

```
#import <appkit/appkit.h>
```

```
// Written by: Jeff Martin (jmartin@bozell.com)
// You may freely copy, distribute and reuse the code in this example.
// Don't even talk to me about warranties.
```

```
// Modified subtly & extensively by RK, 10/15-10/18 94.
// See TextUndo testbed project for details.
```

```
@protocol UndoDelegate
//
// <UndoDelegate> Protocol
//
- undoManagerWillUndo:sender;
- undoManagerDidUndo:sender;
```

@end

[illegible]

```
// Instance Variables
```

//

```
id      undoList;           // The list that holds undo records
id      redoList;          // The list that holds redo records
int     disabled;          // Whether the UndoManager is accepting events
BOOL    undoing;           // Whether the UndoManager is currently undoing
BOOL    redoing;           // Whether the UndoManager is currently redoing
BOOL    recordGrouping;    // Whether UndoRecords are being grouped
int     levelsOfUndo;     // How many levels of undo/redo to record

id      target;            // Current target of registered undo messages
id      delegateList;      // List of objects to be notified of UM changes
unsigned int freeArgsMask; // Stores which args to free
unsigned int copyArgsMask; // Make undo manager copy pointer args
NXAtom  actionName, currentActionName;
}
```

////////////////////////////////////

```
// Format of an undo/redo record
```

//

```
typedef struct UndoRecord {
    marg_list  args;
    int        freeArgsMask;
    int        argSize;
} UndoRecord;
```

```
typedef struct RecordGroup {
    Storage      *recordList;
    NXAtom      actionName;
} RecordGroup;
```

```
- init;
```

```
// Grouping multiple UndoRecords into an undo event
//
- beginUndoRecordGrouping;
- endUndoRecordGrouping;
```

```
//
// Disable/Reenable UndoManager to prevent events from being added to undo list
//
- disableUndoRegistration;
- reenableViewRegistration;

//
// Setting the current target for events that are received
//
- setUndoTarget:object;

//
// Setting the current name of RecordGroup
//
- setActionName:(const char *)aName;

//
// Querying the Undo/Redo status for menu validation
//
```



```
// ~~~~~  
// Setting the target or args of the next registered method to be freed when  
// they are executed  
//  
- freeUndoTargetOnRecordExecute;  
- freeUndoArgsOnRecordExecute;  
- freeUndoArgOnRecordExecuteAt: (int) pos;  
  
// ~~~~~  
// Make UndoManager copy arguments(like objects or strings) for convenience  
//  
- copyUndoArgs;  
- copyUndoArgAt: (int) pos;  
  
// ~~~~~  
// Make UndoManager copy arguments and free them when record is discarded  
//  
- copyUndoArgsFreeOnDiscard;  
- copyUndoArgFreeOnDiscardAt: (int) pos;
```

```
//
// Make UndoManager copy arguments and free them when record is executed
//
- copyUndoArgsFreeOnExecute;
- copyUndoArgFreeOnExecuteAt:(int)pos;

//
// Copies the pointer args in undoRecord as requested by copyArgsMask
//
- copyUndoArgsForRecord:(UndoRecord *)undoRecord;

//
// Overridden to capture undo/redo messages to be added to current record
//
- forward:(SEL)aSelector :(marg_list)argFrame;

//
// Removes a record from the undo/redo list and dispatches the messages in it.
//
- undo:sender;
```

```
- redo: sender;
```

[illegible]

```
// Query and set the maximum length of the undo/redo list
```

//

```
- (int) levelsOfUndo;
```

```
- setLevelsOfUndo: (int) value;
```

[illegible]

```
// These methods add and remove objects that are to receive undo notification.
```

//

```
- addUndoDelegate:object;
```

```
- removeUndoDelegate:object;
```

```
- sendNotification: (SEL) action;
```

[illegible]

```
// Used internally to free the space used for an undo/redo groups and records
```

//

```
- discardRecordGroup: (RecordGroup *) group;
```

```
- executeRecordGroup: (RecordGroup *) group;
```


