PRODOMUS "ONE STEP" DATE AND SCHEDULING TOOLS 4

FEATURE SUMMARY 5

ONE TOUCH DATE HANDLING 5
DATE AND TIME HANDLING 5
CALENDARING 5
INTERNATIONALIZATION 6
IMPLEMENTATION 6

STEPS 7

CONVERTING FROM OTHER POPUP CALENDARS 8
UPGRADING FROM PRIOR VERSIONS OF PD DATE TOOLS 8
STEPS - DICTIONARY 9
STEPS – GENERAL 9
STEPS - POPUP CALENDAR 9
Calling the Calendar from a Control 9
Calendar Button Control Template 9
Drag and Drop 10
STEPS - DATE SCROLLING 10
STEPS - TIME ZONE INFORMATION 10
STEPS - 16-BIT APPLICATIONS 10
Date Information (DTE Files) 10

TRANSLATION 11

PDCAL.TRN, PDAPPT.TRN, AND PDSCHED.TRN 11
MULTI-LANGUAGE 11

TEMPLATE SUMMARY 11

GLOBAL EXTENSION TEMPLATE 12

CALENDAR PROCEDURE EXTENSION TEMPLATE 14

ADD HOLIDAY CODE TEMPLATE. 16

ADD SCHEDULED DATE CODE TEMPLATE 16

GETSCHEDULEQ CODE TEMPLATE 17

DELETE SCHEDULEQ CODE TEMPLATE 18

CHANGE SCHEDULEQ CODE TEMPLATE 18

TIME DROP CONTROL CONTROL TEMPLATE 19

RECURRENCE TEMPLATE 19

RECURRENCE TEMPLATE ENTRIES 20
ADVANCED BUTTON 21

PD CLASS REMOVAL TEMPLATE 22

CLASS SUMMARY 22

CONVENTIONS...22
DATE AND TIME DATA TYPES 22
DATETIME GROUPS..23
DATE CLASS..23
SCROLL CLASS 23
CALENDAR CLASSES...23
SCHEDULE CLASSES 23
PD LOCK CLASS..24

PDDATECT DATE CLASS 24

CONCEPTUAL EXAMPLE 24
Global embeds 24
Initialization procedure..24
FILES..24
PROPERTIES..24
METHODS..25
Initialization and Locale Information Handling...25
Display Functions 27
General Date Functions 27
General True/False Date Functions..27
Relative day Procedures 28
Time Zone Information...29

PDSCHEDCT SCHEDULE CLASS 29

CONCEPTUAL EXAMPLE 29
FILES..31
PROPERTIES..31
METHODS..32
Initialization and Locale Information Handling...32
General Date Methods 33
Holidays and Schedule 33
General True/False Date Functions..35

PDSCROLLCT SCROLL CLASS...36

CONCEPTUAL EXAMPLE 36
FILES..36
PROPERTIES..36
METHODS..37

SCROLL CLASS ALERT KEY ARRAY..38

PDCALCT CALENDAR BASE CLASS...38

FILES..38
PROPERTIES..38
METHODS..39

PDLISTCALCT LIST CALENDAR CLASS 41

CONCEPTUAL EXAMPLE 41
FILES..42
PROPERTIES..42

- METHODS....43

PDDROPCALCT DROP CALENDAR CLASS....44

- CONCEPTUAL EXAMPLE....44
- FILES....45
- PROPERTIES....45
- METHODS....45

PDPOPCALCT POPUP CALENDAR CLASS....46

- CONCEPTUAL EXAMPLE....46
- FILES....48
- PROPERTIES....48
- METHODS....49

PDQTMCT POPCAL THREAD MANAGER....50

- FILES....50
- PROPERTIES....50
- METHODS....50

HOLIDAYS AND SCHEDULED DATES....51

COLOR HANDING....52

DATE FILE CREATION UTILITY....52

SAMPLE DATE CLASS PROCEDURES....52

- SETTING THE LOCALE: 16 AND 32 BIT....52
- HOLIDAY AND BUSINESS DAY MESSAGE....53
- SETTING THE CLARION DATE ENVIRONMENT....53
- DISPLAYING FUNCTION RESULTS IN A QUEUE....54

SAMPLE SETTING RUNTIME POPUP CALENDAR OPTIONS....55

SAMPLE TIMEZONE PROCEDURE....56

SAMPLE APPOINTMENT CALENDAR....58

PRODOMUS ADVANCED SCHEDULING ADD ON....63

- ADVANCED SCHEDULING FEATURES....63
- EXAMPLE PROGRAM....63
- ADVANCED SCHEDULING CLASSES....63
  - *PDMonYrCT – Month Year File Drop Class....63*
  - *PDMonDayYrCT – Month Day Year File Drop Class....63*
  - *PdAppDayListCT – Appointment Day Calendar Class....63*
- STEPS – SCHEDULING FILE DEFINITION....63
- STEPS - APPOINTMENT PROCEDURE....66
- SCHEDULING TEMPLATES....67
  - *PD Month-Day-Year Entry Control Template....67*
  - *PD Day Appointment List Control Template....67*
- PDDT DAY APPOINTMENT DEFAULT DATA EXTENSION TEMPLATE....67
- PD MONTH-DAY-YEAR ENTRY CONTROL TEMPLATE....67

PD DAY APPOINTMENT LIST CONTROL TEMPLATE....68
  Display Date Tab....*68*
  Appointment File Tab....*69*
  Advanced - Additional ScheduleQ Assignments Button.....*70*
  Advanced - File Lookups.....*70*
  Appointment Queue Tab....*71*
  Advanced – Nesting....*71*
  Hot Fields Tab....*72*
  Color, Icon, and Classes Tabs....*72*
PDDT DAY APPOINTMENT DEFAULT DATA EXTENSION TEMPLATE....73

GLOBAL DATA....74

EQUATES....74
  Locale Equates....*74*
  Recurrance Flags....*74*
  First Week Rules....*74*
DATE GROUP....74

FREQUENTLY ASKED QUESTIONS....75

QUESTIONS AND ANSWERS....75
KNOWN ISSUES....77
  Drop Calendar....*77*

SAMPLE APPLICATIONS....77

FILE LIST....78

WARRANTY....78

LICENSE AGREEMENT....78

ABOUT....78

ProDomus “One Step” Date and Scheduling Tools

Feature Summary
Steps
Templates
Classes
Advanced Add Ons
- Appointment Template and Class
- (More to Come)

Sample Coding
- Sample Applications
- Sample Date Class Procedures
- Sample Time Zone Procedure

- Sample Appointment Calendar

Known Issues

File List

Warranty

License Agreement

Feature Summary

One Touch Date Handling

Goal: Create a “One Step” global template solution to date handling.

Result:

- Define range limits and initial values in the dictionary.
- Add one global extension. *Without further coding, fully localized date handling will have been added to your application: all date entries will have drop down calendars with buttons, entry scrolling, and, where appropriate, range limits*. *Change a global option and it will cascade to all entries which have not been overridden locally*.

Date and Time Handling

Goal: Create a tool that handles all types of date and time calculations and provides the basis for applications requiring scheduling procedures.

Result:

- An Internationalized Date Class library handles all date functions.
- A Schedule Class provides a basic tool for solving time and scheduling requirements. It includes methods for scheduled dates, holidays, business days, and recurring items. It is used by the Calendar Classes and is instantiated by the templates.
- Functions are included for calculating US Holidays and Easter.
- An example application shows how to import holidays from Microsoft Outlook.
- Scheduled items and holiday names can easily be displayed for any selected date.
- Schedule entries include options for requiring a unique schedule period or a warning message if there’s a conflict. Conflicts are examined for both recurring and uniquely scheduled items.

Calendaring

Goal: Create a base integrated set of calendaring classes that can be used as the basis for any type of calendar.

Result:

- A Calendar Base Class provides scheduling, range handling, day and month naming, and international date handling common to all calendars.
- A List Box Calendar applys the base class to all calendars using a list box. This is implemented with a control template that can populate a list calendar on a window. The List Box Calendar Class has all the functionality of the Calendar Base Class plus the functionality of a scrolling date class.
- A Drop Calendar applies the List Box Calendar to a special drop list created at run time.
- A Popup Calendar Class implements the list calendar in a threaded popup calendar.

Internationalization

Goals:

- Use information available in the Windows registry for internationalizing date handling. This includes, for example, rules for calculating the week of the year, the first day of the week, as well as full and abbreviated month and day names.
- Provide hooks for translation of class based text.

Results:

- The Date Class provides full 32-bit access to Calendar related registry information.
- If your application does not have access to the windows registry, you can use the 32-bit ***Date File Creation Utility*** to create an a file of registry information.
- Translation files and translation methods make single and multi-language translation easy.

Implementation

Templates. The PD "One Step" templates handle all date entries by simply populating one global extension template.

- This applies any range limits set up in Dictionary Field User Options.
- An Automatically populated procedure calendar extension templates can override the global settings for the procedure as a whole or any field.
- Several Code and Control Templates can be used to populate calendar buttons, recurring date entry, special date and time drop lists, and appointment lists.

Calendar Classes. Calendar Classes support a file drop calendar, a window calendar, and a popup calendar.

- All share basic features including display of holidays and scheduled dates, drag and drop capability, a popup menu, and help screen explaining calendar navigation.
- The popup calendar can be threaded using a built-in calendar thread management class.
- All calendars use the Date Class to support international information
- All calendars us the Scrolling Class to handle navigation.
- The popup calendar and window calendar also display the week of the year opposite each line of dates.
- The popup menu associated with the calendars provides a variety of "Jump To" functions.

Drop Calendars. The drop calendar and its associated button are created at run-time.

- Behaves like a file drop list.
- Unlike a "Popup Calendar", the drop calendar always locates itself immediately below or above the entry.
- The user also continues to be able to move to other locations on the window with a mouse click.
- The entry control is updated immediately as the user moves to different dates in the calendar.
- Pressing the escape key returns the original date.

Scrolling Class. The Scrolling Class uses an array of alerted keys to scroll change the current day by specified amounts. The Scrolling Class is used for date entries and all of the calendar List controls.

- With an array of alerted keys, dates may be scrolled by day, week, half month, month, same day of the month, first day of the month, last day of the month, first day of the year, last day of the year, quarter, same day of the quarter, year, same day of the year, first day of the first week of the year, and first day of the last day of the year.

- Includes Quicken alert keys, press “Y” to go to the end of the year and then the end of the next year or the Plus and Minus keys to increase or decrease the date by one.
- Tool tips display additional date information including the long date (day, month name, date, and four digit year), week of the year, day of the year, days left, holiday name, and scheduled date description.

Date Class Date Rules. Rules for date calculations vary from country to country -- the US standard of Sunday being the first day of the week and the first week of the year being the one in which January 1 appears does not apply in most other countries. The Date Class functions for calculating the Week Of The Year or the Day Of The Week recognize these rules.

True or False Date Functions. Is it a holiday? The Date Class will calculate US holidays and any other holidays added at run time. Holidays affect the calculation of business days: whether a date is a business day and the number of elapsed business days between dates. The Date Class also provide functions for determining whether it is a leap year, valid date, or week day.

Recurring Date. A special True or False function determines whether a date B is a recurring instance of date A

- Recurrency can be daily, weekly, every two weeks, every half month, monthly, bimonthly, quarterly, and yearly.
- Recurrency may have starting and ending dates specified.
- Recurrency can be on the same day of the month where applicable, i.e. the third Wednesday.
- Recurrence can also be specified for specific days of the week, i.e. every Monday, Wednesday and Friday every other week with a starting and ending date. Only one record is required for the scheduled item.

Time Zone Information. 32-bit applications may retrieve local names for Standard and Daylight Savings time, the UTC (Universal Time) bias, the date and time of transitions, and the user's current set up. The Time Zone Code Template retrieves the user's time zone information. The template allows you to perform different actions depending on function call results: unknown time zone, standard time, daylight time, and function failure.

Source Code. ProDomus Date Tools include source code for all classes except the Date Class.

Appointment Class Add On. An Appointment Class, derived from the Schedule Class, provides a file handling interface that makes creation of an appointment list virtually the same as creating any other browse list.

- A control template populates the list plus add, change, and delete buttons.
- Methods can be overridden in the embed editor.
- Tabs are provided for icons, color, and cell level styles.
- Currently ABC only.

Steps

The following are covered in separate topics:

- Upgrading. Steps for upgrading from prior versions of PD Date Tools.
- Converting. Steps to simplify the conversion from other Third Party Calendars
- Dictionary. Steps for setting up the dictionary including Field User Options.
- Popup Calendar. Steps for using the calendars.
- Date Scrolling. Steps for using the date scrolling class.
- Time Zone Information. Steps for getting time zone information.

- 16-Bit Applications. Steps when creating 16-bit applications.

Converting from Other Popup Calendars

If you are using other popup calendars that are populated from templates, we suggest modifying their template code to simply hide the buttons at run time. This will avoid you having to delete all buttons at the cost of some code overhead. You may, of course, delete them one at a time. Also check for any embed code that UNHIDEs the buttons at run time.

Example Template Code to Hide Other Third Party Calendar Buttons:

```
ABC:
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE'),PRIORITY(8100)
HIDE(%CalendarLookupButtonEquate)
DISABLE(?CalendarLookupButtonEquate)
#ENDAT

LEGACY:
#AT(%AfterWindowOpening)
HIDE(%CalendarLookupButtonEquate)
DISABLE(?CalendarLookupButtonEquate)
#ENDAT
```

Upgrading from Prior Versions of PD Date Tools

The “One Step” upgrade rewrites most of the code for the templates and libraries. To upgrade an application will require at least the following.

1. **Function Library Class Name**. Change the prefix to function calls from PDDT_ to the class “object name” PDDate. For example, PDDT_GetMonthName(MyDate) should be changed to PDDateC.GetMonthName.(MyDate).
2. **Initialization**. All initialization is now done from the global extension template. All code generation from the initialization code template has been eliminated. Note your initialization code and enter it in the Global Template in the Date Class button. The template, by default, initializes the date class to the local user’s machine.
3. **Global Extenstion Template/One Step Setup Tab/Class To Use Entry**. Select “Scrolling Entries” from the drop down. This will prevent the auto creation of drop down buttons which migh conflict with previously populated calendar buttons.   You can override this at a procedure level. If you have not populated calendar buttons, use the default entry.
4. **Scroll Class Extension Template**. This has been removed. Scrolling entries are created automatically unless you select “none in the Class to Use entry.
5. **Exclamation.** The new library does not require a “!” before “Default” date calendar entries to use a variable. The template code removes prior exclamation point characters so that you do not have to do this manually.
6. **Holidays and Scheduled Dates**. The scheduling methodology has change that the Date Class maintains an internal queue of scheduled dates. Any holidays and scheduled dates should be added using the AddHoliday and AddScheduled methods. This can take a descriptive name as the second parameter which will display in scrolling date fields and calendars.
7. **PDCal.trn**. Some default values have been moved from the template entries to the PDCal.trn file. Use this file to change icons and text.
8. Please notify ProDomus of any other issues that arise and check www.prodomus.com for any additional instructions.

Steps - Dictionary

1. **Range Limites**. For each date field with range limits, select Field Properties/Options/User Options. Add PDDTRangeLimits([Low][,High]). Parameters may be fields, functions, or numbers. Example: PDDTRangeLimits(TODAY()-60,g:date+365). Note that the range limit checks will allow no entry if the field is not required.
2. **DateTimeGroups.** The scheduling functions work best with groups consisting of a date and time field defined consistently as either two LONG data types or a DATE and TIME data type.
3. **DateTime Created Group.** The schedule class works best with a key based on the date and time created. This can be initialized by entering *PDSchedC.GetDateTime()* to the Initial Value entry in the dictionary. This entry can be used at relatively low risk of duplication as the unique key for the schedule. You may alternatively use the first field in the group or the combination as autoincremented field.
4. **DateTime Modified Group.** You may include a date and time modified group in any file. This will be automatically set in any form if you add *PDDTModified* in the User Options for the DateTimeModified group in the dictionary.
5. **DateTime Schedule Group.** If scheduling dates with times, add a DateTime group for the scheduled date and time.
6. **DateTime Ending Group.** If scheduled items will have and ending date and time, add and Ending DateTime group.

Steps – General

1. **Register the Templates**. Register PDDT_AB.TPL and PDDT.TPL (ABC and Legacy respectively).
2. **"One Step" Date Tools Global Extension Template**. Populate this global extension template. For detailed instructions for overriding the default entries, see the related template help topic.
3. **Calendar Procedure Extension Template**. This template will be populated automatically when the global extension template is populated. For detailed instructions for overriding the default entries, see the PD Calendar Procedure Extension topic.
4. **Multi-DLL Applications.**
    a. Add the global extension to the application the "File" APP file.
    b. Add the global extension to any procedures with date or time handling.
    c. Add the global extension to the EXE APP file.

Steps - Popup Calendar

Calling the Calendar from a Control.

See the Procedure Extension Template: Steps -- General

Calendar Button Control Template

The calendar button control template adds a button to a window.

1. A*dding the Button.* Select Populate/Control Template and select the PDDTPopCalButton -- International Calendar Button.
2. *Button Properties.* Right click the button and select Actions/Button Properties.
    a. *Field To Update.* If the calendar is to update a field, enter the field.
    b. *Default Value.* Enter a default value, typically TODAY(). The default value will be used when the calendar displays if there is not value in the field to update.

    c. *Open as MDI Window.* Check to thread the calendar. Only one threaded calendar may be opened on a thread, i.e. only one on a form.
    d. *Center Window.* Check to center the window when opened.
    e. *Window Xpos*. Enter code to locate the XPOS of the window. Note that the calendar will actually be called from the procedure frame. You will need to add the calling control position to the position of the calling window, i.e. 0{Xpos}+?{Xpos}+Offset.
    f. *Window Ypos*. Enter code to locate the YPOS of the window. Note that the calendar will actually be called from the procedure frame. You will need to add the calling control position to the position of the calling window, i.e. 0{Ypos}+?{Ypos}+Offset.
    g. *Use Range Limits*. Check and enter range limits to override any global limits.

Drag and Drop

A date may be selected from the popup calendar and dropped on a control. The calendar uses the clipboard to pass the selected date. You can set DROPIDs from the Calendar Procedure Extension template.

Steps - Date Scrolling

Scrolling entries may be created from the Procedure Extension Template. See the Procedure Extension Template: Steps -- General

Steps - Time Zone Information

Time zone information is available to 32-bit programs. A Code Template "PDDTTimeZone, Get Time Zone Information" may be placed in an embed point. This places time zone information into a group structure. The elements of the time zone group may be use directly or in any assignment statement.

The template allows you to make use of the return value of the function call. There are four possible return values: Unknown Information Standard Time, Daylight Time, and Failure.

1. *Assign Result Text to a Field (Field).* Text may be assigned to a field depending on the result of the function call. Enter the field to receive the result.
2. *Use Text Equates* (Check). You may use text contained the PDDT.INC file.
3. *Assign &Unknown Time* (Check). Assign text for this result.
4. *Assign Standard/Daylight Time (Check)*. Assign text for this result.
5. *Assign Failed* (Check). Assign text for this result.
6. *Do actions after function call* (Check). You may code specific responses to the function call. If you check this box, enter code into one or more of the following:
    a. *Unknown Zone Action*
    b. *Standard Zone Action*
    c. *Standard Zone Action*
    d. *Failed Action*

Steps - 16-Bit Applications

See also steps under:
Global Extension Template
Procedure Extension Template

The steps for creating 16-bit applications are the same as those for 32-bit applications with the following exceptions:

Date Information (DTE Files)

The registry is not available for date information: month names, day names, first day of the week, and first week of the year rules. This information is also not available in the win.ini file. The Date File Creation Utility, a 32-bit program, can be used to create ini type files that can be read by the GetDateGroup method. This file has the following structure:

```
[enuDateGroup]
FirstWeekRule=0
FirstDayRule=6
FirstBusDOW=1
LastBusDOW=5
Month=January   etc
Mon=Jan etc.
Day=Sunday  etc.
AbbrevDay=Sun etc.
```

The section heading consists of a three character language code and "DateGroup." The variables provide information of date rule, month names, and day names. The business day rules are not part of the windows registry; default values are provided.

As suggested way to obtain the users language code from the Win.ini file is:

```
sLanguage     STRING(3)
CODE
...
sLanguage = GETINI('Intl','sLanguage','enu')
!Where enu is the default.  You may vary this.
```

Alternatively, you might create a file with language codes and allow the user to make one or more selections which are supported in the DTE file.

This information is retrieved from the DTE file with the PDDT_GetDateGroup(DateGroup,Prefix,DteFile) procedure.

Translation

PDCal.trn, PDAppt.trn, and PDSched.trn

1. Copy these to your application directory. All strings and certain other default values used by the date tools are set up as equates in this file.
2. Modify the entries in this file as needed. Changes will be compiled into the application.

Multi-Language

The Scroll Class and Calendar Base Class contain Virtual methods to assist in translation. You may create derived Clarion ABC Compliant Classes that implement these methods using your translation tool of choice.

1. Scroll Class. Derive a TranslateString method from using the PDScrollCT as the base method.
2. Enter the new class in the global template by selecting the Classes tab and then the Scroll Class Button.
3. Calendar Classes. The base calendar class has Translate(Window) and TranslateString Methods that apply to all other derived calendar classes. Create derived classes using PDListCalCT, PDDropCalCT, and PDPopupCalCT as base classes, applying methods using your translation tool of choice.
4. Enter the new derived classes by selecting the Classes tab on the Global Extension and then the Calendar Button.

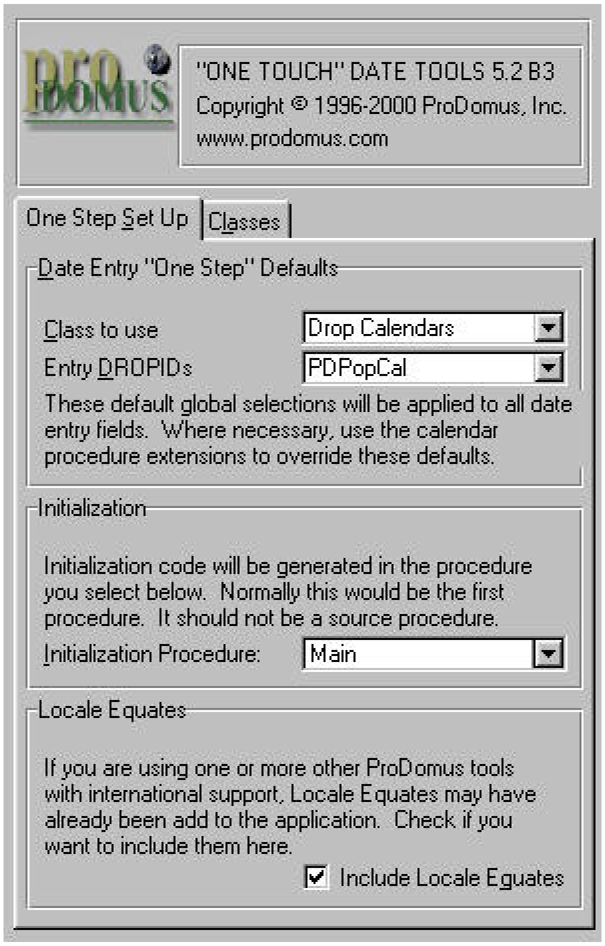
Template Summary

PD One Step Date Tools contain the following templates

- **"One Step" Date Tools Global Extension Template**. Simply populating this template provides basic date handling for all date entries in template generated window procedures. You may easily override any default entries.
- **Calendar Procedure Extension Template**. Procedure templates are generated by the global by adding the global extension. By default, these add functionality to all date entries as specified in the global extension. These may be overridden locally. This template creates drop calendars, scrolling entries, drop ids, and can be used to call a popup calendar from a menu.
- **Calendar List Control Template**. This template populates a small calendar on a window. This calendar may be used as a source for dragging dates to an entry.
- **International Calendar Button Control Template**. This template populates a popup calendar button. The button may be used to update a specific entry field. It may also be threaded and used as a source for dragging dates to a date entry.
- **Add Holiday Code Template**. This template may be used to add a holiday to the Date Class.
- **Add Scheduled Date Code Template**. This template adds scheduled dates to the Date Class.
- **Get ScheduleQ**. Returns a Schedule Queue with or without times and timed items or To Do Items.
- **Delete ScheduleQ**. Deletes an item from the Class Schedule Queue and related queues.
- **Change ScheduleQ**. Changes the Class Protected Schedule Queue..
- **Time Drop Control**. Creates a file drop list for entering time.
- **Get Time Zone Information Extension Template**. This retrieves time zone information.
- **Time Drop Control Template**. Creates an file drop entry with a list of times starting at a specified time with intervals and number of intervals as specified.
- **Date and Recurrence Entry Control Template**. Creates entry fields for the ULONG recurrence bit-mapped flag field. For use with forms.
- **Day Appointment List Control Template.** Creates an Appointment List with Add, Change, and Delete Buttons.
- **Month-Day-Year Entry Control Template**. Creates a date entry fields consisting of drop down lists for the Monday, Day, and Year with Next and Previous Buttons.
- **Day Appointment Default Data Extension**. Creates default local data for the Appointment List Control bracket fields.
- **PD Class Removal Template.** This template is provided separately. It can be added to any application to remove classes. For example, if you do not want to use the Date Tools classes in another application, add the template and check Remove Date Tools.

