The Pine Creek Modula-3 Library

FieldList, v. 1.0

Breaks text strings into fields, which can be processed as strings, characters, or numbers (integer and floating-point). Provides a quick and efficient way to parse input lines and allow free-form input of text and numbers.

November 13, 1991

The FieldList interface allows you to declare objects called *field lists*. You can place a text string into a field list, or read a line into it from a reader, and the text will be divided into whitespace-separated *fields*. The fields can then be processed in any order, as text or numbers. You can use the same field list many times; each new input line replaces the previous one. For example:

```
MODULE Sum EXPORTS Main;
IMPORT FieldList, Wr, Stdio, Fmt;

PROCEDURE Put(t: TEXT) =
  BEGIN Wr.PutText(Stdio.stdout, t); END Put;

VAR fl := NEW(FieldList.T).init();
BEGIN
  LOOP
    Put("Type two numbers: ");
    fl.getLine();
    IF fl.number() # 2 THEN
      Put("Exactly two, please!\n");
    ELSE
      Put(Fmt.F("The sum of %s and %s is %s\n",
        fl.text(0), fl.text(1),
        Fmt.LongReal(fl.real(0)+fl.real(1))));
    END;
  END;
END Sum.

Type two numbers: 3, 56.8
The sum of 3 and 56.8 is 59.8
```

1.0 Installation

FieldList is provided in source form. The distribution file, fl–1.0.tar.Z, is unpacked as directory fl–1.0 with the UNIX™ command

```
> uncompress –c fl–1.0.tar.Z | tar xvf –
```

The enclosed Makefile will build the FieldList module and some interactive test programs. Just type the UNIX command “make” from inside the directory fl–1.0. (You must have SRC Modula–3 version 1.6 or later installed on your system.)

To use FieldList with other programs, be sure the file FieldList.i3 is on the compiler’s search list, and link the files FieldList.io and FieldList.mo with your programs.

All correspondence should be directed to modula3@bert.pinecreek.com, or to:

Modula-3 Library
Pine Creek Software
305 South Craig Street, Suite 300
Pittsburgh, PA 15213
USA

Phone: +1 (412) 681-9811

2.0 Using FieldList

Follow these basic steps to use field lists:

1. Import the FieldList interface by including the line

```
IMPORT FieldList;
```

2. Create a field list object fl with the declaration

```
VAR fl := NEW(FieldList.T).init();
```

3. Read a line into the field list from a reader:

```
fl.getLine(rd);   (* read the next line from rd into fl *)
```

4. Obtain individual fields from the field list as text or numbers, using the field list methods. For example:

```
(* Sum all the numbers on the line *)
sum := 0.0D0;
FOR field := 1 TO fl.number() DO
   sum := sum + fl.real(field);
END;
```

5. Repeat steps 3 and 4 as necessary for subsequent lines.

The remainder of this section goes into more detail on how to use field lists.

2.1 Reading Lines

To treat a string as a list of fields, create a field list and place the string in it. The new string replaces any previous one.

2.1.1 Creating a Field List

Create a field list object just as you would create any Modula–3 object:

```
VAR fl := NEW(FieldList.T).init();
```

A field list initially contains no fields. You only need one object if you process lines sequentially. When you read in a new line, the previous line's fields are lost. If you want to hold two or more lines at the same time, you must create a field list object for each one.

2.1.2 Reading a Line Into a Field List

You can read a line from any reader into a field list object. If your reader is rd (type Rd.T), then simply write

```
fl.getLine(rd);   (* Reads the next line from rd into fl *)
```

The effect of the getLine method on rd is exactly the same as a single call to Rd.GetLine to read a line. Any exceptions raised by that procedure are propagated out of the get method.

If you omit the reader when using the getLine method, the standard input reader is assumed. That is, writing fl.getLine() is the same as writing fl.getLine(Stdio.stdin).

2.1.3 Converting a Text Object to a Field List

If you have a text string and you want to treat it as a field list, use the getText method. If t is a value of type TEXT, you can store t in a field list fl by writing:

```
fl.getText(t);   (* Stores t in fl *)
```

The string can contain any characters; it does not have to be a single "line."

2.1.4 Retrieving the Input Line

After a line or text string has been stored in a field list, you can retrieve the entire line using the line method:

```
t := fl.line();   (* Store the line in TEXT variable t. *)
```

2.1.5 Determining the Length of the Input Line

The length of the current line is returned by the lineLength method:

```
len := fl.lineLength();   (* Length of input line *)
```

The result is the same as Text.Length(fl.line()).

2.1.6 Deleting a Field List

A field list is deleted simply by discarding it or letting the variable holding it go out of scope:

```
fl := NIL;   (* Discard field list fl. *)
```

It is more efficient to use the same field list many times, rather than, say, to create a new field list for every line.

