Edit Field Library API Reference

1. Introduction

These notes provide a programmer's reference for the EditFields library, a code library that implements editable text fields in any window, which work similarly to editable fields in dialogs, except that because they can be used in any window, they are more flexible. The library described here contains relatively low-level code routines. It is envisaged that a higher level resource based code library which is built on this one will be written shortly.

2. API code level interface.

```
EditFieldHdl	NewEditField(Rect *bounds,int IDnum,Str255 *theText);
```

Creates a new edit field with the given ID number, bounding rectangle and string. The bounds rect is in the window's local coordinates, and will be used for hit testing the field, etc. Field type is set to fieldNoRestriction, which is a free-form field. You must link a field into a set of fields after creation.

```
void		LinkEditField(EFListHdl efList,EditFieldHdl eField);
```

Links a newly created edit field into the field set *efList,* which must also have been created. Fields are stored in memory in a linked list structure.

```
void		Draw1Field(EditFieldHdl theField);
```

Draws the given field in the current port within its bounding rect. n.b. When drawn, fields are framed 3 pixels outside the boundsRect. This is identical to the dialog manager's behaviour for dialogs. When declaring a field's boundsrect, allow for the extra space.

```
void		SetFieldText(EditFieldHdl theField,Str255 *theText);
```

Sets the given field's text to the string passed. This only changes the field's data, it does not cause a change to the field on screen. It also sets the fieldDataChanged flag in the field.

```
void		GetFieldText(EditFieldHdl theField,Str255 *theText);
```

Returns the field's text string. If this is the current field, the text may be out of date if an editing change has taken place, in which case you need to update the field data before fetching it.

```
void SetFPField(EditFieldHdl theField,extended fpValue,int nDecPlaces);
```

Sets the given field to a string representation of the given floating point value, with nDecPlaces decimal places.

```
extended        GetFPField(EditFieldHdl theField);
```

Returns a floating point value which is the given field's text converted to fp format.

```
void            SetEdFlags(EditFieldHdl theField,long theFlags);
```

Sets the field's flags to the given value.

```
long            GetEdFlags(EditFieldHdl theField);
```

Returns the current flags from the field.

```
EFListHdl       NewEditFieldList(WindowPtr ownerWindow);
```

Creates a new, fully initialised edit field set with the given window as its 'owner'. This call initialises a TextEdit record for editing the text, but does not create any fields itself. You create fields with *NewEditField* and link them to the field set with *LinkEditField*.

```
void            DisposeEditFieldList(EFListHdl theList);
```

Disposes of an edit field set and any fields linked to it.

```
EditFieldHdl    GetIndexedField(EFListHdl efList,int theIndex);
```

Returns the handle to the field with the given ID number, or NIL if none.

```
void            IncDecCurrentField(EFListHdl efList,Boolean incDecFlag);
```

Increments or decrements the current field in a set by one. The field type is taken into account, so that time value fields are correctly adjusted. This routine applies to numeric type fields only.

```
void            DrawEditFields(EFListHdl efList);
```

Updates all edit fields in a set. The current field is updated by calling TEUpdate, and all others by calling Draw1Field. Normally called when needed as part of a window's update procedure.

```
void            SelectEditField(EFListHdl efList,int theIndex);
```

Changes the current edit field to the field with the given index number. The current field, if different, is updated before switching, so any editing changes made to the field will be correctly recorded.

```
int             FindEditField(EFListHdl efList,Point mClick);
```

Returns the ID number of the edit field containing the given point, or zero if none do. Normally called for a mouse-down event to determine which filed, if any, was clicked.

```
void		EFIdle(EFListHdl efList);
```

Call repeatedly during your main loop so that the caret in the current field is blinked correctly.

```
int		CheckFieldLimits(EditFieldHdl theField,EFListHdl efList);
```

For a numeric field, the current value is checked against its bounds value, and pinned accordingly. Normally called when switching the current field, this function returns FALSE if a value had to be pinned- in this case the text is set to the pinned value and highlighted. If no pinning was necessary, the function returns TRUE. You should NOT SWITCH to a new field if a value of FALSE was returned, as the user should be given the opportunity to correct the value before moving on.

```
void		KeyField(EFListHdl efList,char theKey);
```

Passes the given key character to the current field, filtering out characters that cannot be accepted. For example, numeric field types will not accept alphabetic characters. The keys used to change fields, for example TAB, are not handled here and should already have been filtered out before calling this function.

```
void		SetAndUpdateField(EFListHdl efList,int theFieldID,long fValue);
```

Sets the field with the given ID to the value passed, updating on screen as necessary.