Arguments

The arguments of a method consist of as many input arguments as there are terminals and as many output arguments as there are roots on the calling operation.

Each four byte input argument is either a handle to a Prograph data item or NULL.
Each output argument is the address of where a handle to a Prograph data item will be put. If the method does not set the output to a Prograph data handle, it must set it to NULL.

Supplied Functions

AddPrimitive

Adds a primitive to the Prograph interpreter's primitives table. In interpreted code AddPrimitive must be called from the main routine once for each external primitive. The first argument is the resource number of the primitive's associated 'STR#' resource calling operation when created in a Prograph program. The upper byte of the arity is the number of input terminals and the lower byte is the number of output roots. The third argument is a set of flags. All XPrims need to have the value PF_USER set as one of the flags. If the primitive has variable-arity, the flag PF_VAR should be set. If the calling operation is to be created with a default next-case control, the flag PF_CTRL should be set. Use a bit-or operation to specify multiple flags as shown in the example below. The fourth argument should always be zero. The last argument is the address of the primitive's code.

AddPrimitive and its arguments are discussed in more detail in the section "Writing XPrims" below.

Availability:

Interpreted XPrims

Syntax:

```
#include "X_includes.h"

void AddPrimitive( nameID, arity, flags, select, code )

Int2 nameID;
/* # of 'STR#' resource with primitive name & info */
Int2 arity;
  /* minimum arity of primitive - four digit hex
        value:  first two hex digits = input arity
        second two hex digits = output arity */
Int2 flags;
  /* PF_USER = user defined, PF_CTRL = with
        control, PF_VAR = variable arity */
Int2 select;
 /* not used. Should always be 0 */
ProcPtr code;
/* address of prim's code */
```

Example:

```
extern U_point_2D_in_2D_rect_3F_();
/* assume the primitive point-in-rect? is */
/* defined in a separate file */

void main()

 ...
 AddPrimitive( 3, 0x0200, PF_USER | PF_CTRL | PF_VAR, 0
          &U_point_2D_in_2D_rect_3F_);
 ...
```

Bless *537*

Assigns an data item to a particular class. Use this function for data items which have been read from a resource or a data file. Do not use Bless to change a data item's class arbitrarily. (Bless is not to be confused with the THINK function bless.)

Availability:

XCode

Syntax:

```
#include "X_includes.h"

void Bless(  object, class )

C_object *object;
/* object to bless (must be a Handle) */
 Class
*class;
/* new class for object */
         /* address of class identifier */
```

Example:

```
C_object *myObject;
/* assume myObject contains a handle */
Bless( myObject, &C_string__ );

...
```

CallPrimitive *537*

Calls a Prograph primitive by name in interpreted code. This function is unnecessary in compiled code, since primitives are called directly. Note that there is some overhead involved for the lookup in the table of primitives, and for copying the arguments into and out of the stack. Also, you are responsible for maintaining the use counts of any data items returned by the called primitive.

If a primitive of the given name is not found, CallPrimitive returns the value PRIMERR_NAME. If the call to the primitive fails for any other reason, one of the other PRIMERR values is returned. If the error

returned is PRIMERR_TYPE or PRIMERR_VALUE, the ordinal value of the offending input is encoded in the least-significant byte of the returned error value. If the value returned is PCF_FALSE or PCF_TRUE, the call to the primitive has succeeded.

Availability: "538"

Interpreted XPrims

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"
Int2 CallPrimitive( name, arity, args )

Nat1 * name;
/* pascal string holding the primitive's name */
Int2 arity;
/* actual # of input and output args passed */
Int4 args;
 /* place holder for the actual arguments */
```

Example 1:

```
C_real * value;
/* assume that value is a valid C_real */
C_real * cosine;
/* value returned by call */
Int2 error;
  /* result of the call to the primitive */

error = CallPrimitive( "\pcos", 0x0101, value, &cosine );

if ( error != PCF_TRUE)
 /* then there is something wrong with your primitive */
else

