Dokumentation der Änderungen FIX und FIX/Win

Stand: 05.10.11

Dieses Dokument dient als Zusatz zu den Handbüchern von FIX 3.1.0 und FIX/Win 3.1.0. Es beschreibt die Änderungen von FIX und FIX/Win in den Versionen 4.0.0, 4.1.0, 4.2.0 und 4.5.0.

Version 4.5.0

Allgemeine Änderungen

1 Kompatibilität

FIX/Win und FIX der Versionen 4.5.0, 4.2.0, 4.1.0, 4.0.0 und 3.1.0 sind zueinander kompatibel. Das bedeutet, dass die Version von FIX/Win nicht mit der Version von FIX übereinstimmen muss. Allerdings stehen dann die mit der Version eingeführten Features nicht vollständig zur Verfügung.

2 Standard-C

Sämtliche Prototypen in den Headerdateien unterstützen ab Version 4.5.0 nur noch Standard-C. Die K&R Prototypen wurden entfernt.

3 Neue Meldungen

Folgende Meldungen sind in fix_msg in der Version 4.5.0 neu:

```
10222 : "Objekt zu id %ld nicht gefunden"
12033 : "Rahmen <%d/%d-%d/%d> l\366schen"
13312 : "kein Speicher f\374r frames-Vektor verf\374gbar"
13313 : "kein Speicher f\374r frames-Struktur verf\374gbar"
13316 : "Frames-Limit pro Maske erreicht. MaxFramesOnMask erh\366hen"
10490 : "_Suchen_|Suchen|Suche starten"
10491 : "_OK_|OK|Satz ausw\344hlen und \374bernehmen"
10492 : "_Abbruch_|Abbruch|Auswahl abbrechen"
10493 : "Nochmal|Nochmal|Suche neu starten"
10494 : "Suche"
```

Folgende Meldungen haben sich in der Version 4.5.0 geändert:

```
12020 : "akt. Feld positionieren|akt. Paintarea positionieren| ..."
```

led mit mehreren FXHOME Verzeichnissen

Der Layouteditor led kann jetzt zwischen verschiedenen Anwendungsverzeichnissen (FXHOME) umschalten. Dazu ist wie üblich ein Startskript zu erstellen. In diesem Startskript muss nicht (darf jedoch schon) FXHOME gesetzt werden. Statt dessen muss vor dem Start des Binaries led in ein Verzeichnis gewechselt werden, in dem sich die Datei

```
ledfxhomes.fix
```

befindet. Diese Datei definiert die Verzeichnisse, die als FXHOME ausgewählt werden können. Dabei ist die Form

```
<Name>=<Verzeichnis>
```

für eine mögliche Definition zu wählen. Jede Zeile legt damit ein mögliches Verzeichnis fest. <Name> definiert den symbolischen Namen, der im ersten Feld der led-Auswahlmaske angezeigt wird und über choice ausgewählt werden kann. <Verzeichnis> definiert das Verzeichnis, das bei der Auswahl als FXHOME für die Dateiauswahl und für die Bearbeitung verwendet wird. Das Verzeichnis wird auch im ersten Feld als Tooltip verwendet. <Name> darf höchstens 30 Zeichen lang sein und <Verzeichnis> höchstens 256 Zeichen.

Neben diesen Definitionen kann die Datei Kommentarzeilen enthalten, die durch ein #-Zeichen an der ersten Position zu markieren sind. Beim Start wird die erste Definition als Default verwendet.

Als Startskript kann je nach Betriebssystem (Windows oder UNIX) eine der Dateien

```
fwled.sample
fwled.cmd.sample
```

als Vorlage verwendet werden.

Neben dieser neuen Möglichkeit kann der led auch gestartet werden, ohne das die Datei ledfxhomes.fix im aktuellen Verzeichnis existiert. In diesem Fall muss - wie vor dieser Änderung - FXHOME gesetzt werden. Der Wert wird als einzige Auswahlmöglichkeit in das neue erste Feld der Auswahlmaske geschrieben. Damit kann ein bestehendes Startskript ohne Änderung weiterverwendet werden.

Zur Einstellung des led wird wie bisher die Datei fix.rc aus FXHOME, wenn vorhanden oder aus FXDIR gelesen. Im Unterschied zu vorher wird jedoch das Lesen aus FXHOME unterlassen, wenn FXHOME nicht gesetzt ist. In diesem Fall sind alle Einstellungen des led in $FXDIR/fix.rc vorzunehmen. Dabei ist es sinnvoll - wie auch schon in der ausgelieferten Datei - den Ressourcen, die den led betreffen, den Programmnamen led voranzustellen. Ein erneutes Lesen der Ressourcen beim Wechsel von FXHOME innerhalb des led erfolgt nicht.

Im Zuge dieser Änderung ist die Ressource

```
led.load2: TRUE
```

weggefallen. Die damit einstellte Maske zur Verwendung der inkrementellen Suche wird jetzt immer verwendet. Die Verwendung der alten Maske kann nicht mehr eingestellt werden.

Selos mit Buttons und Suchfeld

Mit Hilfe dieses Features ist es möglich, Selos mit Buttons und einem Suchfeld zu versehen. Damit ein Selo dieses Feature nutzen kann, müssen folgende Bedingungen erfüllt sein:

- Die Anzahl der Suchfelder (Felder in der restrict Klausel) darf maximal 1 sein. Selos mit mehreren Feldern verhalten sich wie bisher.
- Das Selo muss die SQL-Schreibweise verwenden. Selos in CISAM-Schreibweise sind umzustellen, um Suchfelder und Buttons zu nutzen.
- Das Feature muss eingeschaltet werden

1 Aktivierung

Zur Aktivierung des Features ist die Ressource

```
UseExtSelos: FALSE
```

in fix.rc auf TRUE zu setzen.

2 Layout des Selos

In der obersten Zeile des Selos wird das Suchfeld hinter einem Bezeichner platziert. Darunter wird eine Linie gezeichnet, die aus Grafikzeichen mit dem Code "-" besteht.

Die Ressource

```
SeloMinSearchfield: 30
```

bestimmt die Mindestbreite eines Suchfeldes, auf die es gekürzt werden darf, wenn es zu lang ist. Bei langen Feldern vom Typ FXCHARTYPE wird versucht, die Displaylänge soweit zu reduzieren, dass das Feld hinter die Bezeichnung passt. Wenn dabei die festgelegte Mindestlänge unterschritten würde, dann wird wird das Feld nicht weiter gekürzt. Statt dessen wird das Selo selbst und die letzte Spalte der Ergebnistabelle verbreitert.

Wenn ein Selo keine restrict-Klausel besitzt, dann entfallen die oberen beiden Zeilen mit dem Suchfeld.

In der letzten Zeile werden die Buttons "Suchen", "OK" und "Abbrechen" zur Steuerung untergebracht. Über dieser Zeile wird ebenfalls eine Linie aus Grafikzeichen mit dem Code "-" gezeichnet. Das Selo wird also im oberen und im unteren Bereich um jeweils zwei Zeilen erweitert.

3 Beschriftung des Suchfeldes

Die Beschriftung des Suchfeldes wird anhand des Datenbankspaltennamens der restrict-Klausel und der Namen der Ergebnisspalten bestimmt. Wird zwischen dem Namen der restrict-Klausel und dem einer Ergebnisspalte eine Übereinstimmung gefunden, dann wird die Überschrift der Ergebnisspalte als Text für das Suchfeld verwendet. Leerzeichen am Anfang und am Ende werden entfernt.

Beispiel

```
selo
     select artiknr, artiknam, preis
     from art;

     restrict artiknr >= ?POS_ARTIKNR;

     artiknr "Artikel" "%7ld" #POS_ARTIKNR,
     artiknam "Bezeichnung         " "%s" #POS_BEZEI,
     preis  "   Preis" "%8.2f" #POS_PREISEINZEL
end
```

Der Text der Ergebnisspalte artiknr ("Artikel") wird auch als Beschriftung des Suchfeldes verwendet.

Alternativ kann der Text auch hinter der restrict-Klausel in geschweiften Klammern angegeben werden. Dazu können bis zu drei durch | getrennte Texte angegeben werden. Der erste wird als Token, der zweite als Text und der dritte als Tooltip verwendet.

Beispiel

```
selo
     select artiknr, artiknam, preis
     from art;

     restrict artiknr >= ?POS_ARTIKNR{__ART__|Artikel|Nummer des Artikels};

     artiknr "Artikel" "%7ld" #POS_ARTIKNR,
     artiknam "Bezeichnung         " "%s" #POS_BEZEI,
     preis  "   Preis" "%8.2f" #POS_PREISEINZEL
end
```

"__ART__" wird als Token verwendet und bestimmt damit die Breite. "Artikel" wird als Text verwendet und "Nummer des Artikels" als Tooltip.

Alle drei Texte können in dieser Schreibweise auch in die Meldungsdatei der Anwendung geschrieben werden. Im Selo ist dann die Angabe {#<Nummer>} zu machen.

Wenn kein Text im Selo angegeben wurde und auch eine Zuordnung nach dem obigen Verfahren nicht gelingt, dann wird ein Defaulttext aus der FIX-Meldungsdatei verwendet:

```
10494 : "Suche"
```

Bei Selos mit procedure-Aufrufen gibt es keine restrict-Klausel. Der Text für die Beschriftung des Suchfeldes ist deshalb in geschweiften Klammen hinter dem Suchfeld in der Parameterliste anzugeben. Fehlt diese Angabe, dann wird auch hier der Text aus der FIX-Meldungsdatei verwendet.

Beispiel

```
selo
execute procedure getartikel(#POS.POS_ANZ, ?POS.POS_ARTIKNR{#270})
     into sel_artiknr long, sel_artiknam char(60), sel_preis decimal(16,2);
sel_artiknr "Artikel" "%7ld" #POS_ARTIKNR,
sel_artiknam "Bezeichnung         " "%s" #POS_BEZEI,
sel_preis  "   Preis" "%8.2f" #POS_PREISEINZEL
end
```

4 Texte für die Buttons

In ähnlicher Schreibweise wie der Bezeichner des Suchfeldes können die Texte für die Buttons als Liste von drei durch | getrennten Texten definiert werden. Da die Texte für jedes Selo gleich sind, werden sie in der FIX-Meldungsdatei hinterlegt und können bei Bedarf angepasst werden.

```
10490 : "_Suchen_|Suchen|Suche starten"
10491 : "_OK_|OK|Satz ausw\344hlen und \374bernehmen"
10492 : "_Abbruch_|Abbruch|Auswahl abbrechen"
10493 : "Nochmal|Nochmal|Suche neu starten"
```

5 Angabe von Überschriften

Als Nebeneffekt dieses Features ist es möglich, das Selo mit einer Überschrift zu versehen, die am oberen Rand dargestellt wird. Die Überschrift ist hinter dem Objektnamen anzugeben. Auch hier ist es möglich durch #<Nummer> auf einen Text der Meldungsdatei zu verweisen.

Beispiel

```
selo ART #23456
     ....
```

6 Bedienung des Selos

Die Bedienung des Selos kann über zwei Ressourcen gesteuert werden.

```
EraseSeloSearchfield: TRUE
```

bestimmt, ob bei einer erneuten Suche das Suchfeld geleert (TRUE) wird oder ob der alte Wert erhalten bleibt (FALSE) und editiert werden kann.

```
LeaveSeloWithClick: TRUE
```

bestimmt, ob beim Anklicken einer Zeile der Wert übernommen wird und das Selo verlassen wird (TRUE) oder ob dazu ein Doppelklick notwendig ist (FALSE).

Von diesen Ressourcen sind alle Selos betroffen, auch die ohne Buttons, damit sie sich einheitlich verhalten.

Typsichere Funktionen zum Zugriff Memory Relation Spalten

Die im folgenden Funktionen dienen als Ersatz für die Funktion mr_colval().

```
int mr_set_char(MemRelType mrp, int n, TupelType tp, char *val);
int mr_get_char(MemRelType mrp, int n, TupelType tp, char *adrval);
int mr_set_graphics(MemRelType mrp, int n, TupelType tp, char *val);
int mr_get_graphics(MemRelType mrp, int n, TupelType tp, char *adrval);
int mr_set_truth(MemRelType mrp, int n, TupelType tp, char val);
int mr_get_truth(MemRelType mrp, int n, TupelType tp, char *adrval);
int mr_set_short(MemRelType mrp, int n, TupelType tp, short val);
int mr_get_short(MemRelType mrp, int n, TupelType tp, short *adrval);
int mr_set_long(MemRelType mrp, int n, TupelType tp, long val);
int mr_get_long(MemRelType mrp, int n, TupelType tp, long *adrval);
int mr_set_double(MemRelType mrp, int n, TupelType tp, double val);
int mr_get_double(MemRelType mrp, int n, TupelType tp, double *adrval);
int mr_set_float(MemRelType mrp, int n, TupelType tp, float val);
int mr_get_float(MemRelType mrp, int n, TupelType tp, float *adrval);
int mr_set_money(MemRelType mrp, int n, TupelType tp, double val);
int mr_get_money(MemRelType mrp, int n, TupelType tp, double *adrval);
int mr_set_decimal(MemRelType mrp, int n, TupelType tp, dec_t val);
int mr_get_decimal(MemRelType mrp, int n, TupelType tp, dec_t *adrval);
int mr_set_date(MemRelType mrp, int n, TupelType tp, long val);
int mr_get_date(MemRelType mrp, int n, TupelType tp, long *adrval);
int mr_set_datetime(MemRelType mrp, int n, TupelType tp, dtime_t val);
int mr_get_datetime(MemRelType mrp, int n, TupelType tp, dtime_t *adrval);
int mr_set_interval(MemRelType mrp, int n, TupelType tp, intrvl_t val);
int mr_get_interval(MemRelType mrp, int n, TupelType tp, intrvl_t *adrval);
```

Alle Funktionen bekommen als Parameter die Memory Relation (mrp), den Index der Spalte (n = 1 ... max) und das Tupel (tp).

Im Unterschied zu mr_colval() wird nicht die Adresse einer Zelle geliefert. Statt dessen wird ein übergebener Wert in die Zelle geschrieben (mr_set_...()) oder aus einer Zelle gelesen (mr_get_...()) und an einer Variablenadresse abgelegt. val definiert den Wert, der in das Tupel geschrieben werden soll. Wenn ein Spaltenwert gelesen wird, dann wird er an die Adresse adrval kopiert.

Die Typprüfung erfolgt an zwei Stellen. Zum einen muss der übergebene Wert den richtigen Typ besitzen. Das wird durch den Compiler anhand des Prototyps sichergestellt. Zum anderen muss der in der Memory Relation definierte Typ mit dem der Funktion übereinstimmen. Mit mr_set_long() kann beispielsweise nicht der Wert einer Zelle vom Typ double geändert werden. Diese Prüfung findet zur Laufzeit statt. Im Fehlerfall liefert die Funktion den Code TYPE_ERR zurück. Wenn das Setzen oder Auslesen der Zelle erfolgreich war, dann wird SUCCESS zurückgeliefert. Bei anderen Fehlern, wenn beispielsweise die Spalte oder das Tupel nicht existiert, wird LOGIC_ERR zurückgeliefert. Da es sich in beiden Fällen um einen Programmierfehler handelt, wird zusätzlich eine Fehlermeldung ausgegeben, so dass es nicht unbedingt notwendig ist, den Fehlercode auszuwerten. Es ist jedoch sinnvoll, den Fehlercode auszuwerten, wenn weitere Aktionen folgen, die ein Setzen/Lesen des Zellenwertes voraussetzen.

