TDate   TFIELD.H tdate.cpp

 ____
 | TDate |

*** NOTE ***
This class is derived from the DATECL.ZIP file found in the C++ section (lib 6) of the BPROGB forum on compuserve. I have just added the setDate functions and the coversion to PXDate functions. The rest of the class hasn't changed (much, if at all). I haven't found any problems with it, but I haven't attempted to verify all the algorithm's, either!
****

Data Members

day_of_week   int day_of_week   PROTECTED

Holds the current day of the week: Sunday = 1 ... Saturday = 7, or 0 if an invalid date is specified.

displayFormat   static int displayFormat   PRIVATE

Specifies the format used to display the date. Current enumerated values for displayFormat are MDY, FULL or EUROPEAN. The values DAY and MONTH are also provided, but are not supported by the setDate string parser.

See Also: TDate::setFormat

displayOptions   static unsigned char displayOptions PRIVATE

Specifies options which modify the format used to display the date. Current options are NO_CENTURY -- Suppress the printing of the century when in the MDY format (ex: 01/01/91 instead of 01/01/1991). DATE_ABBR -- Abbreviate month and day names when printing in the MONTH, DAY, FULL or EUROPEAN formats. (ex. MON, TUE, JAN, FEB, etc.) The length or the abbreviation is controlled by a DEFINED constant in TFIELD.H named ABBR_LENGTH, preset to 3.

See Also: TDate::setOption

julian   unsigned long julian PROTECTED

Holds the count of the number of days that have passed since 1/1/4713 B.C.

tdate_day   int tdate_day   PROTECTED

Holds the current day of the month, or 0 if an invalid date is specified.

tdate_month   int tdate_month   PROTECTED

Holds the current month, or 0 if an invalid date is specified.

tdate_year   int tdate_year   PROTECTED

Holds the current year, or 0 if an invalid day is specified.

Member Functions

constructors TDate ();

Initializes displayFormat to MDY. Initializes day, day_of_week, displayOptions, julian, month and year to 0.

TDate (const long aJulDate, const long offset = 0);

Initializes displayFormat to MDY and displayOptions to 0. Sets julian to aJulDate + offset. PXOffset is the only offset currently support. This allows a TDate to be initialized by calling TDate(PXDate, PXOffset). Initializes day, day_of_week, month and year to coincide to the julian date.

TDate (const int m, const int d, const int y);

Initializes displayFormat to MDY and displayOptions to 0. Sets the initial day, month and year from the values provides. Computes julian for the specified date and computes the day_of_week for the date.

TDate (char *dat);

Initializes displayFormat to MDY and displayOptions to 0. Calls setDate to attempt to parse the dat. If successful, day, day_of_week, julian, month and year are all initialized to the specified date.

TDate (const date &ds);

Initializes displayFormat to MDY and displayOptions to 0. Copies the day, month and year from the DOS date structure and sets julian and day_of_week for the specified date.

TDate (const TDate &dt);

Copies the day, displayFormat, displayOptions, month and year from the specified TDate. Calculates the julian and day_of_week for the specified date.

char_to_month int char_to_month(const char *charMonth); PRIVATE

Converts month name to month value. Only the first two or three characters of the charMonth are checked for a match. Matching values are JA FE MAR AP MAY JUN JUL AU SE OC NO DE. If the value doesn't match, a 0 is returned.

day int day(void) const;

Returns the value of the day for the date.

dayOfYear int dayOfYear(void) const;

Returns relative date since Jan. 1

decrMonth void decrMonth(void);

Decrements the month by 1. Adjusts the year and day if necessary.

decrYear void decrYear(void);

Decrements the year by 1. Adjusts the day if necessary.

dow	int dow(void) const;

Returns the value of the day_of_week for the date.

eom	date eom(void) const;

Returns last day of month in object. ***NOTE*** This function returns a date not a TDate!!!

formatDate	char *formatDate (int type=-1) const;

Return a string value for the current date in the specified format. If type = -1, the format specified in displayFormat will be used. Other valid types are MDY, FULL and EUROPEAN. Types MONTH and DAY are valid but not be used to generate a new date from the returned string, as insufficient information is present for the setDate(char *) parser.

