AN EXPRESSION COMPILER

MICHAEL E. HENDERSON*

Abstract. This paper describes a set of subroutines for compiling and evaluating string expressions and their derivatives. The expression is first translated into "object code", which consists of a symbol table, and a list of operations which evaluates the expression. Subroutines are provided for querying and filling in the symbol table, for constructing object code for derivatives of the expression, and for executing the object code.

1. Introduction. This document describes a set of subroutines which take an expression and generate a set of instructions ("object code") which can be executed to evaluate the expression and its derivatives. The routines are written in C, and are callable from Fortran.

The "object code" includes a symbol table, which contains an entry for each non-numeric identifier in the expression. Entries in the symbol table may be set to real or integer values, or when appropriate, to the address of an external routine. When all identifiers have been given values, the "object code" can be executed to return a single floating point value. Routines are also included which take "object code" and create new "object code" which evaluates the derivative with respect to one of the non-numeric identifiers in the expression.

To avoid unreasonable expectations, let me briefly point out what this is not. First, this is not a symbolic manipulation program. All identifiers must be set before an expression can be evaluated, and the differentiation provided never produces an expression for the derivative. Second, this is not a programming language. The only source statement allowed is a simple expression. No loops, no branches or assignments.

This set of routines can be used to provide the user with an easier interface to programs which require user defined functions. As a simple illustration, consider a program which plots a function. Here are three approaches to designing the code.

> *Method 1.* The user is asked to write a subroutine called "f" which returns the value of the function. This subroutine is linked (or dynamically loaded) into a program called "plotter" which calls "f" to plot the function.
>
> *Method 2.* The user writes a main program, which calls a subroutine called "plotter" and passes the address of a subroutine which is called by "plotter" to plot the function.

The routines provided here allow a third approach.

> *Method 3.* The user runs a program called "plotter" which asks him to enter an expression containing the function to be plotted (e.g. "sin(x**2)/4.")

Evaluating the expression in this way is of course slower than optimized compiled code, so the third approach may not always be appropriate. For the routines described here evaluation is about a factor of 5 slower than executing source compiled in "C" with the `-O` flag.

* IBM T.J. WATSON RESEARCH CENTER, P.O. BOX 218, YORKTOWN HEIGHTS, NY 10598

These routines provide users with a programming-free interface to define functions to a program. This eliminates the need for the developer to dynamically load user subroutines, or provide "object code" so that the user can link in his own routines. Two programs using this approach are included as examples in the `examples` subdirectory) (`plotter.f` and `calc.c`).

2. Basic Routines. Three sets of routines are provided, which perform the following functions:

- *Compilation of a string*
- *Filling in the symbol table*
- *Evaluation of the "object code"*

The *Compilation* routine "compiles" a character expression into "object code". The routines which *Fill in the symbol table* assign values to the non-numeric identifiers in the expression, and query the symbol table. The *Evaluation* routine computes the value of the expression.

2.1. Compilation. The `ECCompileExpression` routine compiles a string containing an expression into "object code". A zero return code indicates that the compilation was successful. (Appendix A gives the error codes and their meaning. The `ECGetErrorMessage` routine returns the text of a message corresponding to the return code.)

For example, to compile an expression which evaluates the function "sin(x)"

```
#include <ExpCmp.h>

int rc;
struct ECObjectCode *object;

main()
 {

  rc=ECCompileExpression("sin(x)",&object);
  if(rc!=EC_NO_ERROR)
   {
    printf("%s\n",ECGetErrorMessage(rc));
    return;
   };
 };
```

Note that the "object code" is passed as a pointer to a pointer. This was done so that if compilation fails the pointer can be set to NULL, rather than to an "object code" with a flag indicating failed compilation. The compile routine takes care of allocating memory for the "object code". The "object code" data structure should be free'd when it is no longer needed using the routine `ECFreeObjectCode`. This assures that the storage pointed to by the structure also gets free'd.

The syntax of the expression is similar to Fortran.

- Identifiers are case sensitive, so that "x" and "X" are different identifiers.
- The characters $[a-z, A-Z, \%]$ are valid in Identifiers. The % was included to distinguish special identifiers like π (*%pi*).

- The binary operations +, −, ∗, / and ∗∗ (exponentiation) are allowed, along with unary negation −.
- All arithmetic is done in single precision floating point, and all values are converted to floating point before the expression is evaluated. So for example, $2/3 = .666667 \neq 0$.
- Parentheses (and) are used for grouping in the normal fashion.
- Embedded blanks are ignored except in identifiers. So "x * y" is ok, but "x y" is not legal, since it is parsed as "*identifier identifier*".
- Function calls are allowed, with either zero or one argument. The syntax is "*identifier* ( )" or "*identifier* (*expression*)".
- Real constants expressed in exponential notation are not legal.

Some examples of valid expressions are

```
4*x-6.
x**2
x**ln(x)
x**ln(x)/A+2/(-4+atan(%pi*x))
```

while the following expressions are invalid

```
4*x y-6.
x**2*(
x**ln(x,y)+1.e5
```

The compilation produces a structure containing "object code". This consists of a symbol table and a list of instructions for evaluating the expression. The symbol table contains each identifier in the source, its type and value. Before the object code can be executed all identifiers must be set (see Section 2.2).