Global Extension Template

1. *Register the Templates.* PDDT includes two template chains: PDDA.TPL for ABC applications and PDD_.TPL for the clarion class chain. Select Setup/Template Registry and then select the Register template. These templates will be located in the Template folder.
2. *Add the ProDomus Date Tools Global Extension.* From the application properties window, select Global/Extensions. From the Class - PDDTDateTools select ProDomus Date Tools Global Extension.



"ONE TOUCH" DATE TOOLS 5.2 B3
Copyright © 1996-2000 ProDomus, Inc.
www.prodomus.com

Date Entry "One Step" Defaults

Class to use: Drop Calendars
Entry DROPIDs: PDPopCal

These default global selections will be applied to all date entry fields. Where necessary, use the calendar procedure extensions to override these defaults.

Initialization

Initialization code will be generated in the procedure you select below. Normally this would be the first procedure. It should not be a source procedure.

Initialization Procedure: Main

Locale Equates

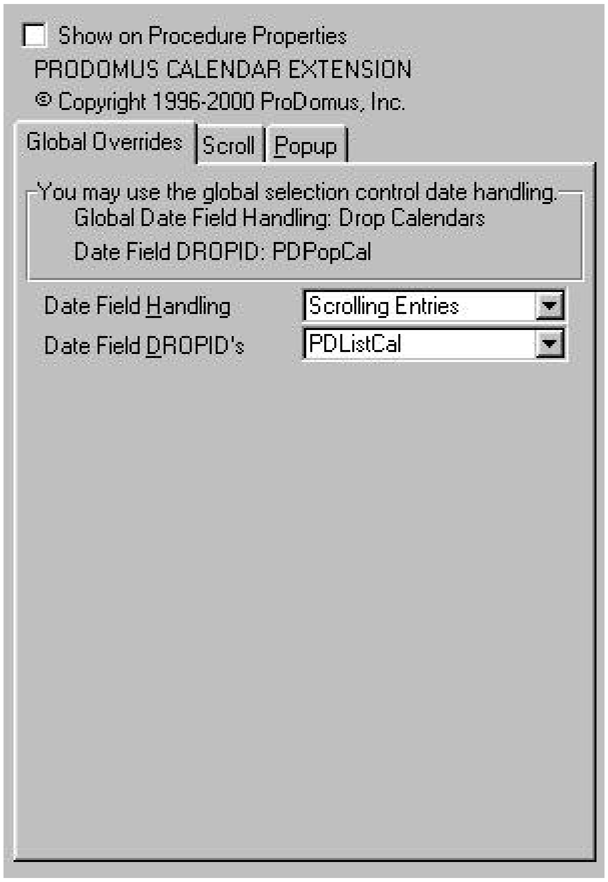
If you are using one or more other ProDomus tools with international support, Locale Equates may have already been add to the application. Check if you want to include them here.

☑ Include Locale Equates

3. *One Set Setup Tap.*
    a. *One Step Setup Defaults.* Select the default class to use for date entry and the default drop id.
    b. *Initialization. (Main EXE)* Select the first procedure in the application. This procedure must include embed where initialization code may be placed.
    c. *Locale Equates.* Check this to include locale equates. If these have already been added by other prodomus templates, uncheck it.
4. *Classes Tab.*
    a. *Date Class Button.*
        i. *Date Class Setup – 32-bit*. Defaults use Monday through Friday as business days and set the library to the user's locale. You may change the business week and optionally add a Locale Field. If you use a field, you must code this to contain a valid locale id number.
        ii. *Date Class Setup – 16-Bit*. Initialization File (Filename.dte). You may specify an initialization file to replace the US defaults. This requires information from the registry that is not available to 16-bit applications. A substitute file may be provided using the DTE extension. This file may be created from a 32-bit application using the CreateDTE(FileName) function or by running the Date File Creation Utility. You may specify a variable using an exclamation point or a specific file.
    b. *Calendar Classes Button.*
        i. *Base Class To Use.* You may select alternative classes to use for the popup, drop, and window list calendars.
        ii. *Event Calendar Equate.* You may also change the default valued used for EVENT:Calendar, an event used by the popup calendar.
]

        iii. *Override Button.* Use this to change default calendar captions and colors.
    c. *Scroll Class Button.* Enter an alterative scroll class.
    d. *16-Bit Applications*. The 16-bit version of the global template allow you to enter alternative classes. You must also enter the class include file.

Calendar Procedure Extension Template



☐ Show on Procedure Properties

PRODOMUS CALENDAR EXTENSION

© Copyright 1996-2000 ProDomus, Inc.

You may use the global selection control date handling.

Global Date Field Handling: Drop Calendars

Date Field DROPID: PDPopCal

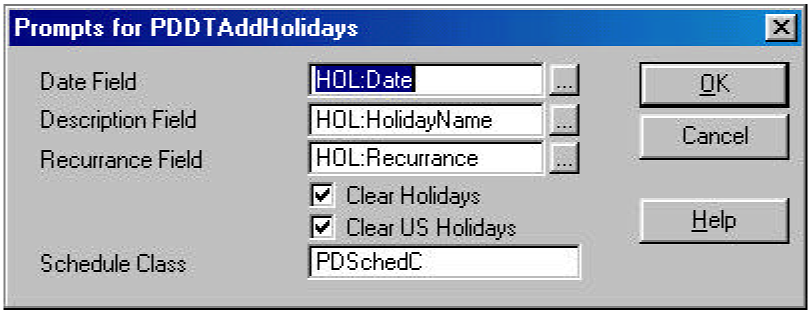
Date Field Handling: Scrolling Entries

Date Field DROPID's: PDListCal

1. *Global Overrides.* The procedure calendar template is generated automatically when the global template is added. Date handling defaults to the global settings. These may be changed as follows:
    a. *Date Field Handling.* You may select Global, Drop Calendars, Scrolling Entries, or No Action. These entries will affect all date fields on the window.
    b. *Date Field DropIDs.* All date fields will have the drop id specified. Popup and drop calendars use PDPopCal. The window calendar uses PDListCal. If the window has a calendar populated, then changing the drop id to PDListCal and the Field Handling to Scrolling Entries is recommended.
2. *Drop Tab.* If date handling is set to drop calendars, the Drop Tab will be visible.
    a. *Alternative Date Class.* Enter an alternative date class if desired. You must populate this class if entered.
    b. *Use default Colors.* If checked, colors set in the global template will be used.
    c. *Drop Calendar Overrides Button.* Use this to change field level behavior and add additional controls to hide if they show through the drop calendar.
        i. *Date Control.* The control to override.
        ii. *Action After Date Selection.* Enter code to generate after a date is selected. Suggestion: Use this to call a routine.
        iii. *Show Drop Calendar.* (reserved)

        iv. *Range Low and Range High.* Enter range limits. This will override any dictionary settings.
        v. *Force Above.* Forces the list to display above rather than below the entry control.
        vi. *Use Default Colors.* Select to use to global colors. Use clear to use class defined colors.
        vii. *Alternative Date Class.* Enter an alternative date class for the field if desired.
        viii. *Controls to Hide Button.* Press to add controls that should be hidden when the drop calendar drops. Use this to add controls only when testing shows that they are displaying when the calendar drops.
3. Scroll Tab.
    a. *Alternative Date Class*. Enter an alternative date class if desired. You must populate this class if entered.
    b. *Scrolling Date Class Overrides*. Press this to enter field level overrides.
        i. *Date Entry Control*. Enter override control.
        ii. *Initial Value*. Enter the default value. This will be used by the HOME key if the entry does not contain a date.
        iii. *Action After Scroll*. Enter code to generate after a date is selected. Suggestion: Use this to call a routine.
        iv. *DropID*. Use this to change the drop id for the field.
        v. *Date Range Limits*. Check to override global date ranges. Enter range limits.
        vi. *Alter Key Overrides*.
            1. *Clear Standard Key Array Values*. You can clear all alert keys and add you own.
            2. *Use Quicken Alert Keys*. Check to use Quicken alert keys.
            3. *Use the insert button to add or change alerted keys*.
4. Popup Tab. Use this tab to call the popup calendar from a menu item.
    a. *Call Popup Calendar from control*. Check
    b. *Calling Control*. Select the calling control.
    c. *Field to Update*. Enter a field to update if applicable.
    d. *Default Value*. Enter a default value.
    e. *Clear Standard Flags*. Check to clear standard flags.
    f. *Open as MDI*. Check to open the calendar as a threaded window.
    g. *Center Window*. Check to center the window. If not checked and no Xpos or Ypos are entered, the calendar will open in the vicinity of the calling control.
    h. *Window Xpos*. Enter code to locate the XPOS of the window. Note that the calendar will actually be called from the procedure frame. You will need to add the calling control position to the position of the calling window, i.e. 0{Xpos}+?{Xpos}+Offset.
    i. *Window Ypos*. Enter code to locate the YPOS of the window. Note that the calendar will actually be called from the procedure frame. You will need to add the calling control position to the position of the calling window, i.e. 0{Ypos}+?{Ypos}+Offset.
    j. *Use Range Limits*. Check and enter range limits to override any global limits.
    k. *Overrides Button*. Use to change default captions and calendar colors.

Add Holiday Code Template.



Date Field: HOL:Date
Description Field: HOL:HolidayName
Recurrance Field: HOL:Recurrance
☑ Clear Holidays
☑ Clear US Holidays
Schedule Class: PDSchedC

This template may be used to add a holiday to the Date Class.

Date Field. Enter a date (recurring items will appear starting with this date).

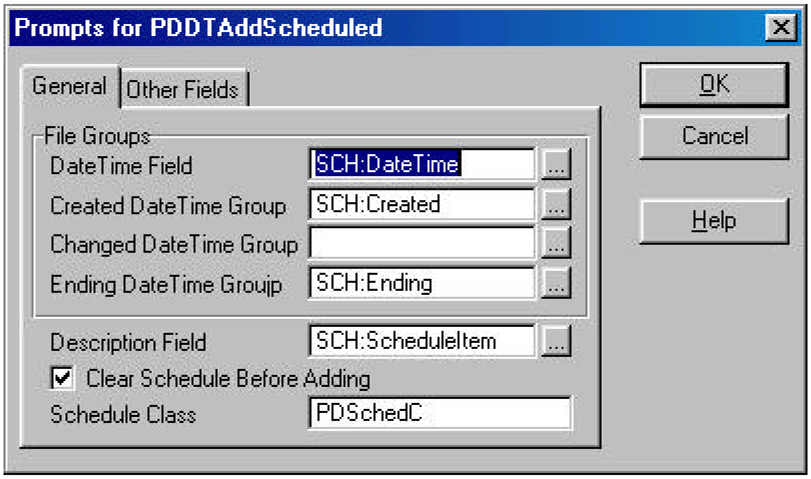
Description Field. Enter a short name for the holiday.

Recurrance Field. Enter the ULONG recurrance field.

Clear Holidays. Enter to clear any previously loaded holidays.

Clear US Holidays. Check to disable the use of US Holidays added by the USHolidays class method.

Add Scheduled Date Code Template



DateTime Field: SCH:DateTime
Created DateTime Group: SCH:Created
Changed DateTime Group:
Ending DateTime Grouip: SCH:Ending
Description Field: SCH:ScheduleItem
☑ Clear Schedule Before Adding
Schedule Class: PDSchedC

This template adds scheduled dates to the Date Class.

Created DateTime Group. A DateTime group. This field is used to relate internal schedule information and related files.

- The group should be declared in the dictionary.
- The primary key does not need to use both fields.
- Note that if you use a DateTime group, its value may be initialized by entering PDSchedC.GetDateTime in the initial value for the group in the Dictionary.
- Note that if the entry is a recurring date, recurrence will not begin before the specified date.

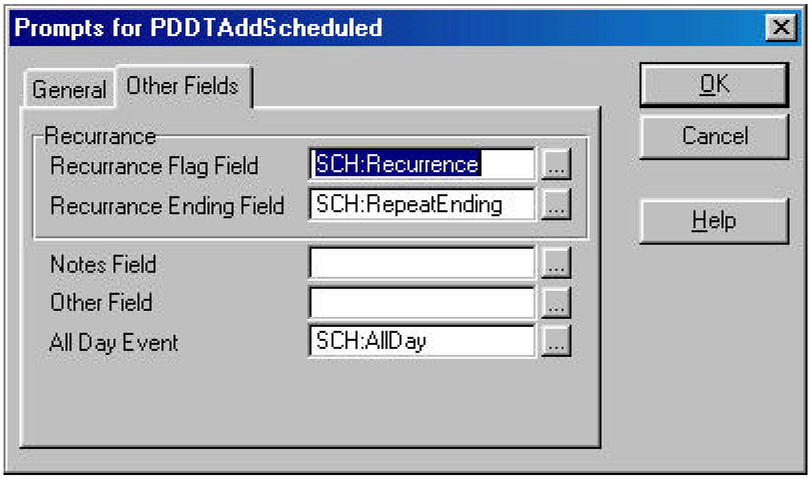
Changed DateTime Group. Optional DateTime group. You can use this to monitor the date and time of schedule changes or to add additional file relationships. Note that you can automatically set the value for this field by adding *PDDTModified* in the Field User Options entry in the Dictionary.

Ending DateTime Group. Optional ending date and time of the scheduled item.

Description Field. Enter a short name for the holiday.

Clear Schedule. Clear existing schedules previously loaded. Note that several schedules such as group schedules can be added.

Schedule Class. The schedule class to use. By default this is the template created PDSchedC.



Prompts for PDDTAddScheduled

Recurrance

Recurrance Flag Field: SCH:Recurrence

Recurrance Ending Field: SCH:RepeatEnding

Notes Field:

Other Field:

All Day Event: SCH:AllDay

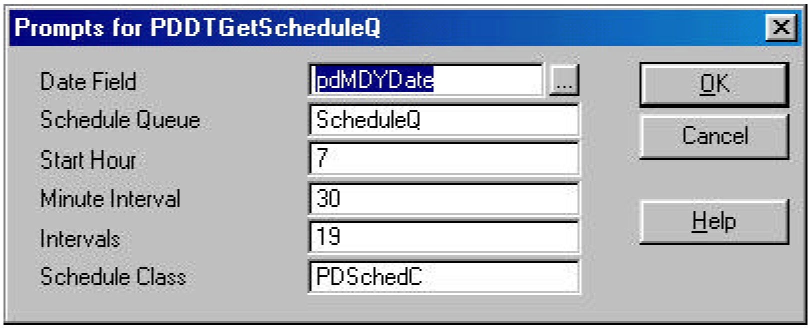
Recurrence Flag Field. Enter the ULONG recurrence field.

Recurrence Ending Field. A date after which a recurring date that will no longer recur.

Notes Field. Optional Notes field.

Other Field. Optional Other field.

GetScheduleQ Code Template



Prompts for PDDTGetScheduleQ

Date Field: pdMDYDate

Schedule Queue: ScheduleQ

Start Hour: 7

Minute Interval: 30

Intervals: 19

Schedule Class: PDSchedC

Returns a schedule Q with time increments.

Date Field. Date of the ScheduleQ

Schedule Queue. Locally declared ScheduleQ.

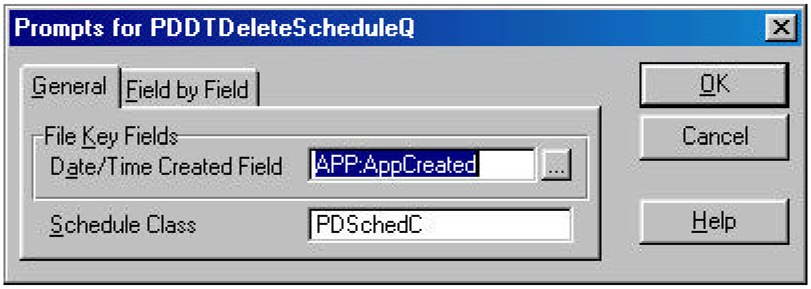
Start Hour. Optional start hour. If not entered, the schedule will include times and untimed items without time intervals. This may be a literal or variable entry.

Minute Interval. Optional time interval in minutes. This may be a literal or variable entry.

Intervals. Optional number of time intervals. This may be a literal or variable entry.

Date Class. The schedule class. This defaults to ***PDSchedC*** as created by the templates.

Delete ScheduleQ Code Template



Date/Time Created Field APP:AppCreated

Schedule Class PDSchedC

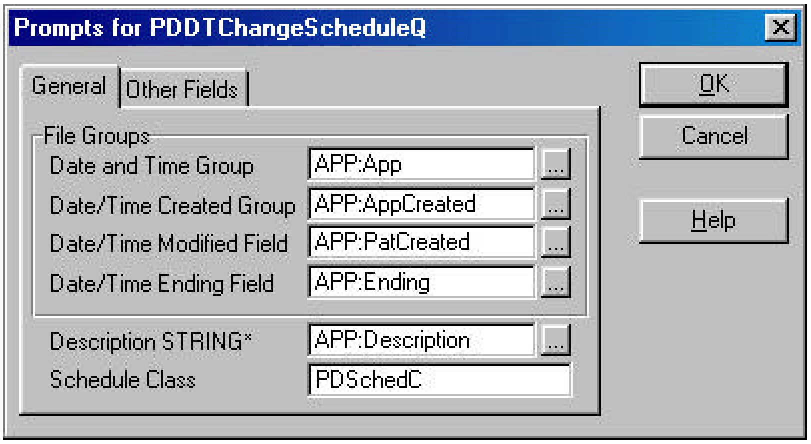
Deletes an item from schedule Q and related queues. DateTime groups must be declared to use this template.

Date/Time Field. The DateTime group key field.

Recurrance Field. Recurrance field.

Date Class. The schedule class. This defaults to ***PDDateC*** as created by the templates.

Change ScheduleQ Code Template



Date and Time Group APP:App

Date/Time Created Group APP:AppCreated

Date/Time Modified Field APP:PatCreated

Date/Time Ending Field APP:Ending

Description STRING* APP:Description

Schedule Class PDSchedC

Changes an item in the scheduleQ.

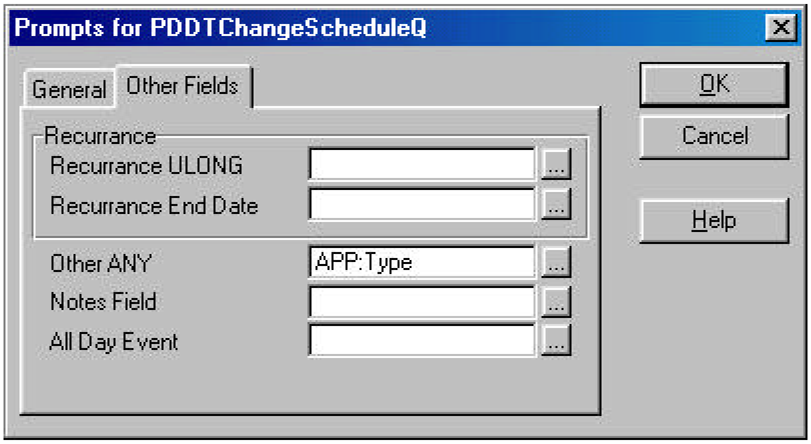
Date and Time Group. Group containing the scheduled Date and Time

Date/Time Modified Group. Optional modified DateTime group.

Date/Time Ending Group. Optional ending Date and Time Group.

Description STRING*. Required short description field.

Schedule Class. Schedule Class to use. Defaults to PDSchedC.



Recurrance ULONG	
Recurrance End Date	
Other ANY	APP:Type
Notes Field	
All Day Event	

Recurrence ULONG. Recurrance field.

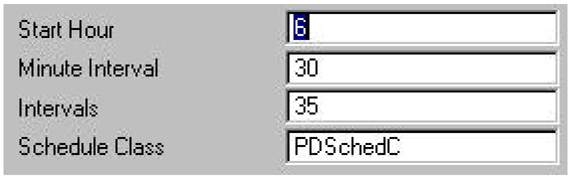
Recurrence End Date. The recurrence ending date field.

Other ANY. Optional Other field.

Notes STRING. Optional Notes field.

All Day Event. A Byte field with a TRUE value if the scheduled event is an all day event.

Time Drop Control Control Template



Start Hour	6
Minute Interval	30
Intervals	35
Schedule Class	PDSchedC

Creates an file drop list for entering time.

Start Hour. Starting time.

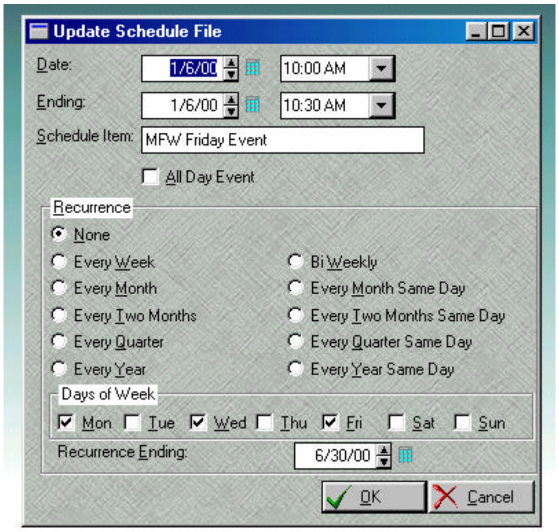
Minute Interval. Interval in minutes.

Intervals. Number of intervals.

Date Class. The schedule class. This defaults to ***PDSchedC*** as created by the templates.

Recurrence Template

The PDDT Recurrence Template is designed for use with a Form Template. It adds a recurrence OPTION control, check boxes for recurring day of the week entries, a ENTRY control for entering a recurrence ending date, and a check box for all day events.



Update Schedule File

Date: 1/6/00 10:00 AM

Ending: 1/6/00 10:30 AM

Schedule Item: MFW Friday Event

☐ All Day Event

Recurrence

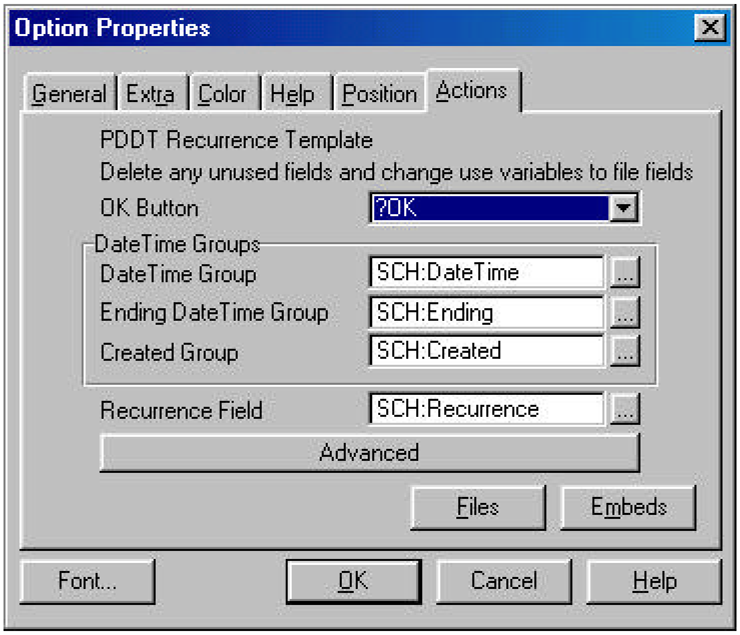
● None
○ Every Week ○ Bi Weekly
○ Every Month ○ Every Month Same Day
○ Every Two Months ○ Every Two Months Same Day
○ Every Quarter ○ Every Quarter Same Day
○ Every Year ○ Every Year Same Day

Days of Week

☑ Mon ☐ Tue ☑ Wed ☐ Thu ☑ Fri ☐ Sat ☐ Sun

Recurrence Ending: 6/30/00

Recurrence Template Entries



Option Properties

PDDT Recurrence Template
Delete any unused fields and change use variables to file fields

OK Button: ?OK

DateTime Groups
DateTime Group: SCH:DateTime
Ending DateTime Group: SCH:Ending
Created Group: SCH:Created

Recurrence Field: SCH:Recurrence

OK Button. The OK button for the scheduled item.

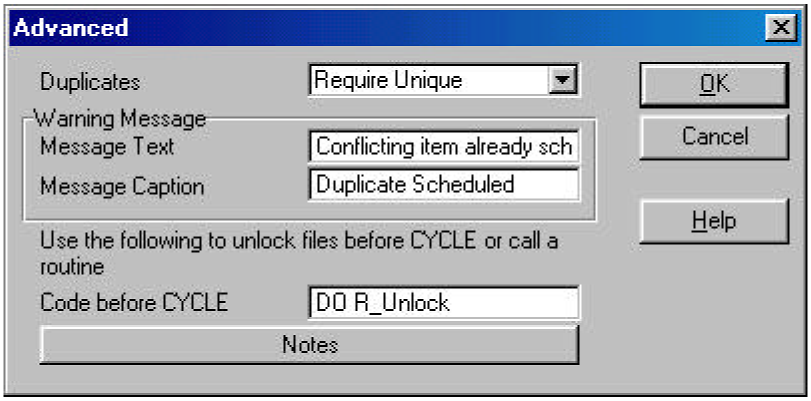
DateTime Group. Required. The DateTime group for the scheduled starting date and time.

Ending DateTime. Required. The DateTime group for the scheduled ending date and time.