 /* use the cosine value and then clean up unneeded values */
 DecUse( value );
 DecUse( cosine );
```

Example 2: "538"

```
C_real * input1;
  /* a real to be formatted */
C_real * input2;
  /* a real to be formatted */
C_string * formString; /* format string,eg."\99.99\ \99.99\"*/
C_string * resString;
/* result of format operation */
Int2 error;
   /* result of the call to the primitive */

error = CallPrimitive( "\pformat", 0x0301, formString, input1,
```

```
          input2, &resString);

if ( error != PCF_TRUE)
 /* then there is something wrong with your primitive */
else

 /* formString is finished with */
 DecUse( formString );
 /* if resString is finished with and is not assigned to
   an output or to a list or instance slot */
 DecUse( resString );
```

DecUse "539"

Decrements the use field of the given data item by one and, if the use count is zero, releases the memory it occupies. Note that it is not the user's responsibility to destroy Prograph data items, but only to ensure that use counts are properly maintained. You should never decrement the use field directly.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

void DecUse( data )

C_object * data;
 /* the data item to be decremented */
```

Example:

```
C_string * myString;
    /* assume myString is "abc" and has a use count of 1 */
DecUse( myString );
    /* myString's use count will be decremented to
     0 & so its memory will be released */
```

Duplicate "539"

Copies any of the Prograph data types. The second parameter, SHALLOW_COPY or DEEP_COPY, controls the way in which Duplicate copies complex objects (lists and instances) and simple objects (everything else). A SHALLOW_COPY returns a copy of a simple object with a use count of one. If the object is complex, the object itself will be copied and the objects in its slots will have their use counts incremented. A DEEP_COPY will copy complex objects to arbitrary depth. Simple objects contained in the complex objects will not be copied, but their use counts will be incremented.
Copies of active instances of system classes will have their owner fields set to NULL. Active instances of subclasses of C_Menu, C_Window and C_Application will have their active? fields set to FALSE.

Availability: "539"

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_object * Duplicate( object, mode )

C_object * object;
 /* data to be copied */
CopyMode mode;
  /* DEEP_COPY or SHALLOW_COPY */
```

Example:

```
C_list * oldData, * newData;
 /* assume that oldData is NULL or a valid data item */
newData = Duplicate( oldData, SHALLOW_COPY );
 /* the list is copied but the items contained in the
   list just have  their use count incremented */
```

GetTypeName "540"

Takes a data item of any type and returns a C_string with its type, or in the case of an instance of a class, returns the class name. The C_string is returned with use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

void GetTypeName( object, string )

C_object * object;
/* a data item */
C_string ** string;
/* name of the data type or class */
```

Example:

```
C_string *myString;
C_object *myDataItem;
/* assume myDataItem to be an
      item of any arbitrary data type */
GetTypeName( myDataItem, &myString );
```

GetRefLevel "540"

Returns the level of indirection of an object: one if the object is a C_Ptr, two if it is a C_Handle, and zero otherwise.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
 #include "X_includes.h"

 Nat2 GetRefLevel( object )

 C_object *object;
/* the object to be tested */
```

Example:

```
 C_object *input;

 switch ( GetRefLevel( input ) )

  case 0:

/* input is direct  */
  case 1:
   /* input is a C_Ptr */
  case 2:
   /* input is a C_handle */
```

HasType

Determines whether a data item is a given data type, or an indirect reference to that type. For example, HasType(object, C_REGION) returns TRUE if object is a C_Region, a C_Ptr that points to a region, or a C_Handle that contains a handle to a region. Class identifiers are defined in X_struct_ids.h.

In XPrim code HasType's second parameter, the type, is of the form C_CLASSNAME (all capitals). In XCode the second parameter is the address of a class identifier and has the form &C_classname__ (mixed case).

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
 #include "X_includes.h"

 Bool HasType( object, type )

 C_object *object;
/* the object to be tested */
 Class
```

```
*type;
/* the type to check */
```

XPrim Example:

```
C_object *input;

if ( HasType( input, C_REGION ) )
 ...
  /* input is either a C_Region, or a C_Ptr or C_Handle which

     reference a region*/
```

XCode Example:

```
C_object *input;

if ( HasType( input, &C_Region__ ) )
 ...
  /* input is either a C_Region, or a C_Ptr or C_Handle which

     reference a region*/
```

INCLASS [542]

INCLASS is a macro which is used to determine the class of a data item. INCLASS returns TRUE if the data item's class is equal to the class specified. Unlike Member, INCLASS does not check the ancestors of the data item's class. In interpreted code IsType must be used in place of INCLASS.