2.2 Retrieving Fields

After a string is placed in a field list with the getLine or getText methods, it is treated as a sequence of text and number fields. Fields are separated in the input string by one or more whitespace characters. The whitespace characters initially consist of the space, tab, newline, formfeed, and comma characters. Also, a field can be enclosed in quotation marks, in which case it can contain embedded whitespace characters. For example, the input line

```
"Billy Jones" 324 Calhoun St., Pgh, PA 15213
```

contains 7 fields:

```
Billy Jones
324
Calhoun
St.
Pgh
PA
15213
```

2.2.1 Determining the Number of Fields

You can find out how many fields are in a field list with the number method:

```
nFields := fl.number();   (* the number of fields *)
```

Fields are numbered consecutively, with the first field in the line being number 0 and the last being number fl.number()–1. (0-based numbering is a Modula–3 convention.)

2.2.2 Retrieving a Field as Text

The text method returns an individual field as a string:

```
SixthField:= fl.text(5);
```

If n >= fl.number(), then the empty string ("") is returned.

2.2.3 Determining the Kind of Field

A field can have one of three kinds: text, integer, or real (i.e., floating-point). A field that is either integer or real is also considered a "number." Any field that is not a number is text. You can determine whether a field is a certain kind by calling special methods:

fl.isInteger(n) returns TRUE if field n in fl is an integer.

fl.isReal(n) returns TRUE if field n in fl is a floating-point number.

fl.isNumber(n) returns TRUE if field n in fl is either an integer or a floating-point number.

If fl.isNumber(n) returns FALSE then field n is a text field.

2.2.4 Retrieving a Field as an Integer

The method integer returns the integer value of a field:

```
val := fl.integer(n);   (* integer value of n'th field *)
```

If n is an integer field, the value is just what was present. If n is a real field, the integer value is obtained by rounding the floating-point value. In all other situations (e.g., if n is a text field or out of range), 0 is returned.

2.2.5 Retrieving a Field as a Real Number

The method real returns the floating-point value of a field:

```
fpVal := fl.real(n);   (* floating-point value of n'th field *)
```

If n is a floating-point field, the value is just what was present. If n is an integer field, the floating-point value is obtained by converting the integer to a floating-point type. In all other situations, 0.0 is returned. The return value has type LONGREAL.

2.2.6 Determining the Length of a Field

The length method returns the number of characters in a field:

```
width := fl.length(n);   (* length of field n *)
```

2.2.7 Retrieving Characters from a Field

The char method returns an individual character from a field. The characters of a field are numbered beginning with 0.

```
ch := fl.char(n, 1);   (* get the 2'nd character of field n *)
```

2.2.8 Determining the Position of a Field

The pos method returns the number of characters that preceded the first character of the field in the original line.

```
offset := fl.pos(n);   (* position of field n *)
```

2.2.9 Testing the Value of a Field

The equal method compares a string with the contents of a field and returns TRUE if they are the same. Comparisons are case sensitive unless the third (optional) parameter is FALSE.

```
fl.equal(n, "fred")          (* FALSE if field is "Fred" *)
fl.equal(n, "–C", FALSE)   (* will match "–c" *)
```

2.3 Customizing Fields

There are several ways to affect how fields are scanned by the getLine and getText methods. All involve changing "attributes" associated with the field list.

2.3.1 Changing the Whitespace Characters

Each field list can have its own set of whitespace characters. When a field list is created, its whitespace characters are given by the value of the constant FieldList.DefaultWS, whose type is SET OF CHAR. If newWS has type SET OF CHAR and contains the desired whitespace characters, writing:

```
fl.setAttr("whitespace",
  FieldList.RefSetOfChar(newWS));
```

will change the whitespace characters beginning with the next call to getLine or getText.

2.3.2 Treating Whitespace as Fields

Whitespace is normally used only to separate fields. However, a field list can be directed to keep the sequences of whitespace characters as fields by calling the method

```
fl.setAttr("keep_whitespace",
  FieldList.RefBoolean(TRUE));
```

This call takes effect beginning with the next call to getLine or getText. As a result, the field list will contain approximately twice as many fields as normal, with whitespace and non-whitespace fields alternating. The method fl.isWhitespace(n) returns TRUE if field n is a whitespace field.

2.3.3 Quoted Strings

Fields that begin with a quotation mark or an apostrophe are assumed to be quoted text fields. All the text up to and including the next matching quote are considered part of the field. The first character following the closing quote (if not whitespace) will be the first character of the next field.