Created Group. Required. The DateTime group for the unique key for the schedule record.

Recurrence Field. The Recurrence Field in the Schedule File.

Advanced Button



Duplicates: Require Unique

Warning Message

Message Text: Conflicting item already sch

Message Caption: Duplicate Scheduled

Use the following to unlock files before CYCLE or call a routine

Code before CYCLE: DO R_Unlock

Duplicates. This entry determines how conflicting scheduled items are handled. Options are:

- Allow Duplicates
- Warning Message
- Require Unique

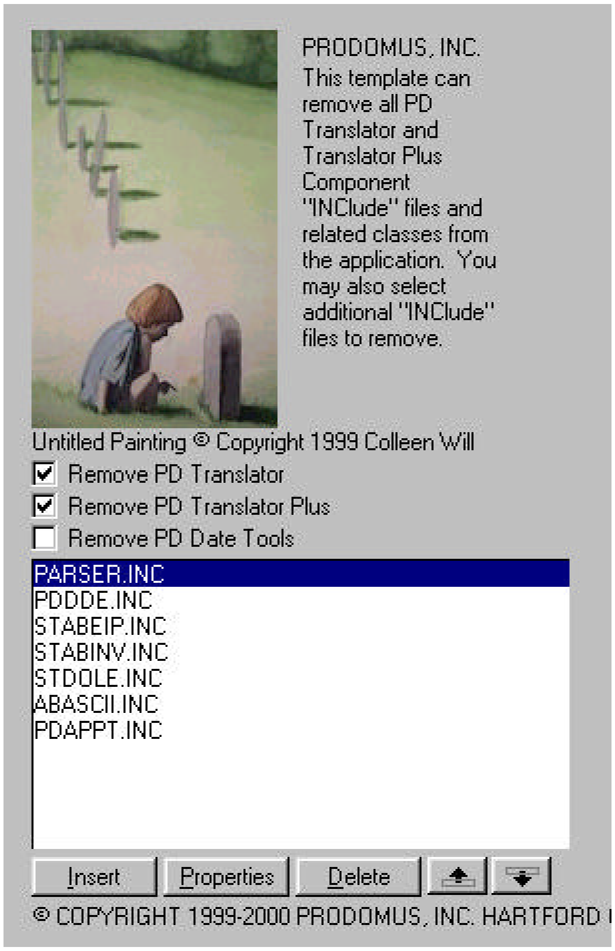
If Warning Message or Require Unique are selected, a message will be displayed indicating that a conflicting item has been scheduled and the date and time of the conflicting item. This uses currently loaded schedule information. In a multi user environment, it is suggested that the schedule file(s) be locked and the schedule reloaded prior to the PDSchedC.IsSchedule method call generated by the template. The file should be unlocked after the call.

Message Text. Default: “Conflicting item already scheduled:” This entry will be followed by the item’s description, date, and time.

Message Caption. Defautl: “Duplicate Scheduled”.

Code before CYCLE. Use this hook to unlock files before CYCLE after the warning message.

PD Class Removal Template


PRODOMUS, INC.
This template can remove all PD Translator and Translator Plus Component "INClude" files and related classes from the application. You may also select additional "INClude" files to remove.

Untitled Painting © Copyright 1999 Colleen Will

☑ Remove PD Translator
☑ Remove PD Translator Plus
☐ Remove PD Date Tools

PARSER.INC
PDDDE.INC
STABEIP.INC
STABINV.INC
STDOLE.INC
ABASCII.INC
PDAPPT.INC

© COPYRIGHT 1999-2000 PRODOMUS, INC. HARTFORD

This template is provided to remove classes from an application. ProDomus classes can be removed by checking a check box. Other classes can by removed by adding class include files to the include file list. Check the www.prodomus.com for the latest version.

Class Summary

Conventions

Suffix. PD Date Tool Classes types consist of a name with a suffix of CT indicating Class Type. They are usually instantiated with the name in C indicating a Class.

Groups. Group types are consist of a name with a suffix of GT. They are instantiated with a suffix of G for Group.

Queues. Queues types are consist of a name plus QT. They are instantiated with a suffix of Q for Queue.

Passed and Local Data and Procedures. Passed parameters are generally prefixed with P or p.

Local variables are generally prefixed with L or l.

See also Date and Time Data Types and DateTime Groups.

Date and Time Data Types

Data Types. Most classes support both LONG, DATE, and TIME data types. When passing DateTime Group parameters, the Schedule Class (pdSchedCT) requires the use of LONGs, To use DATE and TIME fields, use the derived pdSchedDCT. Pairs withing the application should be defined consistently as either LONGs or DATE and TIME.

DateTime Groups

PDDateTimeGT. This group used by the PDSchedCT consists of Date and Time fields declared as LONGs.

PDDateTimeDGT. This groups used by the derived PDSchedDCT consists of Date and Time fields declared as DATE and TIME types.

Date Class

PDDateCT. The date class provides internationalized date handling functions for displaying dates and doing date calculations. The templates instantiate one Date Class as PDDateC that is used by all other classes except the thread management class.

Scroll Class

PDScrollCT. The scroll class alerts keys that change the date in entry or spin fields by various amounts. They are instantiated for fields where they are used as PDSC:[Use]

Calendar Classes

The following classes comprise a set of Calendar Classes:

PDCalCT Calendar Base Class: Provides basic calendar functionality.

PDListCalCT Calendar List Class -- Derived from PDCalCT, provides basic functionality for using a queue and list control. List calendars are instantiated as PDLC:[Instance].

PDDropCalCT Drop Calendar Class – Derived from PDListCalCT, provides the drop calendar functionality. Drop calendars are assicated with entry and spin date entry fields. They are instantiated as PDDC:[Use]

PDPopCalCT Popup Calendar Class – Derived from PDListCalCT, provides the popup calendar functionality. The popup class is instantiated within a generated popup procedure called PDDT_Popup as PDPopupC. The calendar uses configuration information maintained the PopCalQ, a queue maintained by the thread manager.

PDQTmCT Queue Thread Manager – A base thread management class which handles the threading of the popup calendar. The thread management class uses a global queue instantiated as PopCalQ. This contains field, thread, and configuration information used by the popup calendar.

Schedule Classes

PDSchedCT – Provides date and time handling methods. Date and time handling make extensive use of DateTime groups consisting of date and time fields declared as LONGs. The class includes methods for handling holidays and scheduled items.

PDSchedDCT – Derived from PdkSchedCT, this class uses Date and Time fields groups declared as DATE and TIME data types.

PDMonYrCT – Provides support for selection of a month and year from droplist controls.

PDMonDayYrCT – Provides support for selection of month, day, and year from droplist controls.

PDApptDayListCT – Provides support of an appointment listbox with add, delete, and change buttons (ABC Only).

PD Lock Class

The PD Lock Class is not provided with this took kit but is available from www.prodomus.com. This class and template are a simple way to lock multiple files without creating a “deadly embrace”. If you are checking schedules for conflicting dates in a multi-use environment, the schedule entry form should lock schedule files and reloaded the schedule into memory before checking for conflicts and adding or changing the schedule record.

PDDateCT Date Class

Conceptual Example

Global embeds

```
      !-- Add include file to the global embed
INCLUDE('PDDate.inc'),ONCE
      !-- Instantiate the date class.
```

Initialization procedure

```
      !-- to use information for the date group, create a group.  This is not required.
  DateGroup               LIKE(PDDateGT)
      !-- Intialize the class in the first procedure.
  PDDateC.Init()   ! Will initialize the class to the default locale.
      !-- Set date class to other locales
  CASE Choice(?Pictures)
  OF eFrench
    DO French
  OF eGerman
    DO German
  OF eSpanish
    DO Spanish
  END
French ROUTINE
  PDDateC.GetDateGroup(DateGroup,LOC_French)
  ! Modify date group elements as need.
  PDDateC.SetLocale(DateGroup)!,Loc_German)
German ROUTINE
  PDDateC.GetDateGroup(DateGroup,LOC_German)
  ! Modify date group elements as need.
  PDDateC.SetLocale(DateGroup)!,Loc_German)
Spanish ROUTINE
  PDDateC.GetDateGroup(DateGroup,LOC_SpanishModern)
  ! Modify date group elements as need.
  PDDateC.SetLocale(DateGroup) !,Loc_German)
```

Files

PDDate.inc

PDDate.clw

Properties

Initialized	***BYTE***

True if the library has been initialzed.

`DG                    PDDateGT`

Group containing international information.

`SchedQ                &SchedQT`

Queue containing scheduled dates.

`HolidQ                &HolidQT`

Queue containing holiday dates.

`RepeatQ               &RepeatQT`

Queue containing recurring holiday and schedule dates.

`USOff                 BYTE(0),PROTECTED`

Flag set to true if using US calculated holidays.

Methods

Initialization and Locale Information Handling

`Init                  PROCEDURE`

32-bit only.

Initializes date library using user default locale.

`Init                  PROCEDURE(PDDateGT P_DG,ULONG P:LCID)`

32-bit only.

P_Dg: Passes back the date group.

P:LCID: Locale identifier.

`Init                  PROCEDURE(ULONG P:LCID)`

32-bit only.

P:LCID: Locale identifier

`Init                  PROCEDURE(STRING P:Prefix,STRING P:DteFile)`

P:Prefix: Three letter language code to use.

P:DteFile: File containing date group information.

`Init                  PROCEDURE(PDDateGT P_DG)`

P_Dg: Passes back the date group.

`Kill                  PROCEDURE,VIRTUAL`

Disposes created properties.

`Version               PROCEDURE,STRING`

Return: Current Date Class version.

`CreateQ               PROCEDURE,PROTECTED`

Creates the date class queues.

`GetDateGroup          PROCEDURE(PDDateGT P_DG,ULONG P:LCID=0)`

32-bit only.

P_Dg: Passes back the date group.

P:LCID: Locale identifier

`GetDateGroup          PROCEDURE(PDDateGT P_DG,STRING P:Prefix,STRING P:DteFile)`

P_Dg: Passes back the date group.

P:Prefix: Three letter language code to use.

P:DteFile: File containing date group information.

```
GetCurrDateGroup     PROCEDURE(PDDateGT P_DG)
```

P_Dg: Passes back the date group.

```
SetLocale            PROCEDURE(ULONG P:LCID=0)
```

32-bit only.

P:LCID: Locale identifier

Sets the date group property for the passed LCID.

```
SetLocale            PROCEDURE(STRING P:Prefix,STRING P:DetFile)
```

Sets the date group to the passed prefix and date group file.

P:Prefix: Three letter language code to use.

P:DteFile: File containing date group information.

```
SetLocale            PROCEDURE(PDDateGT P_DG)
```

Sets the locale to the passed date group. Use this to modify individual elements in the date group.

P_Dg: The date group to use.

```
GetLongDateFormat    PROCEDURE(*STRING P:Format,ULONG P:LCID)
```

32-bit only.

P:Format: Returns the long date format for the passed LCID.

P:LCID: Locale identifier

```
GetLongDateFormat    PROCEDURE(*STRING P:Format)
```

P:Format: Returns the long date format from the win.ini file.

```
! LCID Parm ignored in 16-bit applications.
CreateDTE            PROCEDURE(PDDateGT P_DG,<STRING P:Prefix>,STRING P:DteFile)
```

Creates a DTE file using the active date group.

P_Dg: The date group to use.

P:Prefix: Three letter language code to use.

P:DteFile: File containing date group information.

```
GetMonthArray        PROCEDURE(STRING[] P:Array,ULONG P:LCID=0)
```

Gets an array of month names.

P:Array: Array to receive month names.

P:LCID: Locale identifier

```
GetMonArray          PROCEDURE(STRING[] P:Array,ULONG P:LCID=0)
```

Gets an array of abbreviated month names.

P:Array: Array to receive month names.

P:LCID: Locale identifier

```
GetDayArray          PROCEDURE(STRING[] P:Array,ULONG P:LCID=0)
```

Gets an array of day names.

P:Array: Array to receive day names.

P:LCID: Locale identifier

```
GetAbbrevDayArray    PROCEDURE(STRING[] P:Array,ULONG P:LCID=0)
```

Gets an array of abbreviated day names.

P:Array: Array to receive day names.

P:LCID: Locale identifier

```
MonthName            PROCEDURE(? P:Date),STRING
```
Return: the month name of the passed date.

```
MonName              PROCEDURE(? P:Date),STRING
```
Return: the abbreviated month name of the passed date.

```
DayName              PROCEDURE(? P:Date),STRING
```
Return: the day name of the passed date.

```
AbbrevDayName        PROCEDURE(? P:Date),STRING
```
Return: the abbreviated name of the passed date.

```
GetFirstOfYearRules  PROCEDURE(*BYTE P:WeekRule,*BYTE P:DayRule,ULONG P:LCID=0)
```
P:WeekRule: returns by address the week rule and first day of the week for the passed LCID (Locale Identifier).

P:DayRule: First day of week.

P:LCID: Locale identifier

Display Functions

```
GetLongDate          PROCEDURE(? P:Date),STRING,VIRTUAL
```
Return: the long date for the passed date.

```
SetSLongDate         PROCEDURE(STRING P:Format)
```
Sets the date group long date format.

P:Format: Date format string.

General Date Functions

```
Week                 PROCEDURE(? P:Date,<*? StartOfFirstWeekParm>), SIGNED, PROC, VIRTUAL
```
StartOfFirstWeekParm: First day of the week – Monday = 0

Return: the week of the year for the passed date.

```
DayOfWeek            PROCEDURE(? P:Date),SIGNED,PROC,VIRTUAL
```
Return: the day of the week where 1 = the first day of the week according the locale's first day of the week rules.

```
DayOfYear            PROCEDURE(? P:Date),USHORT,VIRTUAL
```
Return: the day of the year.

```
DaysLeftInYear       PROCEDURE(? P:Date),USHORT,VIRTUAL
```
Return: the number of days left in the year.

```
DAYS360              PROCEDURE(? P:Begin,? P:End),LONG,VIRTUAL
```
Return: the number of days from the P:Begin Date to P:End based on a 30 day month (360 day year).

General True/False Date Functions

```
IsLeapYear           PROCEDURE(? P:Date),BYTE,VIRTUAL
```
Return: True if P:Date occurs in a leap year.

```
IsValidDate          PROCEDURE(? P:Date),SIGNED,VIRTUAL
```
Return: True if P:Date is within a Clarion standard date range for valid dates.

```
IsWeekDay            PROCEDURE(? P:Date),BYTE,VIRTUAL
```

Return: True if date is a week day.

```
IsRecurringDate      PROCEDURE(? P:Date,? P:FromDate,ULONG P:Flags),BYTE,VIRTUAL
```

Return: True if P: Date is a recurrance of P:FromDate of the type specified the P:Flags.

P:FromDate: Date by which recurrance is measured.

P:Date: Date to test.

P:Flags: Recurrance flag. See equates contained in PDDate.inc

```
IsRecurringDate      PROCEDURE(? P:Date,? P:FromDate,ULONG P:Flags,? P:ToDate),BYTE,VIRTUAL
```

Return: True if P: Date is a recurrance of P:FromDate of the type specified the P:Flags.

P:FromDate: Date by which recurrance is measured.

P:Date: Date to test.

P:DoDate: Dates after this date will not be considered as recurring dates.

P:Flags: Recurrance flag. See equates contained in PDDate.inc

Relative day Procedures

The following methods change the P:Date parameter as indicated by the procedure name. The “X” in the procedure name indicates that the procedure will find a date with the same day of the week as the original date.

```
SemiMonMinus         PROCEDURE(*? P:Date),VIRTUAL
SemiMonPlus          PROCEDURE(*? P:Date),VIRTUAL
MonthMinus           PROCEDURE(*? P:Date),VIRTUAL
MonthPlus            PROCEDURE(*? P:Date),VIRTUAL
MonXMinus            PROCEDURE(*? P:Date),VIRTUAL
MonXPlus             PROCEDURE(*? P:Date),VIRTUAL
BiMonPlus            PROCEDURE(*? P:Date),VIRTUAL
BiMonMinus           PROCEDURE(*? P:Date),VIRTUAL
BiMonXPlus           PROCEDURE(*? P:Date),VIRTUAL
BiMonXMinus          PROCEDURE(*? P:Date),VIRTUAL
QuarterMinus         PROCEDURE(*? P:Date),VIRTUAL
QuarterPlus          PROCEDURE(*? P:Date),VIRTUAL
QuarterXMinus        PROCEDURE(*? P:Date),VIRTUAL
QuarterXPlus         PROCEDURE(*? P:Date),VIRTUAL
YearMinus            PROCEDURE(*? P:Date),VIRTUAL
YearPlus             PROCEDURE(*? P:Date),VIRTUAL
YearXMinus           PROCEDURE(*? P:Date),VIRTUAL
YearXPlus            PROCEDURE(*? P:Date),VIRTUAL
FirstOfMonth         PROCEDURE(*? P:Date),VIRTUAL
EndOfMonth           PROCEDURE(*? P:Date),VIRTUAL
FirstOfYear          PROCEDURE(*? P:Date),VIRTUAL
FirstOfFirstWeek     PROCEDURE(*? P:Date),VIRTUAL
FirstOfLastWeek      PROCEDURE(*? P:Date),VIRTUAL
EndOfYear            PROCEDURE(*? P:Date),VIRTUAL
```

Time Zone Information

```
GetTimeZone          PROCEDURE(TimeZoneType TimeZoneParm),UNSIGNED,PROC,VIRTUAL
```

Return: returns the following values:

```
PDDT_TZUnknown              EQUATE(0)
PDDT_MsgUnknown             EQUATE('Cannot determine the current time zone.')
                            !The operating system cannot determine the
                            !current time zone. This is usually because
                            !a previous call to the SetTimeZoneInformation
                            !function supplied only the bias (and no
                            !transition dates).
PDDT_TZStandard             EQUATE(1)
PDDT_MsgStandard            EQUATE('Now operating under Standard Time.')
                            !The operating system is operating in the range
                            !covered by the StandardDate member of the structure
                            !pointed to by the lpTimeZoneInformation parameter.
PDDT_TZDaylight             EQUATE(2)
PDDT_MsgDaylight            EQUATE('Now operating under Daylight Time.')
                            !The operating system is operating in the
                            !range covered by the DaylightDate member of the
                            !structure pointed to by the lpTimeZoneInformation
                            !parameter.
PDDT_TZFails                EQUATE(0FFFFFFFFh)
PDDT_MsgFails               EQUATE('PDDT_GetTimeZone Failed')
```

See PDDate.inc for a definition of the TimeZoneType.

PDSchedCT Schedule Class

The Schedule Class is a base class that includes a reference to a Date Class. It adds simple scheduling functionality. Holiday and Schedule functions can be used to determine if a specific date is a holiday or contains a scheduled item. It can also return queues (HolidayQ and ScheduleQ) that contain all scheduled and holidays including recurring items for a specific date. These queues can be displayed in a list box or be used as header type queues with relational fields for file lookups.

Conceptual Example

Add holidays in TakeRecord (ABC Process Procedure)

```
PDDateC.AddHoliday(HOL:Date,HOL:HolidayName,HOL:Recurrance)
```

Add Scheduled Items in TakeRecord (ABC Process Procedure)

```
PDDateC.AddSchedule(SCH:Date,SCH:ScheduleItem,SCH:Ending,SCH:Recurrance,SCH:Time,,,,,
,,)
```

Declare and Display Holiday and Schedule Queues:

```
!Date Section
```

```
lHolidayQ      QUEUE(HolidayQT)
               END
lScheduleQ     QUEUE(pdScheduleQT)
               END
!Window Controls (Lists)

LIST,AT(0,149,102,10),USE(lHolidayQ.Name,,?HolidayQName),HIDE,HVSCROLL,LEFT,FONT(,,CO
LOR:Red,,CHARSET:ANSI),COLOR(COLOR:Black),ALRT(EscKey),FORMAT('320LS(320)@s80@E(0FFH,
00H,00H,0FFH)'),DROP(5,120),FROM(lHolidayQ)
LIST,AT(0,149,102,10),USE(lHolidayQ.Name,,?HolidayQName),HIDE,HVSCROLL,LEFT,FONT(,,CO
LOR:Red,,CHARSET:ANSI),COLOR(COLOR:Black),ALRT(EscKey),FORMAT('320LS(320)@s80@E(0FFH,
00H,00H,0FFH)'),DROP(5,120),FROM(lHolidayQ)
```

!To load the queue

```
PDDateC.GetScheduleQ(L:Date,lScheduleQ,7*PDTime:Hour+PDTime:Midnight,L:Interval,L:Tim
es)
!Or
PDDateC.GetScheduleQ(L:Date,lScheduleQ)
```

!To display records from a related file:

```
R_AssignScheduleQ ROUTINE
  FREE(AppQ)
  LOOP I#=1 TO RECORDS(ScheduleQ)
    GET(ScheduleQ, I#)
    CLEAR(AppQ)
    AppQ.AppQTime    =ScheduleQ.Time
    Pat:PatCreated   =ScheduleQ.Modified
    AppQ.Type        =ScheduleQ.Other
    AppQ.Description =ScheduleQ.Name
    AppQ.AppCreated  =ScheduleQ.Created

    ! Appointment File Assignments
    APP:AppCreated=ScheduleQ.Created
    IF NOT ACCESS:Appointments.TryFetch(APP:KAppCR)
      AppQ.EndDate=App:EndDate
      AppQ.EndTime=App:EndTime
    END

    ! Other File Lookups
    IF NOT ACCESS:Patients.TryFetch(PAT:KPatCR)
      AppQ.LastName    =Pat:LastName
    END
  END
  GET(AppQ,1)
  ?AppQList{PROP:Selected}=1
```

!**To get record from related file**

```
   GET(AppQ,CHOICE(?AppQList))
   App:AppCreated=AppQ.AppCreated
   ThisWindow.Response=RequestCompleted
   IF App:AppCdate+App:AppCTime
     IF Access:Appointments.TryFetch(APP:KAppCR)
       MESSAGE('Error Getting Appointment Record','Browse
Appointments',ICON:Exclamation,BUTTON:Ok)
       ThisWindow.Response=RequestCancelled
     END
   END
```

! To process a change of date

```
RV"=PDDateC.GetScheduleQ(pdMDYDate,ScheduleQ,7*PDTime:Hour+PDTime:Midnight,30,19)
```

```
    DO R_AssignFields

    ?LongDate{PROP:Text}=pdDateC.GetLongDate(pdMDYDate)

! To Delate a record (similar to change or add)
      DO R_GetAppQ
      IF NOT SELF.Response=RequestCompleted
        SELF.Response=RequestCancelled
        CYCLE
      ELSIF NOT AppQ.AppCreated.AppDate+AppQ.AppCreated.AppTime
        MESSAGE('Not appointment is scheduled for this time.','No
Appointment',ICON:Asterisk,BUTTON:Ok)
        CYCLE
      ELSE
        SELF.Response=RequestCancelled
      END
      GlobalRequest=DeleteRecord
      UpdateAppointments()
      ! [Priority 4499]
      IF NOT GlobalResponse = RequestCompleted
        CYCLE
      END
      PDDateC.DeleteSchedule(APP:AppCreated,)

RV"=PDDateC.GetScheduleQ(pdMDYDate,ScheduleQ,7*PDTime:Hour+PDTime:Midnight,30,19)
      DO R_AssignScheduleQ
```

Files

PDSched.inc
PDSched.clw

Properties

Date
The current date.

DateC
The date class being used by the Schedule Class.

SchedQ ***&SchedQT***
Queue containing scheduled dates.

HolidQ ***&HolidQT***
Queue containing holiday dates.

RepeatQ ***&RepeatQT***
Queue containing recurring holiday and schedule dates.

USOff ***BYTE(0),PROTECTED***
Flag set to true if using US calculated holidays.

ScheduleQ ***&pdScheduleQT***
Queue containing one day's scheduled dates including recurring items. This queue can be instanitated in procedures and retrieved from the library in a variety of formats. .It includes header information that can be related to files and the protected SchedQ and HolidQ

HolidayQ ***&pdHolidayQT***
Queue containing on day's holiday dates. This queue can be instantiated in procedures and retrieved from the library.

Methods

Initialization and Locale Information Handling

`Init                    PROCEDURE`
32-bit only.
Initializes date library using user default locale.

`Init                    PROCEDURE(PDDateGT P_DG,ULONG P:LCID)`
32-bit only.
P_Dg: Passes back the date group.
P:LCID: Locale identifier.

`Init                    PROCEDURE(ULONG P:LCID)`
32-bit only.
P:LCID: Locale identifier

`Init                    PROCEDURE(STRING P:Prefix,STRING P:DteFile)`
P:Prefix: Three letter language code to use.
P:DteFile: File containing date group information.

`Init                    PROCEDURE(PDDateGT P_DG)`
P_Dg: Passes back the date group.

`Kill                    PROCEDURE,VIRTUAL`
Disposes created properties.

`CreateQ                 PROCEDURE,PROTECTED`
Creates the class queues.

`GetDateTime             PROCEDURE(*GROUP pG),STRING,PROC`
pG: DateTime group consisting of a LONG date and LONG time..Current date and time are inserted into the group.
Return: A string containing that may be assigned to a DateTime group. As an example, this procedure could be placed in the initial value attribute of a field's property:

```
DateTime=PDDateC.GetDateTime(DateTime)
```

`GetDateTime             PROCEDURE(*? P:Date,*? P:Time),BYTE,PROC`
Gets a new date time different from the last one retrieved by the function
P:Date: Date Field
P:Time: Time.
Return: . This returns TRUE if the method succeeds..