It is not necessary to inspect the "object code". However, a routine `ECPrint-ObjectCode` has been provided, which prints a textual version to `stdout`. For the previous example

```
ECPrintObjectCode(object);
```

produces the following output:

```
Object Code:

            6 statements
            2 identifiers
            1 real constants
            1 integer constants

Executable Code:

 Register 1 = Identifier "x"
 Register 2 = IntegerConstant 2
 Register 3 = Register 1 ** Register 2
 Register 4 = Call Identifier "sin" ( Register 3 )
```

```
Register 5 = RealConstant 4.000000
Register 6 = Register 4 / Register 5
Return(Register 6)
```

2.2. Filling in the Symbol Table. There are four types of Identifier: undefined, real (float), integer (int), and function (ECPointerToFunction, a type defined in ExpCmp.h). The following routines set the type and give values to the Identifiers in the symbol table.

```
ECSetIdentifierToUndefined
ECSetIdentifierToReal
ECSetIdentifierToInteger
ECSetIdentifierToFunction

ECSetIdentifier
```

The `ECSetIdentifier` routine takes a string of the form "left=right". The identifier name is read from the string "left" (using `sscanf(left,"%s",identifier)`). The string "right" is compiled, evaluated, and the identifier is set to the result with `ECSetIdentifierToReal`. Before the right hand side is evaluated the routines `ECSetStandardMathConstants` and `ECSetStandardMathFunctions` are called (see Section 2.2.1). Examples are

```
ECSetIdentifier("x=5.4/3.1", object);
ECSetIdentifier("x=3.*sin(-1.8*%pi)",object);
ECSetIdentifierToReal("x",3.14,object);
ECSetIdentifierToInteger("x",3,object);
```

2.2.1. "Standard" Identifiers. There is a group of identifiers that I have found that I set all the time. Two routines are provided for setting these "standard" identifiers: `ECSetStandardMathConstants` and `ECSetStandardMathFunctions`. The "standard math functions" and "standard math constants" are listed in Tables 1 and 2.

Identifier	Function
`sin`	sin
`sinh`	hyperbolic sin
`asin`	arcsin
`cos`	cos
`cosh`	hyperbolic cos
`acos`	arccos
`tan`	tangent
`tanh`	hyperbolic tangent
`atan`	arctangent
`sqrt`	square root
`abs`	absolute value
`exp`	exponential
`log`	natural log
`ln`	natural log
`log10`	log base 10

Table 1. Standard Math Functions

Identifier	Value
%pi	π
%e	e

Table 2. Standard Math Constants

2.2.2. Querying the Symbol Table. An interactive program can print the symbol table using the routine `ECPrintSymbolTable`, and then ask the user to set the identifiers. Continuing our previous example,

```
rc=ECSetStandardMathFunctions(object);
if(rc!=EC_NO_ERROR)
 {
  printf("%s\n",ECGetErrorMessage(rc));
  return;
 };

ECPrintSymbolTable(object);

printf(" Set which Identifier?\n");
scanf("%s",identifier);
printf(" to what value?\n");
scanf("%f",&value);

rc=ECSetIdentifierToReal(identifier,value,object);
```

produces the following output:

```
Symbol Table:

 Identifier "sin", is a Function
 Identifier "x", is Undefined
```

It may also be necessary for a program to access the symbol table and set the identifiers directly. A second set of routines is provided which allows access to information in the symbol table. These include

```
ECNumberOfIdentifiers
ECGetIdentifierName
ECGetIdentifierType
ECIsIdentifierSet

ECNumberOfUnsetIdentifiers
ECGetUnsetIdentifierName

ECNumberOfRealIdentifiers
ECGetRealIdentifierName
ECGetRealIdentifierValue

ECNumberOfIntegerIdentifiers
ECGetIntegerIdentifierName
ECGetIntegerIdentifierValue

ECNumberOfFunctionIdentifiers
ECGetFunctionIdentifierName
ECGetFunctionIdentifierValue
```

The routine `ECGetIdentifierType` returns a string containing "Float", "Integer", "Function", or "Undefined" according to the type of the identifier. If the identifier is not present in the symbol table an empty string is returned and the return code is set. (see Appendix A and the `ECGetErrorMessage` routine.)

2.3. Evaluation. After all identifiers have been given values, the routine `ECEvaluateExpression` can be used to execute the "object code" and evaluate the expression. For the previous example,

```
rc=ECSetIdentifierToReal("x",1.0,object);

f=ECEvaluateExpression(object,&rc);
if(rc==EC_NO_ERROR)printf("sin(1.)=%f\n",f);
```

3. Derivatives. There is a algorithm for constructing the derivative of a program (e.g. [1]). For our expressions, the "program" stored in the "object code" is very much simpler than most programming languages, and this is a very simple operation.

For example, the "object code" for computing the product of two numbers is

```
Register 1 = Identifier "x"
Register 2 = Identifier "y"
Register 3 = Register 1 * Register 2
Return(Register 3)
```

The algorithm for computing the derivative duplicates each Register. This duplicate stores the derivative of its associated Register. For example, the object code below evaluates the derivative of the code above.

```
Register 1 = Identifier "x"
DRegister 1 = 1.
Register 2 = Identifier "y"
```

```
DRegister 2 = 0.
Register 3 = Register 1 * Register 2
DRegister 3 = DRegister 1 * Register 2 + Register 1 * DRegister 2
Return(DRegister 3)
```

Without some sort of optimization, this is not an efficient process. However, it is simple to implement, and does calculate the correct result.

The routine `ECCreateExpressionDerivative` constructs "object code" for the derivative from "object code" for an expression. The symbol table is copied, and for each function call in the original expression a function identifier is added with a "D" prepended to the identifier. A "log" identifier is also added if it is not present (it is needed for differentiating exponentials). For example, the "object code" for the derivative of "cos(x)" with respect to the symbol "x" will have extra identifiers "Dcos" and "log". The `ECSetStandardMathFunctions` sets the derivatives of the standard functions. It does not set the second derivatives. Differentiating a derivative will generate valid "object code", but executing it will fail because of undefined identifiers. For "cos(x)" the user would have to call `ECSetIdentifierToFunction` for the identifier "$\widehat{\text{D}}$Dcos".

For example,

```
rc=ECCompileExpression("sin(x)",&object);
rc=ECCreateExpressionDerivative("x",object,&derivative);

rc=ECSetStandardMathConstants(object);
rc=ECSetStandardMathConstants(derivative);
rc=ECSetStandardMathFunctions(object);
rc=ECSetStandardMathFunctions(derivative);

rc=ECSetIdentifierToReal("x",1.,object);
rc=ECSetIdentifierToReal("x",1.,derivative);

printf("sin(1.)=%d, d/dx(sin)(1.)=%d\n",
               ECEvaluateExpression(object,&rc),
               ECEvaluateExpression(derivative,&rc));
```

4. Examples. Two example codes are supplied in the `examples` subdirectory. These demonstrate the C and Fortran interfaces.