**** NOTE ****
This function returns a newStr'd string pointer. Storage has been allocated for this string from the heap. To avoid memory leaks, make sure to delete the string after you're done using it!

getDate	date getDate(void) const;

Returns a date structure. ***NOTE*** This function returns a date not a TDate!!!

incrMonth	void incrMonth(void);

Increments the month by 1. Adjusts the year and day if necessary.

incrYear	void incrYear(void);

Increments the year by 1. Adjusts the day if necessary.

isLeapYear	int isLeapYear(void) const;

Returns 1 if leap year, 0 if not

julDate	long julDate(void) const;

Returns julian date.

julian_to_mdy	void julian_to_mdy (void);	PRIVATE

convert julian day to mdy

julian_to_wday	void julian_to_wday (void);	PRIVATE

convert julian day to day_of_week

mdy_to_julian	void mdy_to_julian (void);	PRIVATE

convert mdy to julian day

month	int month(void) const;

Returns the value of the month for the date.

operators

\+ TDate operator + (const long i);

Returns the value date + i days.

\- TDate operator - (const long i);

Returns the value of date - i days.

\- long operator - (const TDate &dt);

Returns the number of days between two dates.

+= TDate &operator += (const long i);

Adds i days to the date.

-= TDate &operator -= (const long i);

Subtracts i days from the date.

++ TDate &operator ++ (void);

pre increment function

++ TDate &operator ++ (int);

post increment function

-- TDate &operator -- (void);

pre decrement function

-- TDate &operator -- (int);

post decrement function

< friend int operator < (const TDate &dt1, const TDate &dt2);

Returns 1 if dt1 is less than dt2, returns 0 otherwise.

<= friend int operator <= (const TDate &dt1, const TDate &dt2);

Returns 1 if dt1 is less than or equal to dt2, returns 0 otherwise.

> friend int operator > (const TDate &dt1, const TDate &dt2);

Returns 1 is dt1 is greater than dt2, returns 0 otherwise.

>= friend int operator >= (const TDate &dt1, const TDate &dt2);

Returns 1 if dt1 is greater that or equal to dt2, returns 0 otherwise.

== friend int operator == (const TDate &dt1, const TDate &dt2);

Returns 1 if dt1 is equal to dt2, returns 0 otherwise.

!= friend int operator != (const TDate &dt1, const TDate &dt2);

Returns 1 if dt1 is not equal to dt2, returns 0 otherwise.

<< friend ostream &operator << (ostream &os, const TDate &dt);

Places the character date on the output stream in the format MM/DD/YY.

<< friend ostream &operator << (ostream &os, const date &dt);

Places the character date on the output stream in the format MM/DD/YY.

PXDate long PXDate(void) const;

Returns date in paradox format (i.e. julian - PXOffset)

setDate int setDate (const char *charDate);

setDate will attempt to translate the charDate into a valid date format. charDate is first parsed into seperate tokens by calling TParser::parse. The tokens are then scanned to see if a valid character month has been specified (JA FE MAR AP MAY JUN JUL AU SE OC NO DE). The tokens are then rescanned to interpret any numbers into month, day and year values in that order. If a day is not specified, the first will be assumed. If a year is not specified the current year is assumed. The strings "TODAY" or "*" will return the current computer date. Returns 1 if successful, 0 if not. (Parsing does not always return the desired date. If anyone has suggestions for ways to improve this process, please let me know. This was the best I could come up with. MBB)

setDate int setDate (const long aJulDate, const long offset = 0);

Set the date to the value specified in aJulDate + offset. PXOffset is the only offset which is currently defined. To set the date from a date in paradox format, call setDate(PXDate, PXOffset). Returns 1 if successful, 0 if not.

setFormat static void setFormat (const int format);

Specify the default format to be used in displaying the date. Possible values are MDY, FULL, EUROPEAN. To other values are provided, DAY and MONTH, but are not supported by the setDate(char *) parser.

setOption static int setOption (const int option, const int
action=ON);

Specify the display options to be used in displaying the date. Possible values are NO_CENTURY and DATE_ABBR. Action can be ON or OFF to enable or disable the option.

year int year(void) const;

Returns the value of the year for the date.

