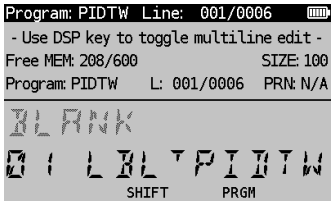
# PIDTW

## Display


*(HP-41CX, Hewlett Packard 1983 and DM41X, SwissMicros 2020)*

## Overview[1]

The `PIDTW` program can calculate the decimals of PI according to the method by Beeler et al. in 1972. It approximates PI that gives an additional bit of precision with 14 terms giving 4 decimal digits of precision each time because $2^{14} > 10^4$. Therefore 2800 = 14 · 200 terms are used. The author of a very compact C program, Dik T. Winter (hence DTW in the program name), wrote the following 160 characters' code:

```
int a=10000,b,c=2800,d,e,f[2801],g;main(){for(;b-c;)f[b++]=a/5;
for(;d=0,g=c*2;c-=14,printf("%.4d",e+d/a),e=d%a)for(b=c;d+=f[b]*a,
f[b]=d%--g,d/=g--,--b;d*=b);}
```

which can be reformatted to something more readable:

```
int a=10000,b,c=2800,d,e,f[2801],g;
int main() {
    b=0;
    while(b<c) {
        b=b+1;
        f[b]=a/5;
    }
    d=0;
    g=c*2;
    while(g>0) {
        b=c;
        while((b>0)) {
            d=d*b+f[b]*a;
            g=g-1;
            f[b]=d%g;
            d=d/g;
            g=g-1;
            b=b-1;
        }
        c=c-14;
        printf("%.4d", e+d/a);
        e=d%a;
        g=c*2;
    }
    return 0;
}
```

which gives the terms as 4 digits with tens to hundreds of iterations around the *variable-g-while-loop* and hundreds to thousands of iterations around the *variable-b-while-loop*. For this program to

[1] *This program is copyright and is supplied without representation or warranty of any kind. The author assumes no responsibility and shall have no liability, consequential or otherwise, of any kind arising from the use of this program material or any part thereof*

understand its feasibility, the requirements for the array size and the number of iterations were analysed before writing a program for the HP-41CX. It turns out that for each decimal it requires 3.5 times as many terms as well as array size. For example, for 800 decimals, it requires 2800 terms to be calculated and an array with 2800 values. The number of loops for b increases exponentially which, together with the required memory for the array f[..] brings limitations due to the capability of the calculator. Having mentioned that, it is feasible to determine the decimals up to 120. From the C program, it has been verified that the values in array f[..] are positive integers that remain under 10000. The solution for storing and updating array f[..] is found in using an ASCII file, thereby converting the 4 digits number in one pair of ASCII converted bytes. Each number in f[..] requires therefore 2 bytes. The alternative of storing each number in a register would require too much memory. The design of the program is explained later.

| Decimals | Array f[..] | Bytes | Records | Bytes/Record | Registers | g-Loops | b-Loops |
|---|---|---|---|---|---|---|---|
| 40 | 140 | 280 | ***02*** | 140 | 41 | *10* | *770* |
| 80 | 280 | 560 | ***04*** | 140 | 81 | *20* | *2940* |
| 120 | 420 | 840 | ***04*** | 210 | 121 | *30* | *6510* |
| 160 | 560 | 1120 | ***05*** | 224 | 161 | *40* | *11480* |
| 200 | 700 | 1400 | ***07*** | 200 | 202 | *50* | *17850* |
| 240 | 840 | 1680 | ***07*** | 240 | 242 | *60* | *25620* |
| 280 | 980 | 1960 | ***10*** | 196 | 282 | *70* | *34790* |
| 320 | 1120 | 2240 | ***10*** | 224 | 322 | *80* | *45360* |
| 360 | 1260 | 2520 | ***12*** | 210 | 362 | *90* | *57330* |
| 400 | 1400 | 2800 | ***14*** | 200 | 403 | *100* | *70700* |
| 440 | 1540 | 3080 | ***14*** | 220 | 443 | *110* | *85470* |
| 480 | 1680 | 3360 | ***14*** | 240 | 483 | *120* | *101640* |
| 520 | 1820 | 3640 | ***20*** | 182 | 523 | *130* | *119210* |
| 560 | 1960 | 3920 | ***20*** | 196 | 563 | *140* | *138180* |
| 600 | 2100 | 4200 | ***20*** | 210 | 603 | *150* | *158550* |
| 640 | 2240 | 4480 | ***20*** | 224 | 643 | *160* | *180320* |
| 680 | 2380 | 4760 | ***20*** | 238 | 683 | *170* | *203490* |
| 720 | 2520 | 5040 | ***24*** | 210 | 724 | *180* | *228060* |
| 760 | 2660 | 5320 | ***28*** | 190 | 765 | *190* | *254030* |
| 800 | 2800 | 5600 | ***25*** | 224 | 804 | *200* | *281400* |