4.1. Calc. The C program `calc` takes an expression as an argument, and allows the user to interactively set identifiers and evaluate the expression. An abbreviated version of the source is listed below. The full source is located in `examples/calc.c`.

```
#include "ExpCmp.h"

main( int *argc, char *arg[])
{
 struct ECObjectCode *object;
 char assignment[256];
 float Value;
 int i,n;
 int rc;

/* Retrieve the argument string and compile it */

 if(ECCompileExpression(arg[1],&object)!=EC_NO_ERROR)exit(8);

/* Set the standard identifiers */

 rc=ECSetStandardMathFunctions(object);
 rc=ECSetStandardMathConstants(object);

 if(ECNumberOfUnsetIdentifiers(object,&rc)==0)
  {

/* If all identifiers have been set, execute the object code */

   Value=ECEvaluateExpression(object,&rc);
   printf("%e\n",Value);
  }else{

/* Otherwise, prompt for assignment statements or other commands. */

   while(TRUE)
    {
     printf("calc: ");
     scanf("%s",assignment);

     if(!strcmp(assignment,"quit"))exit(0);

     if(!strcmp(assignment,"execute"))
      {
       n=ECNumberOfUnsetIdentifiers(object,&rc);
       if(n==0)
        {
         printf("\n");
```

```
         Value=ECEvaluateExpression(object,&rc);
         printf("      \"%s\"=",arg[1]);
         printf("%e\n",Value);
         printf("\n");
        };
      }else{
       ECSetIdentifier(assignment,object);
      };
    };
  };

 ECFreeObjectCode(object);
};
```

The listing below shows the output of a session using the `calc` program.

```
$calc "sin(abs(x)**sin(3*x)-A)+A/abs(ln(abs(y)))"
calc: ?

Source Code:

sin(abs(x)**sin(3*x)-A)+A/abs(ln(abs(y)))

Symbol Table:
 Symbol "sin", is a Function
 Symbol "x", is Undefined
 Symbol "A", is Undefined
 Symbol "abs", is a Function
 Symbol "ln", is a Function
 Symbol "y", is Undefined

Enter one of the following commands at the calc: prompt:

       Identifier=value    assign a value to an Identifier
       execute             evaluate the expression
       quit
       ?                   print this information

calc: A=1.
calc: x=%pi/2.
calc: execute

1 Identifier has yet to be set.

 y
      "sin(abs(x)**sin(3*x)-A)+A/abs(ln(abs(y)))"

calc: y=4.
calc: execute
```

```
      "sin(abs(x)**sin(3*x)-A)+A/abs(ln(abs(y)))"=0.365912

calc: x=9.
calc: execute

      "sin(abs(x)**sin(3*x)-A)+A/abs(ln(abs(y)))"=1.501058

calc:quit
```

4.2. Plotter. `plotter` is a Fortran program which prompts the user for an expression, and then plots it and its derivatives. It assumes that the identifier which is used to plot the function is "x". A simplified source for this program is listed below. The full source can be found in examples/plotter.f.

```
      program plotter
      character*256 sourceCode
      character*256 inputString
      character*256 format
      integer object
      integer derivative
      logical equalSignPresent

c Compile a default function

      sourceCode='sin(x)'
      call ECCompileExpression(sourceCode,object,ierr)
      if(ierr.ne.0)then
        write(6,*)' Compilation of default source failed!',ierr
        call exit
      endif

      call ECSetStandardMathFunctions(object,ierr)
      call ECSetStandardMathConstants(object,ierr)

      defaultObject=object

      call ECCreateExpressionDerivative(object,'x',derivative,ierr)
      call ECSetStandardMathFunctions(derivative,ierr)
      call ECSetStandardMathConstants(derivative,ierr)

      defaultDerivative=derivative

c Prompt the user for commands

    1 continue

        write(6,'(a)')'Command:'
```

```
      read(5,'(a)')inputString
      if(inputString.eq.'quit')call exit

      if(inputString(1:7).eq.'symbols')then
         call ECPrintSymbolTable(object)
         go to 1
      endif

      if(equalSignPresent)then
         call ECSetIdentifier(inputString,object,ierr)
         call ECSetIdentifier(inputString,derivative,ierr)
         go to 1
      endif

      if(inputString(1:4).eq.'plot')then
         ip=3
         dx=(xmax-xmin)/nsteps
         do i=0,nsteps
            x=xmin+i*dx

            call ECSetIdentifierToReal('x',x,object,ierr)
            call ECSetIdentifierToReal('x',x,derivative,ierr)

            y=ecevaluateexpression(object,ierr)
            f=ecevaluateexpression(derivative,ierr)

            if(ip.eq.2)then
              call move(x0,y0)
              call line(x,y)
              call move(x0,f0)
              call line(x,f)
            endif
            ip=2
            x0=x
            y0=y
            f0=f
         enddo

         go to 1
       endif

       sourceCode=inputString
       if(object.ne.defaultObject)call ECFreeObjectCode(object)
       call ECCompileExpression(sourceCode,object,ierr)
       call ECSetStandardMathFunctions(object,ierr)
       call ECSetStandardMathConstants(object,ierr)

       if(derivative.ne.defaultDerivative)
*             call ECFreeObjectCode(derivative)
```

```
          call ECCreateExpressionDerivative
     *                 (object,'x',derivative,ierr)
          call ECSetStandardMathFunctions(derivative,ierr)
          call ECSetStandardMathConstants(derivative,ierr)
          go to 1

        end
```

The listing below shows the output of a session using the `plotter` program.

```
$plotter
Command:
?

 Enter:

  ?              For this help
  quit           Exits the program
  print          Prints the current state
  symbols        Prints the symbols in the current expression
  Dsymbols       Prints the symbols in the derivative
  clear          Clears the screen
  plot           Plot using the current values

  xmin=value     Sets the left side of the plot
  xmax=value     Sets the right side of the plot
  ymin=value     Sets the top side of the plot
  ymax=value     Sets the bottom side of the plot
  nsteps=value   Sets the number of steps

  symbol=value   Sets the symbol to value

  expression     Makes this the current expression

Command:
xmin=-%pi
Command:
xmax=%pi
Command:
sin(abs(x)**sin(3*x)-A)+A/abs(ln(abs(y)))
Command:
A=1.
Command:
y=4.
Command:
plot
Command:
quit
```