Availability:

XCode

Example:

```
C_object *myObject;
/* Determine whether myObject is a real */

if ( INCLASS( myObject, &C_real__ ) )
  ...
else
  ...
```

IncUse [542]

Increments the use field of the given data item by one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

void IncUse( data )

C_object *data;
 /* the data item to be incremented */
```

Example:

```
C_string *myString;
  /* assume myString is "abc" and has a use count of 1 */
IncUse( myString );
  /* myString's use count will be incremented by 1 */
OR
(**myString).use++;
```

IsType

Determines whether an object is of a given type. In XCode format the macro INCLASS serves the same function.

In XPrim code IsType's second parameter, the type, is of the form C_CLASSNAME (all capitals). In XCode the second parmeter is the address of a class identifier and has the form &C_classname__ (mixed case).

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

Bool IsType( object, type )

C_object *object;
/* the object to be tested */
Class
 *type;
/* the type to check */
```

XPrim Example:

```
C_object *input;

if ( IsType( input, C_STRING ) )
 ...
  /* check to see if input is a string */
```

XCode Example:

```
C_object *input;
```

```
if ( IsType( input, &C_string__ ) )
 ...

  /* check to see if input is a string */
```

ListDeleteSlot[543]

Removes a slot from a C_list data item, thereby shortening the list, and optionally decrements the data item's use count.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"

void ListDeleteSlot( list, index, use )

C_list * list;
   /* the list involved */
Nat4 index;
     /* slot in the list; 1st index is 0 */
Bool use;
     /* TRUE = apply DecUse to
                any object in the deleted slot */
```

Example:

```
C_list * myList;
   /* assume myList is ( a b c ) */
ListDeleteSlot( myList, 0, TRUE );
   /* list is now ( b c ) */
```

ListEmptySlot[544]

Finds an empty slot (in other words, one whose value is NULL) in a C_list data item. If no empty slot is found, an empty slot is appended to the end of the list.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

void ListEmptySlot( list, index )

C_list * list;
```

```
  /* the list involved */
Nat4 * index;
   /* 0 based index to the empty slot */
```

Example:

```
C_list * myList;
Nat4 index;
  /* assume myList is ( a b c ) */
ListEmptySlot( myList, &index );
  /* list is now ( a b c NULL ) and index = 3 */
```

ListGetSlot[544]

Returns the contents of the given slot, and optionally increments the use count on the data item returned.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"

void ListGetSlot( list, index, use, item )

C_list * list;
 /* the list involved */
Nat2 index;
  /* 0 based index to the slot */
Bool use;
   /* TRUE = apply IncUse to the item */
C_object ** item;
/* address of the data item or NULL
            if slot empty */
```

Example:

```
C_list * myList;
C_string * myString;
  /* assume myList is ( a b c ) */
ListGetSlot( myList, 1, TRUE, &myString );
  /* list is still ( a b c ), myItem is the string b now
   with a use count of 2 */
```

ListInsertSlot[545]

Adds a slot to a C_list data item (thereby lengthening the list), assigns the given data item to the slot, and optionally increments the data item's use count.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"

void ListInsertSlot( list, index, use, item )

C_list * list;
/* the list involved */
Nat4 index;
 /* 0 based index to the slot */
Bool use;
  /* TRUE = apply IncUse to the item */
C_object * item;
/* the data item to be inserted */
```