The values of PI are displayed in the table below for reference. This grouping of 4 decimals is how the program will show the results. The rows up to 120 decimals have been marked in both tables.

| *1-4* | *5-8* | *9-12* | *13-16* | *17-20* | *21-24* | *25-28* | *29-32* | *33-36* | *37-40* | *Decimals* |
|---|---|---|---|---|---|---|---|---|---|---|
| 3141 | 5926 | 5358 | 9793 | 2384 | 6264 | 3383 | 2795 | 0288 | 4197 | ***40*** |
| 1693 | 9937 | 5105 | 8209 | 7494 | 4592 | 3078 | 1640 | 6286 | 2089 | ***80*** |
| 9862 | 8034 | 8253 | 4211 | 7067 | 9821 | 4808 | 6513 | 2823 | 0664 | ***120*** |
| 7093 | 8446 | 0955 | 0582 | 2317 | 2535 | 9408 | 1284 | 8111 | 7450 | ***160*** |
| 2841 | 0270 | 1938 | 5211 | 555 | 9644 | 6229 | 4895 | 4930 | 3819 | ***200*** |
| 6442 | 8810 | 9756 | 6593 | 3446 | 1284 | 7564 | 8233 | 7867 | 8316 | ***240*** |
| 5271 | 2019 | 0914 | 5648 | 5669 | 2346 | 0348 | 6104 | 5432 | 6648 | ***280*** |
| 2133 | 9360 | 7260 | 2491 | 4127 | 3724 | 5870 | 660 | 6315 | 5881 | ***320*** |
| 7488 | 1520 | 9209 | 6282 | 9254 | 0917 | 1536 | 4367 | 8925 | 9036 | ***360*** |
| 0011 | 3305 | 3054 | 8820 | 4665 | 2138 | 4146 | 9519 | 4151 | 1609 | ***400*** |
| 4330 | 5727 | 0365 | 7595 | 9195 | 3092 | 1861 | 1738 | 1932 | 6117 | ***440*** |
| 9310 | 5118 | 5480 | 7446 | 2379 | 9627 | 4956 | 7351 | 8857 | 5272 | ***480*** |
| 4891 | 2279 | 3818 | 3011 | 9491 | 2983 | 3673 | 3624 | 4065 | 6643 | ***520*** |
| 0860 | 2139 | 4946 | 3952 | 2473 | 7190 | 7021 | 7986 | 0943 | 7027 | ***560*** |
| 7053 | 9217 | 1762 | 9317 | 6752 | 3846 | 7481 | 8467 | 6694 | 0513 | ***600*** |
| 2000 | 5681 | 2714 | 5263 | 5608 | 2778 | 5771 | 3427 | 5778 | 9609 | ***640*** |
| 1736 | 3717 | 8721 | 4684 | 4090 | 1224 | 9534 | 3014 | 6549 | 5853 | ***680*** |
| 7105 | 792 | 2796 | 8925 | 8923 | 5420 | 1995 | 6112 | 1290 | 2196 | ***720*** |
| 0864 | 344 | 1815 | 9813 | 6297 | 7477 | 1309 | 9605 | 1870 | 7211 | ***760*** |
| 3499 | 9999 | 8372 | 9780 | 4995 | 1059 | 7317 | 3281 | 6096 | 3185 | ***800*** |

## Design

The key problem with this algorithm is handling the values of the array f[..]. The solution built for this program is to store each value as ASCII bytes in a file. One file record can store up to 254 bytes which implies a maximum 127 values of f[..]. To make the process repeatable, the number of bytes must be even and fit precisely in each defined number of records (without empty space at the end). In the first table the best fit of number of records and number of bytes per record is given. This is the basis for storing all values of f[..]. As this is not linear, the number of records is retrieved from 2 alpha-based lookup vectors in lines 15-20. These have been marked with dotted lines in the table.