TDateFieldTFIELD.H tdfield.cpp

```
  ___________
  | TField |
  ----------
      I
 _______I________
 | TDateField |
 --------------
```

TDateField provides the means for manipulating dates in dialogs and windows. Upon entering a TDateField, the current data string is selected. While the data string is completely selected, the + key will increment the date one day, the - key will decrement the date one day, the <PageDown> key will increment the date's month by 1 and the <PageUp> key will decrement the date's month by 1, unless the minimum or maximum date boundaries would be crossed (if enabled.)

Data Members

maxValue TDate maxValue

Maximum allowable date if tfValidateMax is enabled.

minValue TDate minValue

Minimum allowable date if tfValidateMin is enabled.

value TDate value

Current date value for this field.

Member functions

constructor TDateField(const TRect& bounds, int aMaxLen);

convertData virtual void convertData(void);

Attempts to convert the char string data to a date by calling value.setDate(data). If successful, data is updated to the current displayFormat for the date and the window is redrawn.

dataSize virtual ushort dataSize();

Returns the size of the date storage in memory (sizeof(long)) in the current implementation.

filterCharCode virtual ushort filterCharCode(ushort charCode);

getData virtual void getData(void *rec);

isValid	virtual Boolean isValid(ushort command);
processKeyCode	virtual ushort processKeyCode(ushort keyCode);
setData	virtual void setData(void *rec);
setJulian	void setJulian(long julDate);
	Set the date value from a julian date (1/1/4713 BC offset)
setMax	virtual void setMax(TDate newMax);
	Set the maxValue to newMax and enable the tfValidateMax option.
setMin	virtual void setMin(TDate newMin);
	Set the minValue to newMin and enable the tfValidateMin option.
setParadox	void setParadox(long pxDate);
	Set the date value from a date in Paradox format. (1/1/1 AD offset)

TFieldTFIELD.H tfield.cpp

```
 __________
 | TView |
 ---------
     |
 _____|______
 | TField |
 ----------
```

A TField object provides a basic input line string editor. It handles keyboard input and mouse clicks and drags for block marking and a variety of line editing functions (see TField handleEvent). The selected text is deleted and then replaced by the first text input.

The tfOptions data member controls various options based on which bits are set. The current tfOptions supported are:

tfRight -- the field is right justified when this bit is set
tfNoScroll -- scrolling is disabled when this bit
tfHideSecure -- masks out the data with securityChar if this bit is set and the field's fieldSecurity < currentSecurity
tfMustFill -- displays an error message when leaving the field if the field does not contain aMaxLen - 1 characters. Also returns a False from the valid routine so the TDialog/TWindow will not be able to close.
tfNotEmpty -- checks that the field contains at least 1 character when leaving the field, if it doesn't an error message is displayed and valid returns False. The error message from tfMustFill overrides this error message if tfMustFill is enabled.
tfBeepError -- controls whether a beep sounds on data entry errors. Enable tfBeepError to get a beep.
tfEnterTabs -- controls whether the enter key should move the cursor to the next field or not. Enable tfEnterTabs for the enter key to advance the focus to the next field.
tfStayError -- controls whether the field will lose the focus when it contains invalid data (or doesn't meet the requirements of tfMustFill or tfNotEmpty.) If this bit is enabled, the keyboard control keys (tab/shiftTab/enter) will not allow the focus to leave the field while it contains invalid data. ***NOTE*** this version DOES NOT stop mouse events from causing the field to lose its focus. The lostFocus routine will still be called for mouse events, but the way the current TView::Select function operates precludes trapping all events unless the TView source is modified (requested of Borland, but beyond the scope of the design specifications for this class.)
tfPreValidate-- controls whether the isValid function is called before or after the "standard" validations are performed. By default, isValid will be called after the length (and min/max in derived classes) validations have occurred successfully. Setting this flag forces the isValid function to be called before the other validations have occurred.