Example:

```
C_list * myList;
C_string * myString;
  /* assume myList is ( a c ) */
myString = MakeC_string( "\pb" );
ListInsertSlot( myList, 1, FALSE, myString );
 /* list is now ( a b c ), the use parameter was FALSE
   since the C_string is created with a use of 1 and
   nothing else points to it */
```

ListSetLength *545*

Changes the number of slots in the given list. Slots can be added or deleted. This function does not adjust the use count on items in deleted slots. Slots that are added are initialized to NULL.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"

void ListSetLength( list, newLength )

C_list * list;
 /* the list involved */
Nat4 newLength;
/* total # of slots list should have */
```

Example:

```
C_list * myList;
  /* assume myList is ( a b c ) */
```

```
ListSetLength( myList, 6 );
  /* list is now ( a b c NULL NULL NULL ) */
```

ListSetSlot "546"

Sets the contents of the given slot, optionally decrements the use count on the data item that was in the slot, and optionally increments the use count on the data item that is set in the slot.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"

void ListSetSlot( list, index, oldUse, newUse, item )

C_list * list;
/* the list involved */
Nat4 index;
 /* 0 based index to the slot */
Bool oldUse;
/* TRUE = apply DecUse to item in slot */
Bool newUse;
 /* TRUE = apply IncUse to new item */
C_object * item;
/* the data item to set in the slot */
```

Example:

```
C_list * myList;
C_string * myString;
  /* assume myList is ( a b c ) */
myString = MakeC_string( "\px" );
ListSetSlot( myList, 1, TRUE, FALSE, myString );
  /* myList is now ( a x c ), newUse = FALSE since myString is
     made with a use count of 1 */
```

ListStretch "546"

Expands the first list to include all items of the second list, and optionally increments the use count of the added items.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
```

```
#include "X_includes.h"

void ListStretch( list1, list2, use )

C_list * list1;
/* the list to stretch */
C_list * list2;
/* the list whose elements will be added */
Bool use;
 /* TRUE = apply IncUse to added items */
```

Example:

```
C_list * myList1, * myList2;
  /* assume myList1 & 2 are (a b c) (d e) */
ListStretch( myList1, myList2, TRUE );
  /* myList1 is now (a b c d e) myList2 is still (d e)
  and the use counts of items d & e have been incremented */
```

MakeC_boolean[547]

Creates a C_boolean data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_boolean * MakeC_boolean( value )

Bool value;
    /* FALSE = 0, TRUE = 1 */
```

Example:

```
C_boolean * myBool;
myBool = MakeC_boolean( FALSE );
```

MakeC_integer[547]

Creates a C_integer data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"
```

```
C_integer * MakeC_integer( value )

Nat4 value;
    /* a four byte signed long integer */
```

Example:

```
C_integer * myInt;
myInt = MakeC_integer( 12345 );
```

MakeC_list "548"

Creates a C_list data item with the given number of slots, with a use count of one.
Each slot is initialized to NULL.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_list * MakeC_list( slots )

Nat4 slots;
/* the # of slots to make in the list */
```

Example:

```
C_list * myList;
myList = MakeC_list( 10 );
```

MakeC_none "548"

Creates a C_none data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_none * MakeC_none( )
```

Example:

```
C_none * myNone;
myNone = MakeC_none( );
```

MakeC_real[548]

Creates a C_real data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_real * MakeC_real( value )

Real10 value;
    /* a ten byte floating point value */
```

Example:

```
C_real * myReal;
myReal = MakeC_real( 1.2345 );
```

MakeC_string [549]

Creates a C_string data item, with a use count of one, from a Pascal formatted string. Use StrFromText to create a C_string data item from a character string and a size.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_string * MakeC_string( text )

Nat1 * text;
  /* address of size byte of Str255*/
```

Example:

```
C_string * myString;
myString = MakeC_string( (Nat1 *) "\pYo Ho Ho !" );
```

MakeC_undefined[549]

Creates an C_undefined data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_undefined * MakeC_undefined( );
```