The values of f[..] are stored in the ASCII file PIF via routine 05 (line 154) and retrieved via routine 06 (line 178). To make these routines fast a number ***wxyz*** is encrypted and decrypted as follows:

- Encryption of **f[..] = *wxyz*** starts with splitting it into two numbers ***wx*** and ***yz***
- To avoid the NULL character, 100 is added to ***wx*** and 1 is added to ***yz***
- The ASCII characters are concatenated to the Alpha register: **CHR(*wx*+100) & CHR(*yz* + 1)**
- *Decryption* to get **f[..]** is the reverse process: **f[..] = VAL(*wx*) - 100 + VAL (yz) – 1**

The initialisation of f[..] with value 2000 is a sequential process thereby creating records and appending characters as per definition of the number of records and bytes per records from the lookup vector.

For retrieving and updating the value of f[..] the pointer depends on the value of b. This calculation is done from line 89 onwards and is based on the algorithm. Please note that the integer part points to the record and the decimals (division by 1000) point to the first character of the two-byte value of f[..]:

$$int(\frac{(b-1)}{(bytesperrecord/2)}) + \ 2 \times \frac{(b-1)\ mod\ (bytesperrecord/2)}{1000}$$


Several tests have been conducted to verify this pointer calculation process with sample values:

| | | | | | Pointer value for specific b | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Decimals | Array size | Bytes | Records | Bytes/Record | 1 | 2 | 70 | 71 | 140 | 142 | 199 | 201 | 210 | 223 | 240 | 279 | 280 | 419 | 421 | 2800 |
| 40 | 140 | 280 | 2 | 140 | 0,000 | 0,002 | 0,138 | 1,000 | 1,138 | MAXERR | MAXERR | MAXERR | MAXERR | MAXERR | MAXERR | MAXERR | MAXERR | MAXERR | MAXERR | MAXERR |
| 80 | 280 | 560 | 4 | 140 | 0,000 | 0,002 | 0,138 | 1,000 | 1,138 | 2,002 | 2,116 | 2,120 | 2,138 | 3,024 | 3,058 | 3,136 | 3,138 | MAXERR | MAXERR | MAXERR |
| 120 | 420 | 840 | 4 | 210 | 0,000 | 0,002 | 0,138 | 0,140 | 1,068 | 1,072 | 1,186 | 1,190 | 1,208 | 2,024 | 2,058 | 2,136 | 2,138 | 3,206 | MAXERR | MAXERR |
| 160 | 560 | 1120 | 5 | 224 | 0,000 | 0,002 | 0,138 | 0,140 | 1,054 | 1,058 | 1,172 | 1,176 | 1,194 | 1,220 | 2,030 | 2,108 | 2,110 | 3,164 | 3,168 | MAXERR |
| 200 | 700 | 1400 | 7 | 200 | 0,000 | 0,002 | 0,138 | 0,140 | 1,078 | 1,082 | 1,196 | 2,000 | 2,018 | 2,044 | 2,078 | 2,156 | 2,158 | 4,036 | 4,040 | MAXERR |
| 240 | 840 | 1680 | 7 | 240 | 0,000 | 0,002 | 0,138 | 0,140 | 1,038 | 1,042 | 1,156 | 1,160 | 1,178 | 1,204 | 1,238 | 2,076 | 2,078 | 3,116 | 3,120 | MAXERR |
