Time Template B/timet

Purpose
Support for parsing and printing times of day.

§**1. Rounding.** The following rounds a numerical value `t1` to the nearest unit of `t2`; for instance, if `t2` is 5 then it rounds to the nearest 5.

```
[ RoundOffTime t1 t2; return ((t1+t2/2)/t2)*t2; ];
```

§**2. Square Root.** This is an old algorithm for extracting binary square roots, taking 2 bits at a time. We start out with `one` at the highest bit which isn't the sign bit; that used to be worked out as `WORD_HIGHBIT/2`, but this caused unexpected portability problems (exposing a minor bug in Inform and also glulxe) because of differences in how C compilers handle signed division of constants in the case where the dividend is -2^{31}, the unique number which cannot be negated in 32-bit twos complement arithmetic.

```
[ SquareRoot num
    op res one;
    op = num;
    if (num < 0) { RunTimeProblem(RTP_NEGATIVEROOT); return 1; }
    ! "one" starts at the highest power of four <= the argument.
    for (one = WORD_NEXTTOHIGHBIT: one > op: one = one/4) ;
    while (one ~= 0) {
        !print "Round: op = ", op, " res = ", res, ", res**2 = ", res*res, " one = ", one, "^";
        if (op >= res + one) {
            op = op - res - one;
            res = res + one*2;
        }
        res = res/2;
        one = one/4;
    }
    !print "Res is ", res, "^";
    return res;
];
```

§**3. Cube Root.** The following is an iterative scheme for finding cube roots by Newton-Raphson approximation, not a great method but which, on the narrow ranges of integers we deal with, is good enough. The square root is used only as a sighting shot.

```
[ CubeRoot num x y n;
    if (num < 0) x = -SquareRoot(-num); else x = SquareRoot(num);
    for (n=0: (y ~= x) && (n++ < 100): y = x, x = (2*x + num/x/x)/3) ;
    return x;
];
```

§**4. Digital Printing.** For instance, “2:06 am”.

```
[ PrintTimeOfDay t h aop;
    if (t<0) { print "<no time>"; return; }
    if (t >= TWELVE_HOURS) { aop = "pm"; t = t - TWELVE_HOURS; } else aop = "am";
    h = t/ONE_HOUR; if (h==0) h=12;
    print h, ":";
    if (t%ONE_HOUR < 10) print "0"; print t%ONE_HOUR, " ", (string) aop;
];
```

§**5. Analogue Printing.** For instance, “six minutes past two”.

```
[ PrintTimeOfDayEnglish t h m dir aop;
    h = (t/ONE_HOUR) % 12; m = t%ONE_HOUR; if (h==0) h=12;
    if (m==0) { print (number) h, " o'clock"; return; }
    dir = "past";
    if (m > HALF_HOUR) { m = ONE_HOUR-m; h = (h+1)%12; if (h==0) h=12; dir = "to"; }
    switch(m) {
        QUARTER_HOUR: print "quarter"; HALF_HOUR: print "half";
        default: print (number) m;
            if (m%5 ~= 0) {
                if (m == 1) print " minute"; else print " minutes";
            }
    }
    print " ", (string) dir, " ", (number) h;
];
```