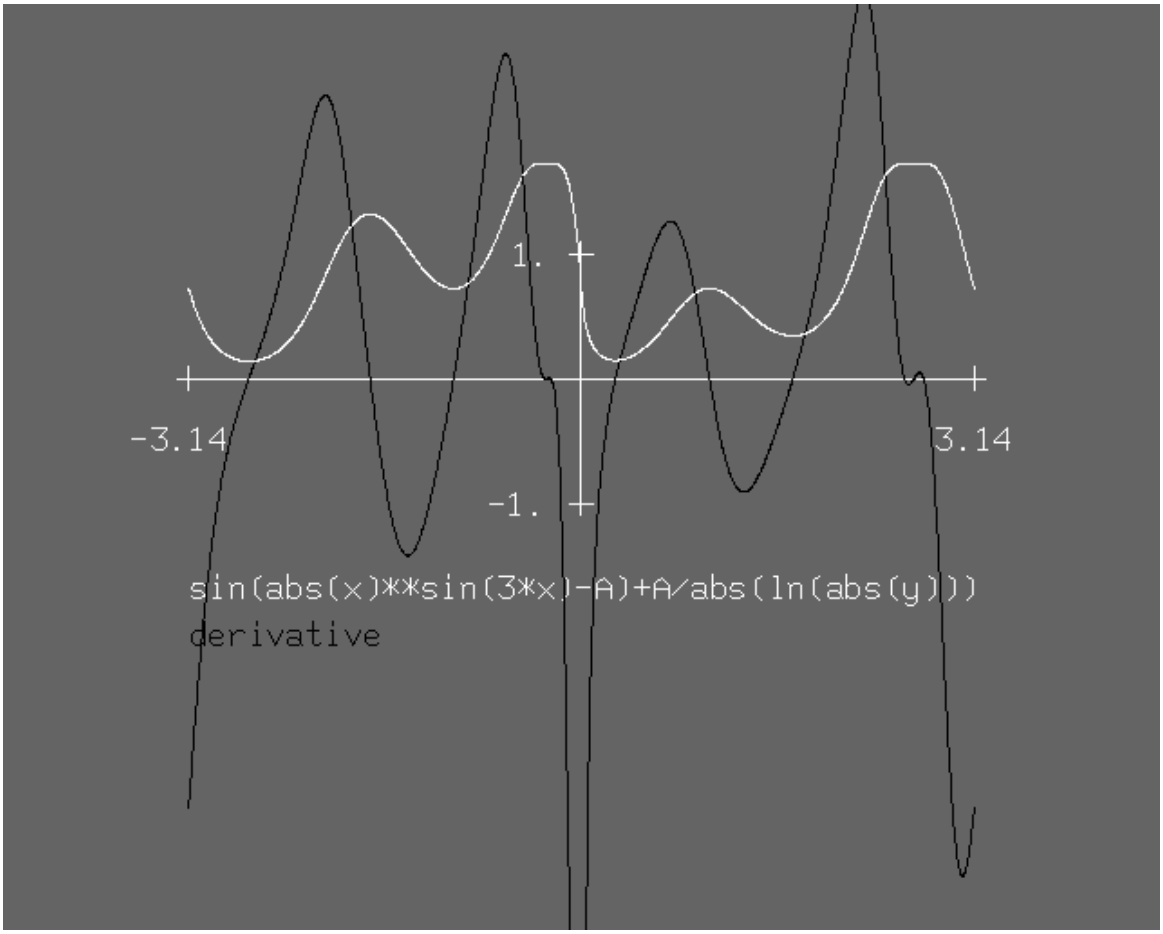

FIG. 1. *The output from the plotter program, with the input given in Section 4.2.*

The result is shown in Figure 1.

5. Conclusion. We have described a set of routines for compiling and evaluating expressions. The preceding sections should serve as a user's guide, and together with the provided examples, should enable you to use the subroutine library.

The following appendices describe

- **Appendix A** – the error codes returned by the various routines.
- **Appendix B** – a detailed description of the interface to each subroutine.

6. Appendix A. Error codes

Constant name (ECMessages.h)	code	Message text
EC_NO_ERROR	0	No Error.
EC_INVALID_OPCODE_IN_DERIVATIVE	1	Invalid opcode.
EC_INVALID_EXPRESSION	2	Expression is invalid.
EC_EXTRA_CHARACTERS	3	Extra characters following valid expression.
EC_LONG_SOURCE	4	sourceCode is longer than 256 characters.
EC_NULL_SOURCE	5	sourceCode is NULL.
EC_TOO_MANY_TOKENS	6	sourceCode contains more than 256 tokens.
EC_INVALID_CHARACTER	7	Invalid character in source.
EC_TOO_MANY_IDENTIFIERS	8	sourceCode contains more than 256 Identifiers.
EC_TOO_MANY_INTEGERS	9	sourceCode contains more than 256 integer constants.
EC_TOO_MANY_REALS	10	sourceCode contains more than 256 real constants.
EC_BAD_CONSTANT_TYPE	11	Invalid ConstantType.
EC_IDENTIFIER_NOT_FOUND	12	Identifier not found.
EC_IDENTIFIERS_NOT_SET	13	Some Identifier not set.
EC_INVALID_OPCODE	14	Bad opcode in object code.
EC_NO_STATEMENTS	15	No Statements in object code.
EC_IDENTIFIER_NOT_FUNCTION	16	Identifier not a Function.
EC_INVALID_ASSIGNMENT	17	Invalid assignment string.
EC_NULL_OBJECT_CODE	18	objectCode is NULL.
EC_INVALID_CONSTANT_TYPE	19	Invalid constant type.
EC_INVALID_IDENTIFIER_TYPE	20	Invalid Identifier type.