`GetEnding               PROCEDURE(*? P:EndDate,*? P:EndTime,? P:FromDate,? P:FromTime,LONG P:Days=0,LONG P:Time=0),REAL,PROC,VIRTUAL`
Return. A DateTime real with Days in the integer portion and the fraction of a day in the decimal protion.
P:EndDate: Returns the ending date.
P:EndTime..Returns the endikng time.
P:FromDate. The starting date..
P:FromTime.. The starting time.
P:Days. The number of days in the period.
P:Time. The elapsed time.

`GetEnding               PROCEDURE(*GROUP pG,LONG P:FromDate,LONG P:FromTime,REAL P:Duration=0),REAL,PROC,VIRTUAL`
This method is gets and ending places the ending date into a DateTime group and takes a DateTime REAL as a duration parameter.

`GetDuration PROCEDURE(*GROUP pFrom,*GROUP pTo),REAL`
pFrom: From DateTime group containing a LONG data types.
pTo: To DateTime group containing a LON Time.
Return: A real containing the duration between the dates.

`GetDuration PROCEDURE(? pfDate,? pfTime,? ptDate,? ptTime),REAL`

Duration taking either a DATE TIME or LONG data types.
Return: A DateTime Real containing the duration from the From DateTime to the To DateTime.
pFDate. From Date
pfTime: From TimePDSchedCT.GetDuration PROCEDURE(*? P:Days,? P:Time,REAL P:Duration)
ptDate: To Date
ptTime: To TIme.

GetDuration PROCEDURE(*? P:Days,*? P:Time,? P:FromDate,? P:FromTime,? P:ToDate,? P:ToTime),REAL,PROC

Return: A DateTime Real containing the duration from the From DateTime to the To DateTime.
P:Days: Number of days between from and to dates and times.
P:Time: Elapsed time between from and to dates and times.
P:FromDate. From Date
P:FromTime: From Time
P:ToDate: To Date
P:ToTime: To TIme.

GetDuration PROCEDURE(*GROUP pG,*GROUP pFrom,*GROUP pTo),REAL

Return: Date Time REAL.
pG: DateTime with a LONG date. Contains the number of days and elapsed time from the pFrom DateTime group to the pTo DateTime group.
pFrom: DateTime group.
pTo: To DateTime group.

GetDuration PROCEDURE(*? P:Days,? P:Time,REAL P:Duration)

P:Days: Returns the number of days contained in hte P:Duration parameter.
P:Time: Returns the elapsed time contained in the P:Duration parameter.
P:Duration: A DateTime REAL.

General Date Methods

BusinessDay: PROCEDURE(? P:Begin,? P:End),LONG,VIRTUAL

Return: the number of business days from the P:Begin date to the P:End date.

Holidays and Schedule

GetScheduleQ   PROCEDURE(? P:Date,pdScheduleQT P_SQ,ULONG P:Flags=0)

This method populates for the specified date a passed scheduleQ as defined in pdScheduleQT. The flags define what to include in the queue.
P:Date: Date to use.
P:SQ: ScheduleQ declared as a pdScheduleQT.
P:Flags: Flags defined in PDSched.Inc. If 0 both items with times and no time are included.

```
     pdSched:Time        EQUATE(1)  Timed items only
     pdSched:NoTime      EQUATE(2) Untimed items only.
```

GetScheduleQ   PROCEDURE(? P:Date,pdScheduleQT P_SQ,LONG P:Start,LONG P:Minutes,LONG P:Times),STRING

This method populates for the specified date a passed scheduleQ as defined in pdScheduleQT. The procedure adds time increments from a specified start time. Only items containing a time are included in the queue.
P:Date: Date to use.
P:SQ: ScheduleA declared as a pdScheduleQT.
P:Start: Start time (Clarion time)
P:Minutes: Minutes in each interval.
P:Times: Number of intervals.

GetHolidayQ   PROCEDURE(? P:Date,HolidayQT P_HQ)

This method fills a Holiday queue for the passed date. This allows there to be more than one holiday to be scheduled for a specific date.

P:Date: Date to use.

P_HQ: Holiday queue of HolidayQT.

```
TakeSchedRecord      PROCEDURE,VIRTUAL
```

This method assigns SchedQ records to the HolidayQ.

```
TakeRepeatRecord     PROCEDURE(BYTE P:IsHoliday),VIRTUAL
```

This method assigns recurring items in the RepeatQ to the Holiday and Schedule queues.

```
TakeHolidayRecord    PROCEDURE,VIRTUAL
```

This method assigns items in the HolidQ to the HolidayQ.

```
TakeScheduleRecord   PROCEDURE(pdScheduleQT,ULONG P:Flags),VIRTUAL
```

This method filters the internal ScheduleQ records according to the GetHolidayQ method. The flags filter the schedule.

```
TakeScheduleRecord   PROCEDURE(pdScheduleQT P_SQ),VIRTUAL
```

This method peforms the assignment of the filtered ScheduleQ to the passed queue.

```
GetTimeQ             PROCEDURE(PDTimeQT pQ,LONG P:Start,LONG P:Minutes,LONG P:Times),BYTE,PROC,VIRTUAL
```

This method creates a Time queue starting with the P:Start time with the specified number and size of internvals

pQ: Queue as defined in PDTimeQT.

P:Start: Starting time.

P:Minutes: The number of minutes in each interval.

P:Times: The number of intervals.

```
ClearSchedule        PROCEDURE()
```

Clears the SchedQ and RepeatQ.

```
AddSchedule          PROCEDURE(*LONG P:CDate,LONG P:CTime=0,LONG P:MDate=0,LONG P:MTime=0,LONG P:Date,LONG P:Time=0,LONG P:EndDate,LONG P:EndTime=0,STRING P:Name,ULONG P:Repeat=0,LONG RepeatEnd=0,<STRING P:Notes>,<? P:Other>,ULONG P:Flags=0)
```

Adds passed fields to the SchedQ and RepeatQ.

```
AddSchedule          PROCEDURE(*GROUP pCreated,<*GROUP pChanged>,*GROUP P:DateTime,<*GROUP P:Ending>,STRING P:Name,ULONG P:Repeat=0,LONG RepeatEnd=0,<STRING P:Notes>,<? P:Other>,ULONG P:Flags=0),VIRTUAL
```

Adds passed fields to the SchedQ and RepeatQ. Passed groups are DateTime groups with LONG dates.

```
ChangeSchedule       PROCEDURE(*GROUP pCreated,<*GROUP pChanged>,*GROUP P:DateTime,<*GROUP P:Ending>,STRING P:Name,ULONG P:Repeat=0,LONG RepeatEnd=0,<STRING P:Notes>,<? P:Other>,ULONG P:Flags=0),VIRTUAL,BYTE,PROC
```

Changes entries in the SchedQ and RepeatQ.

```
ChangeSchedule       PROCEDURE(*LONG P:CDate,LONG P:CTime=0,LONG P:MDate=0,LONG P:MTime=0,LONG P:Date,LONG P:Time=0,LONG P:EndDate=0,LONG P:EndTime=0,STRING P:Name,ULONG P:Repeat=0,LONG RepeatEnd=0,<STRING P:Notes>,<? P:Other>,ULONG P:Flags=0),BYTE,PROC
```

Changes entries in the SchedQ and RepeatQ.

```
DeleteSchedule       PROCEDURE(*GROUP pCreated,ULONG P:Repeat=0)
```

Deletes entries in the SchedQ and RepeatQ

```
DeleteSchedule       PROCEDURE(*LONG P:CDate,LONG P:CTime=0)
```

Deletes entries in the SchedQ and RepeatQ

```
Easter               PROCEDURE(? P:Date),LONG,VIRTUAL
```

Return: Easter date for the year of the passed date.

```
IsHoliday            PROCEDURE(? P:Date,<*STRING P:Name>),BYTE,VIRTUAL
```

Return: True if P:Date is holiday.

P:Name: Holiday name.

```
IsUSHoliday          PROCEDURE(? P:Date,<*STRING P:Name>),BYTE,VIRTUAL
```

Return: True if P:Date is a US Holiday.

P:Name: Holiday name.

```
SetUSOff             PROCEDURE(BYTE P:Off=1)
```

P:Off: Turns US Holiday calculation off or on.

`AddHoliday          PROCEDURE(? P:Date,STRING P:Name,ULONG P:Repeat=0)`

Adds holidays to the Date Class.
P:Date: (First or sample) Date of the Holiday.
P:Name: :Holiday name.
P:Repeat: Recurrence flag. See equates for valid bit mapped flags.

`ClearHolidays       PROCEDURE(BYTE P:ClearUS=1),VIRTUAL`

Clears Date Class library Holiday Queue.
P:ClearUS. Turns US holiday calculation on and off.

`ClearSchedule       PROCEDURE()`

Clears Schedule Queue.

`IsScheduled PROCEDURE(? P:Date,<*STRING P:Name>,BYTE P:AddScheduleQ=0),BYTE`

Return: True if date is a scheduled date.
P:Name: :Scheduled item name.
P:AddScheduleQ: If TRUE, the parameter cases the method to populate the ScheduleQ.

`GetName             PROCEDURE(BYTE P:Flag,STRING P:Name),STRING,VIRTUAL`

Place holder for returning the holiday or scheduled name. This procedure can be used to translated passed names.
Return: Name
P:Flag: 0 returns a blank. 1 to return passed name.
P:Name: Holiday Name.

`ResetSchedQ PROCEDURE !,PROTECTED,VIRTUAL`

Frees the queue.

`ResetRepeatQ PROCEDURE !,PROTECTED,VIRTUAL`

Frees the queue.

`ResetScheduleQ PROCEDURE !,PROTECTED,VIRTUAL`

Frees the queue.

`AddHoliday          PROCEDURE(*GROUP pCreated,<*GROUP pChanged>,LONG P:Date,STRING P:Name,ULONG P:Repeat=0)`

Adds a holiday to the SchedQ and RepeatQ.

`AddHoliday          PROCEDURE(? P:Date,STRING P:Name,ULONG P:Repeat=0),VIRTUAL`

Adds a holiday to the SchedQ and RepeatQ.

`ChangeHoliday       PROCEDURE(*GROUP pCreated,<*GROUP pChanged>,LONG P:Date,STRING P:Name,ULONG P:Repeat=0)`

Changes a holiday in the SchedQ and RepeatQ.

`DeleteHoliday       PROCEDURE(*GROUP pCreated,ULONG P:Repeate=0)`

Deletes a holiday in the SchedQ and RepeatQ.

`ClearHolidays       PROCEDURE(BYTE P:ClearUS=1),VIRTUAL`

Clears the holdiays fromt the SchedQ and RepeatQ

General True/False Date Functions

`IsBusinessDay       PROCEDURE(? P:Date),BYTE,VIRTUAL`

Return: True if P:Date is a business day (not a weekend date or holiday).

`IsRecurringDate     PROCEDURE(? P:Date,<*STRING P:Name>,BYTE P:IsHoliday=0,BYTE P:Add=0),BYTE,PROTECTED,VIRTUAL`

Return: True if P: Date is a recurrance of P:FromDate of the type specified the P:Flags.
P:FromDate: Date by which recurrance is measured.
P:Date: Date to test.
P:Flags: Recurrance flag. See equates contained in PDDate.inc
P:IsHoliday. If passed, the method checks where the passed date is a recurring Holiday.
P:Add: If TRUE, the method populates the ScheduleQ or HolidayQ.
NOTE: IsRecurringDate   PROCEDURE(LONG P:Date,LONG P:FromDate,ULONG P:Flags,LONG P:ToDate),BYTE,VIRTUAL is still part of the Date Class.

PDScrollCT Scroll Class

Date Scroll Class assigns a set of alerted keys to a date entry. Each alerted key is associated with a date calculation method. This home property is used when the HOME key is pressed.   The Quicken property assigns Quicken alert keys to the array.

```
Quicken Keys:
MinusKey –Minus one day.
PlusKey –Plus one day.
Mkey – First day of  month.
Hkey – Last day of month.
Ykey – First day of year.
Rkey – End of year.
Tkey – Home key (Quicken uses today’s date).  The scroll class uses the default date.
```

Conceptual Example

```
    Gloal Data
INCLUDE('PDScroll.inc'),ONCE

    !-- At procedure data for each date entry control.
DE                        PDScrollCT                   !Date Entry Class

    !-- At procedure initialization
DE.Init(DateEntry,PDDateC,?DateEntry,TODAY())
    !-  At field handling.
ACCEPT
  DE.TakeEvent()
  …
END
DE.Kill
```

Files

PDScroll.inc

PDScroll.clw

PDCal.trn (Jump to popup menu translation file)

Properties

Initialized ***BYTE,PROTECTED***

True if class is initialized.

Date ***ANY***

Date to update.

Home ***LONG***

Default date. The library returns to this when the HOME key is pressed.

PDDateC ***&PDDateCT***

Date class to use.

FEQ ***SIGNED,PROTECTED***

Field equate of the control to update.

RangeLow ***LONG,PROTECTED***

Range limit lowest value.

`RangeHigh` `LONG,PROTECTED`

Range limit highest value.

`KeyArray` `LIKE(PDDT_KeyArrayType[])`

Array of alerted keys.

`Holiday` `STRING(40),THREAD`

Name of the holiday associated with a selected date.

Methods

`Init` `PROCEDURE(*LONG P:Date,PDDateCT P:Functions,SIGNED P:Feq=0,LONG P:Home=0,LONG P:Lower=4,LONG P:Upper= 2994626)`

P:Date: Date to scroll.

P:Functions: Date Class to use.

P:Feq: Equate label of the control to scroll.

P:Home: Home date. This will default to Today's the current P:Date value if available. If not it will use Today's date.

P:Low: Lower range limit.

P:Upper: High range limit. Note that the class will allow a 0 value entry. Use the required attribute to force an entry.

`AddDropID` `PROCEDURE(STRING P:DropID),VIRTUAL`

Dropid for the control.

`Jumpto` `PROCEDURE,VIRTUAL`

Currentyly empty class.

`SetPDDateC` `PROCEDURE(PDDateCT P_PDDateC)`

Sets the Date Class property.

`GetKeyArray` `PROCEDURE(UNSIGNED[] P:KeyKeyArray),VIRTUAL`

Assigns values to the passed key array.

`SetKeyArray` `PROCEDURE(UNSIGNED[] P:Array),VIRTUAL`

Sets the Scroll Class KeyArray property.

`SetAlerts` `PROCEDURE,VIRTUAL`

Sets the control alert keys to the values in the KeyArray property.

`SetAlert` `PROCEDURE(BYTE P:Element,UNSIGNED P:Key)`

Changes the specified value in the keyarray.

`ClearAlerts` `PROCEDURE`

Clears the KeyArray property.

`SetTip` `PROCEDURE(SIGNED P:Feq=0),VIRTUAL`

Sets the tool tip of the control according the the current date. This displays the following information:

```
Long Date
Week of the year.
Day of the year.
Days left in the year.
Holiday name.
Scheduled date name.
```

`PopUp` `PROCEDURE,PROTECTED,VIRTUAL`

A popup menu that allows the user to jump to various dates. This is currently used only in the Calendar class.

`SetHome` `PROCEDURE(LONG P:Home)`
Sets the Home property.

`SetRange` `PROCEDURE(LONG P:Lower,LONG P:Upper),VIRTUAL`
Sets the lower and upper range limits.

`TakeEvent` `PROCEDURE,BYTE,PROC,VIRTUAL`
Takes window events. Calls TakeFieldEvent and TakeWindowEvent.

`TakeWindowEvent` `PROCEDURE,BYTE,PROC,VIRTUAL`
Handles window events.

`TakeFieldEvent` `PROCEDURE,BYTE,PROC,VIRTUAL`
Handles field events.

`TakeDropFieldEvent` `PROCEDURE,BYTE,VIRTUAL,PROC`
Supports drag and drop by handling Drop events.

`TranslateString` `PROCEDURE(STRING P:Str),STRING,VIRTUAL`
An empty method to handle translation of text.

`Kill` `PROCEDURE`

Destroys created properties.

Scroll Class Alert Key Array

See PDDate.inc

PDCalCT Calendar Base Class

The Calendar Base Class handles the basic functionality of all calendars.

Files

PDCal.inc

PDCalb.inc

PDCal.clw

PDCal.trn

Properties

`PDDateC` `&PDDateCT,PROTECTED`
The Date Class used by the calendar.

`DG` `LIKE(PDDateGT)`
The Date Group used by the Date Class and Calendar.

`FirstDOW` `BYTE,THREAD,PROTECTED`
The first day of the week for the Calendar.

`CalQ` `&CalQT,PROTECTED`
A queue containing the dates for the displayed month.

`BtnQ` `&BtnQT,PROTECTED`
A queue containing button equates and actions for calendar buttons.

`ReturnDate` `ANY,THREAD`
A pointer the the date to be returned by the calendar.

`Date` `LONG,THREAD,PROTECTED`
The current date displayed by the calendar.

`MinDate` `LONG,THREAD,PROTECTED`
The earliest date that may be returned by the calendar (rangelow).

`MaxDate` `LONG,THREAD,PROTECTED`
The latest date that may be returned by the calendar (rangehigh).

`HomeDate` `LONG,THREAD,PROTECTED`
The default date to display when the HOME key is pressed.

`Imm` `BYTE,THREAD,PROTECTED`
A flag indicating that the return date should be set with each new selection.

`Holiday` `STRING(40),THREAD`
The Holiday name if the currently selected date is a holiday.

`Header` `STRING(13),DIM(7),THREAD,PROTECTED`
An array of letters for the day names.

`HeaderFEQ` `SIGNED,DIM(7),THREAD,PROTECTED`
An array of header field equate numbers. These must be sequential.

`Month` `STRING(20),THREAD,PROTECTED`
The name of the month and year of the currently displayed month.

Methods

`Init` `PROCEDURE(PDDateCT P:DT,*? P:Date,LONG P:Min=eDateMinimum,LONG P:Max=eDateMaximum)`
Initializes the calendar.

P:DT: The Date Class to use.

P:Date: The date to update.

P:Min: The rangelow value.

P:Max: The rangehiigh value.

`Kill` `PROCEDURE,VIRTUAL`
Disposes of created fields.

`SetRange` `PROCEDURE(<? P:Min>,<? P:Max>),VIRTUAL`
Sets the MinDate and MaxDate properties.

`SetHome` `PROCEDURE(? P:HomeDate)`
Sets the HomeDate property.

`SetImmediate` `PROCEDURE(BYTE P:Flag)`
Sets the IMM property.

`Bound` `PROCEDURE,VIRTUAL`
Checks that the currently selected date is within the bounds set by the MinDate and MaxDate properties. The calendar restricts selections to the specified range.

`Jumpto` `PROCEDURE,VIRTUAL`
A window procedure allowing the user to jump to dates by various amounts in various ways:

Window. Entries include:

1. Jump Amount

2. Jump Period
    a. Days
    b. Business Days
    c. Weeks
    d. Months
    e. Years

The JumpTo method is called by the popup method in the scroll class. Window text is set in the PDCal.trn file. You may also use the Translate method to translate the window.

SetFirstDOW ***PROCEDURE(BYTE P:FirstDay=0),VIRTUAL***

Sets the first day of the week. Use this to override the Date Group first day of the week.

GetDate ***PROCEDURE,LONG***

Returns the current return active date.

AddButton ***PROCEDURE(BYTE P:Type,SIGNED P:Feq)***

Adds buttons to the the BtnQ.

P:Type: The type of button.

P:Feq: The field equate label of the button.

Button Type Equates are:

```
PDBTN       ITEMIZE(),PRE(PDBTN)
OK            EQUATE
Home          EQUATE
PrevYr        EQUATE
PrevMo        EQUATE
NextMo        EQUATE
NextYr        EQUATE
Help          EQUATE
            END
```

AddHeaders ***PROCEDURE(SIGNED P:StartFeq),VIRTUAL***

Adds header field equates to the header array.

P:StartFEQ: The first field equate number.

Home ***PROCEDURE***

Sets Date Property to the Home date.

PrevYr ***PROCEDURE***

Sets Date Property to the Previous Year date.

PrevMo ***PROCEDURE***

Sets Date Property to the Previous Month date.

NextMo ***PROCEDURE***

Sets Date Property to the Next Month date.

NextYr ***PROCEDURE***

Sets the Date Property to the Next Year date.

Reset ***PROCEDURE,VIRTUAL***

Frees the CalQ Property.

Load ***PROCEDURE(LONG P:From,LONG P:To),VIRTUAL***

An empty virtual procedure.

TakeEvent ***PROCEDURE,BYTE,VIRTUAL,PROC***

Handles all window events. Calls TakeWindowEvent and TakeFieldEvent.

`TakeWindowEvent`   `PROCEDURE,BYTE,VIRTUAL,PROC`

Handles window events. There is no code in this class.

`TakeFieldEvent`   `PROCEDURE,BYTE,VIRTUAL,PROC`

Handles field events. This uses the BtnQ and calls procedures related to each button type.

`TakeAccepted`   `PROCEDURE,VIRTUAL`

Sets the return date.

`FillCalendar`   `PROCEDURE,VIRTUAL`

Fills the CalQ. If the immediate attribute is set, sets the return date.

`AssignHeaders`   `PROCEDURE`

Assigns header text the header controls.

`GetMonth`   `PROCEDURE,STRING`

Returns the month name of the currently selected month.

`IsHoliday`   `PROCEDURE(? P:Date),BYTE,VIRTUAL`

Return: True if a date is a holiday. The uses to Date Class to determine whether a date is a holiday.

`IsScheduled`   `PROCEDURE(? P:Date,<*STRING P:Description>),BYTE,VIRTUAL`

Return: True of the date is a scheduled date. If a description is available, it also returns a description.

`IsBusinessDay`   `PROCEDURE(? P:Date),BYTE,VIRTUAL`

Return: True if the date is a business day.

`CalendarMap`   `PROCEDURE,VIRTUAL`

A help screen showing alert keys and buttons associated with the calendars. This procedure is called by prssing the help button on popup calendars or select calendar map from the popup menu.

`Translate`   `PROCEDURE(<WINDOW P_Win>),VIRTUAL`

Hook for use of the Translator Class.

`TranslateString`   `PROCEDURE(STRING P:String),STRING,VIRTUAL`

Hook for use of the Translator Class.

PDListCalCT List Calendar Class

The List Calendar Class supports the window calendar list control template and is a base class for drop and popup calendars. It is derived from the PDCalCT.

Conceptual Example

```
ShowList PROCEDURE

L:date   LONG

Window WINDOW('Show Calendar
List'),AT(,,272,158),FONT('Arial',8,,FONT:regular,CHARSET:ANSI),COLOR(COLOR:WINDOW),
|
         CENTER,GRAY
       GROUP,AT(5,5,103,75),USE(?Group1),BOXED
         STRING(@s20),AT(19,10,71,10),USE(?MonthYrTitle),CENTER
         IMAGE(ICON:VCRback),AT(7,10,10,8),USE(?PreviousMonthIMG),CENTERED
         REGION,AT(7,10,10,8),USE(?PreviousMonth)
         IMAGE(ICON:VCRplay),AT(93,10,10,8),USE(?NextMonthIMG),CENTERED
         REGION,AT(93,10,10,8),USE(?NextMonth)
         STRING('* '),AT(13,21,10,8),USE(?Heading1),LEFT(1)
         STRING('* '),AT(23,21,10,8),USE(?Heading2),LEFT(1)
         STRING('* '),AT(35,21,10,8),USE(?Heading3),LEFT(1)
         STRING('* '),AT(45,21,10,8),USE(?Heading4),LEFT(1)
         STRING('* '),AT(57,21,10,8),USE(?Heading5),LEFT(1)
         STRING('* '),AT(67,21,10,8),USE(?Heading6),LEFT(1)
         STRING('* '),AT(79,21,10,8),USE(?Heading7),LEFT(1)
         IMAGE('Pddt_q.ico'),AT(93,21,10,8),USE(?TodayIMG),CENTERED
         REGION,AT(93,21,10,8),USE(?TodayButton)

LIST,AT(9,29,77,48),USE(?PopList),TRN,ALRT(F2Key),COLUMN,FORMAT('11R(1)*@n_2b@11R(1)*
@n_2b@11R(1)*@n_2b@11R(1)*@n_2b@11R(1)*@n_2b@11R(1)*@n_2b@12' &|
             'R(1)*@n_2b@'),FROM('*'),DRAGID('PDListCal')

LIST,AT(93,29,9,48),USE(?WeekList),SKIP,TRN,NOBAR,FONT(,,COLOR:Gray,,CHARSET:ANSI),FO
RMAT('10L@n_2b@#36#'), |
             FROM('*')
       END
         PROMPT('Selected Date: '),AT(11,93),USE(?Prompt1)
         STRING(@d2),AT(69,93),USE(L:date)
     END

  CODE
  OPEN(Window)
  LC.Init(?PopList,PDDateC,L:Date,eDateMinimum,eDateMaximum)
  LC.AddButton(PDBTN:Home,?TodayButton)
  LC.AddButton(PDBTN:PrevMo,?PreviousMonth)
  LC.AddButton(PDBTN:NextMo,?NextMonth)
  LC.SetImmediate(TRUE)
  LC.AddHeaders(?Heading1)
  LC.Open
  LC.AddWeekList(?WeekList)
  LC.AddMonthTitle(?MonthYrTitle)
  ACCEPT
    LC.TakeEvent()
  END
  LC.Kill
```

Files

PDCal.inc

PDCal.clw

PDCal.trn

Properties

Feq ***SIGNED,PROTECTED***

The field equate of the calendar list control.