The status of these bits may be directly changed by the programmer, but using the setTFOptions member function insures that all views are appropriately redrawn as required by the changes to tfOptions.

The following tfOption flags are reserved for future use in derived classes:

tfValidateMin
tfValidateMax
tfPopUp

Field security is controlled by two member variables: fieldSecurity (the security level for this field) and currentSecurity (the applications current authorized security level.) The cmUpdateSecurity broadcast message can be used to update the currentSecurity level of any fields derived from TField (not completely functional at rev 1.0.1). Alternatively, the setCurrentSecurity member function can be called to update individual field's currentSecurity levels.

Two character fields (and a tfOptions flag) control the security of displayed data. hideChar will always replace the displayed data with strlen(data) copies of itself (ie. if data = "hello" and hideChar = '*' then "*****" will be displayed) when hideChar > ' '. securityChar contains the character that will replace the displayed data only if tfHideSecure is enabled and fieldSecurity < currentSecurity. This allows for dynamic control of what fields will display data based on a user's security level.

The getData and setData member functions are available for writing and reading data strings (referenced via the data string data member) into the given record. TField::setState simplifies the redrawing of the view with appropriate colors when the state changes from or to sfActive and sfSelected.

Field validation can be accomplished using a combination of virtual functions. filterCharCode can be used to filter out unacceptable characters for a field. convertData can be used for custom data conversion from string data values to other internal types. convertData is called before any valid ation is performed. isValid is called from the valid routine an may be used to perform special validations tasks (eg. table lookups, etc.) either before or after the standard validations have been performed for the field. Finally the gotFocus and lostFocus routines can be used to perform various tasks upon entering/leaving the field.

Data members

curPos int curPos;

Index to insertion point (that is, to the current cursor position).

See also: TField::selectAll

currentSecurity ushort currentSecurity;

Holds the current security level for the form (or program or user, etc.) This field is used in conjunction with tfHideSecure option to determine if secure data should be masked or not.

See also: TField::fieldSecurity, TField::securityChar, TField::setCurrentSecurity, cmUpdateSecurity

data char *data;

The string containing the edited information.

fieldHint char *fieldHint;

Pointer to the hint text for this field. (Not used in this implementation. In the future though...)

See also: TField::setHint

fieldName char *fieldName;

Pointer to the name text for this field.

See also: TField::setName

fieldSecurity ushort fieldSecurity;

Holds the security level for the current field. If the tfHideSecure option is enabled, then the data for the field is masked if fieldSecurity < currentSecurity.

See also: TField::currentSecurity, TField::SecurityChar, tfHideSecure

firstPos int firstPos;

Index to the first displayed character.

See also: TField::selectAll

hideChar char hideChar;

Character to use to hide the field's display. If hideChar < ' ' (space) then the field's data is displayed, if hideChar >= ' ' then the field's data is masked by the hideChar.

maxLen int maxLen;

Maximum length allowed for string to grow (excluding the final 0). This value is initialized to aMaxLen - 1 in the TField constructor.

See also: TField::dataSize

securityChar char securityChar;

Holds the character to be used as a mask if tfHideSecure is enables and fieldSecurity < currentSecurity.

See also: TField::fieldSecurity, TField::currentSecurity, TField::tfOptions, tfHideSecure

selEnd int selEnd;

Index to the end of the selection area (that is, to the last character block marked).

See also: TField::selectAll

selStart int selStart;

Index to the beginning of the selection area (that is, to the first character block marked).

See also: TField::selectAll

tfOptions ushort tfOptions;

Bitmapped flag field for various field control states.