7. Appendix B. Subroutine Reference

ECCompileExpression

Purpose

Compiles an expression from a string.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECCompileExpression(source, &object);
```

`char`	**source*	Specified by user. Source code
`struct ECObjectCode`	**object*	Returned to user. Object code, or NULL if the compilation fails.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
call ECCompileExpression(source, object, rc)
```

`integer`	*rc*	Returned to user. Return code.
`character`	*source*	Specified by user. Source code.
`integer`	*object*	Returned to user. Object code, or NULL if the compilation fails.

Description

Use `ECCompileExpression` to create "object code" that can be evaluated by `ECEvaluateExpression`. The source code consists of a Fortran-like expression.

Note that the "object code" is passed as a pointer to a pointer to the structure. `ECCompileExpression` allocates space for the "object code", but the user must allocate space for the pointer.

Constants may be integer or floating point. Exponential notation for real constants is not supported.

Identifiers may contain the characters

[a-z,A-Z,%].

Identifiers may not contain embedded blanks. The expression "a b" contains two identifiers, "a" and "b" (and is not a valid expression).

There is one unary operation

-expression,

and there are five binary operations (with the usual order of precedence):

expression1+*expression2*,
expression1-*expression2*,
*expression1***expression2*,
expression1/*expression2*,
and *expression1****expression2*.

Functions may have either with one or no arguments. The syntax of a function call is

identifier()
or *identifier(expression)*.

Examples of valid expressions are

```
a+5.
a+sin(b**3)
sin(a**(x*ln(x)))/my()
```

Return Codes

EC_NO_ERROR	0	No error.
EC_INVALID_EXPRESSION	2	Expression is invalid.
EC_EXTRA_CHARACTERS	3	Extra characters following valid expression.
EC_LONG_SOURCE	4	sourceCode is longer than 256 characters.
EC_NULL_SOURCE	5	sourceCode is NULL.
EC_TOO_MANY_TOKENS	6	sourceCode contains more than 256 tokens.
EC_INVALID_CHARACTER	7	Invalid character in source.
EC_TOO_MANY_IDENTIFIERS	8	sourceCode contains more than 256 Identifiers.
EC_TOO_MANY_INTEGERS	9	sourceCode contains more than 256 integer constants.
EC_TOO_MANY_REALS	10	sourceCode contains more than 256 real constants.
EC_BAD_CONSTANT_TYPE	11	Bad value for ConstantType.

ECCreateExpressionDerivative

Purpose

Creates "object code" which evaluates the derivative of the expression associated with another piece of "object code".

Library

`libExpCmp.a`

C Syntax

rc=`ECCreateExpressionDerivative(`*object*, *variable*, `&`*derivative*`);`

`int`	*rc*	Returned to user. Return code.
`struct ECObjectCode`	`*`*object*	Specified by user. Object code to be differentiated.
`char`	`*`*variable*	Specified by user. Identifier with respect to differentiate.
`struct ECObjectCode`	`*`*derivative*	Returned to user. Object code for derivative, or NULL if compilation failed.

Fortran Syntax

`call ECCreateExpressionDerivative(`*object*, *variable*, *derivative*, *rc*`)`

`integer`	*rc*	Returned to user. Return code.
`integer`	*object*	Specified by user. Object code to be differentiated.
`character`	*variable*	Specified by user. Identifier with respect to differentiate.
`integer`	*derivative*	Returned to user. Object code for derivative, or 0 if compilation failed.

Description

Use `ECCreateExpressionDerivative` to create "object code" which evaluates the derivative of an expression with respect to an identifier. It uses a code differentiation algorithm, which never constructs source code for the derivative.

Note that the derivative "object code" is passed as a pointer to a pointer to the structure. `ECCreateExpressionDerivative` allocates space for the derivative "object code", but the user must allocate space for the pointer.

Return Codes

EC_NO_ERROR	0	No error.
EC_INVALID_OPCODE_IN_DERIVATIVE	1	An unknown opcode was found in object.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECEvaluateExpression

Purpose

Executes a piece of "object code" and returns the result.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
value=ECEvaluateExpression(object, &rc);
```

`float`	*value*	Returned to user. Result of evaluating the code.
`struct ECObjectCode`	**object*	Specified by user. Object code to be evaluated.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
value = ECEvaluateExpression(object, rc)
```

`real`	*value*	Returned to user. Result of evaluating the code.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECEvaluateExpression` to execute a piece of "object code". All identifiers must have been set before the "object code" can be executed.

If an error code is set, `ECEvaluateExpression` returns `QNaN`.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIERS_NOT_SET	13	There are unset identifiers in the "object code".
EC_INVALID_OPCODE	14	An unknown opcode was found in "object code".
EC_NO_STATEMENTS	15	No statements in "object code".
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECFreeObjectCode

Purpose

Frees the storage associated with a piece of "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
ECFreeObjectCode(&object)
```

`struct ECObjectCode` *`*object`* Specified by user. Object code to be free'd

Fortran Syntax

```
call ECFreeObjectCode(object)
```

`integer` *`object`* Specified by user. Object code to be differentiated.

Description

Use ECFreeObjectCode to free the storage associated with a piece of object code. ECFreeObjectCode sets the pointer to NULL or 0 so that subsequent attempts to use the "object code" result in an error. Each expression compilation allocates storage, and if it is not freed the user may eventually run out of storage.

Note that the "object code" is passed as a pointer to a pointer to the structure. `ECFreeObjectCode` sets the pointer to `NULL`, and does nothing if passed a pointer to `NULL`. Freeing an "object code" twice is therefore harmless, but pointless.

Return Codes

EC_NO_ERROR 0 No error.

ECGetErrorMessage

Purpose

Retrieves a the text of a message associated with a return code.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
message=ECGetErrorMessage(rc);
```

`char`	*`*message`*	Returned to user. Message text.
`int`	*`rc`*	Specified by user. Return code.

Fortran Syntax

```
call ECGetErrorMessage(rc, message)
```

`integer`	*`rc`*	Returned to user. Return code.
`character`	*`message`*	Specified by user. Message text.

Description

Use `ECGetErrorMessage` to retrieve the text of a message associated with a return code. For the C interface, a pointer to a static character string is returned, so it should not be freed. For the Fortran interface the text is truncated to the length of the character string provided, or is padded with blanks.