Eine Ausnahme bei der Typprüfung machen die Funktionen mr_get_double(), mr_set_double(), mr_get_money() und mr_set_money(). Sie können sowohl auf money-Spalten als auch auf double-Spalten angewendet werden.

Eine weitere Besonderheit besteht bei den Funktionen mr_set_char() und mr_set_graphics(). Bei mr_set_char() wird geprüft, ob der übergebene Wert nicht länger ist, als die Spalte. In diesem Fall wird LOGIC_ERR zurückgeliefert. Bei mr_set_graphics() muss die Länge des Wertes mit der Länge der Spalte übereinstimmen. Wenn die Bedingungen erfüllt sind, wird der Wert mittels strcpy() in die Spalte kopiert. Ein Überprüfung auf korrekte (UTF-8) Zeichen (wie beim Schreiben des gebundenen Variablenwertes) findet analog zu dem Verfahren mit mr_colval() nicht statt.

Beim Setzen eines Wertes wird die Änderungsmarkierung (mit isCellUpdateMarkSet() auslesbar) gesetzt. Ein Aufruf des Memrel-Listeners unterbleibt jedoch. Dieser Aufruf war auch bei der Verwendung von mr_colval() nicht möglich.

Zugriff auf Paintareas der pan-Datei

Beim Laden einer pan-Datei wird aus den dort beschriebenen Paintareas ein Vektor mit Daten gebildet, der an den Objektzeiger (obj->obj_paptr) gebunden wird. Wenn das Objekt angezeigt wird, dann werden anhand der Daten die Paintareas im Layout und auf dem Bildschirm erzeugt. Nach dem Entfernen des Objektes vom Bildschirm werden die Paintareas wieder aus dem Layout entfernt.

Die im folgenden beschriebenen Funktionen bieten die Möglichkeit, Manipulationen an dem Vektor, der aus der pan-Datei gebildet wird, vorzunehmen. So können Paintareas gelöscht, verändert oder hinzugefügt werden. Schlüssel zum Zugriff ist der Index innerhalb des Vektors.

```
int pan_add_pa(obj *objp, int type, int y1, int x1,
     char *token, char *text, char *tooltip,
     short attrib, short align, int bt_mask, long l1, l2, l3, l4);
```

Diese Funktion fügt dem Vektor des Objektes objp Daten für eine Paintarea vom Typ type an der Position y1, x1 mit dem Token token und dem Text text hinzu. Als Tooltip wird der in tooltip stehende Tooltip verwendet. attrib beschreibt das Attribut, align die Ausrichtung und bt_mask die möglichen Maustasten. In l1 bis l4 stehen die longval-Werte.

Anhand der gespeicherten Daten wird später mittels pa_put_type eine Paintarea erzeugt. Deshalb ist es möglich, für token und text den Wert NULL zu übergeben und für type, attrib, align und bt_mask die in fixwin.h definierten Konstanten zu verwenden.

Die Funktion liefert 0 bei Erfolg und -1 bei einem Fehler. Für ein Objekt kann nur die durch MaxPaintareasOnMask definierte Anzahl Paintareas definiert werden.

```
int pan_next_pa(int idx, obj *objp, int type, int y1, int x1,
     char *token, char *text, char *tooltip,
     short attrib, short align, int bt_mask, long l1, l2, l3, l4,
     unsigned long validmask);
```

Mit Hilfe dieser Funktion können die Daten zu einer Paintarea mit bestimmten Eigenschaften ermittelt werden. Die Eigenschaften werden als Parameter übergeben. Die Parameter entsprechen denen von pan_add_pa. validmask bestimmt, welche Eigenschaften ausgewertet werden. Hier sind die Makros aus fixwin.h zu verwenden. Die Bedeutung der Makros ist die gleiche wie bei pa_next(). idx definiert den Startindex, ab dem gesucht wird. Für den ersten Satz im Vektor ist 0 zu übergeben. Die Funktion liefert -1, wenn keine (weiteren) Daten mit diesen Eigenschaften gefunden werden konnten. Ansonsten liefert die Funktion den Index innerhalb des Vektors. Dieser kann als Startwert für weitere Suchen und als Schlüssel zum Zugriff verwendet werden.

```
int pan_read_pa(int idx, obj *objp, int *type, int *y1, int *x1,
     char *token, char *text, char *tooltip,
     short *attrib; short *align, int *bt_mask, long *l1, *l2, *l3, *l4);
```

Die Funktion ermittelt die Daten der Paintarea mit dem Index idx in dem Objekt objp. Die Daten werden an den als Zeiger übergebenen Parametern abgelegt. Die Bedeutung der Parameter entspricht der von pan_add_pa() mit dem Unterschied, dass hier Zeiger auf Variablen übergeben werden müssen. Es ist auch möglich statt dessen den Wert NULL zu übergeben. In diesem Fall wird der betreffende Wert ignoriert.

```
int pan_update_pa(int idx, obj *objp, int type, int y1, int x1,
     char *token, char *text, char *tooltip,
     short attrib, char  align, int bt_mask, long l1, l2, l3, l4);
```

Diese Funktion ändert die Werte für eine Paintarea des Objektes objp. idx ist der Index innerhalb des Vektors. Die übrigen Parameter entsprechen denen der Funktion pan_add_pa().

```
int pan_delete_pa(int idx, obj *objp);
```

Die Funktion löscht die Daten zu einer Paintarea des Objektes objp. idx definiert den Index der Paintarea.

Beep abschalten

Der Beep-Ton, den FIX bei unbekannten Events erzeugt, kann durch Setzen von

```
BeepOnUnknownEvent: FALSE
```

in der Datei fix.rc abgeschaltet werden.

Variationen von Objektfarben

Mit Hilfe dieses Features ist es möglich, bestimmte Objekte mit anderen Farben darzustellen. Dazu sind drei Schritte notwendig:

- In der Farbzuordnungstabelle müssen andere Farben zum Zeichen der Objekte definiert werden.
- Die Bitmaps zur Darstellung der Grafikzeichen müssen andere Farben verwenden.
- Die Objekte, die andere Farben verwenden sollen müssen entsprechend gekennzeichnet werden.

1 Definition von Farbvariationen

Die Farben zur Darstellung von Objekten werden ab der Version 4.5.0 als Farbvariationen definiert. Eine Farbvariation besitzt eine Nummer von 0 bis 3. Wenn zu einem Objekt keine Variation angegeben wird, dann wird die Variation 0 verwendet. Diese Variation muss in der Farbzuordnungstabelle auf jeden Fall definiert werden.

Die Syntax der bisherigen Farbzuordnungstabelle wurde erweitert. Die bisherigen Angaben definieren die allen Variationen gemeinsamen Angaben. Nach diesen Angaben erfolgen die Definitionen der Variationen. Die Definition einer Variation wird eingeleitet durch:

```
VARIATION [0|1|2|3]
```

Gleichzeitig beendet das Schlüsselwort VARIATION die allen Variantionen gemeinsamen Angaben. Nach der Angabe der Variation können durch

```
READCOMMON
```

die gemeinsamen Angaben für diese Variation definiert werden. Um die Farbzuordnungstabelle einer alten Version umzustellen muss sie um die Zeilen

```
VARIATION 0
READCOMMON
```

am Ende ergänzt werden. Damit wird die Variation 0 definiert, die alle Farben enthält, die am Anfang der Datei definiert wurden.

Bei den weiteren Variationen ist es sinnvoll Abweichungen für bestimmte Elemente zu definieren. Dazu ist die gleiche Syntax für ein Element zu verwenden, wie bei den allgemeinen Definitionen. Soll beispielsweise in Variante 1 die Farbe für die Texte in Proportionalschrift auf rot definiert werden, kann das folgendermaßen erreicht werden:

```
VARIATION 1
READCOMMON
PA_TEXT_NORMAL          RED2      GRAY2
```

Die Anweisung READCOMMON wirkt so, als ob die allgemeinen Definitionen an diese Stelle kopiert werden. Die weiteren Farbangaben überschreiben dann die allgemeinen Angaben. Prinzipiell ist es auch möglich auf READCOMMON zu verzichten und alle Angaben einer Variation zu definieren. Eine weitere Vorgehensweise kann darin bestehen, bei den allgemeinen Definitionen nur die aufzuführen, die bei allen Variationen gleich sind und dann die READCOMMON Anweisung nach der Definition der abweichenden Farbangaben aufzurufen. Dabei besteht jedoch die Gefahr, dass die allgemeinen Definitionen die speziellen überschreiben.

1.1 Definieren von Farbbezeichnungen

Durch die Angabe

```
DEFINE <name> <farbe>
```

kann für eine Farbe ein logischer Name vergeben werden. <name> ist dabei ein beliebiger Name und <farbe> eine Farbe in der bisherigen Syntax (RGB_XXXXXX oder Name der Farbe). Es kann auch der Name einer bereits definierten Farbe für <farbe> verwendet werden.

Beispiel

```
DEFINE HGCOLOR RGB_ECE9D8
```

Als Farbangabe kann danach der Name statt der Farbe verwendet werden. Da die allgemeinen Farbangaben bis zur Definition der ersten Variation überlesen werden, ist es bei diesen nicht notwendig die Farbbezeichnung vorher zu definieren. Sie muss jedoch bei der Definition der Variation vor dem Einlesen der allgemeinen Angaben mit READCOMMON definiert werden.

Beispiel

Hinweis: In diesem Beispiel sind nicht alle Definitionen vorhanden. Es wird nur die Farbe für Texte definiert, um die Funktionsweise zu zeigen.

```
# Allgemeine Angaben - für alle Variationen gültig
TEXT_NORMAL             BLACK      HGCOLOR
TEXT_INVERS             HGCOLOR    RGB_0048F1
TEXT_LOW                GRAY1      HGCOLOR
TEXT_UNDERLINE          BLACK      HGCOLOR UNDERLINE
TEXT_BLINK              BLUE2      HGCOLOR
TEXT_BOLD               WHITE      HGCOLOR
TEXT_GRAYED             GRAY1      HGCOLOR
TEXT_INVISIBLE          GRAY2      HGCOLOR

# Variation 0 - Default
VARIATION 0
DEFINE HGCOLOR RGB_ECE9D8
READCOMMON

# Variation 1 - Hintergrund weiß, schwarz statt rot
VARIATION 1
DEFINE HGCOLOR WHITE
READCOMMON
TEXT_NORMAL             RED1       HGCOLOR
TEXT_UNDERLINE          RED1       HGCOLOR UNDERLINE
```

Der allgemeine Abschnitt definiert alle Farben für Text. Er verwendet für den Hintergrund die Farbbezeichnung HGCOLOR, obwohl sie noch nicht definiert ist. Sie wird erst vor dem Einlesen der allgemeinen Angaben für Variation 0 definiert. Variation 1 legt für diese Farbdefinition die Farbe weiß fest und ließt dann die allgemeinen Angaben ein. Danach werden die Farbdefinitionen für TEXT_NORMAL und TEXT_UNDERLINE neu definiert.

Es wäre sogar ein Fehler, die Farbbezeichnung HGCOLOR direkt am Anfang zu definieren. Denn dadurch zählt sie zu den allgemeinen Angaben von wird von jeder READCOMMON wieder ausgeführt. Die für die Variation festgelegte Farbe für HGCOLOR wird dann wieder überschrieben.

1.2 Zugriff auf Farben in der Benutzerbibliothek

Zum Zugriff auf Farben in der Benutzerbibliothek ist die Nummer der Variation notwendig.

Deshalb bekommen die Funktionen FwownDrawPaintArea() und FwownDrawMenu() diesen Wert als Parameter.

```
FWOWN_API _TCHAR* CALLBACK FwownDrawPaintArea(
     CallBackFct CallBack, void* p_callID,
     HDC p_hDCPA, int p_width, int p_height,
     unsigned long p_pa_id, short p_pa_type,
     _TCHAR* p_text, unsigned short p_pa_attr, short p_pa_align,
     long p_longval1, long p_longval2,
     long p_longval3, long p_longval4,
     _TCHAR* p_ms_name, short p_objclass,
     int p_size, int p_style, int p_variation);

FWOWN_API _TCHAR* CALLBACK FwownDrawMenu(
     CallBackFct p_callBack,   void* p_callID,
     HDC p_hDC, _TCHAR* p_text, unsigned short p_attr,
     int p_cx, int p_cy, unsigned long p_ob_id,
     int p_size, int p_style, short p_variation);
```

Der Wert kann zum Ermitteln von Farben, über GetColor() verwendet werden. Diese Funktion benötigt jetzt auch die Variation.

```
COLORREF (* GetColorFct)(void* p_callID, ColtabIdx idx,
     ColtabPart colpart, short variation);
```

Neue Funktion zum Zugriff auf Farben

Zusätzlich kann die neue Funktion

```
bool GetColorByName(void* p_callID, _TCHAR* p_name, COLORREF* p_col,
     short p_variation)
```

aus dem FIX/Win Kern verwendet werden. Ein Zeiger auf die Funktion kann auf die übliche Weise mithilfe der Makros

```
FW_GETCOLORBYNAME, FWFKT_GETCOLORBYNAME(callback, varname)
```

ermittelt werden.

Die Funktion liefert zu dem Namen p_name einen Farbwert und legt ihn in p_col ab. Wenn kein Farbwert gefunden wurde, liefert die Funktion false, sonst true. p_name muss entweder ein definierter Farbname sein (RED1, WHITE, BLACK ... gleiche Schreibweise wie in der Farbzuordnungstabelle) oder ein mit RGB_ definierter Farbwert. Wenn der Wert p_variation einen Wert von 0 bis 3 enthält, dann kann p_name auch ein über DEFINE in der Farbzuordnungstabelle definierter Name zu einer Farbvariation sein. Diese Variante ist der Hauptzweck der Funktion. Damit lassen sich die definierten Farben zu jeder Variante auslesen.

Beispiel

```
#define FWOWN_EXPORTS
#include "fwown.h"
#include "fwownStd.h"

...

GetColorByNameFct   GetColorByName = NULL;

...
FWOWN_API _TCHAR* CALLBACK FwownDrawPaintArea( CallBackFct CallBack, void* p_callID,
     HDC p_hDCPA, int p_width, int p_height,       unsigned long p_pa_id, short
p_pa_type,
     _TCHAR* p_text, unsigned short p_pa_attr, short p_pa_align,
     long p_longval1, long p_longval2, long p_longval3, long p_longval4,
     _TCHAR* p_ms_name, short p_objclass, int p_size, int p_style, short p_variation) {

     COLORREF background0;
     COLORREF background1;

     FWFCT_GETCOLORBYNAME(p_callBack, GetColorByName);

     GetColorByName(p_callID, _T("HGCOLOR"), &background0, 0);
     GetColorByName(p_callID, _T("HGCOLOR"), &background1, 1);
}
```

Dieses Beispiel legt in background0 die Farbe der Definitionen von HGCOLOR in Variation 0 und in background1 für HGCOLOR in Variation 1 ab. Für die obige Farbzuordnungstabelle werden die Farben #ECE9D8 und weiß geliefert.

Da die Funktion die Farben über Textvergleiche findet, kann es sich lohnen, bei vielen Farbdefinitionen und vielen Aufrufen die ermittelten Farben zu speichern (cachen) und bei einem Wechsel des Stils (FwownHook()) die ermittelten Farben wieder zurückzusetzen.