See also: tfRight, tfNoScroll, tfHideSecure, tfValidateMax, tfValidateMin, tfMustFill, tfNotEmpty, tfBeepError, tfEnterTabs, tfStayError, tfPopUp, tfPreValidate and TField::setTFOptions

Member functions

constructor TField(const TRect& bounds, int aMaxLen);

Creates an input box control with the given values by calling TView(bounds). state is then set to sfCursorVis, options is set to (ofSelectable | ofFirstClick), and maxLen is set to aMaxLen - 1. Memory is allocated and cleared for aMaxlen bytes and the data data member set to point at this allocation.

**** NOTE ****
Be sure to note that aMaxLen includes the terminating \0 character. For a TField which allows the user to type in 10 characters, use an aMaxLen of 11.

constructor TField(StreamableInit streamableInit); protected

Each streamable class needs a "builder" to allocate the correct memory for its objects together with the initialized vtable pointers. This is achieved by calling this constructor with an argument of type StreamableInit. Refer also to Chapter 8.

See also: TView::TView, sfCursorVis, ofSelectable, ofFirstClick

destructor ~TField();

Deletes the data, fieldHint and fieldName memory allocation, then calls ~TView to destroy the TField object.
See also: ~TView

build static TStreamable *build();

Called to create an object in certain stream-reading situations.

See also: TStreamableClass, ipstream::readData

convertData virtual void convertData(void);

Called from the valid routine to convert from character data to specialized data types (eg. TDate, long, etc.) in derived classes.

See alse: TField::valid

dataSize virtual ushort dataSize ();

Returns the size of the record for TField::getData and TField:: setData calls. By default, it returns maxLen + 1. Override this member function if you define descendants to handle other data types.

See also: TField::getData, TField::setData

draw virtual void draw();

Draws the input box and its data. The box is drawn with the appropriate colors depending on whether the box is sfFocused (that is, whether the box view owns the cursor), and arrows are drawn if the input string exceeds the size of the view (in either or both directions). Any selected (blockmarked) characters are drawn with the appropriate palette.

filterCharCode virtual ushort filterCharCode(ushort charCode)

Used to limit what characters can be input into a field. Default allow all characters in. If the charCode should not be allowed, a value of 0 should be returned. If a value of 1 is returned, the equivalent of a selectAll(true) will be performed. Values >= ' ' will be added to or overwrite the data item.

getData vlrtual void getData (void *rec);

Writes the number of bytes (obtained from a call to dataSize) from the data string to the array given by rec. Used with setData for a variety of applications; for example, temporary storage, or passing on the input string to other views. Override getData if you define TField descendants to handle non-string data types. You can also use getData to convert from a string to other data types after editing by TField.

See also: TField::dataSize, TField::setData

getPalette virtual TPalette& getPalette() const;

Returns the default palette string, cplnputLine, "\x13\x13\x14\x15".

gotFocus virtual void gotFocus(void);

Called when the field receives the cmReceivedFocus message (usually when the field receives the focus).

handleEvent void handleEvent(TEvent& event);

Calls TView::handleEvent, then handles all mouse and keyboard events if the input box is selected. This member function implements the standard editing capability of the input box.

Editing features include: block marking with mouse click and drag; block deletion; insert or overwrite control with automatic cursor shape change; automatic and manual scrolling as required (depending on relative sizes of the data string and size.x); manual horizontal scrolling via mouse clicks on the arrow icons; manual cursor movement by arrow, Home, and End keys (and their standard control-key equivalents); character and block deletion with Del and Ctrl-G. The view is redrawn as required and the TField data members are adjusted appropriately.

See also: sfCursorlns, TView::handleEvent, TField::selectAll

isValid virtual Boolean isValid(ushort command);

Called by the valid function. If False is returned to valid, valid will always fail. If command = cmArriving, validation should be performed based on the field being entered (ie. is the field dependent on other events/values). If command = cmLeaving, validation should be performed based on the field being left.

lostFocus virtual void lostFocus(void);

Called when the field receives the cmReleasedFocus message (usually when the field is losing focus.)

processKeyCode virtual ushort processKeyCode(ushort keyCode);

This routine is called if the default handle event didn't deal with the keyCode. It allows users to easily process non-default kbxxxx codes in derived classes without overriding the handle event routine.