Return Codes

None.

ECGetFunctionIdentifierName

Purpose

Returns the name of a function identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
identifier=ECGetFunctionIdentifierName(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`char`	**identifier*	Returned to user. Name of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
call ECGetFunctionIdentifierName(number, object, identifier, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`character`	*identifier*	Returned to user. Name of the requested identifier.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECGetFunctionIdentifierName` to get the name of a function identifier. The identifier number must be positive and less than the number returned by `ECNumberOfFunctionIdentifiers`.

The C interface returns a point to the character string in the symbol table. It is only valid until the "object code" is free'd, and should not be freed by the user.

For the Fortran interface, the name is truncated to the length of the character string provided. The string is blank filled, and is not zero terminated.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetFunctionIdentifierValue

Purpose

Extracts the value of a function identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
value=ECGetFunctionIdentifierValue(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`ECPointerToFunction`	*value*	Returned to user. Value of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
value=ECGetFunctionIdentifierValue(number, object, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`integer`	*value*	Returned to user. Value of the requested identifier.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECGetFunctionIdentifierValue` to get the name of a function identifier. function. The identifier number must be positive and less than the number returned by `ECNumberOfFunctionIdentifiers`.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetIdentifierName

Purpose

Returns the name of an identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
identifier=ECGetIdentifierName(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`char`	**identifier*	Returned to user. Name of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
call ECGetIdentifierName(number, object, identifier, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`character`	*identifier*	Returned to user. Name of the requested identifier.
`integer`	*object*	Specified by user. Object code to be evaluated.
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECGetIdentifierName` to get the name of a identifier. The identifier number must be positive and less than the number returned by `ECNumberOfIdentifiers`.

The C interface returns a point to the character string in the symbol table. It is only valid until the "object code" is free'd, and should not be freed by the user.

For the Fortran interface, the name is truncated to the length of the character string provided. The string is blank filled, and is not zero terminated.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetIdentifierType

Purpose

Returns the type of an identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
value=ECGetIdentifierType(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`char`	**value*	Returned to user. Type of the requested identifier. "Undefined" "Real" "Integer" "Function" "" if return code not zero.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
call ECGetIdentifierType(number, object, value, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`character`	*value*	Returned to user. Type of the requested identifier. "Undefined" "Real" "Integer" "Function" "" if return code not zero.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECGetIdentifierType` to query the type of an identifier. The identifier number must be positive and less than the number returned by `ECNumberOfIdentifiers`.

The C interface returns a pointer to a static character string, which should not be free'd by the user.

The Fortran interface truncates type to the length of the character string provided, or pads with blanks.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetIntegerIdentifierName

Purpose

Returns the name of a real identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
identifier=ECGetIntegerIdentifierName(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`char`	**identifier*	Returned to user. Name of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
call ECGetIntegerIdentifierName(number, object, identifier, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`character`	*identifier*	Returned to user. Name of the requested identifier.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECGetIntegerIdentifierName` to get the name of an integer identifier. The identifier number must be positive and less than the number returned by `ECNumberOfIntegerIdentifiers`.

The C interface returns a point to the character string in the symbol table. It is only valid until the "object code" is free'd, and should not be freed by the user.

For the Fortran interface, the name is truncated to the length of the character string provided. The string is blank filled, and is not zero terminated.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetIntegerIdentifierValue

Purpose

Extracts the value of a real identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
value=ECGetIntegerIdentifierValue(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`int`	*value*	Returned to user. Value of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
value=ECGetIntegerIdentifierValue(number, object, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`real`	*value*	Returned to user. Value of the requested identifier.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECGetIntegerIdentifierValue` to get the name of an integer identifier. function. The identifier number must be positive and less than the number returned by `ECNumberOfIntegerIdentifiers`.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetMessagePrint

Purpose

Returns a value indicating if error messages are printed.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
value=ECGetMessagePrint();
```

`int` *value* Returned to user. 0 if messages are not printed. 1 if messages are printed.

Fortran Syntax

```
value=ECGetMessagePrint()
```

`integer` *value* Returned to user. 0 if messages are not printed. 1 if messages are printed.

Description

Use `ECGetMessagePrint` to determine the fate of error messages. This value defaults to 0 (no messages printed), and can be changed using `ECSetMessagePrint`. The return code of a command is always set.

ECGetRealIdentifierName

Purpose

Returns the name of a real identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
identifier=ECGetRealIdentifierName(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`char`	**identifier*	Returned to user. Name of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
call ECGetRealIdentifierName(number, object, identifier, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`character`	*identifier*	Returned to user. Name of the requested identifier.
`real`	*object*	Specified by user. Object code to be evaluated.
`real`	*rc*	Returned to user. Return code.

Description

Use `ECGetRealIdentifierName` to get the name of a real identifier. The identifier number must be positive and less than the number returned by `ECNumberOfRealIdentifiers`.

The C interface returns a point to the character string in the symbol table. It is only valid until the "object code" is free'd, and should not be freed by the user.

For the Fortran interface, the name is truncated to the length of the character string provided. The string is blank filled, and is not zero terminated.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetRealIdentifierValue

Purpose

Extracts the value of a real identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
value=ECGetRealIdentifierValue(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`int`	*value*	Returned to user. Value of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
value=ECGetRealIdentifierValue(number, object, value, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`real`	*value*	Returned to user. Value of the requested identifier.
`real`	*object*	Specified by user. Object code to be evaluated.
`real`	*rc*	Returned to user. Return code.

Description

Use `ECGetRealIdentifierValue` to get the name of a real identifier. function. The identifier number must be positive and less than the number returned by `ECNumberOfRealIdentifiers`.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECGetUnsetIdentifierName

Purpose

Returns the name of an identifier that hasn't been set.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
identifier=ECGetUnsetIdentifierName(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`char`	**identifier*	Returned to user. Name of the requested identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
call ECGetUnsetIdentifierName(number, object, identifier, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`character`	*identifier*	Returned to user. Name of the requested identifier.
`real`	*object*	Specified by user. Object code to be evaluated.
`real`	*rc*	Returned to user. Return code.

Description

Use `ECGetUnsetIdentifierName` to get the name of an undefined identifier. The identifier number must be positive and less than the number returned by `ECNumberOfUnsetIdentifiers`.

The C interface returns a point to the character string in the symbol table. It is only valid until the "object code" is free'd, and should not be freed by the user.

For the Fortran interface, the name is truncated to the length of the character string provided. The string is blank filled, and is not zero terminated.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECIsIdentifierSet

Purpose

Queries an identifier to see if it is set.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
answer=ECIsIdentifierSet(number, object, &rc);
```