§**6. Understanding.** This I6 grammar token converts words in the player's command to a valid I7 time, and is heavily based on the one presented as a solution to an exercise in the DM4.

```
[ TIME_TOKEN first_word second_word at length flag
    illegal_char offhour hr mn i original_wn;
    original_wn = wn;
{-call:Plugins::Parsing::Tokens::Values::time}
    wn = original_wn;
    first_word = NextWordStopped();
    switch (first_word) {
        'midnight': parsed_number = 0; return GPR_NUMBER;
        'midday', 'noon': parsed_number = TWELVE_HOURS;
        return GPR_NUMBER;
    }
    ! Next try the format 12:02
    at = WordAddress(wn-1); length = WordLength(wn-1);
    for (i=0: i<length: i++) {
        switch (at->i) {
            ':': if (flag == false && i>0 && i<length-1) flag = true;
            else illegal_char = true;
            '0', '1', '2', '3', '4', '5', '6', '7', '8', '9': ;
            default: illegal_char = true;
        }
    }
    if (length < 3 || length > 5 || illegal_char) flag = false;
```

```
    if (flag) {
        for (i=0: at->i~=':': i++, hr=hr*10) hr = hr + at->i - '0';
        hr = hr/10;
        for (i++: i<length: i++, mn=mn*10) mn = mn + at->i - '0';
        mn = mn/10;
        second_word = NextWordStopped();
        parsed_number = HoursMinsWordToTime(hr, mn, second_word);
        if (parsed_number == -1) return GPR_FAIL;
        if (second_word ~= 'pm' or 'am') wn--;
        return GPR_NUMBER;
    }
    ! Lastly the wordy format
    offhour = -1;
    if (first_word == 'half') offhour = HALF_HOUR;
    if (first_word == 'quarter') offhour = QUARTER_HOUR;
    if (offhour < 0) offhour = TryNumber(wn-1);
    if (offhour < 0 || offhour >= ONE_HOUR) return GPR_FAIL;
    second_word = NextWordStopped();
    switch (second_word) {
        ! "six o'clock", "six"
        'o^clock', 'am', 'pm', -1:
            hr = offhour; if (hr > 12) return GPR_FAIL;
        ! "quarter to six", "twenty past midnight"
        'to', 'past':
            mn = offhour; hr = TryNumber(wn);
            if (hr <= 0) {
                switch (NextWordStopped()) {
                    'noon', 'midday': hr = 12;
                    'midnight': hr = 0;
                    default: return GPR_FAIL;
                }
            }
            if (hr >= 13) return GPR_FAIL;
            if (second_word == 'to') {
                mn = ONE_HOUR-mn; hr--; if (hr<0) hr=23;
            }
            wn++; second_word = NextWordStopped();
        ! "six thirty"
        default:
            hr = offhour; mn = TryNumber(--wn);
            if (mn < 0 || mn >= ONE_HOUR) return GPR_FAIL;
            wn++; second_word = NextWordStopped();
    }
    parsed_number = HoursMinsWordToTime(hr, mn, second_word);
    if (parsed_number < 0) return GPR_FAIL;
    if (second_word ~= 'pm' or 'am' or 'o^clock') wn--;
    return GPR_NUMBER;
];

[ HoursMinsWordToTime hour minute word x;
    if (hour >= 24) return -1;
    if (minute >= ONE_HOUR) return -1;
    x = hour*ONE_HOUR + minute; if (hour >= 13) return x;
    x = x % TWELVE_HOURS; if (word == 'pm') x = x + TWELVE_HOURS;
```

```
    if (word ~= 'am' or 'pm' && hour == 12) x = x + TWELVE_HOURS;
    return x;
];
```

§**7. Relative Time Token.** "Time" is an interesting kind of value since it can hold two conceptually different ways of thinking about time: absolute times, such as "12:03 PM", and also relative times, like "ten minutes". For parsing purposes, these are completely different from each other, and the time token above handles only absolute times; we need the following for relative ones.

```
[ RELATIVE_TIME_TOKEN first_word second_word offhour mult mn original_wn;
    original_wn = wn;
    wn = original_wn;
    first_word = NextWordStopped(); wn--;
    if (first_word == 'an' or 'a//') mn=1; else mn=TryNumber(wn);
    if (mn == -1000) {
        first_word = NextWordStopped();
        if (first_word == 'half') offhour = HALF_HOUR;
        if (first_word == 'quarter') offhour = QUARTER_HOUR;
        if (offhour > 0) {
            second_word = NextWordStopped();
            if (second_word == 'of') second_word = NextWordStopped();
            if (second_word == 'an') second_word = NextWordStopped();
            if (second_word == 'hour') {
                parsed_number = offhour;
                return GPR_NUMBER;
            }
        }
        return GPR_FAIL;
    }
    wn++;
    first_word = NextWordStopped();
    switch (first_word) {
        'minutes', 'minute': mult = 1;
        'hours', 'hour': mult = 60;
        default: return GPR_FAIL;
    }
    parsed_number = mn*mult;
    if (mult == 60) {
        mn=TryNumber(wn);
        if (mn ~= -1000) {
            wn++;
            first_word = NextWordStopped();
            if (first_word == 'minutes' or 'minute')
                parsed_number = parsed_number + mn;
            else wn = wn - 2;
        }
    }
    return GPR_NUMBER;
];
```

§8. During Scene Matching.

```
[ DuringSceneMatching prop sc;
    for (sc=0: sc<NUMBER_SCENES_CREATED: sc++)
        if ((scene_status-->sc == 1) && (prop(sc+1))) rtrue;
    rfalse;
];
```

§9. Scene Questions.

```
[ SceneUtility sc task;
    if (sc <= 0) return 0;
    if (task == 1 or 2) {
        if (scene_endings-->(sc-1) == 0) return RunTimeProblem(RTP_SCENEHASNTSTARTED, sc);
    } else {
        if (scene_endings-->(sc-1) <= 1) return RunTimeProblem(RTP_SCENEHASNTENDED, sc);
    }
    switch (task) {
        1: return (the_time - scene_started-->(sc-1))%(TWENTY_FOUR_HOURS);
        2: return scene_started-->(sc-1);
        3: return (the_time - scene_ended-->(sc-1))%(TWENTY_FOUR_HOURS);
        4: return scene_ended-->(sc-1);
    }
];
```