| 280 | 980 | 1960 | 10 | 196 | 0,000 | 0,002 | 0,138 | 0,140 | 1,082 | 1,086 | 2,004 | 2,008 | 2,026 | 2,052 | 2,086 | 2,164 | 2,166 | 4,052 | 4,056 | MAXERR |
| 320 | 1120 | 2240 | 10 | 224 | 0,000 | 0,002 | 0,138 | 0,140 | 1,054 | 1,058 | 1,172 | 1,176 | 1,194 | 1,220 | 2,030 | 2,108 | 2,110 | 3,164 | 3,168 | MAXERR |
| 360 | 1260 | 2520 | 12 | 210 | 0,000 | 0,002 | 0,138 | 0,140 | 1,068 | 1,072 | 1,186 | 1,190 | 1,208 | 2,024 | 2,058 | 2,136 | 2,138 | 3,206 | 4,000 | MAXERR |
| 400 | 1400 | 2800 | 14 | 200 | 0,000 | 0,002 | 0,138 | 0,140 | 1,078 | 1,082 | 1,196 | 2,000 | 2,018 | 2,044 | 2,078 | 2,156 | 2,158 | 4,036 | 4,040 | MAXERR |
| 440 | 1540 | 3080 | 14 | 220 | 0,000 | 0,002 | 0,138 | 0,140 | 1,058 | 1,062 | 1,176 | 1,180 | 1,198 | 2,004 | 2,038 | 2,116 | 2,118 | 3,176 | 3,180 | MAXERR |
| 480 | 1680 | 3360 | 14 | 240 | 0,000 | 0,002 | 0,138 | 0,140 | 1,038 | 1,042 | 1,156 | 1,160 | 1,178 | 1,204 | 1,238 | 2,076 | 2,078 | 3,116 | 3,120 | MAXERR |
| 520 | 1820 | 3640 | 20 | 182 | 0,000 | 0,002 | 0,138 | 0,140 | 1,096 | 1,100 | 2,032 | 2,036 | 2,054 | 2,080 | 2,114 | 3,010 | 3,012 | 4,108 | 4,112 | MAXERR |
| 560 | 1960 | 3920 | 20 | 196 | 0,000 | 0,002 | 0,138 | 0,140 | 1,082 | 1,086 | 2,004 | 2,008 | 2,026 | 2,052 | 2,086 | 2,164 | 2,166 | 4,052 | 4,056 | MAXERR |
| 600 | 2100 | 4200 | 20 | 210 | 0,000 | 0,002 | 0,138 | 0,140 | 1,068 | 1,072 | 1,186 | 1,190 | 1,208 | 2,024 | 2,058 | 2,136 | 2,138 | 3,206 | 4,000 | MAXERR |
| 640 | 2240 | 4480 | 20 | 224 | 0,000 | 0,002 | 0,138 | 0,140 | 1,054 | 1,058 | 1,172 | 1,176 | 1,194 | 1,220 | 2,030 | 2,108 | 2,110 | 3,164 | 3,168 | MAXERR |
| 680 | 2380 | 4760 | 20 | 238 | 0,000 | 0,002 | 0,138 | 0,140 | 1,040 | 1,044 | 1,158 | 1,162 | 1,180 | 1,206 | 2,002 | 2,080 | 2,082 | 3,122 | 3,126 | MAXERR |
| 720 | 2520 | 5040 | 24 | 210 | 0,000 | 0,002 | 0,138 | 0,140 | 1,068 | 1,072 | 1,186 | 1,190 | 1,208 | 2,024 | 2,058 | 2,136 | 2,138 | 3,206 | 4,000 | MAXERR |
| 760 | 2660 | 5320 | 28 | 190 | 0,000 | 0,002 | 0,138 | 0,140 | 1,088 | 1,092 | 2,016 | 2,020 | 2,038 | 2,064 | 2,098 | 2,176 | 2,178 | 4,076 | 4,080 | MAXERR |
| 800 | 2800 | 5600 | 25 | 224 | 0,000 | 0,002 | 0,138 | 0,140 | 1,054 | 1,058 | 1,172 | 1,176 | 1,194 | 1,220 | 2,030 | 2,108 | 2,110 | 3,164 | 3,168 | 24,222 |

Because the GETREC instruction moves the pointer to the end of the copied part record the pointer must be set twice, for decryption and encryption (after updating f[..]). The pointer is calculated once and stored in R09. In the link to the program package (below) the program listing is also provided with comments.