`int`	*number*	Specified by user. Number of the requested identifier.
`int`	*answer*	Returned to user. 1=identifier set. 0=identifier not set.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried.
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
answer=ECIsIdentifierSet(number, object, rc)
```

`integer`	*number*	Specified by user. Number of the requested identifier.
`integer`	*answer*	Returned to user. 1=identifier set. 0=identifier not set.
`real`	*object*	Specified by user. Object code to be evaluated.
`real`	*rc*	Returned to user. Return code.

Description

Use `ECIsIdentifierSet` to determine if a particular identifier has been given a value. The identifier number must be positive and less than the number returned by `ECNumberOfIdentifiers`.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	The requested identifier doesn't exist.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECNumberOfFunctionIdentifiers

Purpose

Returns the number of function identifiers in an "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
number=ECNumberOfFunctionIdentifiers(object, &rc);
```

`int`	*number*	Returned to user. Number of identifiers.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
number=ECNumberOfFunctionIdentifiers(object, rc)
```

`integer`	*number*	Returned to user. Number of identifiers.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECNumberOfFunctionIdentifiers` to determine the number of identifiers in an "object code" that are functions.

Return Codes

EC_NO_ERROR	0	No error.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECNumberOfIdentifiers

Purpose

Returns the number of identifiers in an "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
number=ECNumberOfIdentifiers(object, &rc);
```

`int`	*number*	Returned to user. Number of identifiers.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
number=ECNumberOfIdentifiers(object, rc)
```

`integer`	*number*	Returned to user. Number of identifiers.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECNumberOfIdentifiers` to determine the number of identifiers in an "object code".

Return Codes

EC_NO_ERROR	0	No error.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failedcompilation.

ECNumberOfIntegerIdentifiers

Purpose

Returns the number of integer identifiers in an "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
number=ECNumberOfIntegerIdentifiers(object, &rc);
```

`int`	*number*	Returned to user. Number of identifiers.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
number=ECNumberOfIntegerIdentifiers(object, rc)
```

`integer`	*number*	Returned to user. Number of identifiers.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECNumberOfIntegerIdentifiers` to determine the number of identifiers in an "object code" that are integers.

Return Codes

EC_NO_ERROR	0	No error.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECNumberOfRealIdentifiers

Purpose

Returns the number of real identifiers in an "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
number=ECNumberOfRealIdentifiers(object, &rc);
```

`int`	*number*	Returned to user. Number of identifiers.
`struct ECObjectCode`	**object*	Specified by user. Object code to be queried
`int`	*rc*	Returned to user. Return code.

Fortran Syntax

```
number=ECNumberOfRealIdentifiers(object, rc)
```

`integer`	*number*	Returned to user. Number of identifiers.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

Use `ECNumberOfRealIdentifiers` to determine the number of identifiers in an "object code" that are reals.

Return Codes

EC_NO_ERROR	0	No error.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECNumberOfUnsetIdentifiers

Purpose

Returns the number of undefined identifiers in an "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
number=ECNumberOfUnsetIdentifiers(object, &rc);
```

`int`	*number*	Returned to user.	Number of identifiers.
`struct ECObjectCode`	**object*	Specified by user.	Object code to be queried
`int`	*rc*	Returned to user.	Return code.

Fortran Syntax

```
number=ECNumberOfUnsetIdentifiers(object, rc)
```

`integer`	*number*	Returned to user.	Number of identifiers.
`integer`	*object*	Specified by user.	Object code to be evaluated
`integer`	*rc*	Returned to user.	Return code.

Description

Use `ECNumberOfUnsetIdentifiers` to determine the number of identifiers in an "object code" that are undefined.

Return Codes

EC_NO_ERROR	0	No error.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECPrintObjectCode

Purpose

Prints a textual version of an "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
ECPrintObjectCode(object);
```

`struct ECObjectCode` *`*object`* Specified by user. Object code.

Fortran Syntax

```
call ECPrintObjectCode(object)
```

`integer` *`object`* Specified by user. Object code.

Description

Use `ECPrintObjectCode` to print a textual version of an object code. The text is written to `stdout`.

ECPrintSymbolTable

Purpose

Prints the symbol table of an "object code".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
ECPrintSymbolTable(object);
```

`struct ECObjectCode` **object* Specified by user. Object code.

Fortran Syntax

```
call ECPrintSymbolTable(object)
```

`integer` *object* Specified by user. Object code.

Description

Use `ECPrintSymbolTable` to print the symbol table of an "object code". The text is written to `stdout`.

ECSetIdentifier

Purpose

Uses an assignment statement in a string to set an identifier.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECSetIdentifier(assignment, object);
```

	`int`	*rc*	Returned to user. Return code.
	`char`	**assignment*	Specified by user. Assigment statement.
	`struct ECObjectCode`	**object*	Specified by user. Object code.

Fortran Syntax

```
call ECSetIdentifier(assignment, object, rc)
```

	`character`	*assignment*	Specified by user. Assigment statement.
	`integer`	*object*	Specified by user. Object code to be evaluated
	`integer`	*rc*	Returned to user. Return code.

Description

The routine `ECSetIdentifier` takes a string of the form "left=right". The identifier name is read from the string "left" (`sscanf(left,"%s",identifier)`). The "right" string is compiled, and evaluated to give the value to which the identifier is set with `ECSetIdentifierToReal`. For example, the assignment might be "x=5.4/3.1", or "x=3.*sin(-1.8*%pi)".

