Proposed syntax for formulas in Manufacturing Scheduler

Prepared by Adam Wrobel of Flux Inc. for Yellow Pages Group

July 29, 2008

Abstract

This document pertains to Manufacturing Scheduler, a computer program produced by Flux Inc. for Yellow Pages Group (YPG). Manufacturing Scheduler provides centralized managing of business-related schedules for YPG products.

In this document we describe a proposed syntax for the representation of *formulas* that are used to calculate variable dates based on single fixed parameter values, herein referred to as *fixed dates*.

Contents

1 Quick overview

1.1 Example formulas

Formulas are evaluated left to right. Every statement changes the date found on the left and produces some date that will be used by statement to its right. The first statement uses the event's *fixed date* as it's argument. The date returned by the last statement is the date that will be visible in calendar.

1.1.1 Single statements

```
+ 2
```

Will add two days to a fixed date.

```
- 4 WORKDAYS
```
Will return the date of fourth work day before the specified fixed date. Work days occur Monday through Friday, and exclude locally-relevant holidays.

```
+ 1 TUESDAYS
```
Will return the date corresponding to the first Tuesday after the specified fixed date. If the fixed date falls on a Tuesday then the return value is exactly one week afterward.

```
+ 0 TUESDAYS
```
Will return the date corresponding to the first Tuesday after the fixed date. If the fixed date falls on a Tuesday then the return value will equal the fixed date.

```
+ 0 WORKDAYS
```
Returns the first working day after the fixed date if the fixed date falls on a holiday or a weekend; otherwise, it simply returns the fixed date itself.

1.1.2 Multiple statements

```
- 0 FIRST_DAYS_OF_MONTH +0 WEDNESDAYS +0 WORKDAYS
```
Returns the first Wednesday during the same month as that during with the fixed date occurs, or the first working day afterward if it happens to fall on a holiday.

1.1.3 Conditional statements

```
[DAY_OF_WEEK < 3; + 0 WEDNESDAYS; - 0 WEDNESDAYS]
```
Returns the closest Wednesday to the fixed date.

```
[IS_WEEKEND; - 0 FRIDAYS]
```
If the fixed date falls on a weekend, returns the previous Friday. If it does not fall on a weekend, returns the unchanged fixed date. This differs from "- 0 WORKDAYS" in that it returns the fixed date if it falls on a holiday but is not during a weekend.

```
[DAY_OF_WEEK < 3; + 0 WEDNESDAYS; - 0 WEDNESDAYS] + 0 WORKDAYS
```
Returns the closest Wednesday to the fixed date or the first working day after it if it happens to be a holiday.

```
- FIRST_DAYS_OF_MONTH [DAY_OF_WEEK < 3; + 0 WEDNESDAYS; - 0 WEDNESDAYS]
+ 0 WORKDAYS
```
Returns the closest Wednesday to the first day of the fixed date's month or, if it happens to be a holiday, the first working day afterward.

1.1.4 Comparison with current excel formulas

Current formula:
`EA78-98`
Can be expressed in proposed syntax as:
`-98`

Current formula:
`WORKDAY(DL78;-60)`
Can be expressed in proposed syntax as:
`-60 WORKDAYS`

Current formula:
```
= IF(WEEKDAY(DL80)=1;DL80-2; IF(WEEKDAY(DL80)=2;DL80-3; IF(WEEKDAY(DL80)=6;DL80-7;
IF(WEEKDAY(DL80)=4;DL80-5; IF(WEEKDAY(DL80)=5;DL80-6; IF(WEEKDAY(DL80)=7;DL80-1;
DL80-4))))))
```
Can be expressed in proposed syntax as:
`- 1 FRIDAYS`

As shown above, the basic examples are almost identical, but while the third original formula is quite complicated, the corresponding new format shows a real advantage of syntax described in this document.

2 Reference manual

2.1 Syntax reference

Syntax is described in Extended Backus-Naur Form, a metasyntax notation used to express formal languages such as those used to program computers.

2.1.1 Quick guide to EBNF

In EBNF we define *production rules* to generate sentences that can be used in a language. Production rules refer to other production rules or terminals. *Terminals* are characters that will be visible in the final sentence. They are enclosed in double quotes. Some examples follow:

```
 word ::= "abc";
other_word ::= "bcd";
two_words ::= word " " other_word;
```

We defined three production rules, two of which expand directly to terminal and the last uses both terminal " " and two previous production rules to expand to a terminal "abc bcd".

```
 alternative ::= "a" | "b";
```

Can be either "a" or "b".

```
 repetition ::= {"a"};
```

Can be "" or "a" or "aa" and so on.

```
option ::= "a" [ "b" ];
```
Can be "a" or "ab".

```
grouping ::= ( "a" | "b" ) "c";
```
Parentheses can be used to group expressions. This production rule expands to "ac" or "bc".

2.1.2 Syntax of formulas described in EBNF

All whitespace characters are optional and will be skipped before parsing.

```
formula ::= {statement};
```
This means that formula can consist of zero or more statements.

```
statement ::= simple_statement | conditional_statement;
```
There are currently two types of statements: simple and conditional. They can be mixed freely.

```
simple_statement ::= direction_operator count [time_unit];
```
Many examples of simple statements can be found in section 1.1.1 on page 1. *time_unit* is optional and defaults to `DAYS`.

```
direction_operator := "+" | "-";
```
Defines in which direction we should move. Minus stands for the past, plus for the future.

```
count ::= number;
```
Can be zero or higher. Zero will return unchanged date that was passed as argument only if it is day described by *time_unit*. If it is not zero will work just like a value of one (`1`) would.

```
time_unit ::= "DAYS" | "WORKDAYS" | "WEEKS" | "MONTHS" |
"MONDAYS" | "TUESDAYS" | "WEDNESDAYS" | "THURSDAYS" |
"FRIDAYS" | "SATURDAYS" | "SUNDAYS" |
"FIRST_DAYS_OF_MONTH" | "LAST_DAYS_OF_MONTH";
```
Units are days in calendar that match specific condition. Using simple statements we can move to one of these days. Which will it be is specified by *count* production rule.

```
conditional_statement ::= "[" condition ";" formula [";" formula] "]";
```

Any formula can be executed conditionally, even one containing other conditional statements. With second semicolon comes “else” part of condition. It is optional and defaults to +0 DAYS which leaves argument date unchanged.

```
condition ::= parameter [ comparison_operator number ];
```
If comparison is omitted condition is considered false if parameter is zero or empty string

```
 parameter ::= "IS_WORKDAY" | "IS_WEEKEND" | "IS_HOLIDAY" | "DAY_OF_WEEK"
| "DAY_OF_MONTH" | "MONTH_OF_YEAR";
```
Description of those parameters can be found in section 2.2.2.

```
 comparison_operator ::= "=" | "<>" | "<" | "<=" | ">" | ">=";
```

```
 number ::= digit | ( digit_without_zero {digit} );
```
Number cannot start with multiple zeros.

```
 digit_without_zero :== "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
"9";
digit ::= "0" | digit_without_zero;
```

2.2 Keywords reference

If the syntax proposed in this document is accepted by YPG, this section will contain an explanation of every time unit and parameter of conditional statements that can be used in formulas.

2.2.1 Time units

To be documented.

2.2.2 Parameters for conditions

To be documented.