1.3 Geänderte/Neue Meldungen

Folgende Meldungen wurden verändert (ResFixwinCore.rc):

```
ERR_VG110, ERR_VG120, ERR_VG130, ERR_VG160, ERR_OB150,  ERR_OB155
```

Folgende Meldungen sind neu (ResFixwinCore.rc):

```
ERR_VG165, ERR_VG185
```

1.4 Einschränkungen

Für Tooltips wird immer die Farbe der Variante 0 verwendet. Ebenso wird zur Darstellung des Hintergrundes, der nicht mehr zum FIX/Win Fenster gehört, die Definition von BACKGROUND aus Variation 0 verwendet.

2 Anpassen der Bitmaps

Um nicht alle Bitmaps für jede Variante zu duplizieren, wurde hier ein anderer Weg gewählt.

Die Bitmaps werden für jede Variante einmal geladen. Dabei werden für jede Variante die gleichen gepackten Bitmaps geladen. Die Bitmaps sind somit viermal vorhanden. Zur Anpassung der Farben auf eine bestimmte Variante wird die Funktion

```
FWOWN_API void CALLBACK FwownCompleteBitmapSet(CallBackFct p_callBack, void* p_callID,
     HDC p_hDC, int p_width, int p_height, short p_variation)
```

in der Benutzerbibliothek aufgerufen. p_hDC zeigt in einen Speicherkontext, in dem alle Bitmaps nebeneinander liegen. In p_width und p_height steht die Größe. p_variation definiert die Variation, für die die Bitmap verwendet wird.

Zur Anpassung können mittels der WIN32-API Funktion GetPixel() die einzelnen Pixel gelesen werden und mit SetPixel() verändert werden. Über die Parameter p_callBack und p_callID ist es möglich, die Funktion GetColor() zu ermitteln und so auf die Farben in der Farbzuordnungstabelle zuzugreifen.

2.1 Beispiel

```
FWOWN_API void CALLBACK FwownCompleteBitmapSet(
     CallBackFct p_callBack, void* p_callID,
     HDC p_hDC, int p_width, int p_height, short p_variation)
{
     int x,y;
     COLORREF pixel;
     COLORREF org_background;
     COLORREF new_background;

     FWFCT_GETCOLOR(p_callBack, GetColor);

     org_background =
           GetColor(p_callID, PA_TEXT_NORMAL, BACK, 0);
     new_background =
           GetColor(p_callID, PA_TEXT_NORMAL, BACK, p_variation);

     for(y = 0; y < p_height; y++) {
           for(x = 0; x < p_width; x++) {
                 pixel = GetPixel(p_hDC, x, y);
                 if (pixel == org_background) {
                       SetPixel(p_hDC, x, y, new_background);
                 }
           }
     }
}
```

Die Funktion läuft über alle Pixel und prüft, ob ein Pixel die Hintergrundfarbe von Variation 0 (org_background) besitzt. Wenn ja, dann wird an diese Stelle die Hintergrundfarbe der aktuellen Variation (new_background) geschrieben.

Statt der Funktion GetColor() kann auch die Funktion GetColorByName() verwendet werden, um die Hintergrundfarbe zu bestimmen, wenn sie in der Farbzuordnungstabelle mittels DEFINE festgelegt wurde.

2.2 Wichtige Hinweise

Die Funktion FwownCompleteBitmapSet() wird nur unmittelbar nach dem Laden der Bitmaps aufgerufen. Wenn sie auf Farben der Farbzuordnungstabelle zugreift, hat das Verändern der Farbzuordnungstabelle und

das Nachladen keine Auswirkungen. Erst wenn die Bitmaps ebenfalls nachgeladen werden, wird auf die neuen Farben der Farbzuordnungstabelle zugegriffen, weil dann die Funktion FwownCompleteBitmapSet() erneut aufgerufen wird.

Wenn ein Bitmap (über Entwickler/Bitmaps editieren) bearbeitet wird, dann wird immer das Original aus dem Verzeichnis lookAndFeel-src bearbeitet. Dieses Bitmap besitzt jedoch nicht unbedingt die Farben der Variation, die gerade angezeigt wird. Beim Bearbeiten sollte der Entwickler wissen, welche Farben durch FwownCompleteBitmapSet() zur Laufzeit ersetzt werden.

Bei der Verwendung von einer Farbtiefe von 16 statt 24 Bit - beispielsweise beim Zugriff über RemoteDesktop - entsteht folgendes Problem:

Beim Laden der Bitmaps werden diese, wie unter Windows üblich, in das interne Format Bitmapformat konvertiert und über einen Memory-DC zugänglich gemacht. Das Handle wird der Routine FwownCompleteBitmapSet() als Parameter p_hDC übergeben. Wenn der Bildschirm nur mit 16 Bit statt 24 Bit arbeitet, dann werden die Farben der Bitmaps beim Laden auf 16 Bit heruntergerechnet. Die Funktion, die die Farben ersetzt muss daher damit rechnen, dass nicht mehr die Farbe vorgefunden wird, die vorher in das Bitmap geschrieben wurde. Aus der Hintergrundfarbe RGB_ECE9D8 wird beispielsweise die Farbe RGB_EFEBDE. Um die Farben zu ermitteln, die nach dem Laden vorhanden sind, kann eine kleine Funktion in der Benutzerbibliothek implementiert werden:

```
static COLORREF GetCompatibleColor(HDC p_hDC, COLORREF pixel) {
      HDC memDC;
      HBITMAP memBM;
      HBITMAP oldBM;
      COLORREF retpixel;

      memDC = CreateCompatibleDC (p_hDC);
      memBM = CreateCompatibleBitmap (p_hDC, 10, 10);
      oldBM = (HBITMAP)SelectObject(memDC, memBM);
      SetPixel(memDC, 0, 0, pixel);
      retpixel = GetPixel(memDC, 0, 0);
      SelectObject(memDC, oldBM);
      DeleteDC(memDC);
      DeleteObject(memBM);
      return retpixel;
}
```

Die Funktion macht im Prinzip das gleiche, was auch beim Laden der Bitmaps passiert, aber nur für 1 Pixel. Sie erzeugt einen Memory-DC und ein passendes Bitmaps dazu. Dann schreibt sie ein Pixel mit dem (24-Bit) Farbwert, der als Argument übergeben wurde. Danach ließt sie das Pixel wieder aus und erhält das durch die Windows-Konvertierung manipulierte Pixel. Bei einer Farbtiefe von 24 Bit wird das der gleiche Wert sein. Bei 16 Bit wird jedoch für einige Farben ein anderer Wert geliefert.

Damit das oben angegebene Beispiel auch bei 16 Bit Farbtiefe richtig arbeitet, muss die Vergleichsfarbe mit Hilfe der Funktion umgerechnet werden.

```
org_background = GetCompatibleColor(p_hDC, org_background);
```

2.3 Ersetzen von einzelnen Grafikzeichen

Neben dem globalen Ersetzen von Farben ist es auch möglich, gezielt bestimmte Grafikzeichen zu manipulieren. Da in p_hDC alle Grafikzeichen nebeneinander stehen, ist es dazu notwendig, die Position und die Größe des Grafikzeichens zu kennen.

Dazu kann die neue Funktion

```
bool GetBitmapPosition(void* p_callID, int p_code, unsigned short p_attr,
      int* p_x, int* p_y, int* p_sx, int* p_sy)
```

aus dem FIX/Win Kern verwendet werden. Ein Zeiger auf die Funktion kann auf die übliche Weise mithilfe der Makros

```
FW_GETBITMAPPOS, FWFCT_GETBITMAPPOS(callback, varname)
```

ermittelt werden.

Die Funktion ermittelt die Position p_x, p_y und die Größe p_sx, p_sy eines Grafikzeichens mit dem Code p_code und dem Attribut p_attr innerhalb des Memory-DCs. Wenn ein Grafikzeichen mit dem Code definiert wurde, dann werden die Variablen an den übergebenen Adressen (p_x, p_y, p_sx, p_sy) gefüllt und die Funktion kehrt mit true zurück. Andernfalls bleiben die Variablen unberührt und die Funktion liefert false.

Der Code

```
GetBitmapPosFct     GetBitmapPos  = NULL;

FWOWN_API void CALLBACK FwownCompleteBitmapSet(
      CallBackFct p_callBack, void* p_callID,
      HDC p_hDC, int p_width, int p_height, short p_variation)
{
      int bmX, bmY, bmSX, bmSY;

      FWFCT_GETBITMAPPOS(p_callBack, GetBitmapPos);
      ...
      if (GetBitmapPos(p_callID, 88, A_NORMAL, &bmX, &bmY, &bmSX, &bmSY)) {
            Rectangle(p_hDC, bmX, bmY, bmX + bmSX, bmY + bmSY);
      }
}
```

ersetzt beispielsweise das Grafikzeichen mit dem Code 88 ('X') und dem Attribut NORMAL (Zeichen für oberen Fensterrand) durch ein einfaches Rechteck.

3 Angabe der Variation für FIX-Objekte

In der Beschreibungsdatei von Menüs, Masken und Choices ist die Nummer der Variation hinter die Angabe der Layoutdatei zu schreiben. Im led ist ein entsprechendes Feld dafür vorgesehen. Fehlt die Angabe der Variation, dann wird Variation 0 verwendet.

Unterschiedliche Maskenvarianten dürfen unterschiedliche Variationen besitzen.

Die Nummer der Variation steht in der Komponenten

```
objp->ob_color_variation
```

Eine Manipulation des Wertes hat nur eine Auswirkung, wenn sie vor dem Anzeigen des Objektes vorgenommen wird. Dabei sollte drauf geachtet werden, keine Werte <0 oder >3 einzutragen.

Wenn für eine Variation keine Farben definiert wurden, dann werden die Default-Farben (rot/grün) verwendet.

Verschieben von Objekten

Mit Hilfe dieses Features ist es möglich, Objekte (Masken, Selos, Menüs und Choices) zur Laufzeit mit der Maus am Bildschirm zu verschieben.

Bezüglich einer Verschiebeaktion werden die Objekte von FIX in drei Klassen eingeteilt:

- Objekte, die im Frontend verschoben werden können: moveable-object.
- Objekte, die nicht einzeln verschoben werden können, aber mit verschoben, wenn ein moveable-object verschoben wird: moved-object.
- Objekte die nicht verschoben werden.

Bei einem Kopfmaske mit einer eingebetteten Tabelle ist die Kopfmaske, ein moveable-object und die Tabelle und die Kopfmaske ein moved-object. Die Kopfmaske kann im Frontend "angefasst" werden und verschoben werden. Dabei werden Kopfmaske und Tabelle verschoben. Die Tabelle kann jedoch einzeln nicht verschoben werden.

Damit ein Objekt verschoben werden kann, muss es folgende Bedingungen erfüllen:

- Es muss sich um das aktive Objekt selbst handeln oder es muss sich in der Liste von Objekten befinden, die dem aktiven Objekt zugeordnet wurden (ob_activeobj_list; gesetzt mit set_activeobj_list()).
- Es muss als verschiebbar definiert worden sein; also ein moveable-object sein. Selos und Choices werden von FIX als verschiebbar definiert. Für Masken und Menüs ist ein expliziter Aufruf einer Funktion notwendig.
- Es muss einen oder mehrere Bereiche besitzen, an denen es angefasst und verschoben werden kann. Diese Bereiche können durch eine Funktion der Benutzerbibliothek bestimmt werden.

Wenn sich der Mauszeiger über einem dieser Bereiche befindet, und wenn das Objekt verschiebbar ist, dann wird der Mauscursor umgeschaltet, so dass der Benutzer erkennen kann, dass das Objekt verschiebbar ist. Wenn der Benutzer dann den linken Mausknopf drückt und festhält, kann er durch das Bewegen der Maus das Objekt verschieben. Dazu wird ein Rahmen gezeichnet, der die neue Position des Objektes markiert. Beim Loslassen des Mausknopfes wird das Objekt verschoben.

Das Verschieben von Objekten außerhalb des Bildschirms ist nicht möglich.

1 Definition von verschiebbaren Objekten

Damit ein Objekt als verschiebbar (moveable-object) gilt, muss eine Liste von Objekten definiert werden, die verschoben werden (moved-objects), wenn der Rahmen des Objektes verschoben wird. Im einfachsten Fall steht in dieser Liste nur das Objekt selbst.

Für Selos und Choices wird die Liste von FIX erzeugt. Für Masken und Menüs kann nach dem Laden die Funktion

```
void set_obj_moveable(obj *objp, int maxsize)
```

aufgerufen werden, um eine Verschiebbarkeit festzulegen. objp definiert das moveable-object, für das die Verschiebbarkeit definiert wird. maxsize definiert die maximale Anzahl an moved-objects, die mit dem Objekt zusammen verschoben werden. Intern wird ein Vektor dieser Größe erstellt, der die verschiebbaren Objekte enthält. Das moveable-object selbst ist dabei auch mit zu rechnen, da es in der Liste enthalten sein muss, damit es verschoben wird.

Wenn die Funktion für ein Menü, ein Selo oder eine Choice aufgerufen wird, dann wird nur das moveable-object selbst in die Liste aufgenommen.

Bei einer Maske werden zusätzlich alle Objekte mit aufgenommen, die:

- sich in der Liste der aktiven Objekte (diese Liste wird mit set_activeobj_list() definiert) befinden.
- als Tabellenvarianten oder Maskenvarianten zu einem der aktiven Objekte geladen wurden.

Maskenvarianten, die zum Zeitpunkt des Aufrufs von set_obj_moveable() noch nicht geladen wurden, werden nicht berücksichtigt.

Außerdem wird jede Maske nur einmal mit aufgenommen, auch wenn sie mehrmals als Variante verwendet wird oder mehrfach in der Liste der aktiven Objekte auftaucht. Damit wird sie bei einer Verschiebeaktion auch nur einmal verschoben.

Nach dem Aufruf von set_obj_moveable() ist es möglich, Objekte, die hinzugefügt wurden, aber nicht verschoben werden sollen, wieder zu entfernen. Dazu kann die Funktion

```
void del_moved_obj(obj *objp, obj *a_obj, int variant, BOOLEAN tablemode)
```

verwendet werden. objp definiert das verschiebbare Objekt, dessen Liste manipuliert werden soll. a_obj und variant definieren die Variante des Objekts, das aus der Liste entfernt werden soll. Wenn es sich dabei um eine Zeilenvariante handelt, dann ist tablemode auf TRUE zu setzen, sonst auf FALSE.

Es können auch weitere Objekte definiert werden, die mit verschoben werden sollen, aber keine Variante sind und sich auch nicht in der Liste der aktiven Objekte befinden. Dazu ist die Funktion

```
void add_moved_obj(obj *objp, obj *a_obj, int variant, BOOLEAN tablemode)
```

Die Parameter haben die gleiche Bedeutung wie bei del_moveable_obj().

Die so definierte Liste hat drei Aufgaben:

- Sie definiert das Objekt als moveable-object, weil es jetzt eine Liste von moveable-objects besitzt. Wenn sich der Mauszeiger darüber befindet, wechselt das Aussehen und das Objekt kann an eine andere Position gezogen werden.
- Sie definiert die moved-objects. Im Gegensatz zu dem Objekt, für das die Liste definiert wurde, können diese Objekte nicht einzeln verschoben werden Es sei denn, dass eine eigene Liste mittels set_obj_moveable() für eines oder mehrere dieser Objekte definiert wurde.
- Sie definiert, welche Objekte zur Berechnung des Rahmens, der beim Verschieben gezeichnet wird, verwendet werden. Der Rahmen wird so groß, dass er alle moved-objects umfasst. Die Berechnung findet bei der Anzeige des moveable-objects statt. Beim Verschieben wird der Rahmen aktualisiert. Aus diesem Grund muss die Liste der Objekte vor dem Anzeigen des Objekts definiert werden.