ScrollC ***&LocScrollCT,PROTECTED***

A locally created scroll class.

Config ***LIKE(PDCalConfigGT),PROTECTED***

A group containing calendar configuration information. See PDCal.inc.

SavColorG ***LIKE(SavColorGT),PROTECTED***

A group saving default color information.

Methods

```
Init                 PROCEDURE(SIGNED P:Feq,PDDateCT P_DT,*LONG P:Date,LONG P:Min=eDateMinimum,LONG P:Max=eDateMaximum)
```

Initializes the class.

P:Feq: The field equate of the list control.

P_DT: The Date Class to be used by the class.

P:Date: The date to be updated by the calendar.

P:Min: The lowest value of the date.

P:Max: The highest value of the date.

```
Kill                 PROCEDURE,VIRTUAL
```

Disposes newly created properties.

```
Open                 PROCEDURE,VIRTUAL
```

Calls the SetFrom method which sets the from property of the calendar list control.

Calls the FillCalendar calendar method.

Calls the AssignHeaders method.

Alerts the EnterKey

```
SetRange             PROCEDURE(LONG P:Min=eDateMinimum,LONG P:Max=eDateMaximum),VIRTUAL
```

Sets the date range limits.

```
SetColors            PROCEDURE,VIRTUAL
```

Assigns colors to the list control’s queue.

```
AssignSchedColor     PROCEDURE,VIRTUAL
```

Assigns colors to holidays and scheduled dates.

```
AddDropID            PROCEDURE(STRING P:DropID),VIRTUAL
```

Adds DROPIDs to the internal scroll class property.

```
SetConfig            PROCEDURE(PDCalConfigGT P_Config),VIRTUAL
```

Sets the Config group properties of the class

P:Config: a group containing configuration values.

```
SetConfig            PROCEDURE(LONG P:Event=0,PDDateCT P_PDDateC),VIRTUAL
```

Sets the Config group properties of the class

P:Event: The EVENT:Calendar equate.

P_PDDateC: Date Class to use.

```
GetConfig            PROCEDURE(PDCalConfigGT P_Config),VIRTUAL
```

Retrieves the current configuration.

P_Config. Group to receive configuration information.

```
SetFROM              PROCEDURE,VIRTUAL
```

Sets the FROM attribute of the list control.

```
AddMonthTitle        PROCEDURE(SIGNED P:Feq),VIRTUAL
```

Assigns the calendars month and year title.

```
AddWeekList          PROCEDURE(SIGNED P:Feq),VIRTUAL
```

Passes a week list field equate to the class.

```
FillCalendar         PROCEDURE,VIRTUAL
```

Fills the calendar.

Calls the parent FillCalenar method.

Assigns colors to the queue used by the calendar.

Selects the current row and column.

Calls the SetTip method.

PopUp ***PROCEDURE,PROTECTED,VIRTUAL***

Displays and executes a popup menu that can be called with the right mouse key.

The popup entries can be translated in the PDCal.trn file.

It also uses the hooked translate string method.

SetTip ***PROCEDURE(SIGNED P:Feq=0),PROTECTED,VIRTUAL***

Calls the internal scroll class set tip method.

TakeEvent ***PROCEDURE,BYTE,VIRTUAL,PROC***

Calls the parent TakeEvent method.

Calls the scroll class TakeEvent method.

Calls the TakeDropEvent method..

TakeFieldEvent ***PROCEDURE,BYTE,VIRTUAL,PROC***

Handles the list control field events.

TakeDropFieldEvent ***PROCEDURE,BYTE,VIRTUAL,PROC***

Handles field drop events.

PDDropCalCT Drop Calendar Class

Conceptual Example

```
CalDrop  PDDropCalCT

ShowDrop PROCEDURE
date     LONG
Window WINDOW('DropCalendar'),AT(,,173,136),FONT('MS Sans
Serif',8,,FONT:regular,CHARSET:ANSI),CENTER, |
         SYSTEM,GRAY
       PROMPT('Date with Drop'),AT(5,9),USE(?Prompt1)
       ENTRY(@d2),AT(67,10,60,10),USE(Date)
       PROMPT('Note this date entry has a message warning the user if the date is a
holiday or ' &|
           'not a business
day.'),AT(8,42,155,47),USE(?Prompt3),FONT(,,COLOR:Navy,,CHARSET:ANSI)
       BUTTON('Cl&ose'),AT(113,106,45,14),USE(?CloseButton),STD(STD:Close)
     END

Holiday STRING(80)

    CODE
    OPEN(Window)
    Date=TODAY()
    CalDrop.Init(?date,PDDateC,Date)
    ! CalDrop.AddHide(?FieldTohide,TRUE)  ! Add any needed here
    CalDrop.Open
    SELECT(?Date)
    ACCEPT
      CalDrop.TakeEvent
      CASE ACCEPTED()
      OF ?Date
```

```
        IF PDDateC.IsHoliday(Date,Holiday)
          CASE MESSAGE(PDDateC.GetLongDate(Date)&' is '&CLIP(Holiday)&'.  Are you
sure you want to enter this date?','Holiday',ICON:Asterisk,BUTTON:Yes+BUTTON:No)
          OF BUTTON:No
            SELECT(?Date)
          END
        ElSIF NOT PDDateC.IsBusinessDay(Date)
          CASE MESSAGE('Please note that '&PDDateC.GetLongDate(Date)&' is not a
business day.  Do you want to select another date?','Selected
Date',ICON:Asterisk,Button:Yes+Button:No)
          OF BUTTON:Yes
            SELECT(?Date)
          END
        END
      END
    END
    CalDrop.Kill
```

Files

PDCal.inc

PDCal.clw

PDCal.trn

Properties

EntryFEQ ***SIGNED***
The field equate of the entry control to be updated.

ButtonFEQ ***SIGNED***
The field equate of the drop button.

EntryScrollC ***&PDScrollCT***
The internally created scroll class for the entry.

Methods

Init ***PROCEDURE(SIGNED P:Feq,PDDateCT P_DT,*? P:Date,LONG P:Min=eDateMinimum,LONG P:Max=eDateMaximum,BYTE P:ForceAbove=0)***

P:Feq: The equate of the date entry field.

P_DT: The date class to use.

P:Date: The field to be updated.

P:Min and P:Max: The date entry range limits.

P:ForceAbove: Forces the drop calendar to display above the field rather than below it.

AddDropID ***PROCEDURE(STRING P:DropID),VIRTUAL***
Adds a DROPID of the entry field.

Kill ***PROCEDURE,VIRTUAL***
Disposes of created properties.

SetRange ***PROCEDURE(LONG P:Min=eDateMinimum,LONG P:Max=eDateMaximum),VIRTUAL***
Sets date range limits.

Open ***PROCEDURE,VIRTUAL***
Creates the drop button and drop calendar.

TakeEvent ***PROCEDURE,BYTE,VIRTUAL,PROC***

Calls the internal scroll class take event method.

Calls the settip method.

HideControl ***PROCEDURE(SIGNED P:Feq),VIRTUAL***

HIDEs a controls in overlaid by drop. Where the AddHide method has been used to force the hiding of a control, it is hidden here. If this is a date entry, the drop button will also be hidden.

P:Feq. The control to hide.

UnhideControl ***PROCEDURE(SIGNED P:Feq),VIRTUAL***

UNHIDEs a controls in overlaid by drop. Where the AddHide method has been used to force the hiding of a control, it is unhidden here. Any controls that were previous disabled or hidden will be restored to their original state.

P:Feq. The control to unhide. If this is a date entry, the button will also be unhidden.

AddHide ***PROCEDURE(SIGNED P:Feq,BYTE P:ForceHide=0),VIRTUAL***

Adds controls to the queue of overlaid controls.

P:Feq. The equate of the control to add.

P:ForceHide. TRUE if the control should be hidden rather than disabled. This will prevent the control from showing through the drop list.

TakeFieldEvent ***PROCEDURE,BYTE,VIRTUAL,PROC***

Handles button and entry control field events.

FillCalendar ***PROcEDURE,VIRTUAL***

Calls the parent FillCalendar method.

Adds headers to the drop list.

TakeAccepted ***PROCEDURE,VIRTUAL***

Hides the drop calendar.

Calls the parent TakeAccepted method.

Selects the calendar entry control.

PDPopCalCT Popup Calendar Class

Conceptual Example

```
ShowThread PROCEDURE


! Sample Data
L:Date      LONG
L:Date2     LONG
L:Date3     LONG

!-- NOTE: Window must have IMM attribute to pass iconize event.
Window WINDOW('Show Threaded Popup
Calendar'),AT(,,191,190),FONT('Arial',8,COLOR:Navy,FONT:regular,CHARSET:ANSI), |
         CENTER,IMM,ICON('Pddt_cal.ico'),GRAY,MDI
       GROUP('Date entry and popup calendar'),AT(4,7,179,83),USE(?Group1),BOXED
         PROMPT('&Start Date:'),AT(11,17,77,10),USE(?DatePMT)
         ENTRY(@d2),AT(105,17,60,10),USE(L:Date),DROPID('PDPopCal')
         BUTTON,AT(165,17,12,10),USE(?CalendarBTN),ICON('PDDT_Cal.ico')
         PROMPT('The threaded calendar can update a specific date as shown
here.'),AT(11,33,162,19),USE(?Prompt4)
         PROMPT('If either the window or calendar is iconized, the other will be
iconized at the ' &|
             'same time.'),AT(11,58,162,27),USE(?NotePrompt1)
       END
       GROUP('Dates with Drop ID''s'),AT(4,95,180,90),USE(?Group2),BOXED
         PROMPT('&End Date:'),AT(11,111,77,10),USE(?DatePMT2)
         ENTRY(@d2),AT(100,111,60,10),USE(L:Date2),DROPID('PDPopCal')
         PROMPT('Date &Confirmed: '),AT(11,123,77,10),USE(?DatePMT3)
         ENTRY(@d2),AT(100,123,60,10),USE(L:Date3),DROPID('PDPopCal')
         PROMPT('These dates are drop enabled.  Dates may be dragged from the
calendar started by' &|
             ' the button above or from the calendard opened from the main
menu.'),AT(11,139,162,38), |
             USE(?NotePrompt2)
       END
     END

  CODE
  L:Date=TODAY()
  L:Date2=TODAY()
  L:Date3=TODAY()
  OPEN(Window)
  0{PROP:Text}=0{PROP:Text}&' '&Thread()
  ACCEPT
    !-- Handle thread management at top of loop.
    PDQtmC.TakeEvent(THREAD())
    !-- Take Field Events
    CASE FIELD()
    !-- Handle Popup Calendar Button
    OF ?CalendarBTN
      CASE EVENT()
      OF EVENT:Accepted
        CLEAR(PopCalQ)
        !-- Check if window already open, pass 1 to restore if it is open.
        IF NOT PDQTmC.IsStarted(THREAD(),1)  ! for restore
          DO R_ConfigCalendar
          !-- Start the calendar (Code in FRAME window)
          POST(EVENT:Calendar,,1)
        END
      END
    !-- Handle Date Fields
    OF ?L:date2
      !-- Drops. Note that if scroll class where applied and you use template, this
is handled for you.
      CASE EVENT()
      OF EVENT:Drop
        CHANGE(?L:Date2,CLIPBOARD())
        DISPLAY(?L:date2)
      END
    OF ?L:date3
      CASE EVENT()
      OF EVENT:Drop
        CHANGE(?L:date3,CLIPBOARD())
        DISPLAY(?L:date3)
      END

    END
  END

!-----------------------------
R_ConfigCalendar ROUTINE
!-----------------------------
! Essentially, parameters used by the thread managemer and calendar are passed by the
! Popcal Queue. These need to be initialized when the calendar is called.
```

```
  !-- Set default date value
  g:Date=l:Date

  !-- Set thread management key data
  PopCalQ.Control=?L:Date
  PopCalQ.Thread=THREAD()

  !-- Set pointer to date
  PopCalQ.PassDate &= g:Date ! L:date if local to the procedure or not threaded.

  !-- Configure PopCalQ configuration group
  PopCalQ.Config :=: SAV::Config ! Most info saved for reuse.
  PopCalQ.Config.LowerLimit=DATE(1,1,YEAR(TODAY())-2)
  PopCalQ.Config.UpperLimit=DATE(1,1,YEAR(TODAY())+2)
  PopCalQ.Config.SchedFG = COLOR:Black
  PopCalQ.Config.SchedBG = COLOR:Yellow
  PopCalQ.Config.Caption = 0{PROP:Text}
  !—Make the window MDI
  PopCalQ.Config.Flags = BXOR(PopCalQ.Config.Flags,PDDT_eCalMDI)

  !-- This code will position the calendar next to the button.
  !   Add the window and button x and y positions pluss any offset.
  PopCalQ.Xpos=0{PROP:Xpos}+?{PROP:Xpos}+16
  PopCalQ.Ypos=0{PROP:Ypos}+?{PROP:Ypos}+12


!-- Frame Code (TakeWindowEvent)
!    Note: See generated PDPopCal.clw for P_Popcal code.
    CASE EVENT()
    OF EVENT:Calendar
      IF BAND(PopCalQ.Config.Flags,PDDT_eCalMDI)
        IF NOT PDQTmC.IsStarted(PopCalQ.thread,1)
          PDQTmC.InsertQ
          START(P_PopCal,35000)
        END
      ELSE
        P_PopCal()
      END
    END
!—- Other Frame
  PDQtmC.Init(PopCalQ)  ! Init Embed
  PDQTmC.Kill           ! Kill Embed
```

Files

PDCal.inc

PDCal.clw

PDCal.trn

Properties

PopWindow **&Window,PROTECTED**

The pop calendar window.

Week **BYTE,THREAD,PROTECTED**

The currently selected week of the year.

DOY **SHORT,THREAD,PROTECTED**

The currently selected day of the year.

`DaysLeft` ***`SHORT,THREAD,PROTECTED`***
The the currently selected number of days left in the year.

`TM` ***`&PDQTmCT`***
The popup calendar thread management class.

Methods

`Init` ***`PROCEDURE(PDQTmCT P_Tm)`***
Initializes the popup calendar class.

P_Tm: The calendars thread management class.

`Kill` ***`PROCEDURE,VIRTUAL`***
Disposes of newly created properties.

`Run` ***`PROCEDURE(PDQTmCT P_Tm,WINDOW P_Win),VIRTUAL`***
Runs the popup calendar.

P_Tm: The thread management class to use.

P_Win: The window to open.

`Ask` ***`PROCEDURE,VIRTUAL`***
Runs the accept loop.

`Open` ***`PROCEDURE,VIRTUAL`***
Sets the calendars configuration.

Calls the OpenWindow method.

Calls the AddWeekList method.

Calls the AddHeaders method.

Calls the ConFigWindow method.

Calls the Thread Manager UpdateQ method.

Calls the parent Open method.

Calls the AddMonthTitle method.

Opens the Popup Calendar Window.

`OpenWindow` ***`PROCEDURE,VIRTUAL`***
Opens the PopWindow.

`AddButtons` ***`PROCEDURE,VIRTUAL`***
Adds all the button types and equates to the BtnQ.

`ConfigWindow` ***`PROCEDURE,VIRTUAL`***
Sets the text and appearance of the window and alerts the Enter key.

`FillCalendar` ***`PROCEDURE,VIRTUAL`***
Calls the parent FillCalendar method.

Passes information to the thread management queue to update the calling controls date entry.

`IsMDI` ***`PROCEDURE,BYTE,VIRTUAL`***
Return: True if the window opened is MDI.

`SetIcon` ***`PROCEDURE(STRING P:Icon)`***
Sets the windows icon.

`SetCaption` ***`PROCEDURE(STRING P:Caption),VIRTUAL`***

Sets the windows caption.

`SetTitle`   ***`PROCEDURE(STRING P:Title),VIRTUAL`***
Sets the windows month and year title.

`SetWallPaper`   ***`PROCEDURE(STRING P:WallPaper)`***
Assigns a wall paper to the window.

`SetFlat`   ***`PROCEDURE(BYTE P:Flat=1),VIRTUAL`***
Sets the flat attribute on all window buttons.

`SetProperty`   ***`PROCEDURE(ANY P:PropToSet,ANY P:Property),VIRTUAL`***
Sets window properties.

`TakeWindowEvent`   ***`PROCEDURE,BYTE,VIRTUAL,PROC`***
Handles window events.

Passes window events to the thread management queue.

`TakeCloseEvent`   ***`PROCEDURE,BYTE,VIRTUAL`***
Updates the thread management class.

`TakeAccepted`   ***`PROCEDURE,VIRTUAL`***
Updates the return value.

Updates the thread manager class.

Posts and event accepted to the control to be upated.

PDQTmCT PopCal Thread Manager

The thread manager links the popup calendar with the calling control and date field to be updated. The queue contains configuration. The class passes events and information between the calling threads and the calendar thread.

Files

PDCal.inc

PDCal.clw

PDCal.trn

Properties

`PopCalQ`   ***`&PDPopCalQT`***
Queue containing thread management information. See PDCalb.inc.

Methods

`Init`   ***`PROCEDURE(PDPopCalQT PQ)`***
Initializes the thread management class.

PQ: The global PopCalQ to use.

`Kill`   ***`PROCEDURE`***
Frees the PopCalQ property.

`IsStarted`   ***`PROCEDURE(SIGNED P:Thread,BYTE P:Restore=0),BYTE,VIRTUAL`***
Return:True if the passed thread has been started.

P:Thread: The thread to start.

P:Restore: 1=restore focus the the popup calendar if it has been started.

TakeEvent PROCEDURE(SIGNED P:Thread),VIRTUAL

Calss the TakeWindowEvent method for the passed thread.

If the thread has been started and the calendar is updating a control, calls the controls TakeFieldEvent method.

TakeWindowEvent PROCEDURE(SIGNED P:Thread),VIRTUAL

Posts window events to the popup calendar if there is a calendar opend from the calling thread.

TakeFieldEvent PROCEDURE,VIRTUAL

Updates the calling control field and passes control events to the popup calendar.

UpdateQ PROCEDURE,VIRTUAL

Updates the PopCalQ.

InsertQ PROCEDURE,VIRTUAL

Adds a record to the PopCalQ.

GetQ PROCEDURE,BYTE,PROC,VIRTUAL

Called from the popup calendar getting a record for the current calendar.

Return: True if the record is found.

GetQ PROCEDURE(SIGNED P:Thread),BYTE,PROC,VIRTUAL

Called from the calling thread. Gets the threads PopCalQ record.

Return: True if the record is found.

Holidays and Scheduled Dates

There are three internal schedule queues maintained by the Date Class: a holiday queue, and scheduled date queue, and a repetitive date queue. The last includes both holidays and scheduled dates. Holiday dates affect the calculation of business days while scheduled dates do not. See the sample application for ans example of adding Holidays and Scheduled Dates. These were created as follows:

1. To add Holidays
    a. Create a Holiday file with a date field (LONG), Name field (Miximum size 80 characters), and a recurrance field (ULONG). This field should be required.
    b. Under validity checks for the recurrance field, enter Must be in List choices for the recurrancy – none|Yearly|Yearly, Same Day of Week.
    c. Create a normal Browse and Form procedure for this file.
    d. In the initialization of section of the Form, after the window is opened, add values for each of the choices of the recurrance field, i.e.
        i. ?NoneRadio{PROP:Value}=0
        ii. ?YearlyRadio{PROP:Value}=EveryYear !Equate from PDDate.inc
        iii. ?YearlySameDay{PROP:Value}=EveryXYear !Equate from PDDate.inc
    e. Create a process procedure that processes the Holiday file.
        i. In Local Objects/This Process/Take Record add the PDDT Add Holiday code template.
        ii. Enter the fields in the entries.
    f.Call this process when the application is initialized and any time holidays are added, changed, or deleted.
    g. If you want to turn US holidays on or off, in the Kill method, add PDDateC.SetUsOff(FALSE) or PDDateC.SetUsOff(TRUE)
2. To add Scheduled items
    a. Create a schedule file with the same fields as found in 1a above plus and Ending (LONG) field which can be used to termate the recurrance of a scheduled item.
    b. Under validity checks, add choices for None|Every Week|Bi Weekly|Every Month|Every Month Same Day|Every Quarter|Every Quarter Same Day|Every Year|Every Year Same Day.
    c. Create a Browse and Form procedures for the file.
    d. In the initalization section of the Form, assign the recurrance equates to the PROP:Values of each of the radio buttons.
    e. Create a process procedure that processes the Schedule file.

        i. In ;Local Objects/This Process?Take Record add the PDDT Add Scheduled Date code template.
        ii. Complete the field entries in the template.

    f. Call this process when the application is initialized and any time scheduled items are added, changed, or deleted.

Color Handing

Calendar colors can be set in three locations:

1. Global extension Calendar Classes/Overrides.
2. Popup Calender Overrides in the Calendar Procedure Extension Template.
3. PDCal.trn

The first creates a Def_Config group which may be used by any of the calendars by checking Use Default Colors in their respective templates. The Popup Calendars will **always** use either this group or their locally defined overrides; they will not use the PDCal.trn configuration. This may change in future releases.

The drop and list calendars use either the Def_Config group. You can see the generated code for this in the R_PDSetDefConfigQ Routine in the first initialization procedure.

Date File Creation Utility

The date file creation utility (MakeDte.exe) will create DTE files that can be used with 16-bit applications. The program lists installed locales. To create one or more DTE files, run the program; a list of installed locales will be displayed. To proceed:

1. Mark one or more locales or check the All Locales box.
2. Press the button to create a DTE file.
3. Select or enter a file name in the file dialog.

Multiple locales can be placed the same file. The section name will be prepended with the language code for the locale. Currently, 32-bit windows supports 93 locales. The number that you create depends on the number that are actually installed on your machine.