See also: TField::handleEvent, TField::filterCharCode

read virtual void *read(ipstream& is);

Reads from the input stream is.

See also: TStreamableClass, TStreamable, ipstream

selectAll void selectAll (Boolean enable);

Sets curPos, firstPos, and selStart to 0. If enable is set to True, selEnd is set to the length of the data string, thereby selecting the whole input line; if enable is set to False, selEnd is set to 0, thereby deselecting the whole line. Finally, the view is redrawn by calling drawView.

See also: TView::drawView

setCurrentSecurity virtual void setCurrentSecurity(ushort newSecurityLevel);

Called to update the current security level. If the new security level > fieldSecurity and the tfHideSecure is enabled, then the field contents will be masked by the securityChar character. This routine is also called when the cmUpdateSecurity broadcast is received.

setData virtual void setData (void *rec);

By default, copies the number of bytes (as returned by dataSize) from the rec array to the data string, and then calls selectAll(True). This zeros curPos, firstPos, and selStart. Finally, drawView is called to redraw the input box.

Override setData if you define descendants to handle non-string data types. You also use setData to convert other data types to a string for editing by TField.

See also: TField::dataSize, TField::getData, TField::selectAll

setHideChar virtual void setHideChar(char aChar);

This routine is used to update the hideChar data member. If the new hideChar is > ' ' then the field's data will be masked by the hideChar character.

setHint virtual void setHint(char *aString);

Sets this field's fieldHint string to aString. The hint options currently aren't implemented in this class.

setName virtual void setName(char *aString);

Sets this field's fieldName string to aString.

setTFOptions virtual void setTFOptions (ushort aOption, Boolean enable);

Called to change the state of various tfOption flags. tfOption flags control the justification, scrolling, security, validation and enter handling for fields.

See also: tfRight, tfNoScroll, tfHideSecure, tfValidateMax, tfValidateMin, tfMustFill, tfNotEmpty, tfBeepError, tfEnterTabs, tfStayError, tfPopUp and TField::tfOptions valid virtual void valid(ushort command);

Called to validate the contents of the field. If command = cmValid, we're being called after the constructor. If command = cmArriving, we're being called before the field is entered. If command = cmLeaving, we're being called before the field is left. Currently performs standard validations (ie. string length checks) when cmLeaving. Also calls the user supplies isValid function.

write virtual void write(opstream& os);

Writes to the output stream os.

See also: TStreamableClass, TStreamable, opstream

Related functions Certain operator functions are related to TField but are not member functions; see page 232 for more information.

Palette

TFields use the default palette, cpField, to map onto the 19th through 21st entries in the standard dialog palette.

```
           1     2     3     4
        ===========================
cpField  || x13 | x13 | x14 | x15 ||
        ===========================
 Passive---^     |     |     ^---Arrow
 Active-----------     ----------Selected
```

TFloatFieldTFIELD.H tffield.cpp

```
     ___________
     | TField |
     ----------
         |
  ________|________
  | TFloatField |
  ---------------
```

TFloatField provides support for floating point numeric entries. The current implementation of TFloatField allows the +/- and PgUp/PgDn keys to increment/decrement the value when the field is selected (selectAll(True)). This +/- will increment/decrement by a value of 1, the PgUp/PgDn by a value of 10.

Data members

value   float value

Holds the fields current value

maxValue   float maxValue

Holds the maximum value to be checked if tfValidateMax is set

minValue   float minValue

Holds the minimum value to be checked if tfValidateMin is set

Member functions

char *formatString; // field's format string for sprintf function call

constructor   TFloatField(const TRect& bounds, int aMaxLen);

Calls the TField constructor and initializes fieldType to TFloatType.

destructor   ~TFloatField();

Release any storage allocated for the formatString.

convertData   virtual void convertData(void);

Converts the string in data to a floating point value. Calls sprintf to reformat the value back into data. Uses formatString for the conversion if it is defined, otherwise uses the default %.2f for the conversion.

dataSize   virtual ushort dataSize();