2 Zurücksetzen der Position

Beim Verschieben von Objekten wird vorher deren originale Position in einer objektinternen Komponente gespeichert. Dies geschieht unmittelbar vor dem ersten Verschieben eines Objektes.

Mit Hilfe der Funktion

```
void restore_objects_position(obj *objp)
```

kann die aktuelle Position zurückgesetzt werden. Dazu werden alle Objekte, die zusammen mit objp verschoben werden auf die gemerkte Position zurückgesetzt. Wenn es keine gemerkte Position gibt, weil das Objekt noch nicht verschoben wurde, bleibt das Objekt an der aktuellen Position.

Durch Setzen der Ressource

```
RestorePosAfterPerform: 1
```

in fix.rc, wird die Funktion für ein Objekt nach perform aufgerufen, wenn die Maske vom Bildschirm entfernt wird.

3 Definition von "Anfassern"

Prinzipiell kann ein moveable-object an allen Zellen "angefasst" und verschoben werden. Das ist jedoch keineswegs sinnvoll. Deshalb muss die Menge der Zellen, an denen ein Objekt zum Verschieben angefasst werden kann, begrenzt werden.

Dazu kann in der Benutzerbibliothek die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownIsMoveableHandle(
     CallBackFct p_callBack, void* p_callID,
     int p_mx, int p_my,
     _TCHAR p_c, unsigned short p_attr, unsigned short p_wp_style,
     _TCHAR* p_ms_name, short p_ob_class,
     int p_ob_x, int p_ob_y, int p_ob_sx, int p_ob_sy,
     short p_pa_type, int p_pa_x, int p_pa_y, int p_pa_len,
     long p_longval1, long p_longval2, long p_longval3, long p_longval4);
```

implementiert werden. Sie wird für das Zeichen unter dem Mauszeiger aufgerufen und muss entscheiden, ob dieses Zeichen zu einem verschiebbaren Rahmen gehört oder nicht.

Dazu kann sie die Parameter auswerten:

- p_mx, p_my - Position des Mauszeigers in Zellenkoordinaten.
- p_c, p_attr, p_wp_style - Zeichencode, Videoattribut und Stil des Zeichens unter dem Mauszeiger.
- p_ms_name, p_ob_class, p_ob_x, p_ob_y, p_ob_sx, p_ob_sy - Name, Klasse und Rechteck des Objekts unter dem Mauszeiger.
- p_pa_type, p_pa_x, p_pa_y, p_pa_len, p_longval1 - 4 - Typ, Position, Länge und Longval-Werte der Paintarea unter dem Mauszeiger. Wenn sich keine Paintarea unter dem Mauszeiger befindet, dann werden diese Werte als 0 übergeben.

Über den Rückgabewert wird das Verhalten von FIX/Win bestimmt:

- "" - FIX/Win bestimmt, ob die Zelle als Anfasser verwendet werden darf. Wenn sich die Zelle in der ersten Zeile des Objekts befindet, dann ist das der Fall.
- "0" - Die Zelle soll nicht als Anfasser gelten.
- "1" - Die Zelle soll als Anfasser gelten.

Alle anderen Rückgabewerte werden als Fehlermeldung interpretiert und im Meldungsfenster ausgegeben.

4 Neuer Mauszeiger

Beim Verschieben wird ein neuer Mauszeiger verwendet. Um den Mauszeiger an eigene Wünsche anzupassen, kann die Ressource

```
CUR_MOVE
```

angepasst werden.

Zeichnen von Linien und Rahmen im led

Linien und Rahmen mussten bisher durch das Eingeben von Grafikzeichen im led erstellt werden. Mit Hilfe dieser Erweiterung ist es möglich, Linien und Rahmen direkt im led zu definieren und mittels Pfeiltasten/Maus zu editieren. Voraussetzung ist die Verwendung von Layoutdateien im ASCII-Format (pan-Format).

1 Grundlagen

Rahmen werden durch einen Typ, die Position (Zeile, Spalte) und die Größe (Zeilen, Spalten) definiert. Da bisher in den unterschiedlichen Anwendungen verschiedene Grafikzeichen zur Darstellung von Rahmen verwendet wurden, bleibt es der Anwendung überlassen, wie ein Rahmen gezeichnet wird. Dazu ist eine Funktion bereitzustellen, die durch das Schreiben von Grafikzeichen den Rahmen zeichnet. Anhand des Typs, der durch eine Nummer dargestellt wird, kann die Funktion bestimmen, welche Grafikzeichen zu verwenden sind. Die Funktion wird von FIX aufgerufen, um die im Layout definierten Rahmen zu zeichnen.

2 Funktionen zum Zeichnen der Rahmen

Damit der led Rahmen zeichnen kann, muss er mit verschiedenen Funktionen gebunden werden. Diese Funktionen werden zusammen mit FIX in der Datei

```
src/framefct_std.c
```

als Vorlage ausgeliefert. Beim Binden mittels des mitgelieferten Makefiles wird die Datei

```
src/framefct.c
```

übersetzt und zum led dazu gebunden. Diese Datei ist entweder selbst anhand der Vorlage zu erstellen, oder sie wird vom Makefile durch Kopieren von framefct_std.c erstellt, wenn sie nicht vorhanden ist. Die Funktionen zeichnen Rahmen, die aus den standardmäßig definierten Grafikzeichen bestehen. Bei Bedarf können die Funktionen an die Bedürfnisse der Anwendung angepasst werden.

2.1 Festlegen der Rahmentypen

Hierzu ruft der led die Funktion

```
void get_frametypes(char **typenames)
```

auf. Im Vektor typenames sind die Namen der verschiedenen Rahmentypen anzugeben, die im led verwendet werden. Dazu kann jedem Element des Vektors ein Name zugewiesen werden. Die Nummern der Typen gehen von 0 - maximal 127. Es sind also bis zu 128 Typen definierbar. Typen, die nicht verwendet werden, können mit NULL belegt werden. Bevor die Funktion aufgerufen wird, werden alle Typen auf NULL gesetzt.

Die Namen werden nur im led und nicht in der Anwendung verwendet. In der Anwendung wird nur mit den Typnummern gearbeitet. Die Typnummer wird auch in der pan-Datei zur Definition des Rahmens abgespeichert. Der für den Typ definierte Text wird nur im led verwendet, um den Typ genauer zu beschreiben, so dass sich der Entwickler nicht die einzelnen Typnummern merken muss.

2.2 Erzeugen von Rahmen

Wenn im led ein Rahmen aufgezogen wird und danach mit Return bestätigt wurde, dann wird eine Funktion aufgerufen, die den Rahmen erzeugt.

```
void create_frame(int y1, int x1, int y2, int x2, int type,
     int    *p_frame_y, int *p_frame_x, int *p_frame_sy, int *p_frame_sx,
     int *p_frame_type, paintarea **paintareas_add)
```

Eine der Aufgaben der Funktion ist es, in den Variablen p_frame_* die Position und die Größe abzulegen, die aus den beiden Eckpunkten, die in y1,x1 und y2,x2 übergeben werden, berechnet werden kann. Das Verfahren dazu ist immer gleich und ist in framefct.c ausprogrammiert. Genauso verhält es sich mit der Festlegung des Typs, der einfach von type nach p_frame_type kopiert werden kann.

Der eigentliche Grund, warum diese Funktion als Quellcode ausgeliefert wird, ist die Erzeugung von zusätzlichen Paintareas. Diese können beispielsweise eine Überschrift oder einen Button zum Schließen des Fensters sein.

Dazu sind im Array paintareas_add bis zu 32 Paintareas zu definieren. Diese werden als eigenständige Paintareas zusätzlich zu dem Rahmen erzeugt. Sie können später - wie alle anderen Paintareas - mit den Mitteln des leds bearbeitet werden. Wenn sie nicht gewünscht werden, dann können sie auch vom Benutzer des led gelöscht werden.

2.3 Zeichnen von Rahmen

Zum Zeichnen von Rahmen ruft der led die Funktion

```
void draw_frame(int y, int x, int sy, int sx, int type, int active)
```

auf. y,x definiert die Position des Rahmens; sy,sx definiert die Größe. Die Typnummer wird als type übergeben. Hier ist es wichtig, dass die Funktionen alle Typnummern behandelt, die durch get_frametypes() definiert wurden. Wenn sich der led in einem Modus befindet, indem der Rahmen durch Verschieben einer der Ecken verändert werden kann, dann besitzt active den Wert 1, sonst 0. Bei active==1 ist es ratsam, die linke obere Ecke und die rechte untere Ecke gesondert darzustellen, um dem Benutzer zu zeigen, dass der Editiermodus für Rahmen aktiv ist und diese beiden Ecken verschoben werden können.

Zum Zeichnen eines Zeichens kann die (neue) FIX-Funktion

```
void write_char(unsigned int attr, int c)
```

verwendet werden. Sie schreibt das Zeichen c mit dem Attribut attr auf die aktuelle Position im Bildschirm. Danach wird die Position um eins nach rechts verschoben.

Um die aktuelle Position zu setzen kann die FIX-Funktion

```
void move(int y, int x)
```

aufgerufen werden.

Es ist auch möglich, das Zeichen an der aktuellen Position zu lesen. Dazu ist die Funktion,

```
void read_char(unsigned int *attr, int *c, int offset_line, int offset_col);
```

aufzurufen. Wenn als offset_col und offset_line 0 angegeben wird, wird die aktuelle Position ausgelesen. Durch Angaben von anderen Werte ist es möglich benachbarte Zellen auszulesen, ohne vorher die aktuelle Zelle mittels move() zu ändern. Dabei sollte aber darauf geachtet werden, dass nicht ins Leere - sprich außerhalb des virtuellen Bildschirms - gegriffen wird. Zu dieser Prüfung können die globalen Variablen v_line, S_lines, v_col und S_cols herangezogen werden.

Die mit FIX ausgelieferte Funktion draw_frame() ermittelt im ersten Schritt, ob es sich bei dem Rahmen um eine Linie handelt. Linien erkennt man daran, dass eine der Größenangaben 1 ist. In diesem Fall wird eine vereinfachte Funktion zum Zeichnen der Linie (draw_v_line() oder draw_h_line()) aufgerufen. Alle drei

Funktionen (draw_frame(), draw_v_line() und draw_h_line()) bestimmen anhand des Typs type einen Satz von Grafikzeichen, die für den Rahmentyp zu verwenden sind. Zum eigentlichen Zeichnen der Zeichen wird dann die Funktion drawchar() verwendet. Diese schreibt das Zeichen nicht einfach den Bildschirm. Statt dessen untersucht sie das an dieser Stelle bereits vorhandene Zeichen und die Umgebung. Sie bestimmt so eventuell ein neues Zeichen. So wird beispielsweise aus einem waagerechten und einem senkrechten Strich ein Kreuz.

2.4 Einbindung in die Anwendung

Im Gegensatz zum led benötigt die Anwendung nur die Funktion draw_frame(). Alle anderen Funktionen (create_frame() und get_frametypes()) werden ausschließlich vom led benutzt.

Um die Migration einer Anwendung zu gewährleisten, ist hier die Funktion über den Zeiger

```
extern void (*S_draw_frame)(int y, int x, int sy, int sx, int type, int active);
```

einzubinden. Das erlaubt es, dass Anwendungen, die die neue Funktionalität zur Darstellung von Rahmen nicht nutzen, nicht mit framefct.c gebunden werden müssen.

Um die Funktionalität zu nutzen, ist die Anwendung mit framefct.c zu binden und S_draw_frame auf die Funktion draw_frame zu setzen. Das mit ausgelieferte Windemo zeigt beispielhaft wie es gemacht wird (makefile und demo.c).

3 Bedienung des led

3.1 Erstellen eines Rahmens

Zur Erzeugung eines neuen Rahmens ist die Taste

```
g2
```

zu drücken. Dadurch wird an der aktuellen Position ein Rahmen in der Größe 3, 3 gezeichnet. Alternativ kann das Erzeugen durch das Auswählen des Menüpunktes

```
Rahmen erstellen/positionieren
```

aus dem Kontextmenü vorgenommen werden. In diesem Fall geht der Rahmen von der aktuellen Cursorposition bis zur aktuellen Mausposition; also der Position an der das Kontextmenü mit der rechten Maustaste geöffnet wurde.

Nach dem Erzeugen befindet sich der led im Rahmen-Editmodus. In der Meldungszeile wird die Position der beiden Eckpunkte des Rahmens und der Typ des Rahmens angezeigt.

Der Typ des Rahmens kann durch die Tasten PD und PU oder über das Kontextmenü durch die rechte Maustaste umgeschaltet werden. Durch Drücken der Tasten 0-9 kann der Typ auch direkt eingegeben werden. Für Typen mit zweistelliger Typnummer ist das nicht möglich.

Mit Hilfe der Pfeiltasten oder durch Positionieren des Cursors mittels der linken Maustaste kann die aktuelle Ecke (dort wo sich der Cursor befindet) verschoben werden. Durch das Anklicken der anderen Ecke, wird diese zur aktuellen und kann verschoben werden. Die aktuelle Ecke kann ebenfalls mit Hilfe der Leertaste gewechselt werden.

Wenn sich die Ecken an der richtigen Position befinden, dann kann der Rahmen übernommen werden. Dazu ist

```
RT
```

zu drücken oder eine der Ecken doppelt anzuklicken. Danach werden die zum Rahmen gehörenden Paintareas erzeugt.

Um das Erstellen abzubrechen, ist die Taste

```
EN
```

zu drücken.

3.2 Editieren eines Rahmens

Um einen bestehenden Rahmen zu ändern, können entweder die Werte mit der Maske geändert werden oder der Rahmen-Editmodus kann nochmals für einen bestehenden Rahmen gestartet werden.

Um die Werte mittels Eingabemaske zu ändern, ist der Rahmen doppelt anzuklicken oder es ist

```
f6
```

zu drücken, wenn sich die Schreibmarke über dem Rahmen befindet. Paintareas, die zusammen mit dem Rahmen erzeugt wurden, gehören nicht zum Rahmen. Es sind eigenständige Objekte, die getrennt bearbeitet werden können. Der Cursor muss also wirklich auf dem Rahmen stehen und nicht auf einer Paintarea, die zusammen mit dem Rahmen erzeugt wurde.

In der Maske können die Werte geändert werden. Es kann auch zu anderen Rahmen geblättert werden, indem die Tasten PD und PU verwendet werden. Die aktuellen Werte werden dann übernommen, was dem Drücken von OK gleich kommt. Sichtbar werden die Änderungen am Rahmen jedoch erst, wenn die Maske verlassen wird. Das Drücken von Abbrechen bezieht sich nur auf die aktuellen Werte.

Um den Rahmen-Editmodus zu starten, ist

```
g2
```

zu drücken, während sich der Cursor über dem Rahmen befindet. Alternativ kann hier - wie auch beim Erstellen - mittels rechter Maustaste über dem Rahmen der Menüpunkt