Within a quoted field, the \ character is used to escape quotation marks. Quoted fields are always considered text, even if the contents would be a number. The field method, which returns a field as a text string, does not return the enclosing quote and does not translate escape sequences.

To disable the quoted string feature, and treat quotes as normal characters, execute the method

```
fl.setAttr("quoted_fields",
  FieldList.RefBoolean(FALSE));
```

3.0 Implementation Characteristics

When a text string is entered into a field list, it is immediately copied into a dynamic character array for faster performance. Since there is little cost in doing so, the original line is also kept as a TEXT object in case the line method is called.

New lines are quickly scanned to locate the enclosed fields and the field positions are entered into a dynamic array of descriptors. Number fields are not analyzed or converted until the values are explicitly needed, so that no penalty is paid for fields that are ignored (except for the initial scan).

When a field must be scanned in more detail (e.g., to convert it to a number or a string), the result is cached with the field, so that repeated calls for the same field have minimal overhead.

The character array and descriptor array are reused for subsequent lines read into the same field list, minimizing calls on NEW. The arrays expand to accommodate the longest line and most fields per line read so far; they are never contracted. Cache records are dynamically allocated, but they are placed on a global free list when released and are available for use in any field list.

Scanning of floating-point values is performed by the C library routine Cstdlib.strtod, which is the same one used in the Scan and Convert interfaces.

4.0 Reference

4.1 Error Handling

In so far as possible, all the methods and procedures in this interface attempt to do something reasonable in all circumstances, rather than reporting an error. For example, requesting the integer value of a text field simply causes a zero to be returned, not an error indication. Specifying a field number greater than the number of fields is treated as an empty field. There are four circumstances in which an error will occur:

1. A checked run-time error will occur if the value of a field list is NIL.

2. An exception or run-time error may be generated by the call to Rd.Getline that is part of the get method, depending on the condition of the specified reader.
3. Calling the methods setAttr or getAttr with an unrecognized attribute name or a parameter of the wrong type will cause the exception FieldList.Error to be raised.
4. Exceeding the resources of the host computer may cause a call to NEW to fail inside the FieldList module and thus cause a run-time error.

4.2 Field Syntax

This section describes the permissible syntax for numbers and quoted strings. There is no such thing as an "illegal" field; if a field is not a legal number, it is simply treated as a text field.

4.2.1 Integers

An integer consists of an optional sign (+ or –) followed by one or more decimal digits. There is no provision for integers in other radices.

4.2.2 Floating-point Numbers

A floating-point number consists of the following parts, in order:

1. an optional sign (+ or –)
2. one or more decimal digits with at most one leading, following, or embedded decimal point
3. optionally, one of the characters E, e, D, d, X, or x, followed by an optional sign, followed by one or more decimal digits

Either a decimal point, an exponent part, or both must be present in order for the number to be considered floating-point and not integer.

4.2.3 Quoted Strings

A quoted string must begin with " or ', followed by zero or more characters, followed by a matching " or '. A " or ' character after the one beginning the quoted string is not considered to be matching if it is immediately preceded by a backslash (\).

Quoted fields are only recognized when they begin the field list or when they follow whitespace characters. A quote within a non-quoted text field is not treated specially. If a field begins with a quoting character but there is no matching quote on the line, then the first quote is treated as a normal character.

Only a quoted field can be empty (i.e., "") with non-empty fields following it.

4.3 Use in Multithreaded Programs

No synchronization is provided for individual field list objects, so the same object should not be used by parallel threads. However, parallel threads may use different field lists safely.

4.4 Exceptions and Types

These exceptions and types are defined in FieldList.

4.4.1 Error

This exception is raised in the setAttr and getAttr methods when an attribute name is unrecognized or when the type of an attribute argument is inappropriate.

4.4.2 Field

This is the type used for field numbers; it is the same as CARDINAL.

4.4.3 T

The field list object type. It contains the methods listed in Section 4.5 below. The type has no visible fields.

4.5 Methods

The object FieldList.T has the following methods.

4.5.1 char

```
char(n: Field; i: CARDINAL := 0): CHAR;
```

Character i from field n is returned. The character '\000' is returned if the field number or character number is out of range.

4.5.2 dump

```
PROCEDURE dump(wr: Wr.T := NIL);
```

Writes the current state of the field list to the indicated writer. If the writer is omitted, Stdio.stdout is used. This procedure is normally disabled; its body is enclosed in comments. To use it, edit the file FieldList.m3 and recompile.

4.5.3 equal

```
equal(n: Field; text: TEXT; caseSensitive := TRUE)
  : BOOLEAN;
```

Returns TRUE if the contents of field n is the same as the text string. Letter case is significant if the caseSensitive argument is TRUE. The string compared against text is the same as is returned by the method text(n).