Sample Date Class Procedures

Setting the Locale: 16 and 32 Bit

```
  COMPILE('@@',TEST16)
!----------------------------
French ROUTINE
!----------------------------
  PDDateC.GetDateGroup(DateGroup,'fra','.\Pddt16.dte')
  PDDateC.SetLocale(DateGroup)!,Loc_German)
!----------------------------
German ROUTINE
!----------------------------
  PDDateC.GetDateGroup(DateGroup,'deu','.\Pddt16.dte')
  PDDateC.SetLocale(DateGroup)!,Loc_German)
!----------------------------
Spanish ROUTINE
!----------------------------
  PDDateC.GetDateGroup(DateGroup,'esp','.\Pddt16.dte')
  PDDateC.SetLocale(DateGroup)!,Loc_German)
  !@@
  COMPILE('@@',TEST32)
!----------------------------
French ROUTINE
!----------------------------
  PDDateC.GetDateGroup(DateGroup,LOC_French)
  PDDateC.SetLocale(DateGroup)!,Loc_German)
!----------------------------
German ROUTINE
!----------------------------
  PDDateC.GetDateGroup(DateGroup,LOC_German)
  PDDateC.SetLocale(DateGroup)!,Loc_German)
!----------------------------
Spanish ROUTINE
!----------------------------
  PDDateC.GetDateGroup(DateGroup,LOC_SpanishModern)
  PDDateC.SetLocale(DateGroup)!,Loc_German)
  @@
```

Holiday and Business Day Message

```
IF PDDateC.IsHoliday(DateEntry,Holiday)
    CASE MESSAGE(PDDateC.GetLongDate(DateEntry)&'||This is a holiday
('&CLIP(Holiday)&').  Are you sure you want to enter this
date?','Holiday',ICON:Asterisk,BUTTON:Yes+BUTTON:No)
    OF BUTTON:No
      POST(EVENT:Selected,?DateEntry)
      SELECT(?DateEntry)
    END
  ELSIF NOT PDDateC.IsBusinessDay(DateEntry)
    CASE MEsSAGE(PDDateC.GetLongDate(DateEntry)&'||This is not a business day.  Are
you sure you want to enter this date?','Not Business
Day',ICON:Asterisk,BUTTON:Yes+BUTTON:No)
    OF BUTTON:No
      POST(EVENT:Selected,?DateEntry)
      SELECT(?DateEntry)
    END
  END
```

Setting the Clarion Date Environment

```
  DateGroup LIKE(PDDateGT)

  PDDateC.GetCurrDateGroup(DateGroup)
  LOCALE('CLAMONTH',CLIP(DateGroup.MonthArray[1]) & ','&|
                    CLIP(DateGroup.MonthArray[2]) & ','&|
                    CLIP(DateGroup.MonthArray[3]) & ','&|
                    CLIP(DateGroup.MonthArray[4]) & ','&|
                    CLIP(DateGroup.MonthArray[5]) & ','&|
                    CLIP(DateGroup.MonthArray[6]) & ','&|
                    CLIP(DateGroup.MonthArray[7]) & ','&|
                    CLIP(DateGroup.MonthArray[8]) & ','&|
                    CLIP(DateGroup.MonthArray[9]) & ','&|
                    CLIP(DateGroup.MonthArray[10])& ','&|
                    CLIP(DateGroup.MonthArray[11])& ','&|
                    CLIP(DateGroup.MonthArray[12]))
  LOCALE('CLAMON',CLIP(DateGroup.MonArray[1])   & ','&|
                    CLIP(DateGroup.MonArray[2]) & ','&|
                    CLIP(DateGroup.MonArray[3]) & ','&|
                    CLIP(DateGroup.MonArray[4]) & ','&|
                    CLIP(DateGroup.MonArray[5]) & ','&|
                    CLIP(DateGroup.MonArray[6]) & ','&|
                    CLIP(DateGroup.MonArray[7]) & ','&|
                    CLIP(DateGroup.MonArray[8]) & ','&|
                    CLIP(DateGroup.MonArray[9]) & ','&|
                    CLIP(DateGroup.MonArray[10])& ','&|
                    CLIP(DateGroup.MonArray[11])& ','&|
                    CLIP(DateGroup.MonArray[12]))
```

Displaying Function Results in a Queue

```
FREE(ResultQ)
ResultQ.Func='Week of Year'
ResultQ.Result=PDDateC.Week(DateEntry)
ADD(ResultQ)
ResultQ.Func='First Day of First Week'
Junk#=PDDateC.Week(DateEntry,FirstDay#)!,FirstWeek,FirstDay)
REsultQ.Result=FORMAT(FirstDay#,@d17)
ADD(ResultQ)
ResultQ.Func='Day Name'
REsultQ.Result=PDDateC.DayName(DateEntry)!,gDayArray)
ADD(ResultQ)
ResultQ.Func='Abbrev Day Name'
REsultQ.Result=PDDateC.AbbrevDayName(DateEntry)!,gAbbrevDayArray)
ADD(ResultQ)
ResultQ.Func='Month Name'
REsultQ.Result=PDDateC.MonthName(DateEntry)!,gMonthArray)
ADD(ResultQ)
ResultQ.Func='Abbrev Month Name'
REsultQ.Result=PDDateC.MonName(DateEntry)!,gMonArray)
ADD(ResultQ)
ResultQ.Func='Is Week Day'
IF PDDateC.IsWeekDay(DateEntry)
  REsultQ.Result='TRUE'
ELSE
  ResultQ.Result='FALSE'
END
ADD(ResultQ)
ResultQ.Func='Is Leap Year'
IF PDDateC.IsLeapYear(DateEntry)
  REsultQ.Result='TRUE'
ELSE
  ResultQ.Result='FALSE'
END
ADD(ResultQ)
ResultQ.Func='Day Of Year'
REsultQ.Result=PDDateC.DayOfYear(DateEntry)
ADD(ResultQ)
ResultQ.Func='Days Left in Year'
REsultQ.Result=PDDateC.DaysLeftInYear(DateEntry)
ADD(ResultQ)
ResultQ.Func='Day of Week'
REsultQ.Result=PDDateC.DayOfWeek(DateEntry)!,FirstDay)
ADD(ResultQ)
ResultQ.Func='Is Business Day (US Federal)'
IF PDDateC.IsBusinessDay(DateEntry) !,PDDT_IsHoliday)
  ResultQ.Result='TRUE'
ELSE
  ResultQ.Result='FALSE'
END
ADD(ResultQ)
ResultQ.Func='Bus. Days from Today'
Today#=TODAY()
```

```
REsultQ.Result=PDDateC.BusinessDays(Today#,DateEntry)
REsultQ.DateValue=DateEntry
ADD(ResultQ)
ResultQ.Func='Days Duration - 30 Day Month'
REsultQ.Result=PDDateC.Days360(Today#,DateEntry)!
ADD(ResultQ)
ResultQ.Func='Is Holiday (US Federal)'
REsultQ.Result=PDDateC.IsHoliday(DateEntry)
ADD(ResultQ)
ResultQ.Func='First of Week Rule'
REsultQ.Result=DateGroup.FirstWeekRule
ADD(ResultQ)
ResultQ.Func='First Day Rule (0=Monday)'
REsultQ.Result=DateGroup.FirstDayRule
ADD(ResultQ)
Display(?)
```

Sample Setting Runtime Popup Calendar Options

```
CLEAR(PopCalQ)
PopCalQ.thread=THREAD()
g:Date=DateEntry
IF NOT g:Date
  g:Date=Today()
END
PopCalQ.PassDate&=g:Date
PopCalQ.Control=?DateEntry
PopCalQ.Config = SAV::Config !SELF.Config
IF NOT CalendarCk
  PopCalQ.Config.Caption='PD Cal'
  PopCalQ.Config.Flags=BOR(PopCalQ.config.Flags,PDDT_eCalCreateFields)
  IF SystemFontCk
    PopCalQ.Config.flags=BOR(PopCalQ.Config.flags,PDDT_eCalSystemFont)
  ELSE
    PopCalQ.Config.flags-=BAND(PopCalQ.Config.flags,PDDT_eCalSystemFont)
  END
ELSE
  PopCalQ.Config.Flags=PDDT_eCalNoCaption
  IF SystemFontCk
    PopCalQ.Config.flags=BOR(PopCalQ.Config.flags,PDDT_eCalSystemFont)
  ELSE
    PopCalQ.Config.flags-=BAND(PopCalQ.Config.flags,PDDT_eCalSystemFont)
  END
END
IF NOT TrnCk
  PopCalQ.Config.Flags-=BAND(PopCalQ.Config.Flags,PDDT_eTRN)
ELSE
  PopCalQ.Config.Flags=BOR(PopCalQ.Config.Flags,PDDT_eTRN)
END
IF NOT WPCk
  PopCalQ.Config.WallPaper=''
ELSE
  PopCalQ.Config.WallPaper=GETINI('Config','WallPaper','','.\pddt.ini')
END
PopCalQ.Config.Flags=BOR(PopCalQ.Config.Flags,PDDT_eCalMDI)
PopCalQ.Config.LowerLimit=eDateMinimum
PopCalQ.Config.UpperLimit=eDateMaximum
PopCalQ.Xpos=?{PROP:Xpos}+0{PROP:Xpos}+16
PopCalQ.Ypos=?{PROP:Ypos}+0{PROP:YPos}+12
PopCalQ.Config.event=EVENT:Calendar
```

Sample TimeZone Procedure

```
TimeZoneProc PROCEDURE

TimeZone             LIKE(TimeZoneType)

TZ                      GROUP!,PRE(TZ)
Bias                     LONG                    !Minutes from UTC
StandardName             USHORT,DIM(32)          !Name of Standard Time
StandardDate             LIKE(StandardTime)      !Standard Date Inforamtion
StandardBias             LONG                    !Standart Time Bias (Minutes)
DaylightName             USHORT,DIM(32)          !Name of Daylight time
DaylightDate             LIKE(DaylightTime)      !Daylingt Time Information
DaylightBias             LONG                    !Daylight Time Bias (minutes)
                        END

!-----------------------
!WINDOW
!-----------------------

TZwindow WINDOW('Time Zone 
Information'),AT(,,221,149),FONT('Arial',8,,FONT:regular),CENTER,IMM,TIMER(20), |
         SYSTEM,GRAY
       STRING('Dummy'),AT(13,22,195,10),FONT(,,,FONT:bold),USE(?TZReturn)
       STRING('UTC Bias'),AT(13,34),USE(?String20)
       ENTRY(@n(15)),AT(69,34,,10),FONT(,,0FFH,),USE(TimeZone.Bias),SKIP,READONLY
       SHEET,AT(6,6,211,132),USE(?Sheet1),SPREAD
         TAB('&Standard Time'),USE(?Tab1)
           STRING('Standard Name'),AT(14,46),USE(?String21)
           STRING('Standard Bias'),AT(14,58),USE(?String22)

ENTRY(@s32),AT(70,46,,10),FONT(,,0FFH,),USE(TimeZone.StandardName),SKIP,READONLY

ENTRY(@n(15)),AT(70,58,,10),FONT(,,0FFH,),USE(TimeZone.StandardBias),SKIP,READONLY
           STRING('Start of Standard 
Time'),AT(14,71),FONT(,,,FONT:bold),USE(?String1)
           STRING('Year'),AT(14,83),USE(?String2)
           STRING('Month'),AT(14,95),USE(?String3)
           STRING('Day Of Week'),AT(14,107),USE(?String4)
           STRING('Day'),AT(14,119),USE(?String5)
           STRING('Hour'),AT(124,83),USE(?String6)
           STRING('Minute'),AT(124,95),USE(?String7)
           STRING('Second'),AT(124,107),USE(?String8)
           STRING('MilliSec'),AT(124,119),USE(?String9)

ENTRY(@n_6),AT(70,83,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sYear),SKIP,READONL
Y

ENTRY(@n_6),AT(70,95,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sMonth),SKIP,READON
LY

ENTRY(@n_6),AT(70,107,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sDayOfWeek),SKIP,R
EADONLY

ENTRY(@n_6),AT(70,119,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sDay),SKIP,READONL
Y

ENTRY(@n_6),AT(162,83,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sHour),SKIP,READON
LY
```

```
ENTRY(@n_6),AT(162,95,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sMinute),SKIP,READ
ONLY

ENTRY(@n_6),AT(162,107,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sSecond),SKIP,REA
DONLY

ENTRY(@n_6),AT(162,119,,10),FONT(,,0FFH,),USE(TimeZone.StandardDate.sMilliseconds),SK
IP,READONLY
         END
         TAB('&Daylight Time'),USE(?Tab2)
           STRING('Daylight Name'),AT(14,46),USE(?String20x)
           STRING('Daylight Bias'),AT(14,58),USE(?String22x)

ENTRY(@s32),AT(70,46,,10),FONT(,,0FFH,),USE(TimeZone.DaylightName),SKIP,READONLY

ENTRY(@n(15)),AT(70,58,,10),FONT(,,0FFH,),USE(TimeZone.DaylightBias),SKIP,READONLY
           STRING('Start of Daylight
Time'),AT(14,71),FONT(,,,FONT:bold),USE(?String1x)
           STRING('Year'),AT(14,83),USE(?String2x)
           STRING('Month'),AT(14,95),USE(?String3x)
           STRING('Day Of Week'),AT(14,107),USE(?String4x)
           STRING('Day'),AT(14,119),USE(?String5x)
           STRING('Hour'),AT(124,83),USE(?String6x)
           STRING('Minute'),AT(124,95),USE(?String7x)
           STRING('Second'),AT(124,107),USE(?String8x)
           STRING('MilliSec'),AT(124,119),USE(?String9x)

ENTRY(@n_6),AT(70,83,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dYear),SKIP,READONL
Y

ENTRY(@n_6),AT(70,95,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dMonth),SKIP,READON
LY

ENTRY(@n_6),AT(70,107,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dDayOfWeek),SKIP,R
EADONLY

ENTRY(@n_6),AT(70,119,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dDay),SKIP,READONL
Y

ENTRY(@n_6),AT(162,83,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dHour),SKIP,READON
LY

ENTRY(@n_6),AT(162,95,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dMinute),SKIP,READ
ONLY

ENTRY(@n_6),AT(162,107,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dSecond),SKIP,REA
DONLY

ENTRY(@n_6),AT(162,119,,10),FONT(,,0FFH,),USE(TimeZone.DayLightDate.dMilliseconds),SK
IP,READONLY
         END
       END
     END
  CODE
  OPEN(TZWindow)
  CASE  PDDateC.GetTimeZone(TimeZone)
  OF PDDT_TZUnknown                     !Set return value display
    ?TZReturn{PROP:TExt}=PDDT_MsgUnknown
  OF PDDT_TZDaylight
    ?TZReturn{PROP:TExt}=PDDT_MsgDaylight
  OF PDDT_TZStandard
```

```
      ?TZReturn{PROP:TExt}=PDDT_MsgStandard
    OF PDDT_TZFails
      ?TZReturn{PROP:TExt}=PDDT_MsgFails
    END
    ACCEPT
    END
```

Sample Appointment Calendar

```
!--------------------------------------------------------------------------------
AppointmentCalendar PROCEDURE
!-------------------------------------------------------------------------------
!|
!| This procedure shows an example of a multi-threaded calendar with a "TODO"
!| list and appointment calendar.  It has the following elements:
!|
!| AppQ --     The appointment queue of times, a short appointment description,
!|             and a long appointment description.
!|
!| AptFileQ -- A Queue, representing a file, in which appointments are stored.
!|
!| ToDoFileQ-- A Queue, representing a file, in which a day's to do list is stored.
!|
!| A date entry field, called pDate, is linked to the calendar and a PD Date
Scrolling
!| class.  This class is defined as
!|
!|
!|
LC                CLASS(PDListCalCT)
TakefieldEvent      PROCEDURE,BYTE,VIRTUAL
                  END

AppQ              QUEUE,STATIC
Time                LONG
Description         STRING(20)
AptNotes            STRING(120)
Pos                 LONG
                  END

!-- Used here for testing.  Add a real file for an application.
AptFileQ          QUEUE,STATIC
Date                LONG
Time                LIKE(AppQ.Time)
Description         LIKE(AppQ.Description )
AptNotes            LIKE(AppQ.AptNotes)
Pos                 LIKE(AppQ.Pos )
                  END
ToDoFileQ         QUEUE,STATIC
Date                LONG
ToDoNotes           STRING(120)
                  END

pDate             LONG,STATIC
AptNotes          STRING(SIZE(AppQ.AptNotes))
Description       STRING(SIZE(AppQ.Description))
ToDoNotes         STRING(SIZE(ToDoFileQ.ToDoNotes))
Date              LONG,STATIC
Time              LONG,STATIC
```

```
window WINDOW('Appointment
Calendar'),AT(,,287,221),FONT('Arial',8,,FONT:regular,CHARSET:ANSI),COLOR(COLOR:WINDO
W), |
         IMM,ICON('CAL2.ICO'),SYSTEM,GRAY,RESIZE,MDI
       GROUP('&Calendar'),AT(2,5,103,82),USE(?Group1),BOXED
         STRING(@s20),AT(16,15,71,10),USE(?MonthYrTitle),CENTER
         IMAGE(ICON:VCRback),AT(4,15,10,8),USE(?PreviousMonthIMG),CENTERED
         REGION,AT(4,15,10,8),USE(?PreviousMonth)
         IMAGE(ICON:VCRplay),AT(90,15,10,8),USE(?NextMonthIMG),CENTERED
         REGION,AT(90,15,10,8),USE(?NextMonth)
         STRING('* '),AT(10,27,10,8),USE(?Heading1),LEFT(1)
         STRING('* '),AT(20,27,10,8),USE(?Heading2),LEFT(1)
         STRING('* '),AT(32,27,10,8),USE(?Heading3),LEFT(1)
         STRING('* '),AT(42,27,10,8),USE(?Heading4),LEFT(1)
         STRING('* '),AT(54,27,10,8),USE(?Heading5),LEFT(1)
         STRING('* '),AT(64,27,10,8),USE(?Heading6),LEFT(1)
         STRING('* '),AT(76,27,10,8),USE(?Heading7),LEFT(1)
         IMAGE('Pddt_q.ico'),AT(90,27,10,8),USE(?TodayIMG),CENTERED
         REGION,AT(90,27,10,8),USE(?TodayButton)

LIST,AT(6,35,77,48),USE(?PopList),TRN,ALRT(F2Key),COLUMN,FORMAT('11R(1)*@n_2b@11R(1)*
@n_2b@11R(1)*@n_2b@11R(1)*@n_2b@11R(1)*@n_2b@11R(1)*@n_2b@12' &|
             'R(1)*@n_2b@'),FROM('*'),DRAGID('PDListCal')

LIST,AT(90,35,9,48),USE(?WeekList),SKIP,TRN,NOBAR,FONT(,,COLOR:Gray,,CHARSET:ANSI),FO
RMAT('10L@n_2b@#36#'), |
             FROM('*')
       END
       PROMPT('&Description'),AT(4,90),USE(?Prompt5)

ENTRY(@s20),AT(4,100,60,10),USE(Description),IMM,ALRT(UpKey),ALRT(EnterKey),ALRT(Down
Key)
       PROMPT('This illustrates the use of the Calendar Control with an appointment
list of tim' &|
           'es.  The entry field has Enter, Up and Down keys alerted to enter and
scroll the' &|
           ' highlithed time.  Each time any entry is completed, it adds the
appointment. Th' &|
           'e appoinment date heading uses the Date Tools LongDate method.  The date
may be ' &|
           'changed by selecting the Calendar
Control.'),AT(3,114,103,102),USE(?Prompt2)
       BUTTON('Close'),AT(237,201,45,14),USE(?CloseBTN)

LIST,AT(112,9,172,183),USE(?CalList),SKIP,VSCROLL,VCR,FORMAT('[40L(1)|_M~Time~L(2)@t7
@320L(2)|_M~Appointment~]|M~***~'), |
           FROM(AppQ)
     END
!----------------------------
  CODE
!----------------------------
  OPEN(Window)
  DO InitList
  pDate=Today()
  DO SampleToDoList
  DO PrimeAppQ
  DO RefreshDayCalendar
  CASE THREAD()
  OF 2
    Name"='Ross (Variable Entry)'
  OF 3
    Name"='Susan (Variable Entry)'
  OF 4
    Name"='David (Variable Entry)'
  ELSE
    Name"='Team '&THREAD() &' (Variable Entry)'
  END
  0{PROP:Text}=ClIP(Name")
  SELECT(?CalList,1,1)
  SELECT(?PopList)
  ACCEPT
    LC.TakeEvent
    CASE FIELD()
    OF ?CloseBTN
      CASE EVENT()
      OF EVENT:Completed
        POST(EVENT:Completed,?Description)
        POST(EVENT:CloseWindow)
      END
    OF ?CalList
      CASE EVENT()
      OF EVENT:NewSelection
        GET(AppQ,CHOICE(?))
        AptNotes=AppQ.AptNotes
        Description=AppQ.Description
        DISPLAY()
      END
    OF ?Description
      CASE EVENT()
      OF EVENT:Accepted
        Update(?)
        AppQ.Description=Description
        DO AddApointment
      OF EVENT:AlertKey
        CASE KEYCODE()
        OF UpKey
          IF CHOICE(?CalList)=1
            SELECT(?CalLIst,RECORDS(AppQ))
          ELSE
            SELECT(?CalList,CHOICE(?CalList)-1)
          END
          POST(EVENT:NewSelection,?CalList)
          SELECT(?)
        OF DownKey
          IF CHOICE(?CalList)=RECORDS(AppQ)
            SELECT(?CalList,1)
          ELSE
            SELECT(?CalList,CHOICE(?CalList)+1)
          END
          POST(EVENT:NewSelection,?CalList)
          SELECT(?)
        OF EnterKey
          SELECT(?CalList,CHOICE(?CalList))
          POST(EVENT:NewSelection,?CalList)
          SELECT(?)
        END
      END
    OF ?PopList
    END
  END

!----------------------------
PrimeAppQ ROUTINE
!----------------------------
```

```
!|
!| This routine fills the appointment queue with
!| empty values starting at 8am through 6pm
!|
  FREE(AppQ)
  CLEAR(AppQ)
  Time=(60*60*8*100)+100 !8am
  LOOP I#=1 TO 41
    AppQ.Time=Time
    Time += 100*60*30  !Add 30 Minutes
    IF TIME>100*60*60*18 THEN BREAK.
    AppQ.AptNotes=LEFT(CLIP(FORMAT(AppQ.Time,@t1))) &': '&'Sample Appointment Entry'
    AppQ.Pos=I#
    ADD(AppQ,AppQ.Pos)
  END
!----------------------------
RefreshAppQ ROUTINE
!----------------------------
!|
!| With each date change, the appointment queue is reprimed with empty times
!| appointments.  These are then replaced if an appointment is found in the
!| appointment file.
!|
  ?CalList{PROPList:Header+PROPList:Group,1}=PDDateC.GetLongDate(pDate)
  DO PrimeAppQ
  LOOP I#=1 TO RECORDS(AptFileQ)
    GET(AptFileQ,I#)
    IF AptFileQ.Date=pDate
      AppQ.pos=AptFileQ.pos
      GET(AppQ,AppQ.pos)
      AppQ :=: AptFileQ
      PUT(AppQ)
    END
  END
!----------------------------
RefreshDayCalendar ROUTINE
!----------------------------
!|
!| This procedure refhreshes all entries with a change in the entry date.
!| If first looks for ToDo items in the ToDoFileQ.  It then repfreshes
!| the appointment queue.
!|
!| It finally posts an event new slection to the appointment list to
!| repfresh values for the current slection.
!|
  ToDoFileQ.Date=pDate
  GET(ToDoFileQ,ToDoFileQ.Date)
  IF NOT ErrorCode()
    ToDoNotes=ToDoFileQ.ToDoNotes
  ELSE
    CLEAR(ToDoNotes)
  END
  DO RefreshAppQ
  POST(EVENT:NewSelection,?CalList)
!----------------------------
AddApointment ROUTINE
!----------------------------
!|
!| This routine adds an appointment to the AptQ and the AptFileQ.
!| The description has already been entered in descritiption entry
!| field.
!|
  PUT(AppQ,AppQ.Pos)
  AptFileQ.Date=pDate
  AptFileQ.Pos=AppQ.Pos
  GET(AptFileQ,AptFileQ.Date,AptFileQ.Pos)
  IF ERRORCODE()
    AptFileQ :=: AppQ
    ADD(AptFileQ,AptFileQ.Date,AptFileQ.Pos)
  ELSE
    AptFileQ :=: AppQ
    PUT(AptFileQ)
  END
  DISPLAY()
!----------------------------
AddToDoItem ROUTINE
!----------------------------
!|
!| This routine adds or replaces a todo entry
!|
  ToDoFileQ.Date=pDate
  GET(ToDoFileQ,ToDoFileQ.Date)
  IF ERRORCODE()
    ToDoFileQ.ToDoNotes=ToDoNotes
    ADD(ToDoFileQ,ToDoFileQ.Date)
  ELSE
    ToDoFileQ.ToDoNotes=ToDoNotes
    PUT(ToDoFileQ)
  END
!----------------------------
SampleToDoList ROUTINE
!----------------------------
!|
!| This routine puts sample values in the todo file queue.
!|
  ToDoFileQ.Date=pDate-15
  LOOP 30 TIMES
    ToDoFileQ.ToDoNotes=FORMAT(ToDoFileQ.Date,@d18)&': '&'Sample To Do Entry'
    ADD(ToDoFileQ,ToDoFileQ.Date)
    ToDoFileQ.Date+=1
  END

InitList ROUTINE
  LC.Init(?PopList,PDDateC,pDate)

  LC.AddButton(PDBTN:Help,?TodayButton)
  LC.AddButton(PDBTN:PrevMo,?PreviousMonth)
  LC.AddButton(PDBTN:NextMo,?NextMonth)
  LC.SetImmediate(TRUE)
  LC.AddHeaders(?Heading1)
  LC.AddWeekList(?WeekList)
  LC.AddMonthTitle(?MonthYrTitle)
  !-- Other items not added
!  LC.SetRange(TODAY(),TODAY()+365)
!  LC.AddButton(eOKBTN,eOkButton)
!  LC.AddButton(ePrevYrBTN,?PreviousYear)
!  LC.AddButton(eNextYrBTN,eNextYear)
  LC.Open

LC.TakeFieldEvent PROCEDURE
RV byte
  CODE
  PARENT.TakeFieldEvent()
  IF FIELD()=?PopList
```

```
      DO RefreshDayCalendar
    END
    return 0
```

ProDomus Advanced Scheduling Add On

Advanced Scheduling Features

Day Appointment List. A control template and class library create a time incremented appointment list containing start and ending dates and times and recurring items. The appointment listing is a queue that may be designed in any format and with any set of fields from an appointment file and related files. The queue may show a single schedule or a group schedule. It also displays schedule durations with brackets from the from time to the to time. The control template populates add, change, and delete buttons and a formatted list which the user may modify as needed.

Month-Day-Year Entry Control Template. A control template and class library create web style date entries controls consisting of separate file drops for the month, day, and year and buttons for the next day and previous day. The entry may be ranged limited. Using the date class, it also translates the months for any supported windows locale. A change in any field updates a date field which may be displayed or not displayed.

Example Program

Doctors.app. Doctors.App and Doctor.Dct illustgrate the use of both the Month-Day-Year and the Appointment Control Templates and Classes.

Advanced Scheduling Classes

PDMonYrCT – Month Year File Drop Class

Month Year class creates and handles a drop listing for a choice of month and year. The month list drops a list of months names translated by the PD Date Class.