Returns sizeof(float);

filterCharCode   virtual ushort filterCharCode(ushort charCode);

Filters out all characters except the digits, minus and decimal point. If the entire field is selected, the +/- keys will increment/decrement the value by 1 respectively unless doing so would exceed a min/max boundary.

processKeycode virtual ushort processKeyCode(ushort keyCode);

If the entire field is selected, the kbPgUp/kbPgDn keys will add/subtract 10 from the current value unless doing so would exceed a min/max boundary.

setFormat virtual void setFormat(char * newFormat);

Specify a sprintf format string to be used to display the value.

setMax virtual void setMax(float newMax);

Sets maxValue to newMax and sets the tfValidateMax flag for the field.

setMin virtual void setMin(float newMin);

Sets minValue to newMin and sets the tfValidateMin flag for the field.

TDoubleFieldTFIELD.H tdbfield.cpp

```
-----------------------------------------------------------------------------
  __________
  | TField |
  ----------
      |
 _______|_______
 | TDoubleField|
 ---------------
```

TDoubleField's functionality is currently the same as TFloatField's. See the member functions of TFloatField for a description of the functionality. TDoubleField does use double values instead of floats, though.

TLongDoubleFieldTFIELD.H tldfield.cpp

```
-----------------------------------------------------------------------------
  __________
  | TField |
  ----------
      |
_________|__________
| TLongDoubleField |
--------------------
```

TLongDoubleField's functionality is currently the same as TFloatField's. See the member functions of TFloatField for a description of the functionality. TLongDoubleField uses long double values instead of floats.

TIntFieldTFIELD.H tifield.cpp

```
--------------------------------------------------------------------------------
    ___________
    | TField |
    ----------
        |
  _______|_______
  | TIntField |
  -------------
```

TIntField's functionality is almost exactly like TFloatField's. The only significant different is filterCharCode does not allow decimals for TIntFields, TIntFields have no formatString and TIntField uses Int values instead of Floats.

TLongFieldTFIELD.H tlfield.cpp

```
--------------------------------------------------------------------------------
    ___________
    | TField |
    ----------
        |
  _______|________
  | TLongField |
  --------------
```

TLongField's functionality is almost exactly like TFloatField's. The only significant different is filterCharCode does not allow decimals for TLongFields, TLongFields have not formatString and TLongField uses Long values instead of Floats.

TParserTFIELD.H tdate.cpp

```
 ______________________
 | TStringCollection |
 ---------------------
        |
   ______|______
   | TParser |
   -----------
```

TParser parses a character string, creating an unsorted TStringCollection of all the words/tokens contained in the string.

Member Functions	
Constructor	TParser(short aLimit, short aDelta)
	Calls the TStringCollection Constructor.
compare	virtual int compare(void *key1, void *key2)
	Returns a -1, causing strings to be added sequentially to the TStringCollection (duplicates are allowed.)
parse	void parse(char *line)
	Parses the line for words/tokens to add to the TStringCollection. The characters /-. will be appended to the end of any word they follow, if no whitespace intervenes, otherwise they are added as seperate tokens.

TUpperFieldTFIELD.H tufield.cpp

```
 ___________
 | TField |
 ----------
     I
________I________
| TUpperField |
---------------
```

TUpperField provides a version of TField which converts all lowercase characters to uppercase.

Member functions	
constructor	TUpperField(const TRect& bounds, int aMaxLen);
	Calls the TField constructor.
filterCharCode	virtual ushort filterCharCode(ushort charCode)
	Converts all lowercase characters to uppercase.

TLowerFieldTFIELD.H tlwfield.cpp

```
 ___________
 | TField |
 ----------
     I
________I________
| TLowerField |
---------------
```

TLowerField provides a version of TField which converts all uppercase characters to lowercase.

Member functions	
constructor	TLowerField(const TRect& bounds, int aMaxLen);
	Calls the TField constructor.
filterCharCode	virtual ushort filterCharCode(ushort charCode)
	Converts all uppercase characters to lowercase.