4.5.4 getAttr

```
getAttr(attribute: TEXT): REFANY;
```

Retrieve the value of the named attribute. Each attribute has a designated type T, and the value returned will have type REF T. The list of attributes are given in Table 1 on page 6.

4.5.5 getLine

```
get(rd: Rd.T := NIL) RAISES
  {Rd.EndOfFile, Rd.Failure, Rd.Alerted, Rd.Error};
```

Reads a line from rd by a call to Rd.GetLine(rd) and stores it in the field list, replacing any previous line. If rd is NIL, Stdio.stdin is used as the reader.

4.5.6 getText

```
getText(t: TEXT);
```

Stores the text string t in the field list.

4.5.7 init

```
init(): T;
```

The init() method currently has no effect, but clients are advised to include a call on init when they create field lists for future compatibility.

4.5.8 integer

```
integer(n: Field): INTEGER;
```

If n is an integer field, return its integer value. If n is a floating-point field, round the field's value to an integer and return it. If the value to be returned in either of these cases exceeds the range of INTEGER, return either FIRST(INTEGER) or LAST(INTEGER) (whichever is closer to the true value). For all other values of n, return 0.

4.5.9 isInteger

```
isInteger(n: Field): BOOLEAN;
```

If the characters of field n form a syntactically legal integer (see “Integers” on page 4), return TRUE. Otherwise, return FALSE.

4.5.10 isNumber

```
isNumber(n: Field): BOOLEAN;
```

Same as isInteger(n) OR isReal(n).

4.5.11 isReal

```
isReal(n: Field): BOOLEAN;
```

If the characters of field n form a syntactically legal floating-point number (see “Floating-point Numbers” on page 4), return TRUE. Otherwise, return FALSE.

4.5.12 isWhitespace

```
isWhitespace(n: Field): BOOLEAN;
```

If field n contains field-separating whitespace characters, return TRUE. Otherwise, return FALSE.

A field can contain whitespace characters only if the attribute "keep_whitespace" of the field list is TRUE. A quoted field that happens to contain only whitespace characters is not considered a whitespace field.

4.5.13 length

```
length(n: Field): CARDINAL;
```

Returns the number of characters in field n. Returns 0 if n is not a valid field number.

4.5.14 number

```
number(): CARDINAL;
```

Returns the number of fields in the field list. This number can be zero if the input line is empty, or if the line contains only whitespace characters and the "keep_whitespace" attribute is FALSE.

4.5.15 pos

```
pos(n: Field): CARDINAL;
```

Returns the number of characters preceding field n in the field list. If n is not a valid field number (i.e., is too large), this method returns the total length of the line.

4.5.16 put

```
put(n: Field; wr: Wr.T := NIL);
```

Write the contents of field n on writer wr. If wr is NIL, Stdio.stdout is used. The text written is the value of field(n).

4.5.17 real

```
real(n: Field): LONGREAL;
```

If n is a floating-point field, return its value. If n is an integer field, convert the field's value to floating-point and return it. For all other values of n, return 0.0D0.

4.5.18 setAttr

```
setAttr(attribute: TEXT; value: REFANY);
```

Change the value of the named attribute to the indicated value. Each attribute has a designated type T, and the value argument should have type REF T when calling setAttr. (No reference to the argument is kept after the call to setAttr.) The list of attributes are given in the table below.

TABLE 1. Attributes of Field List Objects

Name	Type	Effect
"whitespace"	SET OF CHAR	the characters considered to be whitespace
"keep_whitespace"	BOOLEAN	If true, whitespace characters are kept as individual fields
"quoted_fields"	BOOLEAN	If true, quoted fields are recognized by the scanner.

4.5.19 text

```
text(n: Field): TEXT;
```

Return field n just as it appears in the input. If n is not a valid field number, return the empty string. Quoted fields are returned from this method without their enclosing quotes.

4.6 Procedures

The following procedures are included in FieldList. They are included to help you with calls to the setAttr method.

4.6.1 RefBoolean

```
PROCEDURE RefBoolean(val: BOOLEAN): REF BOOLEAN;
```

Returns a reference to a new dynamic variable of type BOOLEAN, initialized to the value of the parameter.

4.6.2 RefInteger

```
PROCEDURE RefInteger(val: INTEGER): REF INTEGER;
```

Returns a reference to a new dynamic variable of type INTEGER, initialized to the value of the parameter.

4.6.3 RefSetOfChar

```
PROCEDURE RefSetOfChar(READONLY val: SET OF CHAR): REF SET OF CHAR;
```

Returns a reference to a new dynamic variable of type SET OF CHAR, initialized to the value of the parameter.