**. Example: **`inSchedule(2017/12/01 8:00, "my_schedule", LOCAL)`** returns **`true`**. Example: **`inSchedule(2017/12/01 17:00, "my_schedule", LOCAL)`** returns **`false`**. Example: **`inSchedule(2017/12/04 17:00, "my_schedule", LOCAL)`** returns **`true`**.
inSchedule(number **time_instant**, string **schedule_name**, string **additional_terms**, timeZone **time_zone**) : boolean	Similar to previous function, but with extra parameter **additional_terms**, which is a string containing extra **Schedules Definition Grammar** clauses that will be attached to schedule with name schedule_name. This function can be used to include personal holidays to an existing schedule. Example without additional terms: **`inSchedule(2017/12/04 9:00, "my_schedule", LOCAL)`** returns **`true`**. Example with additional terms: **`inSchedule(2017/12/04 9:00, "my_schedule", "2017/12/04 {;}", LOCAL)`** returns **`false`**.
timeDifference(number **higher_instant**, number **lower_instant**, string **schedule_name**, timeZone **time_zone**) : number	Returns the number of milliseconds elapsed from **lower_instant** to **higher_instant** within schedule with name **schedule_name** for **time_zone** timezone. Example: **`timeDifference(2017/12/04 10:01, 2017/12/01 01:00, "my_schedule", LOCAL)`** returns **`8 * {HOUR} + 31 * {MINUTE}`**. Example: **`timeDifference(2017/12/04 17:00, 2017/12/04 14:00, "my_schedule", LOCAL)`** returns **`2 * {HOUR} + 30 * {MINUTE}`**.
timeDifference(number **higher_instant**, number **lower_instant**, string **schedule_name**, string **additional_terms**, timeZone **time_zone**) : number	Similar to previous function, but with extra parameter **additional_terms**, which is a string containing extra **Schedules Definition Grammar** clauses that will be attached to schedule with name schedule_name. This function can be used to include personal holidays to an existing schedule. Example without additional terms: **`timeDifference(2017/12/05 18:00, 2017/12/01 9:00, "my_schedule", LOCAL)`** returns **`25 * {HOUR}`**. Example with additional terms: **`timeDifference(2017/12/05 18:00, 2017/12/01 9:00, "my_schedule", "2017/12/04 {;}", LOCAL)`** returns **`15 * {HOUR}`**.
addTime(number **base_instant**, number **offset**, string **schedule_name**, timeZone **time_zone**) : number	Returns the time instant resulting of adding **offset** milliseconds to **base_instant** within schedule with name **schedule_name** for **time_zone** timezone. Example: **`addTime(2017/12/01 01:00, 8 * {HOUR} + 31 * {MINUTE}, "my_schedule", LOCAL)`** returns **`2017/12/04 10:01`**. Example: **`addTime(2017/12/04 14:00, 2 * {HOUR} + 30 * {MINUTE}, "my_schedule", LOCAL)`** returns **`2017/12/04 17:00`**. Since version **2.2.41** negative **offset** values are supported: Example: **`addTime(2017/04/24 09:00, - 2 * {HOUR}, "my_schedule", LOCAL)`** returns **`2017/04/21 14:00`**. Example: **`addTime(2017/04/20 20:30, - 5 * {HOUR}, "my_schedule", LOCAL)`** returns **`2017/04/20 13:00`**.
addTime(number **base_instant**, number **offset**, string **schedule_name**, string **additional_terms**, timeZone **time_zone**) : number	Similar to previous function, but with extra parameter **additional_terms**, which is a string containing extra **Schedules Definition Grammar** clauses that will be attached to schedule with name schedule_name. This function can be used to include personal holidays to an existing schedule. Example without additional terms: **`addTime(2017/12/01 9:00, 25 * {HOUR}, "my_schedule", LOCAL)`** returns **`2017/12/05 18:00`**. Example with additional terms: **`addTime(2017/12/01 9:00, 25 * {HOUR}, "my_schedule", "2017/12/04 {;}", LOCAL)`** returns **`2017/12/06 18:00`**.
nextTime(number **time_instant**, string **schedule_name**, timeZone **time_zone**) : number Available since version 2.2.40	If **time_instant** doesn't belong to schedule with name **schedule_name**, then returns closer time in the future that belongs to the schedule, otherwise returns **time_instant**. Example: **`nextTime(2017/12/01 01:00, "my_schedule", LOCAL)`** returns **`2017/12/01 08:00`**. Example: **`nextTime(2017/12/01 15:00, "my_schedule", LOCAL)`** returns **`2017/12/04 08:00`**. Example: **`nextTime(2017/12/01 08:00, "my_schedule", LOCAL)`** returns **`2017/12/01 08:00`**. Example: **`nextTime(2017/11/30 15:00, "my_schedule", LOCAL)`** returns **`2017/11/30 16:00`**.
nextTime(number **time_instant**, string **schedule_name**, string **additional_terms**, timeZone **time_zone**) : number Available since version 2.2.40	Similar to previous function, but with extra parameter **additional_terms**, which is a string containing extra **Schedules Definition Grammar** clauses that will be attached to schedule with name schedule_name. This function can be used to include personal holidays to an existing schedule. Example without additional terms: **`nextTime(2017/12/01 15:00, "my_schedule", LOCAL)`** returns **`2017/12/04 08:00`**. Example with additional terms: **`nextTime(2017/12/01 15:00, "my_schedule", "2017/12/04 {;}", LOCAL)`** returns **`2017/12/05 8:00`**.

In the examples above we have used schedule `"my_schedule"`, whose definition in **Schedules Definition Grammar** is:

```
MON - THU {
   08:00 - 15:00,
   16:00 - 19:30;
}

FRI {
    08:00 - 15:00;
}
```

Note that **`2017/04/21`** and **`2017/12/01`** are Fridays.

Custom schedules are defined at **`Administration > Add-ons > JIRA WORKFLOW TOOLBOX > Schedules`**.

Backus–Naur Syntax Specification

<**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`

Schedules Usage Examples

- **Automatic work log with start and stop work transitions**