## Runtime

Running the program Is more of a nostalgic interest because the performance is extremely low due to the number of loops (see first table) that needs to be handled. Simulations using the HP-41CX emulator V41 on maximum speed and turbo show that the runtime varies from 15 minutes, 1.5 hours to 4.5 hours for determining 40, 80 respectively 120 decimals:

| Decimals | 1-4 | 5-8 | 9-12 | 13-16 | 17-20 | 21-24 | 25-28 | 29-32 | 33-36 | 37-40 | | hh:mm:ss |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lapse Time 40 | 0:02:58 | 0:02:19 | 0:02:09 | 0:01:57 | 0:01:45 | 0:01:30 | 0:01:15 | 0:00:58 | 0:00:40 | 0:00:21 | **40** | 0:15:52 |
| | | | | | | | | | | | | **0:15:52** |
| Lapse Time 80 | 0:09:27 | 0:07:07 | 0:06:57 | 0:06:44 | 0:06:31 | 0:06:17 | 0:06:01 | 0:05:44 | 0:05:25 | 0:05:05 | **40** | 1:05:18 |
| | 0:04:44 | 0:04:21 | 0:03:58 | 0:04:10 | 0:03:06 | 0:02:38 | 0:02:10 | 0:01:39 | 0:01:08 | 0:00:03 | **80** | 0:27:57 |
| | | | | | | | | | | | | **1:33:15** |
| Lapse Time 120 | 0:18:57 | 0:14:11 | 0:14:01 | 0:13:47 | 0:13:37 | 0:13:20 | 0:13:04 | 0:12:47 | 0:12:28 | 0:12:08 | **40** | 2:18:20 |
| | 0:11:46 | 0:11:24 | 0:10:59 | 0:10:35 | 0:10:06 | 0:09:39 | 0:09:09 | 0:08:39 | 0:08:07 | 0:07:34 | **80** | 1:37:58 |
| | 0:06:59 | 0:06:22 | 0:05:46 | 0:05:08 | 0:04:28 | 0:03:46 | 0:03:03 | 0:02:20 | 0:01:34 | 0:00:48 | **120** | 0:40:14 |
| | | | | | | | | | | | | **4:36:32** |

Compared to the HP-41CX's performance, these runtimes could easily be multiplied by a factor 4.

## Example

| KEYSTROKES | DISPLAY | COMMENTS |
|---|---|---|
| | | Determine PI with 40 decimals |
| [XEQ] [ALPHA] PIDTW [ALPHA] | NR.D= 7 | Number of decimals required (in multiples 40) |
| 40 [R/S] | D 1= 3 14 1 | Enter 40; '3' and decimals 2-4 of PI shown |
| [R/S] | D5= 5926 | Decimals 5-8 |
| [R/S] | D9= 5358 | Decimals 9-12 |
| [R/S] | D 13= 9793 | Decimals 13-16 |
| [R/S] | D 17= 2384 | Decimals 17-20 |
| [R/S] | D21= 6264 | Decimals 21-24 |
| [R/S] | D25= 3383 | Decimals 25-28 |
| [R/S] | D29= 2795 | Decimals 29-32 |
| [R/S] | D33= 0288 | Decimals 33-36 |
| [R/S] | D37= 4 197 | Decimals 37-50 |
| [R/S] | NR.D= 7 | Provide number of decimals, e.g. 120 |
| 120 [R/S] | D 1= 3 14 1 | '3' and decimals 2-4 are shown of PI |
| | … | Continue using R/S to display decimals 5-116 |
| [R/S] | D 117= 0664 | Decimals 117-120 |
| [R/S] | NR.D= 7 | Enter excessive number of decimals, e.g. 800 |
| 800 [R/S] | 80 ,000000 | Program stops if there is insufficient room |