```
Rahmen erstellen/positionieren
```

ausgewählt werden.

Danach kann der Rahmen wie beim Erzeugen verändert werden. Bei der Übernahme der Änderungen mit RT oder Doppelklick werden jedoch keine neuen Paintareas angelegt. Statt dessen werden die Paintareas, die sich auf der oberen oder unteren Linie des Rahmen befinden, mit verschoben. Je nachdem, ob sich die Paintarea näher am linken oder rechten Rand befindet, wird der Abstand zum linken oder rechten Rand beim Verschieben beibehalten. Bei den Paintareas spielt es dabei keine Rolle, ob die zusammen mit dem Rahmen erzeugt wurden oder nachträglich erzeugt und auf den Rahmen gezogen wurden.

3.3 Löschen eines Rahmens

Rahmen können - genau wie Paintareas und Felder - gelöscht werden, indem über dem Rahmen

```
f7
```

gedrückt wird. Zusammen mit dem Rahmen werden die Paintareas gelöscht, die sich auf der oberen oder unteren Linie des Rahmens befinden.

Verwenden von Standardmauszeigern

Durch Setzen von

```
useStdCursor: TRUE
```

in der Datei fixwin.frc verwendet FIX/Win für einige Mauszeiger die vom System definierten Standard-Mauszeiger. Wenn das System andere Zeiger definiert, dann werden diese auch von FIX/Win verwendet.

Folgende Cursor werden dann ersetzt:

- Standardzeiger - fixwin.cur, fixwin2.cur, fixedit.cur (Handsymbol) durch IDC_ARROW (Pfeilsymbol)
- Wartezeiger - fixbusy.cur (Wartemaske) durch IDC_WAIT (Eieruhr)
- Kopierzeiger – cur_mar*.cur durch IDC_IBEAM (I-Beam)
- Zeicheninfo – fixlook.cur durch IDC_CROSS (Kreuz)
- Verschiebecursor - cur_move.cur (Kreuz mit Pfeilen) durch IDC_SIZEALL (Kreuz mit Pfeilen)
- Sprungcursor – cur_jump.cur durch IDC_NO
- Busycursor – cur_busy durch IDC_APPSTARTING

Kontrollieren von FIX-Meldungen

Mit dieser Version wurden alle Meldungen in FIX umgestellt, so dass sie kontrolliert werden können.

Zum Kontrollieren der Meldungen ist an dem Zeiger

```
extern void (*S_FxmsgHook)(char *file, int line, int msgnum,
     BOOLEAN *suppress, int typ, int art, char *txt);
```

eine Funktion zu hinterlegen. Diese Funktion wird vor der Ausgabe jeder Meldung von FIX aufgerufen. Die Programmstelle, von der aus die Meldung ausgegeben wird, wird durch file (Dateiname), line (Zeile) und msgnum (Meldungsnummer) spezifiziert. file und line haben eher dokumentarischen Charakter und können sich innerhalb einer FIX-Version ändern. msgnum ist jedoch eine feste Meldungsnummer, die sich auch über neue FIX-Versionen hinweg nicht ändert (es sei denn, es lässt sich wegen wichtigen Gründen nicht vermeiden). Alle Meldungen wurden bei der Umstellung auf dieses Verfahren nummeriert und mit einer festen Nummer versehen. Neue Meldungen bekommen eine neue Nummer. Anhand der Nummer kann deshalb eine Meldung eindeutig zugeordnet werden.

Über den Wert suppress kann der weitere Verlauf gesteuert werden. Wird der Wert mit TRUE belegt, dann wird die Ausgabe der Meldung durch FIX unterdrückt. Um die Meldungen, die unterdrückt werden sollen, von außen zu steuern, kann die Datei

```
$FXDIR/ignoremsg.fix
```

angelegt werden. Sie enthält in jeder Zeile die Nummer einer Meldung, die unterdrückt werden soll. In der eigenen Funktion kann dann mittels

```
int fxmsg_msgnum_isset(int msgnum);
```

abgefragt werden, ob eine Meldungsnummer in der Datei vorhanden ist. Dabei wird die Datei nicht jedes Mal durchgelesen. Statt dessen wird sie beim ersten Aufruf in ein Bitarray eingelesen, was bei den nächsten Aufrufen verwendet wird.

Beispiel

Die Funktion gibt alle Meldungen in eine Datei aus und unterdrückt die in $FXDIR/ignoremsg.fix definierten Meldungen.

```
void FxmsgHook(char *file, int line, int msgnum,
BOOLEAN *suppress, int typ, int art, char * msg)
{

      f = fopen("msg.log", "a+");
      fprintf(f, "MSG(%d,%s:%d): %s\n", msgnum, file, line, msg);
      fclose(f);
      if (!fxmsg_msgnum_isset(msgnum)) {
            *suppress = TRUE;
      }
}
```

Kontrollieren von FIX/Win-Meldungen

In ähnlicher Weise wie die FIX-Meldungen können ab der Version 4.5.0 auch die Meldungen von FIX/Win kontrolliert werden.

Dazu wurde - wie bei FIX - jede Meldung von FIX/Win mit einer internen Meldungsnummer versehen. Zum Unterdrücken der Meldung ist die Nummer in die Datei

```
$FXDIR/ignoremsg.fixwin
```

in eine eigene Zeile zu schreiben (gleiches Format wie bei FIX).

Zum Ermitteln der Nummern ist der Schalter

```
+MESSAGE
```

in in der Ressource DebugLog in config/fixwin.rc oder zur Laufzeit über das Menü Diagnose/Log Optionen zu setzen und FIX/Win mit /diag zu starten. Alle Meldungen werden dann nach fixwin.log geschrieben. Vor der Meldung steht dann die Meldungsnummer. Wenn die Meldung unterdrückt wurde, steht ein '-' vor der Nummer, sonst ein '+'.

Die Ausgaben lassen sich auch mit DebugView (von sysinternals) oder einem ähnlichem Werkzeug, das Debugmeldungen anzeigt, beobachten, ohne die Logdatei zu öffnen.

Version 4.2.0a

FIX für Windows unter Cygwin

FIX unterstützt ab dem 15.06.2011 die MKS-Tools nicht mehr. Statt dessen kann Cygwin eingesetzt werden.

Gleichzeitig wurde die Installation von FIX für Windows vereinfacht.

1 Installation von FIX

FIX für Windows wird jetzt - genauso wie FIX für UNIX - als ein Satz Archivdateien ausgeliefert. Als Format wird dabei - passend zu Windows - das ZIP-Format verwendet.

Das bietet den Vorteil, dass einzelne Dateien mit Standardtools nachinstalliert werden können.

Zur Installation werden die Dateien nur noch auf der Platte in einem oder zwei Verzeichnissen entpackt. Das Kompilieren der Meldungsdateien wird bereits vor dem Zusammenstellen der Archive vorgenommen. Die Archive enthalten die kompilierten Meldungsdateien.

Zur Auswahl der Pakete und des Zeichensatzes kann das Programm

```
FIXextract.exe
```

verwendet werden. Es fragt zwei Verzeichnisse (FXDIR und FXDIRSYS), die zu installierenden Pakete und den Zeichensatz ab. Danach werden alle *_sys.zip Dateien in $FXDIRSYS und alle anderen in $FXDIR entpackt.

Die Archive enthalten alle Dateien für den Zeichensatz ISO-15 oder UTF-8. Wenn der Zeichensatz ISO-2 oder IBM437 ausgewählt wurde (geht nur bei FIX, der nicht für UTF-8 übersetzt wurde), dann werden danach die Archive T_ISO-2.zip bzw. T_IBM437.zip in $FXDIR entpackt. T_ISO-15.zip wird nur der Vollständigkeit halber mit ausgeliefert. Alle Dateien sind auch in den anderen Pakten enthalten. Beim Entpacken von T_ISO-2.zip und T_IBM437.zip tritt jedoch eine Besonderheit von FIXextract in Kraft: Es werden nur Dateien entpackt, die bereits vorher in der ISO-15 Version vorhanden sind. Wenn beispielsweise das Demopaket nicht entpackt wurde, dann werden auch die ISO-2 (IBM437) Dateien des Demos nicht entpackt.

2 Starten von FIX unter Cygwin

Da nur noch wenige eine Shell unter Windows zum Entwickeln mit FIX einsetzen, wird dies vom Installationsprogramm nicht mehr unterstützt. Wer trotzdem mit einer Shell arbeiten will, der muss sich diese von Hand konfigurieren. Hier eine kurze Anleitung dazu. Die Pfade beziehen sich dabei auf eine Standardinstallation.

2.1 Installation von Cygwin

Hier genügt eine Standardinstallation und das Paket make.

2.2 Installation von Microsoft Visual Studio

Es genügt die Installation des C-Compilers. Er sollte von einer Cygwin-Shell aus aufgerufen werden können. Dazu kann in

```
C:/Programme/cygwin.bat
```

die Zeile

```
call "c:\Programme\Microsoft Visual Studio 9.0\Common7\Tools\vsvars32.bat"
```

aufgenommen werden. Nach dem Start einer neuen Shell sollte sich dann der Compiler durch Eingabe von

```
cl
```

starten lassen.

2.3 Anpassen von FIX

Hierzu ist die Datei

```
C:/Programme/FIX/etc/profile.cygwin
```

anzupassen. Damit FIX richtig arbeitet, sind Mountpoints vom Typ text anzulegen. Dabei ist darauf zu achten, dass die Pfadanteile des Mountpoints und des tatsächlichen Verzeichnisses übereinstimmen. So kann beispielsweise

```
C:/Programme/FIX
```

nach

```
/Programme/FIX
```

FIX gemountet werden aber nicht nach

```
/FIX
```

Somit ist sichergestellt, dass die Pfade für Cygwin und FIX (abgesehen vom Laufwerksbuchstaben) gleich sind.

Ein solcher Mountpoint kann durch das Eintragen von

```
C:/Programme/FIX /Programme/FIX ntfs text,posix=0,user,auto 0 0
```

in

```
C:/Programme/cygwin/etc/fstab
```

angelegt werden. Danach ist mount -a zu starten.

Wenn die binären Komponenten in ein anderes Verzeichnis installiert werden, dann ist ein zweiter Mountpoint anzulegen.

Es ist nicht notwendig, das der Mountpoint direkt auf die FIX Installation zeigt. FIX kann auch in irgendeinem Unterverzeichnis unterhalb des Mountpoints liegen.

2.4 Starten der Umgebung und Kompilieren

Zum Start der Umgebung kann über cygwin.bat eine Shell gestartet werden. Danach kann nach

```
cd /Programme/FIX/etc
```

gewechselt werden und mit

```
PATH=".:$PATH"
export PATH
. profile.cygwin
```

eine FIX-Umgebung aufgebaut werden. Danach kann die Demosoftware wie (unter UNIX) gewohnt gestartet und übersetzt werden (fw, mkdemo). Zum Kompilieren wird dabei der in m.default eingetragene Compilerwrapper benutzt. Er setzt die Aufrufe aus den Makefiles in cl-spezifische Aufrufe um.

Allgemeine Änderungen

1 sync_refresh bei perform

Wenn beim Start einer Maske gleich zu Beginn (bei L_ENTER_OBJECT) eine länger andauernde Aktion gestartet wird, dann wird die Maske erst nach Ablauf der Aktion angezeigt. Dies läßt sich durch den Aufruf von sync_refresh("text", TRUE) unmittelbar vor der Aktion beheben. Dabei kann es vorkommen, dass ein von einer anderen Maske vorhandenes Edit Control in der neuen Maske durchscheint. Dieses kann durch Aufruf von fx_hide_editcontrol() vor dem Aufruf von sync_refresh() abgeschaltet werden.

Um diese Aufrufe von FIX für jeden Aufruf von perform() vorzunehmen, kann die Ressource

```
RefreshBeforePerform: 3
```

in fix.rc gesetzt werden. Der Ressource kann ein Wert von 0-3 zugewiesen werden:

```
0 - default: kein Aufruf von sync_refresh() und fx_hide_editcontrol()
1 - Aufruf von sync_refresh()
2 - Aufruf von fx_hide_editcontrol()
3 - Aufruf von sync_refresh() und fx_hide_editcontrol()
```

Ein ähnliches Problem besteht, wenn nach dem Aufruf von perform() eine längere Aktion ausgeführt wird. Hier bleiben auch Maske und Edit Control stehen, bis FIX eine neue Taste anfordert. Auch hier kann das Verhalten durch Aufruf von fx_hide_editcontrol() und sync_refresh() gesteuert werden. Um diese Aufrufe für jeden Aufruf von perform() vorzunehmen, kann die Ressource

```
RefreshAfterPerform: 3
```

in fix.rc gesetzt werden. Der Ressource kann ein Wert von 0-3 zugewiesen werden. Die Bedeutung entspricht der von RefreshBeforePerform.

2 Anzeige des Carets bei perform

In FIX-Versionen ab dem 01.06.2011 wird das Caret im FIX/Win Fenster abgeschaltet, wenn eine Maske mittels perform() abgearbeitet wird und Fieldareas verwendet werden.

Grund für diese Änderung war, dass das Caret sichtbar wird, wenn das Edit Control mittels fx_hide_editcontrol() weggeblendet wurde.

3 Verhalten bei fehlenden Selos/Choices

Bie fehlenden Selos und Choices wurde bisher die Maske nicht geladen und dann das Programm abgebrochen.

In Versionen ab dem 25.05.2011 werden auch bei fehlenden Selo- und Choice-Dateien die Masken geladen, wenn die entsprechenden Felder nicht die Eigenschaft DBMUST besitzen.

4 Anzeige des Wartecursors

In einigen Fällen kommt es vor, dass trotz länger andauernder Operationen kein Wartecursor angezeigt wird. Der Grund dafür liegt in dem Start des Thread, der den Wartecursor nach einem Timeout anzeigt. Dieser Thread wird gestartet, wenn FIX Daten zu FIX/Win sendet. Er wird beendet, wenn FIX eine Taste anfordert. Wenn FIX eine längerdauernde Operation macht und keine Taste anfordert, dann wird der Thread nicht beendet und er zeigt nach einem Timeout den Wartecursor an.

Wenn nach dem Anfordern einer Taste, keine Bildschirmausgabe an FIX/Win gesendet wird, dann wird der Thread nicht neu gestartet und der Wartecursor wird trotz einer langen Operation nicht angezeigt. Um dieses Problem zu beheben, ist es möglich, den Thread nach dem Senden einer Taste zu starten.

Dazu ist in config/fixwin.frc der Eintrag

```
BusyCursorAfterSendEvent: TRUE
```

zu setzen. Um das Starten des Threads nach dem Senden von Netzdaten abzuschalten kann in in config/fixwin.frc der Eintrag

```
BusyCursorOnNetData: FALSE
```

vorgenommen werden. Wenn keine der beiden Ressourcen gesetzt ist, dann wird der Thread wie bisher nur nach dem Senden von Netzdaten gestartet. Auch wenn der Thread in beiden Fällen gestartet wird, ist sichergestellt, dass er nur einmal gestartet wird, weil vor dem Start geprüft wird, ob der Thread bereits läuft.

Diese Änderung wurde auch in den Versionen 4.1.0 und 4.0.0 implementiert.

5 Protokolloptimierung

Durch Setzen der Ressource

```
DeclareFieldAreasDelayed: TRUE
```

in fix.rc kann eine Protokolloptimierung eingeschaltet werden. Sie macht sich bemerkbar, wenn Masken verwendet werden, die viele unsichtbare Felder (NODISP) besitzen . Für diese Felder ist es nicht notwendig, eine Fieldarea anzulegen. Damit werden für diese Felder keine Daten zu FIX/Win gesendet.

Wenn eines dieser Felder sichtbar wird, dann wird die notwendige Fieldarea unmittelbar vor der Darstellung des Feldes angelegt und an FIX/Win übertragen.

6 Merken von angeklickten Paintareas

Durch Aufruf der Funktion