Return Codes

EC_NO_ERROR	0	No error.
EC_INVALID_ASSIGNMENT	17	Invalid assignment string.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFree ObjectCode, or was returned by a failed compilation.
EC_INVALID_IDENTIFIER_TYPE	20	Attempt to assign a function.

ECSetIdentifierToFunction

Purpose

Changes the type of an identifier to function, and sets the value.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECSetIdentifierToFunction(identifier, value, object);
```

`int`	*rc*	Returned to user.	Return code.
`char`	**identifier*	Specified by user.	Identifier to set.
`ECPointerToFunction`	*value*	Specified by user.	Value for identifier.
`struct ECObjectCode`	**object*	Specified by user.	Object code.

Fortran Syntax

```
call ECSetIdentifier(identifier, value, object, rc)
```

`character`	*identifier*	Specified by user.	Identifier to set.
`external`	*value*	Specified by user.	Value for identifier.
`integer`	*object*	Specified by user.	Object code to be evaluated
`integer`	*rc*	Returned to user.	Return code.

Description

The routine `ECSetIdentifierToFunction` sets the value of a function identifier.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	Identifier not found.
EC_IDENTIFIER_NOT_FUNCTION	16	Identifier is not used as a function.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECSetIdentifierToInteger

Purpose

Changes the type of an identifier to integer, and sets the value.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECSetIdentifierToInteger(identifier, value, object);
```

`int`	*rc*	Returned to user.	Return code.
`char`	**identifier*	Specified by user.	Identifier to set.
`int`	*value*	Specified by user.	Value for identifier.
`struct ECObjectCode`	**object*	Specified by user.	Object code.

Fortran Syntax

```
call ECSetIdentifierToInteger(identifier, value, object, rc)
```

`character`	*identifier*	Specified by user.	Identifier to set.
`integer`	*value*	Specified by user.	Value for identifier.
`integer`	*object*	Specified by user.	Object code to be evaluated
`integer`	*rc*	Returned to user.	Return code.

Description

The routine `ECSetIdentifierToInteger` changes the type of an identifier to integer, and sets the value.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	Identifier not found.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECSetIdentifierToReal

Purpose

Changes the type of an identifier to real, and sets the value.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECSetIdentifierToReal(identifier, value, object);
```

`int`	*rc*	Returned to user. Return code.
`char`	**identifier*	Specified by user. Identifier to set.
`float`	*value*	Specified by user. Value for identifier.
`struct ECObjectCode`	**object*	Specified by user. Object code.

Fortran Syntax

```
call ECSetIdentifierToReal(identifier, value, object, rc)
```

`character`	*identifier*	Specified by user. Identifier to set.
`real`	*value*	Specified by user. Value for identifier.
`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

The routine `ECSetIdentifierToReal` changes the type of an identifier to real, and sets the value.

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	Identifier not found.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECSetIdentifierToUndefined

Purpose

Changes the type of an identifier to "Undefined".

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECSetIdentifierToUndefined(identifier, object);
```

`int`	*rc*	Returned to user.	Return code.
`char`	**identifier*	Specified by user.	Identifier to set.
`struct ECObjectCode`	**object*	Specified by user.	Object code.

Fortran Syntax

```
call ECSetIdentifierToUndefined(identifier, object, rc)
```

`character`	*identifier*	Specified by user.	Identifier to set.
`integer`	*object*	Specified by user.	Object code to be evaluated
`integer`	*rc*	Returned to user.	Return code.

Description

The routine `ECSetIdentifierToUndefined` changes the type of an identifier to "Undefined".

Return Codes

EC_NO_ERROR	0	No error.
EC_IDENTIFIER_NOT_FOUND	12	Identifier not found.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECSetMessagePrint

Purpose

Returns a value indicating if error messages are printed.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
ECSetMessagePrint(value);
```

`int`	*value*	Specified by user. 0 if messages should not be printed. 1 if messages should be printed.

Fortran Syntax

`call ECSetMessagePrint(`*value*`)`

`integer`	*value*	Specified by user. 0 if messages should not be printed. 1 if messages should be printed.

Description

Use `ECSetMessagePrint` to determine the fate of error messages. This value defaults to 0 (no messages printed), and can be queried using `ECGetMessagePrint`. The return code of a command is always set regardless of the vaule given.

ECSetStandardMathConstants

Purpose

Sets some of the common mathematical constants.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECSetStandardMathConstants(object);
```

`int`	*rc*	Returned to user. Return code.
`struct ECObjectCode`	`*`*object*	Specified by user. Object code.

Fortran Syntax

```
call ECSetStandardMathConstants(object, rc)
```

`integer`	*object*	Specified by user. Object code to be evaluated
`integer`	*rc*	Returned to user. Return code.

Description

The routine `ECSetStandardMathConstants` gives values to the identifiers %pi and %e.

Return Codes

EC_NO_ERROR	0	No error.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

ECSetStandardMathFunctions

Purpose

Sets some of the common mathematical constants.

Library

`libExpCmp.a`

C Syntax

```
#include <ExpCmp.h>
rc=ECSetStandardMathFunctions(object);
```

	`int`	*rc*	Returned to user. Return code.
	`struct ECObjectCode`	**object*	Specified by user. Object code.

Fortran Syntax

```
call ECSetStandardMathFunctions(object, rc)
```

	`integer`	*object*	Specified by user. Object code to be evaluated
	`integer`	*rc*	Returned to user. Return code.

Description

The routine `ECSetStandardMathFunctions` gives values to the identifiers

Identifier	Function
`sin`	sin
`sinh`	hyperbolic sin
`asin`	arcsin
`cos`	cos
`cosh`	hyperbolic cos
`acos`	arccos
`tan`	tangent
`tanh`	hyperbolic tangent
`atan`	arctangent
`sqrt`	square root
`abs`	absolute value
`exp`	exponential
`log`	natural log
`log10`	log base 10

and their first derivatives (same names with "D" prepended).

Return Codes

EC_NO_ERROR	0	No error.
EC_NULL_OBJECT_CODE	18	Input object has been deleted by ECFreeObjectCode, or was returned by a failed compilation.

REFERENCES

[1] A. Griewank and G. Corliss, eds., *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, Philadelphia, 1991, SIAM.