PDMonDayYrCT – Month Day Year File Drop Class

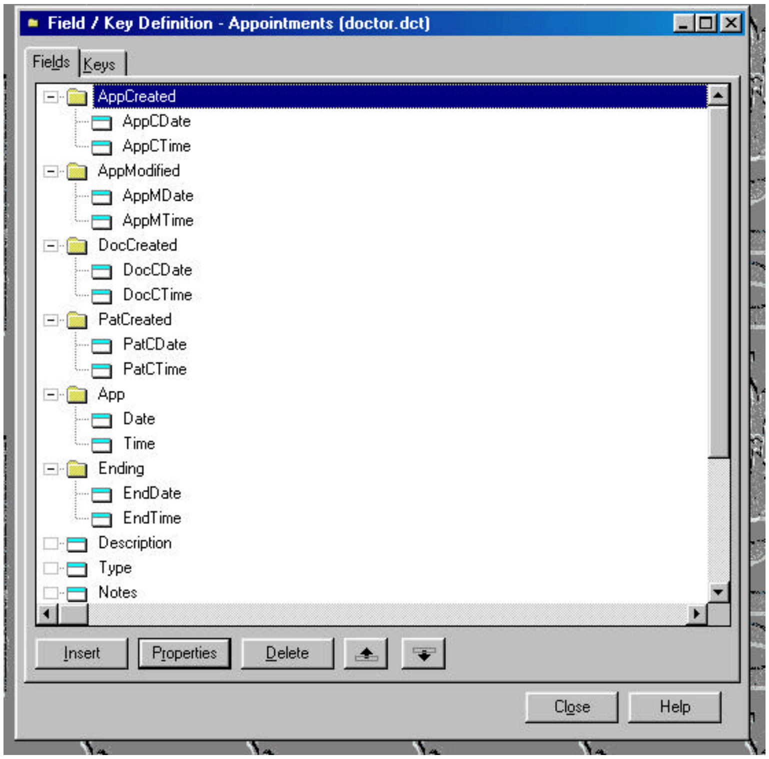
The Month Day Year Class is derived from the Month Year Class. It adds day handling, next and previous buttons, and range limit handling.

PdAppDayListCT – Appointment Day Calendar Class

The Appointment Day Calendar Class handles scheduled events by time increments, with starting dates, ending dates, starting times, and ending times. In also handles brackets displaying the duration of scheduled items.

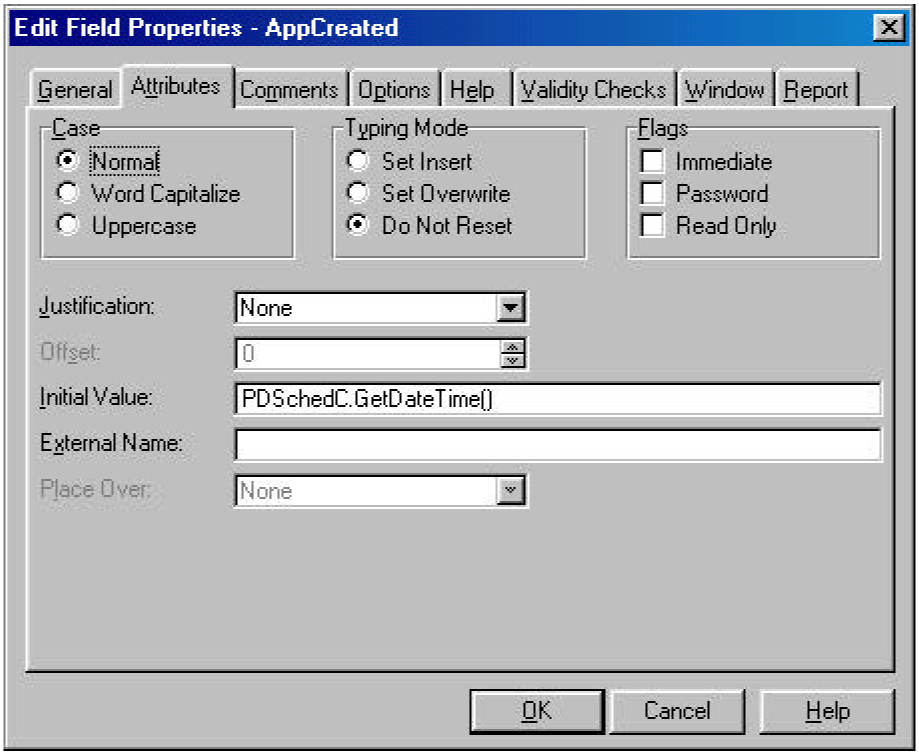
Steps – Scheduling File Definition

Using the Appointment Class library and templates should begin with the proper definition of the appointment file in the dictionary. In particular, the use of DateTime groups will make file management much easier.



- AppCreated
  - AppCDate
  - AppCTime
- AppModified
  - AppMDate
  - AppMTime
- DocCreated
  - DocCDate
  - DocCTime
- PatCreated
  - PatCDate
  - PatCTime
- App
  - Date
  - Time
- Ending
  - EndDate
  - EndTime
- Description
- Type
- Notes

1. Open the dictionary editor and create a new file.
2. Create an AppCreated Group. This group will be used in a unique key.



Case
● Normal
○ Word Capitalize
○ Uppercase

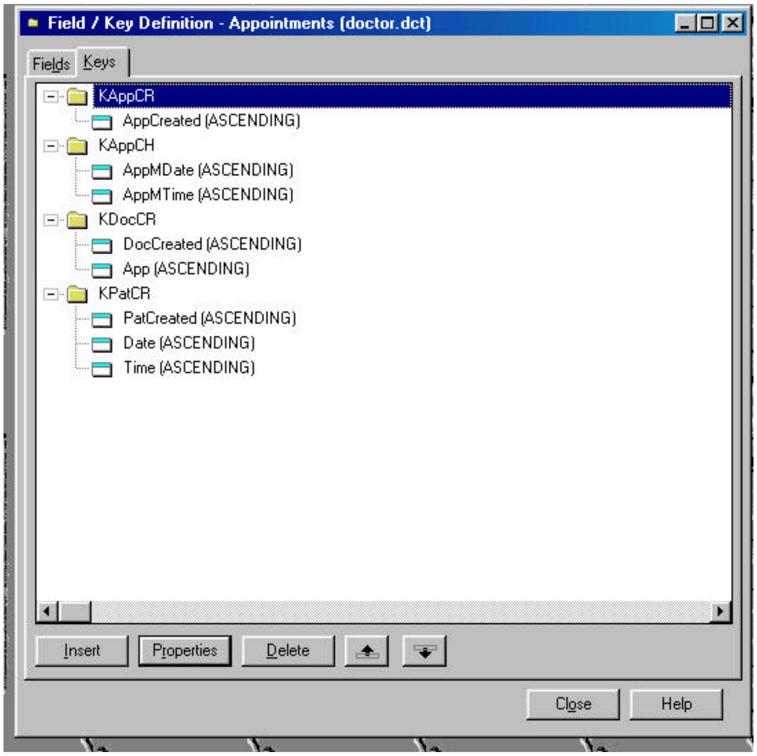
Typing Mode
○ Set Insert
○ Set Overwrite
● Do Not Reset

Flags
☐ Immediate
☐ Password
☐ Read Only

Justification: None
Offset: 0
Initial Value: PDSchedC.GetDateTime()
External Name:
Place Over: None

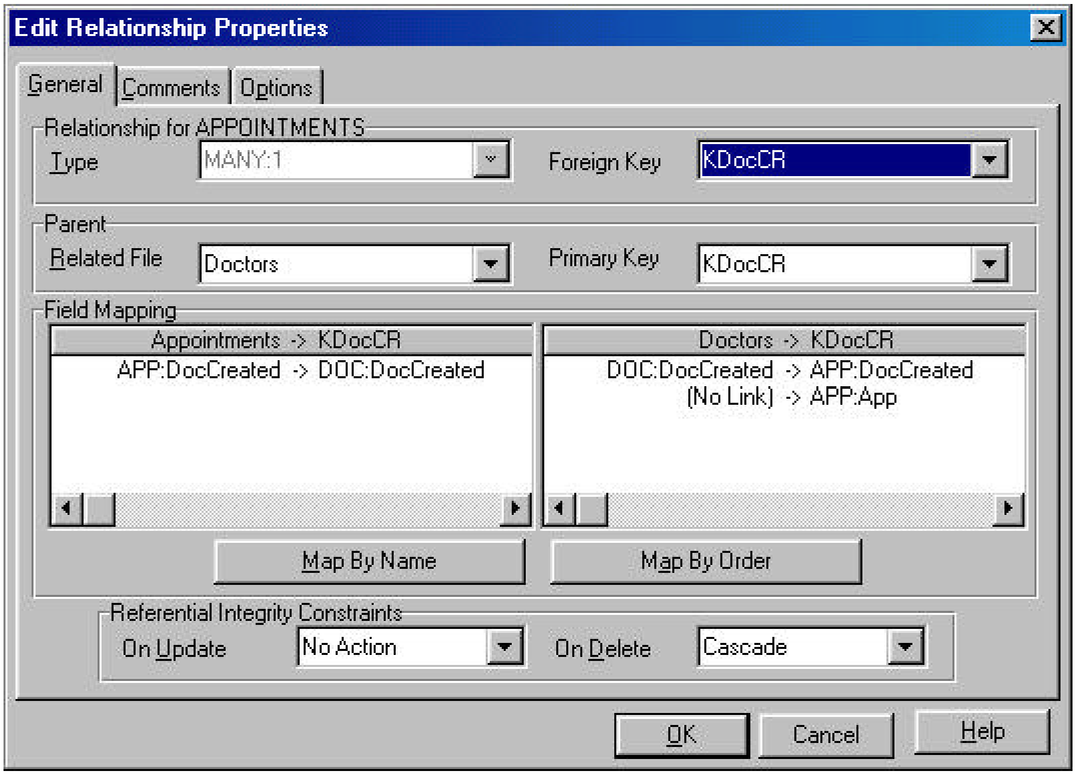
3. Click the attributes tab and add **PDSchedC.GetDateTime()** as the initial value. This will prime the field with the date and time when a record is created.
4. Click the Options Tab and Check Do No Auto-Populate this field.

5. Add an AppDate Field to the AppCreated Group. Define this as a long.
6. Add an AppTime Field to the AppCreated Group. Define this as a long.
7. If you are using ending dates and times, add a group for the ending date and time.
8. Create a similar group for the appointment date and time. (The App group in the screen above). This will be a field in a key allowing duplicates.
9. You can use similar groups for file relationships. In the Doctor example, groups are created for the Patient file and the Doctor File.
10. Add a appointment description. The appointment Queue library provides a STRING(80) for this purpose.
11. Add other fields, including relational fields, as needed.



- KAppCR
  - AppCreated (ASCENDING)
- KAppCH
  - AppMDate (ASCENDING)
  - AppMTime (ASCENDING)
- KDocCR
  - DocCreated (ASCENDING)
  - App (ASCENDING)
- KPatCR
  - PatCreated (ASCENDING)
  - Date (ASCENDING)
  - Time (ASCENDING)

12. Add a key for the Create Group. This should require a unique value and exclude empty keys.
13. Add a key for the appointment. In the example of the KPatCreated is used to relate the appointment file with the Patiend File and allow filtering by the PatCreated DateTime field.
14. Add other relational key fields as needed.



Relationship for APPOINTMENTS
Type: MANY:1
Foreign Key: KDocCR

Parent
Related File: Doctors
Primary Key: KDocCR

Field Mapping

Appointments -> KDocCR	Doctors -> KDocCR
APP:DocCreated -> DOC:DocCreated	DOC:DocCreated -> APP:DocCreated
	(No Link) -> APP:App

Referential Integrity Constraints
On Update: No Action
On Delete: Cascade

15. Add file relationships as need. In the example program, there are one to many relationships between the Patient file and appointments an between the Doctor file and appointments.

Steps - Appointment Procedure

1. Select Procedure/New and create a new Window procedure.
2. Open the Window of the new procedure.
3. Select Populate/Control Templates and select the PDDT Day Appointment List Control Template.
4. Press OK when the list formatter appears.
5. Resize the list control or window as needed.
6. Populate a field for the date entry and a string to display a long date.
7. Right click any of the controls and select Actions.
8. Select the Display Date Tab
9. Enter the Date Field.
10. Enter the Date Entry Control. If there are multiple controls such as a button or Mondth, Day, Year entry, any of them may be entered here.
11. Date Display String. If you have a string displaying the long date, enter the control equate.
12. Appointment Update Form. Enter a form for updating the appointment file.
13. Parameters. If the update form takes parameters, enter the parameters. See the Doctors.app for an example passing the selected date and time.
14. Procedure to load schedule. You may call a procedure to load the schedule using this template or load the shedule separately as done in the Doctors.app.
15. Advanced Button. If you have additional fields affect the date such as a lookup button or a Month-Day-Year type of entry, you may enter them using the Advanced Button.
16. See PD Day Appointment List Control Template for entries on other tabs.

17. To load the schedule, create a process procedure and add the PDDT AddSchedule Code Template to TakeRecord under Local Objects, This Process.

Scheduling Templates

PD Month-Day-Year Entry Control Template

This template populates drop lists for the month, day, and year entries and previous and next day buttons. It generates code that using the PDMonDayYr Class library.

PD Day Appointment List Control Template

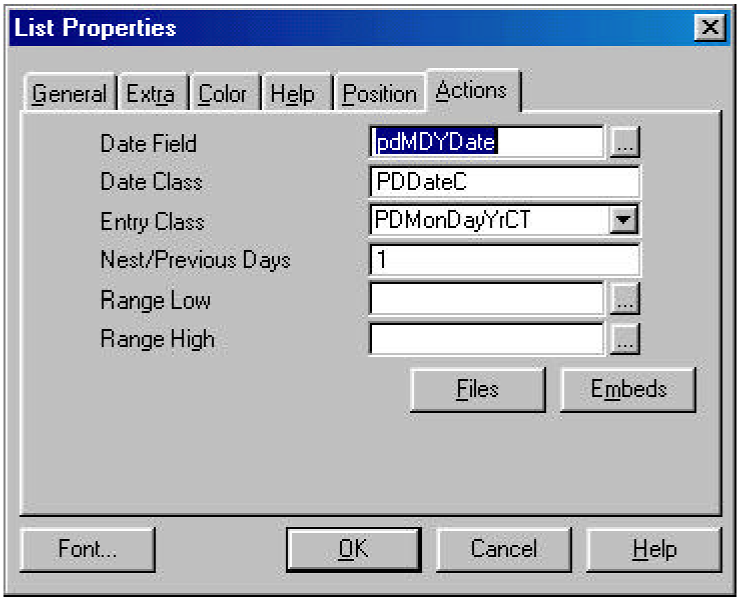
This template populates an appointment list box with update buttons. It generates code using the PdappDayListCT class library. The list box includes a default format which may be modified. The template and class library allow for more than one field to affect a change in date such as might be the case with a date entry plus calendar button, a small calendar list, or a month-day-year set of drop lists. It handles assignment of fields from an appointment file and related lookup files.

PDDT Day Appointment Default Data Extension Template

This unusual template creates default local data for Bracket fields. The template may be populated and then removed to allow modification of the data. It also includes a #FIELD listing that can be pasted into the list format in window properties of the appointment procedure. This links the fields to the listbox format.

PD Month-Day-Year Entry Control Template

This template populates drop lists for the month, day, and year entries and previous and next day buttons. It generates code that using the PDMonDayYr Class library. Note that the Doctor example includes two appointment schedule windows. The BrowseAppointmentsWeb had a Month-Day-Year Entry control and a scheduled coded without the use of the appointment class and template. The BrowseAppointmentsWin uses the Appointment Class and Template.



Date Field: pdMDYDate
Date Class: PDDateC
Entry Class: PDMonDayYrCT
Nest/Previous Days: 1
Range Low:
Range High:

Date Field. The date field to be updated by any change in the month day year controls.

Entry Class. The date entry class to use.

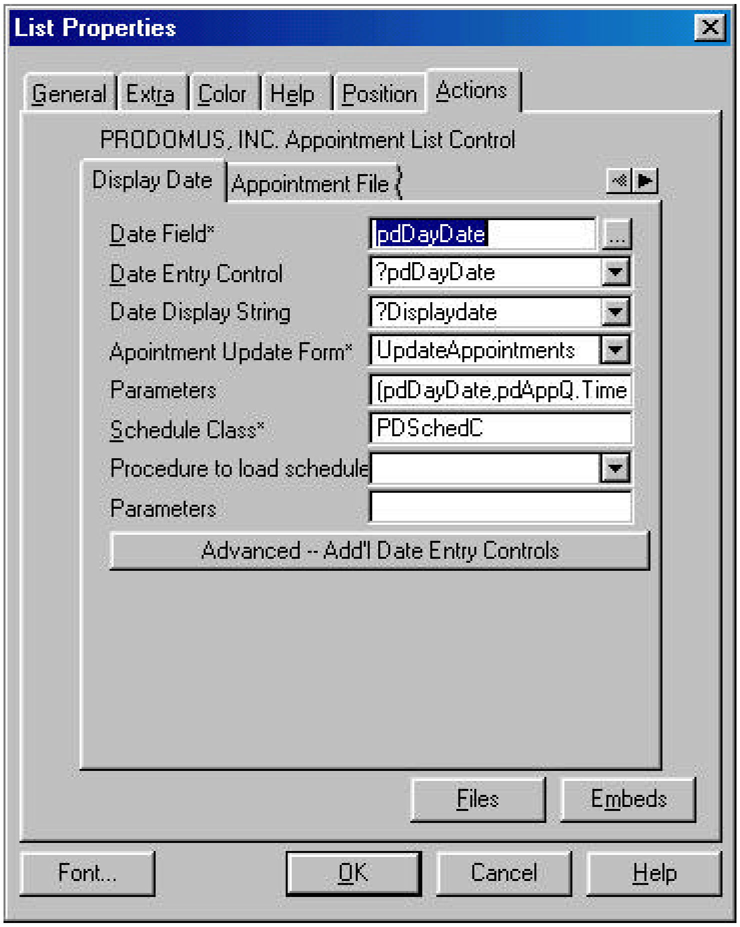
Next/Previous Days. The number of days to scroll when pressing the next/previous buttons.

Range Low. A field containing the high range limit of the date field.

Range High. A field containing the low range limit of the date field.

PD Day Appointment List Control Template

This template populates an appointment list box with update buttons. It generates code using the PdappDayListCT class library. The list box includes a default format which may be modified. The template and class library allow for more than one field to affect a change in date such as might be the case with a date entry plus calendar button, a small calendar list, or a month-day-year set of drop lists. It handles assignment of fields from an appointment file and related lookup files.



Date Field* pdDayDate

Date Entry Control ?pdDayDate

Date Display String ?Displaydate

Apointment Update Form* UpdateAppointments

Parameters (pdDayDate,pdAppQ.Time

Schedule Class* PDSchedC

Procedure to load schedule

Parameters

Display Date Tab

Date Field. Date for which to display the schedule.

Date Display String. Optional long date display string.

Apointment Update Form. Form to be called when updating the schedule.

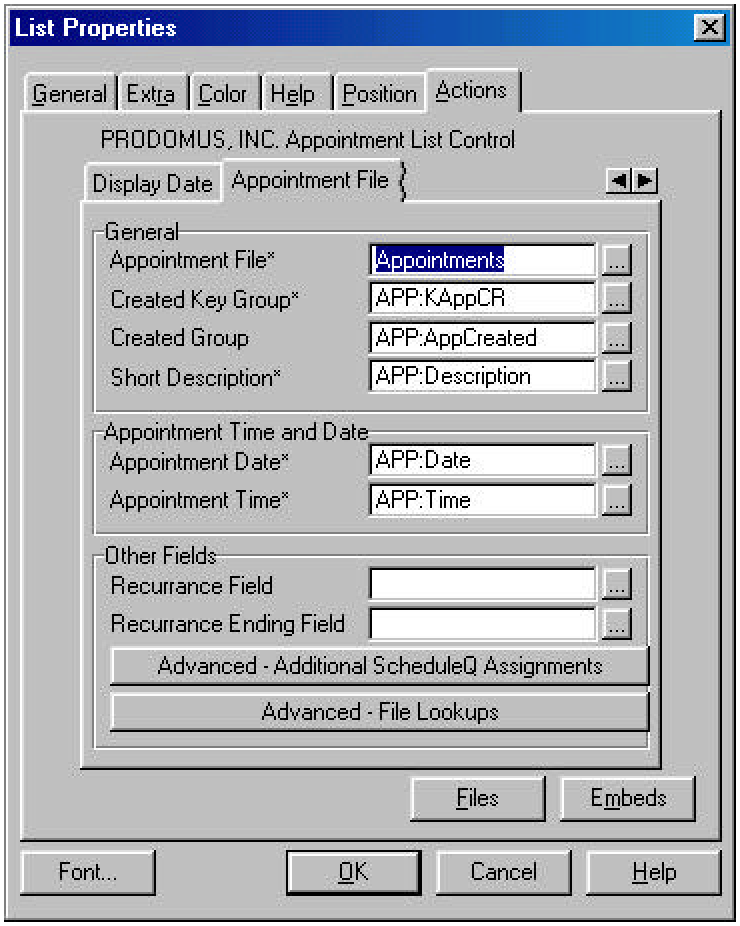
Parameters. Parameters to pass to the update form if any.

Schedule Class. Schedule Class to use. Normally PDSchedC.

Procedure to load schedule. Optional procedure to load the shedule when the window is called.

Parameters. Parameters to pass to the load procedure.

Advanced -- Add''l Date Entry Controls. Additional controls affecting the date to be displayed.



Appointment File* Appointments
Created Key Group* APP:KAppCR
Created Group APP:AppCreated
Short Description* APP:Description
Appointment Date* APP:Date
Appointment Time* APP:Time
Recurrance Field
Recurrance Ending Field

Appointment File Tab

Appointment File. The appointment file. Required.

Created Key Group. A unique key containing one or both of the fields in the Created Group below.

Created Group. The appointment file must have a group of two long fields. The group should consist of two LONG fields (typically date and time or a unused field and an auto incremented field).

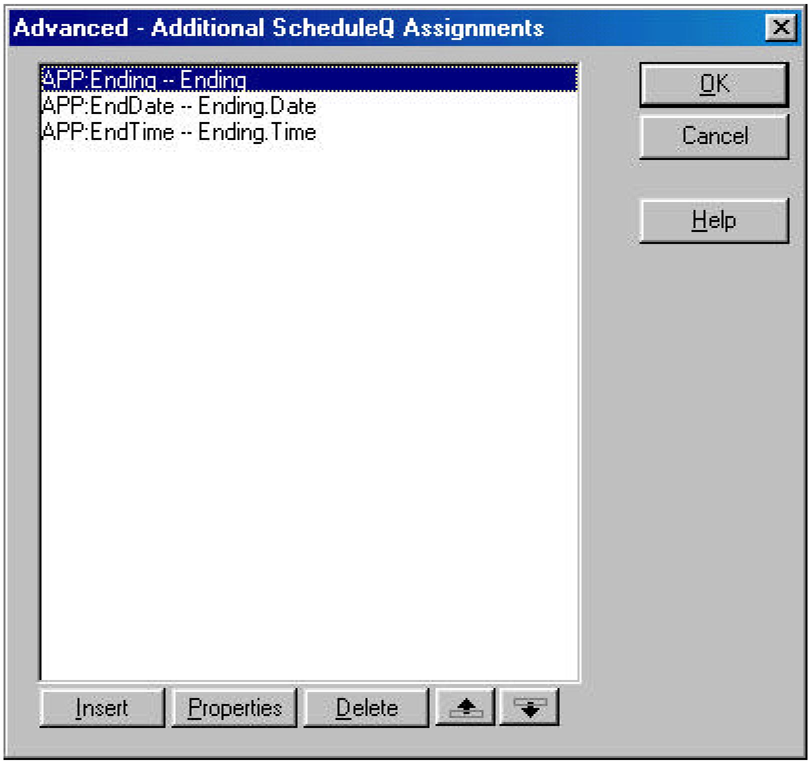
Short Description. A description field from the appointment file. Required.

Appointment Date. The field containing the appointment Date. Required.

Appointment Time. The field containing the appointment time. Required.

Recurrence Field. A ULONG field containing bitmapped recurrence flags as declared in the PDDate.inc file.

Recurrence Ending Field. A field containing the date at which recurrence ends.



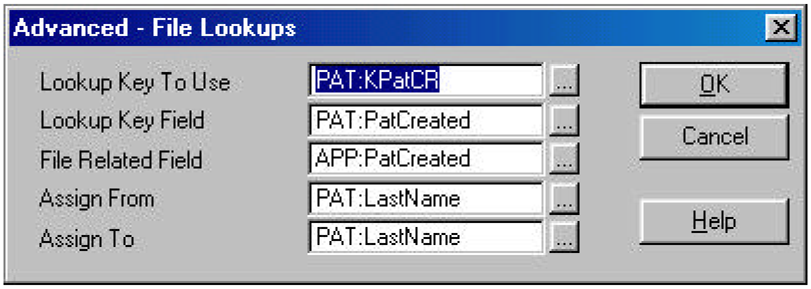
APP:Ending -- Ending
APP:EndDate -- Ending.Date
APP:EndTime -- Ending.Time

Advanced - Additional ScheduleQ Assignments Button.

These entries relate file fields with specific entries in the ScheduleQ defined in PDSched.inc. They are in addition to the created group, date, time, recurrence, and description fields above.

File Field. Field Field.

Schedule Field. The corresponding field from the PDSchedCT ScheduleQ. Choices are: Notes|Other|Flags|Ending|Ending.Date|Ending.Time|Modified|Modified.Date|Modified.Time.



Lookup Key To Use	PAT:KPatCR
Lookup Key Field	PAT:PatCreated
File Related Field	APP:PatCreated
Assign From	PAT:LastName
Assign To	PAT:LastName

Advanced - File Lookups.

These entries use the class library to lookup fields related to those in the Appointment file.