```
void pa_hold_clicked();
```

kann der Verweis auf die angeklickte Paintarea gesichert werden.

Durch Aufruf von

```
void pa_restored_clicked();
```

kann der gesicherte Zustand restauriert werden.

Diese Funktionalität wird benötigt, wenn innerhalb der Abarbeitung von BT_LEFT_PA eine weitere Maske geöffnet wird, in der ebenfalls Paintareas angeklickt werden können.

7 sync_refresh bei fflash

Wenn die Optimierung für Refresh (S_skip_sync_refresh) eingeschaltet wird, dann werden Meldungen, die mit fflash, l_msg und flash ausgegeben werden, weg optimiert. Erst beim Einlesen der nächsten Taste wird die letzte Meldung angezeigt. Diese Meldungen sollen aber in den meisten Fällen sofort sichtbar sein.

Durch Setzen von

```
SyncOnFlash: TRUE
```

in fix.rc kann der Refresh für fflash, flash und l_msg eingeschaltet werden, ohne dass in den Code der Anwendung eingegriffen werden muss.

Diese Änderung wurde auch in den Versionen 4.1.0 und 4.0.0 implementiert.

Version 4.2.0

Erweiterung des led

FIX Versionen ab dem 20.11.2010 enthalten eine erweiterte Version des Layouteditors led. Der folgende Text beschreibt die vorgenommenen Änderungen und die erweiterte Funktionsweise.

1 Tastenbelegung

In der neuen Version verwendet led die Events h3-h8 zum Blättern in Varianten. In der Anwendung liegen diese Events vermutlich auf andere Tasten die man zum Blättern nicht verwenden würde. Es empfiehlt sich daher, für den led eine eigene Tastenbelegung zu verwenden, die FIX/Win über den Schalter

```
/keytab ledkey.fix
```

mit gegeben wird. Als Vorlage kann die Datei config/ledkey.fix verwendet werden. Sie ist in das Gruppenverzeichnis zu kopieren und kann bezüglich der Tasten an die eigenen Bedürfnisse angepast werden

2 Maske zum Zwischenspeichern

Die Maske zum Zwischenspeichern wird jetzt ausschließlich über die Taste K_g3 oder den Button "Kopieren" aufgerufen. Der Aufruf über K_f9 im WYSIWYG-Modus oder über K_h2 innerhalb der Feldliste funktioniert nicht mehr.

Die Maske zum Speichern wurde mit den Buttons "Abbrechen" und "Speichern" versehen. Abgebrochen wird beim Drücken des Buttons "Abbrechen" oder beim Verlassen der Maske nach oben oder mit L_END. Gespeichert wird beim Drücken des Buttons "Speichern" oder beim Verlassen der Maske nach unten.

Die Abfrage, ob gespeichert werden soll, entfällt. Beschreibungsdatei und Layout werden immer zusammen abgespeichert.

Beim Speichen wird jetzt in der mfo-/men-Datei der Pfad der Layoutdatei eingetragen, der in der Maske zum Speichen angegeben wird. Damit passen gespeicherte mfo-Datei und Layoutdatei zusammen (Vor dieser Änderung wurde die Layoutdatei verwendet, die in der Hauptmaske steht).

3 Neue Funktionen in der Dateiauswahl

Die Maske zur Dateiauswahl verfügt jetzt über weitere Funktionen, die über neue Buttons gestartet werden können.

mdemo

Mit diesem Button wird das Objekt (funktioniert auch bei Menüs) von mdemo gestartet.

Kopieren

Diese Funktion dient zum Kopieren einer Maske oder eines Menüs mitsamt Layout. Das Objekt wird dazu in den led geladen und danach wird direkt die Maske zum Zwischenspeichern aufgerufen. Hier kann ein neuer Dateiname für die Maske und das Layout vergeben werden. Danach wird der Bearbeitungsteil sofort verlassen.

Die gleiche Funktionalität lässt sich erreichen, indem das Objekt normal geladen wird, dann die Maske zum Zwischenspeichern aufgerufen wird (Taste K_g3) und die Bearbeitung wieder verlassen wird. Diese Methode hat den Vorteil, dass die Daten des Objekts vor dem Speichern inspiziert werden können.

Varianten

Statt der einzelnen Maske werden alle Varianten zur Bearbeitung geladen. Mehr zu diesem Modus steht im nächsten Abschnitt.

Zur Konfiguration, welche der Funktionen bei Auswahl mit Doppelklick oder Return ausgeführt wird, kann die Ressource

```
led.defaultAction
```

auf einen der Werte

- 1 - mdemo
- 2 - Kopieren
- 3 - Bearbeiten
- 4 - Bearbeiten von allen Varianten

gesetzt werden.

4 Schnelles Setzen von Feldeigenschaften

Zum schnellen Setzen von Feldeigenschaften kann ein Kontextmenü auf dem Feld "Eigenschaften" verwendet werden. Das Menü kann entweder mit der rechten Maustaste oder mit der Menütaste (Event ~102000) gestartet werden.

Die Menüpunkte selbst sind als Ressourcen mit den Namen led.stypMen_<n> zu definieren, wobei <n> eine Zahl zwischen 0 und 9 ist. Als Wert ist eine Zeichenkette mit dem Format

```
<Text>=<Zahl>
```

anzugeben. Der Anteil <Text> definiert den Text der im Menü angezeigt werden soll. Er sollte die Feldeigenschaft oder die Feldeigenschaften kurz beschreiben. <Zahl> definiert die Feldeigenschaften, die gesetzt werden sollen. Die möglichen Werte sind in fix/accept.h definiert. Wenn mehrere Feldeigenschaften gesetzt werden sollen, dann sind die Werte zu addieren.

Beispiel

Der erste Menüpunkt soll die Eigenschaften NOENT (=8) und NODISPL (=2048) setzen.

```
led.stypMen_0: NOENT+NODISPL=2056
```

Die Menüpunkte sind immer lückenlos zu definieren. Das bedeutet, dass es nicht möglich ist, led.stypMen_0 bis led.stypMen_5 zu definieren und dann bei led.stypMen_8 weiter zumachen. Sobald der erste nicht definierte Menüpunkt von led entdeckt wird, werden keine weiteren mehr eingelesen. Um einen Separator zu definieren, ist die Zeichenkette "-" als Wert für die Ressource zu definieren.

Beim Auslösen des Menüpunktes wird geprüft, ob die definierten Eigenschaften bereits vorhanden sind. Wenn nicht, dann werden sie gesetzt. Andernfalls werden sie entfernt.

5 Laden von allen Varianten

Durch das Laden über den entsprechenden Button werden bei Masken mit Varianten alle zugehörigen Varianten geladen. Dies betrifft sowohl die Maskenvarianten als auch die Zeilenvarianten. Wenn eine oder mehrere der Zeilenvarianten selbst wieder Maskenvarianten besitzt, dann werden auch diese geladen.

Aus der Sicht von led liegen alle Varianten - egal ob Zeilenvariante oder Maskenvariante - nebeneinander. Eine Struktur ergibt sich nur durch die beim Laden vergebene Variantennummer und die Reihenfolge beim Laden.

Zuerst wird die Hauptmaske geladen. Sie bekommt die Zeilenvariantennummer 1 und die Maskenvariantennummer 1.

Danach werden alle Maskenvarianten zur Hauptmaske geladen. Sie bekommen ebenfalls die Zeilenvariantennummer 1. Die Maskenvariantennummer wird hochgezählt. Die erste Maskenvariante hat also die Nummer 1/2, die zweite 1/3 u.s.w. Die Ladereihenfolge ergibt sich aus der Angabe in der mfo-Datei (zoom).

Danach wird - sofern vorhanden - die nächste Zeilenvariante geladen. Dabei wird die Zeilenvariantennummer hochgezählt. Die Maskenvariantennummer beginnt wieder bei 1. Sie bekommt als die Nummer 2/1.

Wenn die Zeilenvariante weitere Maskenvarianten definiert, dann werden diese unmittelbar nach der Zeilenvariante geladen. Hierbei wird die Maskenvariantennummer wieder erhöht. Diese Varianten bekommen also die Nummern 2/2, 2/3, usw.

Entsprechend werden die weiteren Zeilenvarianten und deren Maskenvarianten geladen. Die Reihenfolge der Zeilenvarianten ergibt sich aus der Angabe in der mfo-Datei(use).

Wenn beim Laden eine mfo-Datei entdeckt wird, die bereits geladen wurde, dann wird diese nicht nochmals geladen. Es wird jedoch eine Zeilenvariantennummer und eine Maskenvariantennummer vergeben und eine Art Link auf die geladene Maske hergestellt, so dass die Maske später bei der Bearbeitung mehrfach auftaucht.

Die Maximalanzahl an Varianten beträgt 256. Danach werden keine weiteren Varianten mehr geladen.

In der Bearbeitungsmaske im led wird in der ersten Zeile der Name der mfo-Datei und die Zeilenvariantennummer / Maskenvariantennummer angezeigt.

5.1 Blättern durch die Varianten

Durch folgende Tasten kann die aktuelle Variante gewechselt werden:

- h4 (ALT Pfeil runter) - nächste Maskenvariante, wenn nicht vorhanden, nächste Zeilenvariante.
- h5 (ALT Pfeil hoch) - vorherige Maskenvariante, wenn nicht vorhanden, vorherige Zeilenvariante.
- h6 (ALT Seite runter) - nächste Zeilenvariante
- h5 (ALT Pfeil hoch) - vorherige Zeilenvariante

5.2 Suchen einer Variante

Durch die Taste

- h8

oder den Button "Variante" kann eine Maske aufgeblendet werden, in der nach einer Variante gesucht werden kann. Dazu ist im Suchfeld ein Teilbegriff der mfo-Datei einzugeben. Durch Doppelklick auf den Namen der mfo-Datei wird die Variante zur aktuellen Variante. Wenn das Feld "Eindeutig" den Wert X besitzt, werden mfo-Dateien, die in mehreren Varianten verwendet werden, nur einmal angezeigt.

Neben der Auswahl einer Variante bietet die Maske die Möglichkeit, bestimmte Informationen zu den Varianten inspizieren. Zum einen werden neben der mfo-Datei die Nummer der Zeilenvariante und die der Maskenvariante angezeigt. Am Ende der Zeile steht ein Flag, das angibt, welche Teile der Maske beim Speichern auch tatsächlich gespeichert werden (müssen).

- L - nur das Layout wurde verändert und muss gespeichert werden
- M - nur die Maske wurde verändert und muss gespeichert werden
- A - Maske und Layout wurden verändert und müssen gespeichert werden

Die Flags werden gesetzt, wenn eine Veränderung der entsprechenden Daten stattfindet. Dabei haben bestimmte Änderungen auch Auswirkungen auf andere Varianten. Wird beispielsweise ein Feld hinzugefügt, dann sind davon alle Varianten betroffen. Genauere Informationen finden sich im Abschnitt zu den Feldänderungen.

Eine weitere Verwendung der Maske besteht in der Möglichkeit, bestimmte Varianten zu markieren. Dies geschieht durch Doppelklick oder durch Drücken der Leertaste im Feld "Mark".

Zwischen den Varianten, die markiert wurden, kann mittels der Taste

- h3 (Strg-TAB)

gewechselt werden. Diese Vorgehensweise ist sinnvoll, wenn eine Maske viele Varianten besitzt und bei der Bearbeitung aber nur zwei oder einige wenige verändert werden sollen. Auch zum direkten Vergleich des Layouts zweier Masken durch schnelles Umschalten im WYSIWYG-Modus kann dieses Verfahren verwendet werden.

5.3 Ändern von Eigenschaften

Änderungen von Eigenschaften der Maske können sich je nach Art nur auf die aktive Variante, auf alle Zeilenvarianten oder auf alle Varianten (Zeilen- und Maskenvarianten) auswirken.

Änderungen von Maskeneigenschaften

Folgende Änderungen werden in alle anderen Varianten übernommen:

- Mausunterstützung
- with +
- where +
- Fetch +
- Action +

Die mit + gekennzeichneten Werte müssen nicht gleich sein. Sie werden jedoch beim Wechsel der Variante in der Anwendung (nicht im led) ignoriert. Aus diesem Grund werden sie bei einer Änderung im led in alle anderen Varianten kopiert und sind somit in allen Varianten gleich.

Wenn Bit 2 in SwapOptions nicht gesetzt ist, dann werden zusätzlich alle

- Longval-Werte

in alle anderen Varianten übernommen.

Folgende Änderungen werden nur in andere Zeilenvarianten übernommen:

- Name
- Titel
- Dimension
- Position

Ein Problem entsteht, wenn die gleiche mfo-Datei sowohl als Zeilenvariante als auch als Maskenvariante verwendet wird. In diesem Fall werden auch dann alle Zeilenvarianten geändert, wenn nur die Maskenvariante geändert wird.

Änderungen von Feldeigenschaften

Folgende Änderungen werden in alle andere Varianten übernommen:

- Logischer Bezug
- Feldname
- Datentyp
- NULL,
- Auswahl +
- Default +
- Muster +
- Prüfroutine +
- Feld-Link +

Die mit + gekennzeichneten Werte müssen nicht gleich sein. Sie werden jedoch beim Wechsel der Variante in der Anwendung (nicht im led) ignoriert. Aus diesem Grund werden sie bei einer Änderung im led in alle anderen Varianten kopiert und sind somit in allen Varianten gleich.

Wenn Bit 1 in SwapOptions nicht gesetzt ist, dann werden zusätzlich

- Format
- Länge
- Displaylänge

in alle anderen Varianten übernommen.

Wenn Bit 2 in SwapOptions nicht gesetzt ist, dann werden zusätzlich alle

- Longval-Werte

in alle anderen Varianten übernommen.

Einfügen von neuen Feldern

Neue Felder werden an Position 0, 0 (so wie bisher auch) eingefügt. Davon sind alle Varianten betroffen.

In allen Varianten, außer der aktuellen Variante bekommt das neue Feld die Eigenschaft NOMOD/NODISPL.

5.4 Einfügen von neuen Varianten

Das Einfügen von neuen Varianten ist während der Bearbeitung nicht möglich.

Deshalb ist zum Einfügen wie folgt vorzugehen:

- Kopieren einer ähnlichen Variante unter anderem Namen
- Definition der mfo-Datei als Zeilen- oder Maskenvariante
- Neuladen der Maske mit allen (und somit auch der neuen) Varianten

Das Kopieren einer Variante kann während der Bearbeitung erfolgen, indem die gewünschte Variante zur aktuellen gemacht wird (siehe "Blättern durch die Varianten" / "Suchen einer Variante") und mittels K_g3 oder

dem Button "Kopieren" (siehe "Maske zum Zwischenspeichern") unter dem gewünschten Namen gespeichert wird.

Sinnvollerweise wird der Inhalt des Feldes "Beschreibungsdatei" der Maske zum Zwischenspeichern mittels Strg-C in die Zwischenablage kopiert. Er kann dann nämlich mit Strg-V an der gewünschten Stelle im Feld "Varianten" oder "Zeilen-T." eingefügt werden. Vorher ist die mfo-Datei, zu der die Variante hinzugefügt werden soll, zur aktuellen zu machen.

Danach ist die Maske zu speichern und erneut zu laden. Dabei wird die neue Variante mitgeladen.

5.5 Einschränkungen

Wenn eine Maske mit allen Varianten geladen wird, existieren folgende Einschränkungen:

UNDO

Dieses Feature basiert auf dem automatischen Zwischenspeichern vor jedem Bearbeitungsschritt. Wenn alle Varianten geladen werden, dann müssten vor jedem Bearbeitungsschritt alle Varianten zwischengespeichert werden. Dies stellt ein Performanceproblem dar. Aus diesem Grund wird UNDO beim Laden von allen Varianten abgeschaltet.

SORTIEREN

Bei Varianten müssen alle Felder die gleiche Reihenfolge haben. Deshalb ist ein Sortieren nach Zeilen oder Spalten einer bestimmten Variante nicht sinnvoll, da damit alle anderen Varianten unsortiert sind.

6 Anpassen der Statuszeile

Der Text mit der Meldungsnummer 10089 definiert den Prompt im WYSIWYG-Modus von led. Er enthält (wie andere Meldungstexte auch) Tastendefinitionen und Beschriftungen.

Für die neue Version wurden die Beschriftungen dieses Prompts stark gekürzt, damit die neuen Buttons für Feldliste und Varianten angezeigt werden können. Weiterhin wurde Platz benötigt, um oberhalb dieser neuen Buttons den Namen der aktuellen mfo-Datei anzuzeigen. Diese Information ist bei der Arbeit mit mehreren Varianten sehr wichtig.

Da der Platz für die aktuelle Variante und das aktuelle Feld sehr knapp bemessen ist, empfiehlt es sich, mit einer größeren Bildschirmauflösung als 80x25 im led zu arbeiten. In diesem Fall kann der Text mit der Meldungsnummer 10089 vergrößert werden. Damit erhöht sich auch der Platz für die Informationen, die über dem Prompt in der Meldungszeile angezeigt werden (aktueller Feldname und mfo-Datei). Der Platz und die Position der Informationen richtet sich nach den Buttons für die Tasten in der darunter liegenden Promptzeile aus. Um beispielsweise mehr Platz für die mfo-Datei zur Verfügung zu Stellen kann einfach die Beschriftung "Variantenliste" hinter der Tastendefinition #h8 um einige Leerzeichen erweitert werden.

Als Alternative enthält die Meldungsdatei einen Prompt 10089, der zu 125 Zeichen Breite passt. Nach dem Einkommentieren und dem Auskommentieren des alten Textes, ist die Meldungsdatei mittels msgprep neu zu erzeugen.

Es können aber auch eigene Texte für den Prompt definiert werden.

Dabei ist noch anzumerken, das auch der Originalprompt nicht zur Standardgröße von 80x25 passt und eigentlich schon eine größere Bildschirmgröße erforderlich macht. Die fehlende (abgeschnittene) Anzeige des Ende-Buttons und der Variantennummern ist bei der Bearbeitung mit 80x25 in Kauf zu nehmen.

7 Speichern von Dateien

Ein Speichern von Dateien findet nur noch statt, wenn die Daten der entsprechenden Datei tatsächlich geändert wurden. Dabei wird zwischen Layout- und Maskendaten unterschieden. Wenn nur Änderungen am Layout vorgenommen wurden, dann wird nur die Layoutdatei gespeichert. Bei Änderungen an Daten, die nur die Maske (das Menü) betreffen, wird nur die Beschreibungsdatei gespeichert.

Wenn mehrere Varianten geladen wurden, dann werden nur die Varianten gespeichert, die tatsächlich von einer Änderung betroffen sind.

Wenn der led mit dem Event L_END statt durch Klicken auf einen Button verlassen wird, dann erfolgt nach der Nachfrage, ob der led verlassen werden soll, eine Nachfrage, ob die Beschreibungsdatei und das Layout gesichert werden sollen. In der neuen Version erfolgen diese Nachfragen nur, wenn tatsächlich in den entsprechenden Bereichen Änderungen stattgefunden haben.

8 Neue C-Hooks

Die neuen C-Hooks erlauben es eine andere Datei im led zu laden und zu speichern, als die, die der Benutzer angegeben hat b.z.w. als in der mfo-Datei als Layout spezifiziert wird.

Somit ist es beispielsweise möglich statt des sprachspezifischen Layouts, das in der mfo-Datei steht, ein Layout in einer Standardsprache (Metasprache) zur Bearbeitung zu laden.

Zur Realisierung wird der led nach der Installation mit der Datei

src/ledhook.c

gebunden (Ausführung des Makefiles in $FXDIR).

Diese Datei definiert die folgende Funktionen:

```
void led_before_load_object(char* fname);
void led_before_load_layout(char* fname);
void led_before_save_object(char* fname);
void led_before_save_layout(char* fname);
```

Die Funktionen werden unmittelbar vor dem Laden und Speichern der Objekt- und Layoutdateien aufgerufen. Die ausgelieferte Datei src/ledhook.c enthält Funktionen, die keine Änderung vornehmen. Durch Anpassen der Funktionen kann der Dateiename, der an der Adresse fname steht, geändert werden. Der Dateiname darf maximal 42 Zeichen lang werden. Ansonsten kommt es zu einer Speicherüberschreibung.

Der Dateiname wird nur für das Laden und Speichern selbst geändert. Das bedeutet, dass der in led angezeigte Dateiname für das Objekt, das Layout und die Varianten den Originaldateinamen entsprechen.

Wenn beispielsweise eine Layoutdatei aus einem anderen Verzeichnis gelesen werden soll und beim Speichen auch wieder in dieses gespeichert werden soll, dann muss die Änderung des Dateinamens sowohl in led_before_load_layout() als auch in led_before_save_layout() vorgenommen werden. Der im Feld "Layout" angezeigte Dateiname entspricht dem Dateinamen, der in der Beschreibungsdatei (mfo-Datei) steht. Wenn er geändert wird, dann wird der geänderte Name auch an led_before_save_layout() übergeben.

Zusammenspiel mit den Cmd-Hooks

Die Hooks werden in der folgenden Reihenfolge aufgerufen:

Beim Laden für jede Variante:

```
BeforeLoadCmd <objectfilename>
led_before_load_object(<objectfilename>)
load_object(<objectfilename_new>);
led_before_load_layout(<layoutfilename>)
load_layout(<layoutfilename_new>);
```

Beim Speichern für jede Variante, deren Objekt sich geändert hat:

```
led_before_save_object(<objectfilename>)
save_object(<objectfilename_new>)
AfterSaveMfoCmd <objectfilename_new>
```

Beim Speichern für jede Variante, deren Layout sich geändert hat:

```
led_before_save_layout(<layoutfilename>)
save_layout(<layoutfilename_new>)
AfterSavePanCmd / AfterSaveMlyCmd <layoutfilename_new>
```

Beim Speichern für jede Variante:

```
AfterEditCmd <objectfilename>
```

<objectfilename> ist der Name, der in der Maske zur Bearbeitung ausgewählt wurde. <objectfilename_new> ist der von den C-Hooks geänderte Dateiname. <layoutfilename> ist der Name der Layoutdatei, so wie er in der Objektdatei steht. <layoutfilename_new> ist der von den C-Hooks geänderte Dateiname. Beim Speichern ist <layoutfilename> der Dateiname, der im Feld "Layout" steht. Dieser Name wird beim Speichern als Layoutdatei in die mfo-Datei geschrieben.

Die unterschiedliche Reihenfolge beim Laden und beim Speichern ergibt sich aus den bestehenden Routinen des led. Beim Laden werden die FIX-Routinen verwenden, die ein Objekt zusammen mit dem Layout laden. Beim Speichern wird der interne Zustand des led verwendet.

Allgemeine Änderungen

1 Kompatibilität

FIX/Win und FIX der Versionen 4.2.0, 4.1.0, 4.0.0 und 3.1.0 sind zueinander kompatibel. Das bedeutet, dass die Version von FIX/Win nicht mit der Version von FIX übereinstimmen muss. Allerdings stehen dann die mit der Version eingeführten Features nicht vollständig zur Verfügung. Für die Version 4.2.0 bedeutet das, dass sowohl FIX als auch FIX/Win die Version 4.2.0 haben müssen, um ActiveX Controls zu nutzen.

2 Zeilenvarianten anpassen

Um die Anwendung über das Umschalten einer Zeilenvariante zu informieren, wird die Funktion, die an der Adresse

```
void (*S_TableRowSwapped)(obj *o1, obj *o2);
```

hinterlegt ist, aufgerufen. Sie bekommt als Argument die Zeilenvariante von der umgeschaltet wurde(o1) und die Zeilenvariante auf die umgeschaltet wurde(o2).

3 Abschalten des Editors

Durch Aufruf der Funktion

```
fx_hide_editcontrol()
```

kann das Edit Control zur Feldwerterfassung abgeschaltet werden. Sinnvoll ist dies, wenn bei längeren Aktionen kein Feld sichtbar sein soll. Bei der nächsten Felderfassung wird das Edit Control wieder eingeschaltet.

4 Konfiguration des Caches für Fieldareas und Paintareas

Der Cache für Fieldareas und Paintareas wird ab der Version 4.2.0 nicht mehr über die Variablen S_fa_max_tabs, S_fa_tab_size, S_pa_max_tabs und S_pa_tab_size eingestellt. Statt dessen sind die Ressourcen

```
PaintareaMaxTabs : 128
PaintareaTabSize :  32
FieldareaMaxTabs : 128
FieldareaTabSize :  32
```

zu verwenden. Gleichzeitig haben sich die Defaultwerte auf die oben angegebenen Werte geändert.

5 Transparenz von Keylabels

Keylabels können jetzt transparent gezeichnet werden. Dazu ist durch die Ressource

```
KeylabelTransparent
```

in lookAndFeel.frc eine transparente Farbe zu definieren.Als Format für die Farbangabe ist das gleiche Format wie für coltab.fix zu verwenden (RGB_rrggbb).

Damit ist möglich, den Hintergrund von allen Bitmaps durchsichtig zu machen. So wird der Hintergrund durch die Rahmen keyFrmInActive-[M|L|H].bmp und keyFrmActive-[M|L|H].bmp bestimmt und es ist möglich, Buttons mit Farbverlauf zu zeichnen.

Wenn dieses Feature verwendet wird, dann sollte die Größe der Bitmaps auf jeden Fall zur Zellengröße passen. Das bedeutet, dass das Bitmap für den Rahmen in der Höhe die gleiche Höhe wie eine Zelle haben muss und in der Breite die vierfache Breite wie eine Zelle haben muss, da von dem Button vier Zellen beansprucht werden. Bei den Bitmaps für die Keylabels sind von diesen Werten jeweils 4 abzuziehen, da oben, unten, links und rechts jeweils 2 Pixel für den Rahmen verwendet werden. Wenn die Bitmaps nicht diese Maße besitzen, dann skaliert FIX/Win die Bitmaps passend. Dabei kann es passieren, dass Farben zustande kommen, die der Transparenzfarbe entsprechen und somit durchsichtig sind.

6 Setzen der Values-Liste

Durch Aufruf der Funktion

```
int f_set_values(field *f, char *valstr)
```

kann für das Feld f die Werteliste in valstr definiert werden. Das Format in varstr ist das gleiche, wie in der mfo-Datei (durch | getrennte Werte). Anhand dieser Zeichenkette wird das fix-interne Format für die Werteliste gebildet. Ein Zugriff auf bestehende Wertelisten ist deshalb nicht möglich. Die alte Werteliste wird gelöscht. Die aktuelle Werteliste wird beim Freigeben der Maske (und damit des Feldes) gelöscht.

7 Zeichensatz bei fx_exec_frontend()

Die Beschränkung auf den ASCII Zeichensatz für die Zeichenkette, die mit fx_exec_frontend() übertragen wird, fällt mit der Version 4.2.0 weg. Statt dessen muss die Zeichenkette in dem Zeichensatz bereitgestellt werden, der durch FXCHARSET eingestellt wird. Auf der Seite von FIX/Win werden die Zeichen in Unicode (wchar_t) umgewandelt.

Zusammen mit der Beschränkung fällt auch die Prüfung auf den Zeichensatz weg. Es kann nicht in aller Vollständigkeit geprüft werden, ob die Zeichenkette dem geforderten Zeichensatz entspricht.

8 Füllen der clickinfo-Struktur

Mit Hilfe der neuen Funktion

```
int fill_clickinfo(int ev, struct clickinfo *btLeftFromField);
```

kann ein Element der Struktur clickinfo mit aktuellen Werten gefüllt werden. Damit ist es möglich, andere Events wie einen Mausklick zu behandeln. Die Funktion bekommt den Event ev und einen Zeiger auf eine Struktur vom Typ clickinfo als Parameter. Als Ergebnis wird die Struktur gefüllt. Die Komponenten fields_only und activated werden nicht berücksichtigt.

Als Mausposition wird die Position verwendet, die beim letzten Event von FIX/Win übermittelt wurde (Auch das Drücken einer Taste übermittelt die Mausposition). Als Grundlage zur Bestimmung des Objektes, des Feldes und der Zeile innerhalb einer Tabelle wird der zum Zeitpunkt des Aufrufes aktuelle Bildschirmzustand verwendet. Das kann ein anderer sein, als der zum Zeitpunkt an dem das Event eingetroffen ist, wenn zwischenzeitlich Objekte neu dargestellt oder entfernt wurden. Deshalb sollte fill_clickinfo() möglichst frühzeitig aufgerufen werden.

9 Einfügen von Feldwerten

Beim Einfügen von Feldwerten (Paste) und der Verwendung von Proportionalschrift in Feldern wurde bisher der Wert ungeachtet des Feldformates direkt in das Edit Control kopiert. Erst wenn das Feld verlassen wurde, wurde der Wert von FIX geprüft. Damit ist es möglich, Zeichen, die nicht zum Format des Feldes passen (z.B. Buchstaben und Leerzeichen bei numerischen Feldern), in das Feld einzufügen. Erst beim Verlassen des Feldes werden die fehlerhaften Zeichen entdeckt.

Ab der Version 4.2.0 kann die anwendungsabhängige Ressource (in der *.frc Datei der Anwendung)

```
PasteAsChars: TRUE
```

gesetzt werden. In diesem Fall wird der Text der Zwischenablage als Einzelzeichen an das Edit Control gesendet. Das Edit Control kann dann für jedes Zeichen entscheiden, ob es angenommen wird oder nicht. Damit wird die gleiche Logik verwendet, wie sie auch bei der Eingabe über Tastatur aktiv ist.

Wenn die Ressource auf FALSE gesetzt wird, was auch der Default ist, wird das bisherige Verfahren verwendet.

10 Sortierung der Dateiliste im led

Die Dateiliste, die bei der inkrementellen Suche zur Auswahl der mfo-/men-Datei dargestellt wird, wird jetzt sortiert.

11 Verschieben von Prompt- und Messagezeile

Mittels

```
void set_prompt_offset(int y_pos_offset)
```

kann die Promptzeile verschoben werden. Der Wert y_pos_offset wird genauso interpretiert wie die Ressource PromptWindowYPosOffset. Er gibt die Verschiebung zur Defaultposition (in der letzten Zeile am unteren Bildrand) an. Hier sind nur negative Wert sinnvoll, die die Zeile weiter nach oben bewegen.

Mittels