Example:

```
C_undefined * myUndef;
myUndef = MakeC_undefined( );
```

MakeC_Handle "550"

Creates a C_Handle data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_Handle * MakeC_Handle( class, handle )

Class class;
    /* class of object pointed to */
Handle handle;
   /* Macintosh handle */
```

Example:

```
Region ** myRegion;
C_Handle
* myHandle;
  /* assume myRegion contains a handle to a Region */
myHandle = MakeC_Handle( C_REGION, myRegion);
```

MakeC_Point "550"

Creates a C_Point data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "QuickDraw.h"
#include "X_includes.h"

C_Point * MakeC_Point( v, h )

Int2 v;
      /* vertical coordinate */
Int2 h;
      /* horizontal coordinate */
```

Example:

```
C_Point * myPoint;
myPoint = MakeC_Point( 2, 4 );
```

MakeC_Ptr "551"

Creates a C_Ptr data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_Ptr * MakeC_Ptr( class, pointer )

Class class;
    /* class of object pointed to */
Ptr pointer;
    /* Macintosh pointer */
```

Example:

```
Region * myRegion;
C_Ptr
* myPtr;
  /* assume myRegion contains a pointer to a Region */
myPtr = MakeC_Ptr( C_REGION, myRegion);
```

MakeC_Rect "551"

Creates a C_Rect data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "QuickDraw.h"
#include "X_includes.h"

C_Rect * MakeC_Rect( top, left, bottom, right )

Int2 top;
    /* top vertical coordinate */
Int2 left;
    /* left horizontal coordinate */
Int2 bottom;
   /* bottom vertical coordinate */
Int2 right;
    /* right horizontal coordinate */
```

Example:

```
C_Rect * myRect;
myRect = MakeC_Rect( 2, 4, 6, 8 );
```

MakeC_RGBColor -552-

Creates a C_RGBColor data item with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "QuickDraw.h"
#include "X_includes.h"

C_RGBColor * MakeC_RGBColor( red, green, blue )

Nat2 red;
    /* red color */
Nat2 green;
    /* green color */
Nat2 blue;
    /* blue color */
```

Example:

```
C_RGBColor * myRGB;
myRGB = MakeC_RGBColor( 0x0001, 0xF000, 0x1234 );
```

Make_instance -552-

Creates a data item of a user defined class, with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_object * Make_instance( classname )

C_string * classname;
  /* class name */
```

Example:

```
C_object * myInst;
C_string
* myClassName;
  /* assume myClassName contains the name of a prograph class */
myInst = Make_instance( myClassName);
```

Member[553]

Determines whether a data item belongs to a particular class. Member returns TRUE if the data item's class or any of its ancestors is equal to the class specified. (Member is not to be confused with the THINK function member.)

Availability:

XCode

Syntax:

```
#include "X_includes.h"

Bool Member(  object, class )

C_object *object;
/* object to test */
  Class
 *class;
/* class to test */
         /* address of class identifier */
```

Example:

```
C_object *myObject;
/* Member returns TRUE if myObject's class is */
/* a descendant of  C_number */

if ( Member( myObject, &C_number__ ) )
   ...
```

New[553]

Creates an data item of the specified class. New sets the use count of the new data item to one. (New is not to be confused with the THINK function new.)

If New is used to create an instance of a user defined class the attributes of the instance will not be initialized; that is, they will contain garbage values. To create an instance of a user defined class which is initialized to its default value make a deep copy of the default instance with the Duplicate function. The default instance of a user defined class can be found in a global variable of the form C_class__A_default.

Availability:

XCode

Syntax:

```
#include "X_includes.h"

C_object *New( class )

 Class *class;
 /* class of new object */
        /* address of class identifier */
```

Example:

```
C_Person *myPerson;
/* Assume the class C_Person has been defined to have */
/* three attributes */

myPerson = (C_Person *) New( &C_Person__ );

(**myPerson).name = MakeC_string( "\PFred Nitnee" );
(**myPerson).age
= MakeC_integer( 32 );
(**myPerson).address = NULL;
 ...
```

NewN

Used to create data items of class C_list, C_string or C_ABlock. All of these classes have a four-byte length field directly after the use field. The length field is set by NewN.