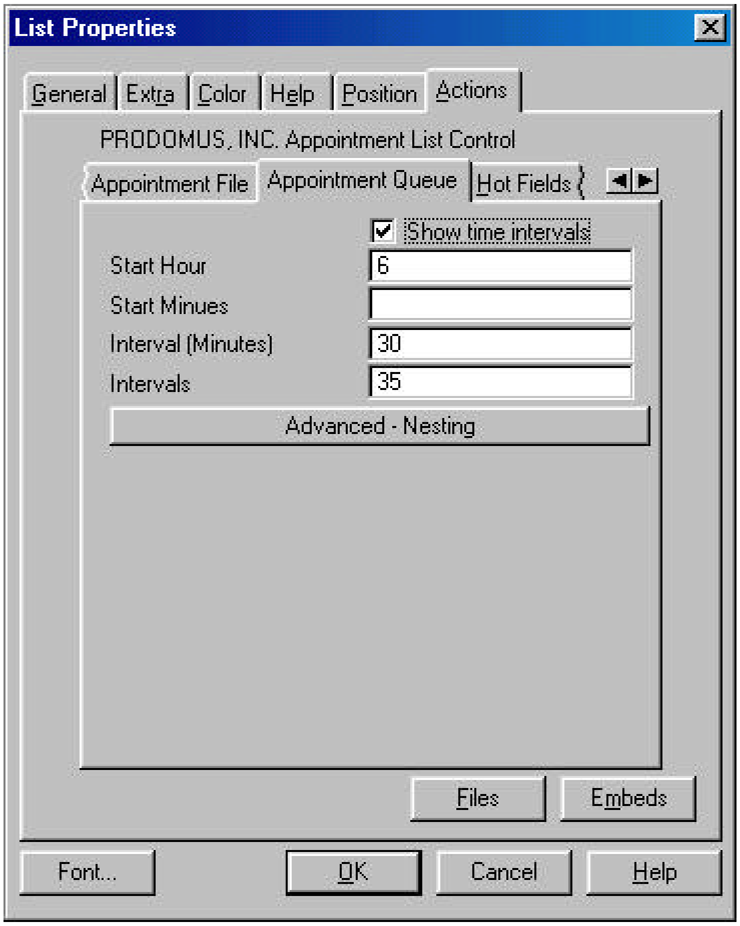
Lookup Key To Use. The lookup key.

Lookup Key Field. The lookup key field.

File Related Field. The field in the appointment file related to the Lookup Key Field.

Assign From. The field containing the value to assign.

Assign To. The field to which to assign the value.



PRODOMUS, INC. Appointment List Control

☑ Show time intervals

Start Hour 6

Start Minues

Interval (Minutes) 30

Intervals 35

Appointment Queue Tab

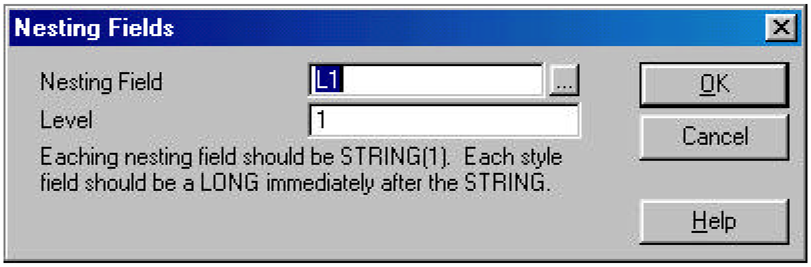
Show time intervals. Check to show time intervals in the schedule.

Start Hour. Enter the starting hour, i.e. 7 or 14 (for 2pm).

Start Minutes. Start minutes after the start hour.

Interval (Minutes). The interval of each time increment.

Intervals. The number of intervals.



Nesting Field L1

Level 1

Eaching nesting field should be STRING(1). Each style field should be a LONG immediately after the STRING.

Advanced – Nesting

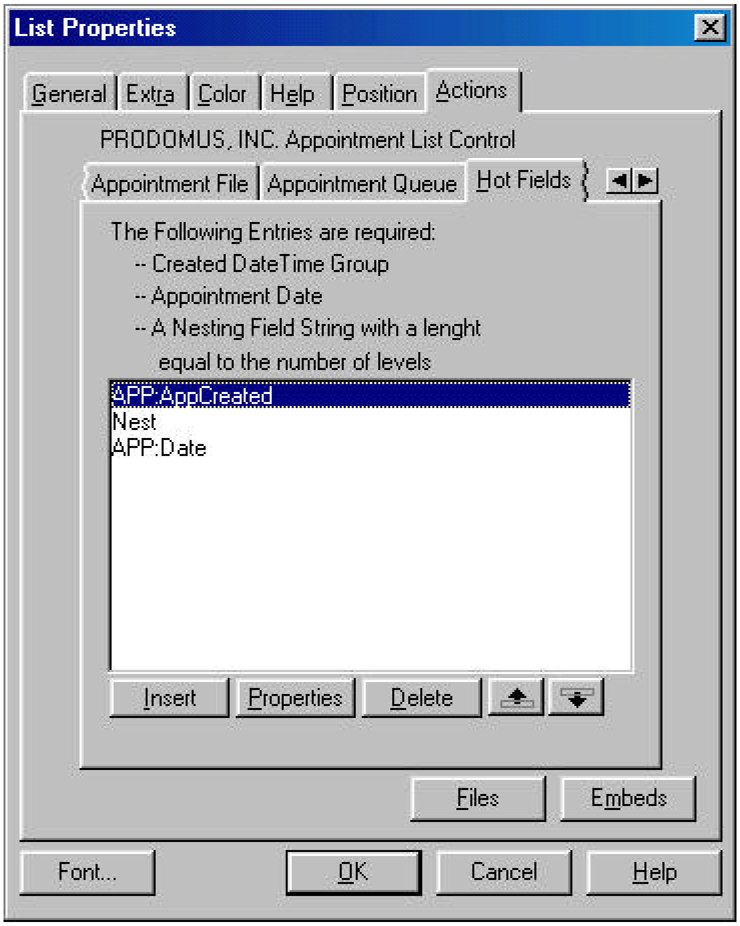
Enable Entry. Enables nesting brackets showing time durations.

Nest String. A String equal to or greater in length than the number of nesting levels to be displayed.

Nesting Fields Button. Each nesting field must be entered.

Nesting Field. The nesting field to display brackets. This should be a STRING(1)

Level. The level associated with the field.



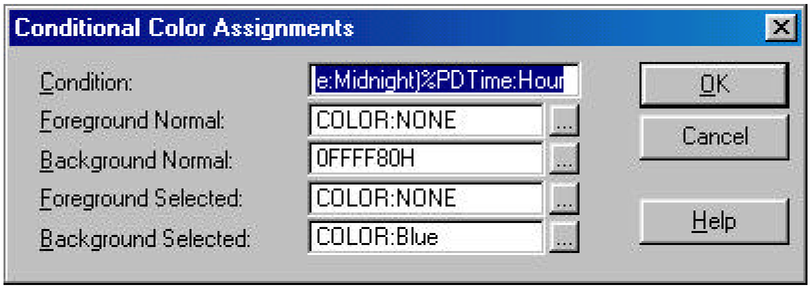
The Following Entries are required:
-- Created DateTime Group
-- Appointment Date
-- A Nesting Field String with a lenght
equal to the number of levels

APP:AppCreated
Nest
APP:Date

Hot Fields Tab

NOTE: that the following entries are required. This fields do not do not have to be displayed, but must be present in the queue:

- Created DateTime Group
- Appointment Date
- A Nesting Field String with a length equal to the number of levels



Condition: e:Midnight)%PDTime:Hour
Foreground Normal: COLOR:NONE
Background Normal: 0FFFF80H
Foreground Selected: COLOR:NONE
Background Selected: COLOR:Blue

Color, Icon, and Classes Tabs

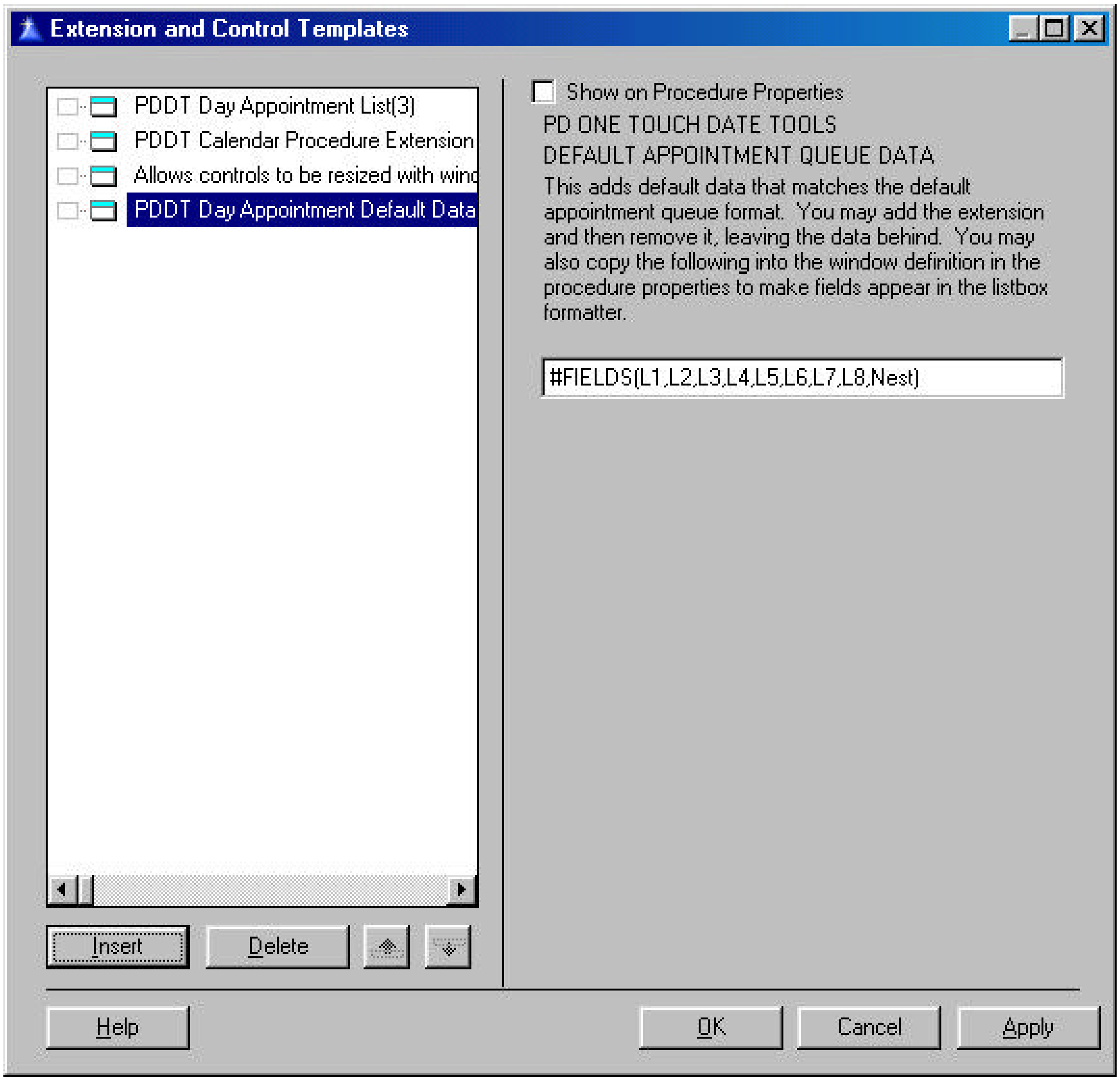
These tabs have the same standard entries as found in the Clarion templates.

The example appointment application uses a filter to color times that occur on the hour:

NOT (pdAppQ.Time-PDTime:Midnight)%PDTime:Hour

PDDT Day Appointment Default Data Extension Template

This unusual template creates default local data that is consistent with the default format created by the Day Appointment List Control Template. The template may be populated and then removed to allow modification of the data. It also includes a #FIELD listing that can be pasted into the list format in window properties of the appointment procedure. This links the fields to the listbox format.



Extension and Control Templates
PDDT Day Appointment List(3)
PDDT Calendar Procedure Extension
Allows controls to be resized with winc
PDDT Day Appointment Default Data
Insert
Delete
Help
OK
Cancel
Apply

☐ Show on Procedure Properties

PD ONE TOUCH DATE TOOLS

DEFAULT APPOINTMENT QUEUE DATA

This adds default data that matches the default appointment queue format. You may add the extension and then remove it, leaving the data behind. You may also copy the following into the window definition in the procedure properties to make fields appear in the listbox formatter.

#FIELDS(L1,L2,L3,L4,L5,L6,L7,L8,Nest)

It populates 8 strings for use as bracket fields plus a Nest field. To use this template:

1. Populate the Appointment List Control Template
2. Populate the extension template.
3. Copy the default #FIELD list.
4. In the Window properties, select the Window Elipsis button.
5. Paste the copied #FIELD entries at the end of the LIST definition.

```
LIST,AT(5,18,406,150),USE(?AppQList),VSCROLL,ALRT(EnterKey),ALRT(MouseLeft2),FORMAT('
4L*Y@s1@4L*Y@s1@4L*Y@s1@4L*Y@s1@4L*Y@s1@4L*Y@s1@4L|*Y@s1@32R(1)|_*~Time~' &|

'C@t7b@[36R(1)|_*~Date~C(0)@d17b@32R(1)|_*~Time~C@t7b@]|~Ending~43L(1)|_*~Type~C(' &|
                 '0)@s10@83L(1)|_*~Last
Name~C(0)@s30@320L(1)|_*~Description~@s80@'),FROM(pdAppQ),#SEQ(3), |
#ORIG(?AppQList),#FIELDS(L1,L2,L3,L4,L5,L6,L7,L8,APP:Time,APP:EndDate,APP:EndTime,APP
:Type,PAT:LastName,APP:Description)
```

Global Data

Equates

Locale Equates

See PDLocale.inc

Recurrance Flags

```
EveryWeek                    EQUATE( 01h)
EveryTwoWeeks                EQUATE( 02h)
EveryMonth                   EQUATE( 04h)
EveryxMonth                  EQUATE( 08h)
EverySemiMonth               EQUATE( 10h)
EveryBiMonth                 EQUATE( 20h)
EveryBiXMonth                EQUATE(040h)
EveryQuarter                 EQUATE(080h)
EveryXQuarter                EQUATE(100h)
EveryYear                    EQUATE(200h)
EveryXYear                   EQUATE(400h)
```

First Week Rules

```
WK_FirstOfFirst              EQUATE(0)
WK_FirstFullWeek             EQUATE(1)
WK_FirstHalfWeek             EQUATE(2)

!First Day Rules
DAY_Sunday                   EQUATE(6)
DAY_Monday                   EQUATE(0)
DAY_Tuesday                  EQUATE(1)
DAY_Wednesday                EQUATE(2)
DAY_Thursday                 EQUATE(3)
DAY_Friday                   EQUATE(4)
DAY_Saturday                 EQUATE(5)
```

Date Group

```
PDDT_GType              GROUP,TYPE
FirstWeekRule               BYTE
FirstDayRule                BYTE
FirstBusDOW                 BYTE
```

```
LastBusDOW                  BYTE
MonthStr                    STRING(156)
MonthArray                  STRING(13),DIM(12),OVER(MonthStr)
MonStr                      STRING(156)
MonArray                    STRING(13),DIM(12),OVER(MonStr)
DayStr                      STRING(91)
DayArray                    STRING(13),DIM(7),OVER(DayStr)
AbbrevDayStr                STRING(91)
AbbrevDayArray              STRING(13),DIM(7),OVER(AbbrevDayStr)
                        END
```

Frequently Asked Questions

Questions and Answers

Can the date tools be used with SQL which requires DATE and TIME data types?

Yes. All date and time fields must be consistently declared, i.e. all must be DATE and TIME data types. To use DATE and TIME data types with the schedule class, select PDSchedDCT as the class to use on the Date Tools Global Classes Tab. This overrides the Schedule Class which otherwise uses LONG data types. This allows the use of DateTime group fields with DATE/TIME definitions.

The other classes (Date Class, Scrolling Date Class, and Calendar Classes) support both LONG and DATE/TIME data types without changing the Schedule Class.

How do I disable a calendar drop button?

The drop buttons will be disabled and hidden by the class library according to the entry properties. The buttons also use the skip attribute. If for some reason you need the equate, the button equate is set the to entry equate plus 2500. The drop list is set to the entry equate plus 1500.

Right now, all we want to do is provide the user with a main menu option calendar (for reference only); and then popup's at various places to select a date for transfer into a date field. We would like the calendar to show weekends and US holidays color coded so that the user has visual cues when making the selection from the popup's. What's the easiest and least time consuming way for us to get this simple calendar done? Where do we look for instructions/guidance/examples?

1. To add the calendar to the Menu,
    a. Add the calling menu item to to the window.
    b. Select Extensions from the procedure properties window.
    c. Highlight the PD Calendar Procedure Extension template and select the Popup Tab.
    d. Check Call popup calendar from control.
    e. Enter the Calling Control created in step 1a.
    f. Check open as MDI window (if you want it threaded). This will enable drag and drop to any date entry field.
    g. Check Center Window (probably preferred here).
2. All the date entry controls will have drop calendars if this is selected from the global template. If you prefer the popup calendar, you can change the One Setp Set Up tab entry Class To Use to “Scrolling Entries.” You will then need to use the Popup Calendar Control template to populate buttons for each entry. You could also not populate buttons, but use the global template to drag and drop entries. The global template, however, will not have any range limits unless you do some additional coding.
3. Colors in drop calendars and popup calendars are red by default as is the inclusion of US holidays.

I would like to commence the weeks with Sunday. I have glimpsed at the code and you seem to get a date group.

The class initializes a date group in the library. You may change any element of this group at run time by creating a group, getting the current group values from the library, changing the element, and then passing the changed group back to the library. Steps are:

1. Add a globally or locally defined date group.
   ```
   GDateGroup          PDDateGT
   ```
2. Get the current date group from the library.
   ```
   PDDateC.GetDateGroup(gDateGroup)
   ```
3. Modify the date group element.
   ```
   GDateGroup.FirstDayRule=DAY_Sunday  ! Use equate in PDDate.inc
   ```
4. Pass the group back to the Class:
   ```
   PDDateC.SetLocale(gDateGroup)
   ```

Note that this will not only change the display of the first day of the week, but it will also the week of the ear calculation.

I would like to create my own popup calendar without the threading and some of my own features. What is the best approach.

The DTX.app example does this except that it is designed as a toolbox rather than a popup calendar. Some general steps are:

1. Create a window procedure.
2. Add (LONG P:Date),LONG as the Prototype and Parameter.
3. Add a local variable to hold the date value while the calendar is display. L:Date, LONG.
4. Populate a Calendar List control.
5. Enter L:Date as the entry field.
6. Add additional buttons as needed including an OK button.
7. Use the AddButton method to add the buttons to the Calendar List Class.
8. Set the return value to L:Date if the OK button is pressed.

I would like to change the locale at run time. How do I do this?

Use the SetLocale method. In 32-bit, you can pass an equate found in PDLocale.inc. See code samples in Sample Date Class Procedures.

```
EXECUTE LocaleChoice
  PDDateC.SetLocale(LOC_FrenchCanada)
  PDDateC.SetLocale(LOC_EnglishCanada)
END
```

The drop list and button are not properly resized if the window is resized.

Use the resize class SetStragegy method to set the resize strategy for the controls. The control Equate number will be the entry plus the following equates. Locate the code in the Resizer.init method.

```
PDDROPCAL:eButtonFeq EQUATE(1500)
PDDROPCAL:eListFeq   EQUATE(2500) plus

Example:
SELF.SetStrategy(?pdDayDate+PDDropCal:eButtonFEQ, Resize:FixLeft+Resize:FixTop,
Resize:LockSize)
SELF.SetStrategy(?pdDayDate+PDDropCal:eListFEQ, Resize:FixLeft+Resize:FixTop,
Resize:LockSize)
```

Known Issues

Drop Calendar

1. Display of Overlapping Controls. Lists that overlap other controls normally continue to display the underlying control. To stop the display, the Drop Calendar disables most overlapping controls. Tabs are exceptions – to properly display, there must be space on the tab for the list to either drop down or up within the confines of the list. Some controls also continue to display through the list – no consistent reason has been found for this behavior. The AddHide(?FieldEquate,TRUE) method can be used to hide these controls when the calendar drops. Controls can be added using Calendar Procedure Extension control overrides. See PD Calendar Procedure Extension.
2. Conflict with Timer. Underlying controls were found to display with timer events occurring on the application frame. The drop controls now toggle the timer off while the drop is displayed and then back on when it is hidden. This uses the EVENT:TimerXO defined in PDCal.inc.
3. Conflict with Tool Tips. When a tool tip is displayed over a dropped calendar, the underlying control displays within a dropped list when the area is repainted. Tool tips on the drop calendar have therefore been disabled. They still display in the entry itself.

Sample Applications

Two sample applications are provided:

1. Events.App. Makes simple use of the date tools: the scrolling date function, calendar button, and time zone function.
2. Dtx.App. Illustrates the use different uses of the date tools and calendars. This has two elements:
    a. Simple sample implementations the global extension, procedure extension, and control templates: ShoDrop, ShoList, ShoPopCalButton, ShoScroll, ShoScroll Popup.
    b. A more complex scheduling example. This uses a heaer file SchHD, a schedule type, and two child files, Schedule, and Holiday.
        i. Files
            1. SchHD File. This file contains a name for the schedule, the schedule’s locale, and a notes entry. When a schedule is selected, it changes the Date Class locale to the locale in this file.
            2. Schedule File. This file contains both recurring and individual scheduled items.
            3. Holiday File. This file contains holidays and specifies their recurrance.
        ii. Procedures
            1. BrowseSHD and UpdateSHD. This browses the header files and sets the currently active schedule.
            2. BrowseSchedule and UpdateSchedule. This lists and updates schedule entries.
            3. BrowseHoliday and UpdateHoliday. This lists and updates Holiday entries.
            4. ProcessHolidays. This adds Holidays to the Date Class using the AddHoliday method.
            5. ProcessSchedule. This adds the Scheduled items to the DateClass using the AddSchedule method.
            6. ToolBarCalendar. This procedure is a Calendar List Control Template this displays the current schedule and holidays. It has the following elements:
                a. The names of the holidays and first scheduled item display at the bottom of the window.
                b. If there are multiple scheduled items for a given date, file drops are display.
                c. The display uses prompts and file drops that are hidden or unhidden depending on the number of scheduled items or holidays to display for the currently selected date.
                d. A popup button calling the browse and processing procedures.
                e. A wizard “Next/Back” button the displays a help screen. This actually uses groups which are hidden and displayed according to the current selectgion.
                f. A EVENT:Refresh which communicates with the Frame procedure to update the calendar when changes are made using the Frame’s toolbar button.

File List

See the file DIR.TXT for a list of files and their locations.

Warranty

ProDomus, Inc. warrants the physical installation file(s) and physical documentation provided by ProDomus to be free of defects in materials and workmanship for a period of ninety days from the date of purchase. If ProDomus, Inc. receives notification within the warranty period of defects in materials or workmanship, and such notification is determined by ProDomus, Inc. to be correct, ProDomus, Inc. will replace the defective files (s) or documentation.

The entire and exclusive liability and remedy under this Limited Warranty shall be limited to replacement of defective file(s) or documentation and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if ProDomus has been specifically advised of the possibility of such damages. In no event will ProDomus, Inc.'s liability for any damages to you or any other person ever exceed the lower of suggested list price or actual price paid for the license to use the software, regardless of any form of the claim.

PRODOMUS, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.

License Agreement

This is a legal agreement between you (either an individual or entity) and ProDomus, Inc. By using PD Date Tools, consisting of various electronic files ("Software") , you agree to the terms of this Agreement. If you do not agree to the terms of this Agreement, promptly remove the Software from any computers where you installed the Software.

This License Agreement ("License") permits you to use one copy of the specified version of the ProDomus software product identified above on a single computer. If the anticipated number of users of the Software will exceed the number of applicable licenses, then you must have a reasonable mechanism or process in place to ensure the number of persons using the software concurrently does not exceed the number of Licenses.

ProDomus gives you the right to reproduce and distribute the run-time modules of the Software provided that you: (a) distribute the run-time module only in conjunction with and as part of your software product; (b) do not use the ProDomus name, logo, or trademark to market your software product; (c) include a valid copyright notice on your software product; and (d) agree to indemnify, hold harmless, and defend ProDomus and its suppliers from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of your software product.

ABOUT

PD Date Tools

16 Owen Street
Hartford, CT 06105

Tel: (860) 523 4032 or (860) 233 0705

Facsimile: (860) 523 4032 or 860 236 7954
pwill@prodomus.com (email only)
pdService@home.com (any files)

Web Site

www.prodomus.com

Other ProDomus products for Clarion for Windows Developers:

CW Translator and CWIntl

International Toolkits for dynamically handling national conventions, language, and int'l environment without the need to recompile.

CWTranslator -- Replace strings through string lookups. Maximum performance.

CWIntl -- Replace strings by control and user. Maximum control.

Better Lookups

PD Lookup -- A Better Browse Button String Lookup

PD Drops -- Better Drop List and Drop Combo

PD Address

International Address Tool Kit for flexible address layout

CWFin

Finance Tool Kit