## Program Listing

```
01▪LBL "PIDTW"
02 CLX
03 STO 03
04 STO 08
05 100
06 STO 11
07 STO 04
08 ST* 04
09 11
10 "NR.D=?"
11 PROMPT
12 STO 00
13 20
14 /
15 "14142020202020"
16 >"242825"
17 X<Y?
18 "02040405070710"
19 X<Y?
20 >"101214"
21 2
22 -
23 AROT
24 ATOX
25 48
26 -
27 10
28 *
29 ATOX
30 48
31 -
32 +
33 STO 01
34 RCL 00
35 3,5
36 *
37 STO 06
38 2
39 *
40 RCL 01
41 /
42 STO 02
43 LASTX
44 *
45 1
46 +
47 7
48 /
49 0,49
50 +
51 XEQ 04
52 FIX 0
53 CF 29
54 RND
55 EMROOM
56 X<>Y
57 X>Y?
58 GTO 04
59 CRFLAS
60 RCL 01
61 STO 05
62 2000
63 SF 00
64 CLA
65 XEQ 05
66▪LBL 00
67 RCL 02
68 2
69 /
70 STO 07
71 INSREC
72 DSE 07
73▪LBL 01
74 APPCHR
75 DSE 07
76 GTO 01
77 DSE 05
78 GTO 00
79▪LBL 02
80 RCL 06
81 STO 05
82 2
83 *
84 STO 10
85 X<=0?
86 GTO 04
87 ISG 03
88▪LBL 03
89 RCL 05
90 1
91 -
92 STO Y
93 RCL 02
94 2
95 /
96 ST/ Z
97 MOD
98 500
99 /
100 X<>Y
101 INT
102 +
103 STO 09
104 XEQ 06
105 RCL 04
106 *
107 RCL 05
108 RCL 07
109 *
110 +
111 STO 07
112 1
113 ST- 10
114 RCL 07
115 RCL 10
116 MOD
117 XEQ 05
118 RCL 07
119 RCL 10
120 /
121 INT
122 STO 07
123 1
124 ST- 10
125 DSE 05
126 GTO 03
127 RCL 07
128 RCL 04
129 /
130 RCL 08
131 +
132 INT
133 "D"
134 ARCL 03
135 >"="
136 RCL 11
137 X>Y?
138 >"0"
139 10
140 *
141 X>Y?
142 >"0"
143 ARCL Y
144 PROMPT
145 3
146 ST+ 03
147 RCL 07
148 RCL 04
149 MOD
150 STO 08
151 14
152 ST- 06
153 GTO 02
154▪LBL 05
155 FC? 00
156 XEQ 07
157 RCL 11
158 /
159 INT
160 LASTX
161 FRC
162 RCL 11
163 *
164 1
165 +
166 XTOA
167 X<>Y
168 RCL 11
169 +
170 XTOA
171 FS?C 00
172 RTN
173 2
174 DELCHR
175 RDN
176 INSCHR
177 RTN
178▪LBL 06
179 XEQ 07
180 GETREC
181 ATOX
182 1
183 -
184 ATOX
185 RCL 11
186 -
187 LASTX
188 *
189 +
190 RTN
191▪LBL 07
192 CLA
193 RCL 09
194 SEEKPTA
195 X<>Y
196 RTN
197▪LBL 04
198 FIX 5
199 SF 29
200 SF 25
201 "PIF"
202 PURFL
203 CF 25
204 END
```

(349 bytes)

## Registers, Labels and Flags

| REGISTERS | COMMENTS |
|---|---|
| R00 | Number of decimals |
| R01 | Number of records |
| R02 | Number of bytes per record |
| R03 | Decimal group indication |
| R04 | Variable a=10000 |
| R05 | Variable b |
| R06 | Variable c |
| R07 | Variable d |
| R08 | Variable e |
| R09 | File pointer to f[b] |
| R10 | Variable g |
| R11 | Constant value 100 |

| LABELS | COMMENTS |
|---|---|
| LBL00 | Init f[b] and create new record |
| LBL01 | Init f[b] and append characters |
| LBL02 | Overall g loop |
| LBL03 | Loop b inside overall loop g |
| LBL04 | Delete text file, program end |
| LBL05 | Store 4-digit value f[b] |
| LBL06 | Retrieve 4-digit value f[b] |
| LBL07 | Set file pointer for b for f[b] |

| FLAGS | COMMENTS |
|---|---|
| 00 | First time access for file f[b] to indicate that pointer not needed |
| 25 | Ignore error to delete text file for f[b] |
| 29 | Hide decimal separator for display |

## References

Website: How to compute digits of pi? by Lopez-Ortiz
Website: Computing Pi in C by Lynn
Website Wolframs: Pi Formulas by Weisstein
Website: The BBP Algorithm for Pi by Baily (2016)
Website: A Brief History of Pi at math.com
Website: An efficient determination of the coefficients in the Chudnovskys' series for 1/PI by Milla (2020)
HP41 program: 560 digits of Pi calculated on the HP-41 by J-F Garnier
HP41 program: Example program: Deep Pi by Fabrice Bellard (1997)
Website MoHPC: PI Decimals calculation on HP41c at hpmuseum.org

## Downloads

The RAW/TXT format of the program is available via the website: PIDTW (in zip file).