Availability:

XCode

Syntax:

```
#include "X_includes.h"

C_object *NewN( class, length )
```

```
 Class *class;
 /* class of new object */
        /* address of class identifier */
Nat4 length;
 /* for C_list:   number of slots */
        /* for C_string:   number of characters */
```

Example:

```
C_list *myList;
/* Create a list with 12 slots */

myList = (C_list *) NewN( &C_list__, 12 );
...
```

NumToStr [554]

Converts a number to a C_string, and if stretch is TRUE, adds it to an existing C_string passed in myStr; otherwise, it creates a new C_string, with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"
#include "MacTypes.h"

void NumToStr( number, stretch, myStr )

Int4 number;
   /* number to convert */
Bool stretch;
   /* stretch or create, TRUE or FALSE */
C_string ** myStr;
 /* C_string input or to return */
```

Example:

```
C_string *myStr;
  /* assume myStr has the value "The number is " */
NumToDtxt( 12345, TRUE, &myStr );
  /* myStr is now "The Number is 12345" */
```

ResToStr [555]

Retrieves a resource of type 'STR ' from the current resource file and converts it to a C_string. If stretch is TRUE, it is added to an existing C_string passed in string; otherwise it creates a new C_string with a use count of one.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"
#include "MacTypes.h"

void ResToStr( resNum, stretch, string )

Int2 resNum;
  /* STR resource ID number */
Bool stretch;
  /* stretch or create, TRUE or FALSE */
C_string **string;
/* C_string input or to return */
```

Example:

```
C_string *myStr;
ResToStr( 128, FALSE, &myStr);
  /* the value of myStr is now whatever STR resource 128 is */
```

StrFromText "555"

Creates a C_string data item, with a use count of one, from a character string and a size. Use MakeC_string to create a C_string data item from a Pascal formatted string.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

C_string * StrFromText( text, size )

Nat1
* text;
  /* address of 1st character of text */
Nat4
size;
   /* number of characters in text */
```

Example:

```
C_string * myString;
myString = StrFromText( (Nat1 *) "Yo Ho Ho !", 10 );
```

StrJoin "556"

Expands the C_string to include the second C_string.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "X_includes.h"

void StrJoin( string1, string2)

C_string * string1;
/* the string to stretch */
C_string * string2;
/* the string which will be added */
```

Example:

```
C_string * myString1, * myString2;
 /* assume myString1 is "Hello " and myString2 is "sailor!" */
StrJoin( myString1, myString2 );
 /* myString1 is now "Hello sailor!" myString2 is unchanged */
```

StrStretchString [556]

Expands a C_string data item to include arbitrary text.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"

void StrStretchString( string, text, textSize)

C_string * string;
 /* the string to stretch */
Nat1 * text;
   /* address of 1st char of text to add */
Nat4 textSize;
  /* # of bytes of text to add */
```

Example:

```
C_string * myStr;
Nat1 * myText;
  /* assume myStr is "Hello " myText points to "sailor!"
StrStretchString( myStr, myText, 7 );
  /* myString is now "Hello sailor!", myText is unchanged */
```

StrToPstring [557]

Copies a maximum of max bytes from a C_string into a given character array, setting the size byte appropriately.

Availability:

Interpreted XPrims, Compiled XPrims, XCode

Syntax:

```
#include "MacTypes.h"
#include "X_includes.h"

void StrToPstring( dtxt, string, max )

C_string * string;
/* the string to copy */
Nat1 * Pstring;
 /* address of the array */
Nat2 max;
   /* number of bytes available */
```

Example:

```
C_string *myStr;
Str255 myStr255;
  /* assume myStr is "slithy toves" */
StrToPstring( myStr, (Nat1 *) myStr255, 255 );
  /* myStr255 is now "\pslithy toves" myStr is unchanged */
```