```
void set_msg_offset(int y_pos_offset)
```

kann die Messagezeile verschoben werden. Der Wert y_pos_offset wird genauso interpretiert wie die Ressource MsgWindowYPosOffset. Er gibt die Verschiebung zur Defaultposition (in der vorletzten Zeile am unteren Bildrand) an. Der Wert 1 verschiebt die Messagezeile in die letzte Zeile. Negative Werte schieben die Zeile weiter nach oben.

Einbindung von ActiveX Controls

1 Ziel

Innerhalb einer Maske soll es möglich sein, ActiveX Controls einzubinden. Sie sollen sich auch über mehrere Zeilen erstrecken können.

2 Probleme und Lösungen

Zentrales Problem bei der Darstellung des ActiveX Controls ist die Überlappung mir anderen Fenstern. Alle Menüs, Masken, Selos und Choices liegen aus Sicht von FIX/Win in einem Fenster - dem FIX/Win-Fenster (siehe Definition im FIX/Win Handbuch). Wenn ein ActiveX Control eingeblendet wird, dann legt es sich über das FIX/Win-Fenster. Alle FIX Objekte liegen damit unter dem ActiveX Control. Es ist nicht möglich, dass sich beispielsweise ein Selo über das ActiveX Control legt.

Zur Lösung dieses Problems wird folgendes Verfahren angewendet:

(1) Für das ActiveX Control wird ein aktiver und ein nicht aktiver Zustand definiert.

(2) Im aktiven Zustand kann das ActiveX Control bedient werden. Es bekommt den Eingabefokus, das zugehörige Fenster wird eingeblendet und in den Vordergrund gebracht.

(3) Im nicht aktiven Zustand wird das ActiveX Control ausgeblendet. Statt des Controls wird ein Bereich angezeigt, der mit Paintareas gefüllt wird. Die Paintareas stellen ein graphisches Abbild des Controls dar.

3 Implementierung

3.1 Schnittstelle zu FIX

Definition

Mittels

```
HAXCTRL axctrl_create(field *f, short variante, long ax_id,
     int pos_y, int pos_x, int height, int width,
     int bt_mask, char**ptrdata, int maxdatasize);
```

kann ein ActiveX Control - oder besser gesagt die Datenstruktur zum Zugriff auf FIX Seite - erzeugt werden. pos_y, pos_x, height und width definieren Position und Größe innerhalb der Maske. FIX legt in dem Bereich die benötigten Paintareas an. ax_id ist eine eindeutige ID, die an FIX/Win weitergegeben wird. FIX/Win bzw. die Benutzerbibliothek (FWOWN) erkennen an der ID, welches ActiveX Control zu erstellen ist. Die ID muss eindeutig innerhalb der Anwendung sein. Bei der Verwendung der gleichen ID kehrt die Funktion mit einem

Fehler zurück.

f definiert das Feld, an das das ActiveX Control gebunden wird. Das ist notwendig, um den logischen Ablauf der FIX Maskenlogik zu gewährleisten. Wenn das Feld betreten wird, dann wird statt der Felderfassung das ActiveX Control auf der Windows-Seite aktiviert. Aus Sicht von FIX und der Anwendung gibt es jedoch keinen Unterschied zur normalen Felderfassung. Damit wird beispielsweise das ActiveX Control während des normalen Maskendurchlaufs aktiviert. Die Events, die dazu an die Anwendung gesendet werden, unterscheiden sich nicht von einer normalen Felderfassung. Der einzige Unterschied ist, dass das Feld nicht dargestellt wird.

Gleichzeitig bestimmt f seinem Vaterobjekt die Maske, in die das ActiveX Control eingebettet wird. Wenn es mehrere Varianten gibt, dann ist die Nummer der Variante als variant zu übergeben. Ansonsten ist der Wert mit 0 zu belegen.

ptrdata definiert die Adresse einer Zeigervariablen, der ein Puffer zur Kommunikation zugewiesen wird. Die gewünschte Größe des Puffers ist als maxdatasize mitzugeben.

Als Ergebnis liefert die Funktion ein Handle zurück, das zur weiteren Kommunikation mit dem ActiveX Control auf FIX-Seite zu verwenden ist. Der Vorteile gegenüber der ax_id liegt im schnelleren Zugriff. Im Falle eines Fehlers wird HAXCTRL_INVALID zurückgeliefert.

Datenaustausch

Mit

```
char* axctrl_transfer_data(HAXCTRL h_ax, int direction, int *realsize);
```

können Daten mit dem ActiveX Control ausgetauscht werden. Das bedeutet, dass Daten an die Benutzerbibliothek gesendet werden und Daten von dieser empfangen werden. h_ax bestimmt, um welches ActiveX Control es sich handelt (Rückgabewert von axctrl_create()). Mit dem Parameter direction kann festgelegt werden, ob die Daten dem ActiveX Control zugewiesen werden (AXCTRL_SEND), oder ob Daten vom ActiveX Control ausgelesen (AXCTRL_RECEIVE) werden. Die Daten, die zu senden sind, sind in dem bei axctrl_create() erzeugten Puffer abzulegen. Hierbei ist darauf zu achten, dass maxdatasize nicht überschritten wird. In *realsize ist die tatsächliche Größe der Daten, die zu übertragen sind, abzulegen. Nach dem Senden der Daten und dem Zurücksenden der Antwort kehrt die Funktionen mit einem Zeiger auf den Datenbereich mit den zurückgesendeten Daten zurück. Die Größe des Datenbereich wird nach *realsize geschrieben.

Es ist zu beachten, dass in beiden Fällen (direction==AXCTRL_RECEIVE/AXCTRL_SEND) Daten gesendet und empfangen werden. Im Fall direction==AXCTRL_RECEIVE dienen die gesendeten Daten dazu, genauer zu spezifizieren, welche Daten vom ActiveX Control ausgelesen werden sollen. Im Fall direction==AXCTRL_SEND liefern die zurückgesendeten Daten eine Bestätigung oder den Hinweis auf einen Fehler.

Wenn mittels direction==AXCTRL_SEND viele kleine Puffer mit Daten gesendet werden, kann es zu Performanceproblemen kommen. Dies liegt zum Teil an den kleinen Datenmengen. Der Netzwerktreiber wartet iin diesem Fall auf weitere Daten, um die Paketgröße zu optimieren. Um dieses Problem zu beheben, kann die Socketoption TCP_NODELAY für die einzelnen Sockets gesetzt werden. FIX/Win und FIX unter UNIX setzen diese Option automatisch, wenn nicht die Umgebungsvariable NO_TCP_NODELAY gesetzt wird. Bei FIX unter Windows muss der FIX-Server fxsrv diese Option setzen. Er tut dies ebenfalls standardmässig, wenn die Variable nicht gesetzt wird.

Ein anderer Grund liegt in den vielen Roundtrips, die durch kleine Puffer entstehen. Um dem entgegen zu wirken, kann statt des Wertes AXCTRL_SEND der Wert AXCTRL_SENDNOWAIT verwendet werden. FIX/Win sendet in diesem Fall keine Antwort zurück. Damit wird die Anzahl der Roundtrips minimiert. Als Rückgabewert liefert die Funktion in diesem Fall einen Zeiger auf den Puffer, der zum Datenaustausch verwendet wird, zurück. Ein weiterer Performancevorteil ergibt sich daraus, das FIX/Win nicht bei jeder Datenübertragung das interne Abbild des ActiveX Controls aktualisiert. Damit das jedoch passiert und der Benutzer die Änderung sehen kann, ist es notwendig, das zum Schluss einmal AXCTRL_SEND verwendet wird.

Weitere wichtige Hinweise zur Verwendung von AXCTRL_SENDNOWAIT sind bei der Beschreibung der Frontend Funktion

Das Format der Daten bleibt der Anwendung überlassen. Letztendlich werden die Daten an die Benutzerbibliothek weitergegeben und sind von dieser an das ActiveX Control weiterzugeben.

Da das ActiveX Control an ein Feld gebunden wird, hätte es sich angeboten, statt dieses Datenaustauschs einfach den Feldwert zu verwenden. Das hätte jedoch folgende Nachteile:

- Die Länge eines Feldes ist auf 2048 Zeichen begrenzt.
- Der Datenaustausch findet nur beim Betreten und Verlassen des Feldes statt. In den meisten Fällen sollen jedoch auch Daten zu anderen Zeitpunkten an das ActiveX Control gesendet werden. Z.B. wenn ein Datensatz gelesen wurde oder wenn sich relevante Daten geändert haben.
- Es ist nicht immer notwendig, alle Daten zu senden oder zu empfangen. Es genügt, die Änderungen auszutauschen. In dem Feld will man jedoch den kompletten Wert halten.

Mausbedienbarkeit

Bezüglich der Mausbedienbarkeit müssen zwei Zustande unterschieden werden:

- Das ActiveX Control ist aktiv.
- Das ActiveX Control ist nicht aktiv und wird durch sein Abbild aus Paintareas dargestellt.

Maustasten im ersten Zustand (aktiv) sind komplett vom ActiveX Control selbst zu behandeln. Im zweiten Zustand (nicht aktiv) werden Mausklicks auf den Bereich, der für das Control reserviert ist, an FIX weitergeleitet und kommen dort als Events:

```
BT_LEFT_AX
BT_MIDDLE_AX
BT_RIGHT_AX
```

an. Mit der Funktion

```
long axctrl_get_clicked_id(void);
```

kann die ID (Wert ax_id bei der Erzeugung mit axctrl_create()) abgefragt werden. Um ein Handle zu erhalten, kann die Funktion

```
HAXCTRL axctrl_get(int ax_id);
```

verwendet werden. Mit Hilfe des Handles und der Funktion

```
int axctrl_getinfo(HAXCTRL h_ax, int *ax_id, field **f,
     int *pos_y, int *pos_x,   int *height, int *width,
     unsigned char *bt_mask,   char **ptrdata, long *maxdatasize);
```

können weitere Daten ermittelt werden. Alle Parameter hinter h_ax sind Zeiger auf Variablen, die mit den bei axctrl_create() übergebnen Werten gefüllt werden. Wenn ein Wert nicht benötigt wird, kann statt eines Zeigers NULL übergeben werden.

Damit bleibt es der Anwendung überlassen, auf Mausklicks zu reagieren. Um in diesem Fall das ActiveX Control zu aktivieren, kann einfach das Feld besucht werden, dem das Control zugeordnet wurde (z.B. mit F_GOTO). Die Anwendung hat jedoch die Möglichkeit, vorher Prüfungen zu starten und eine Aktivierung zu unterbinden.

Damit Mausklicks gesendet werden, ist der Parameter bt_mask beim Aufruf von axctrl_create() entsprechend zu besetzten. Hier können die gleichen Konstanten wie bei Paintareas verwendet werden. Weitere Bedingung ist, das das Objekt, indem das ActiveX Control eingebettet ist, zu dem Zeitpunkt aktiv ist oder zur Liste der aktiven Objekte gehört (set_activeobj_list()).

Ein nur schwer lösbares Problem liegt darin, dass der Mausklick für das ActiveX Control verloren geht. Das bedeutet das beispielsweise der Klick auf einen bestimmten Tag eines Kalender ActiveX Controls nicht diesen Tag auswählt. Stattdessen wird das Control aktiviert. Erst ein zweiter Klick selektiert den Tag.

Freigabe

Die Datenstruktur für ein ActiveX Control kann bereits unmittelbar nach dem Laden einer Maske erzeugt werden. Damit werden auch die dafür benötigten Paintareas angelegt.

Beim Entfernen der Maske vom Bildschirm werden die Paintareas - wie alle anderen Paintareas auch - entfernt. Wenn die Maske neu zur Anzeige gebracht wird, dann werden neue Paintareas erzeugt. Der Paintarea-Cache sorgt dabei für Optimierungen.

Die Datenstruktur für ein ActiveX Control bleibt jedoch beim Entfernen der Maske vom Bildschirm bestehen. Damit bleibt auch das ActiveX Control auf Fix/Win Seite bestehen.

Erst das Entfernen des Objekts mit o_free() entfernt auch das ActiveX Controls und die benötigten Datenstrukturen dafür.

Wenn das Objekt nicht mit o_free() freigegeben wird, kann das ActiveX Control sowie die Datenstruktur durch Aufruf der Funktion

```
int axctrl_delete(HAXCTRL h_ax)
```

explizit freigegeben werden.

Erst wenn ein ActiveX Control freigegegen wurde, kann ein neues mit der gleichen ax_id mit axctrl_create() erzeugt werden.

Als Parameter bekommt die Funktion das beim Erzeugen vergebene Handle. Bei Erfolg liefert sie 1 als Rückgabewert und sonst 0. In diesem Fall ist höchstwahrscheinlich das Handle falsch.

Beispiel

Es soll ein ActiveX Control zur Auswahl von Farben über die RGB-Komponenten in eine Maske eingebunden werden. Die Komponenten können im ActiveX Control bestimmt werden. Zusätzlich werden sie in den (FIX-)Feldern COLOR_RED, COLOR_GREEN und COLOR_BLUE angezeigt und können dort auch eingegeben werden. Wenn die Farben im ActiveX Control geändert wurden, dann sind auch die Feldwerte anzupassen. Wenn umgekehrt die Feldwerte geändert wurden, dann ist der Zustand des ActiveX Controls anzupassen.

Nach dem Laden wird das ActiveX Control für eine Maske erzeugt und an ein Feld gebunden.

```
HAXCTRL h_ax_color;
static char* color_transfer_buffer = NULL;
#define COLOR_TRANSFER_BUFFER_MAX 1024
#define COLOR_CTRL 10

h_ax_color = axctrl_create(MF(COLOR_mskp, COLOR_COLINFO), COLOR_CTRL, 0,
      6, 24, 8, 26, PA_BT_LEFT,
      &color_transfer_buffer, COLOR_TRANSFER_BUFFER_MAX);
```

Der Bereich von 8 Zeilen x 26 Zeichen an der Position 6,24 wurde in der Maske freigelassen. Als ID wurde COLOR_CTRL 10 gewählt. Hier ist jede von 0 verschiedene Zahl möglich. Der 1024 Byte große Puffer zur Kommunikation wird in der Variablen color_transfer_buffer gehalten. Das Control wird an das Feld COLOR_COLINFO gebunden, dessen Aufgabe es ist, den Zeitpunkt der Aktivierens innerhalb des Maskendurchlaufs zu bestimmen.

Zum Senden von Daten an das Control dient die Funktion

```
static void updateAxControl(int part)
{
      int size;

      switch (part) {
      case -1:
            sprintf(color_transfer_buffer, "RGB %d %d %d",
```

```
            m_color.red, m_color.green, m_color.blue);
            break;
      case COLOR_RED:
            sprintf(color_transfer_buffer, "RED %d",
            m_color.red);
            break;
      case COLOR_GREEN:
            sprintf(color_transfer_buffer, "GREEN %d",
            m_color.green);
            break;
      case COLOR_BLUE:
            sprintf(color_transfer_buffer, "BLUE %d",
            m_color.blue);
            break;
      }
      size = strlen(color_transfer_buffer);
      axctrl_transfer_data(h_ax_color, AXCTRL_SEND, &size);
}
```

Sie verwendet ein einfaches Datenformat, indem die Werte einfach hinter das Kürzel für die Komponente geschrieben werden. Für den Rückgabewert interessiert sie sich in diesem einfachen Beispiel nicht. Aufgerufen wird die Funktion nach dem Vorwärts- oder Rückwärtsverlassen eines der Farbfelder:

```
case L_PRVFIELD:
      ...
      if (prg_elemnr == COLOR_RED ||
            prg_elemnr == COLOR_BLUE ||
            prg_elemnr == COLOR_GREEN) {
            updateAxControl(prg_elemnr);
      }
      ...
      break;

case L_NXTFIELD:
      ...
      if (prg_elemnr == COLOR_RED ||
            prg_elemnr == COLOR_BLUE ||
            prg_elemnr == COLOR_GREEN) {
            updateAxControl(prg_elemnr);
      }
      ...
      break;
```

Zum Lesen von Daten wird die Funktion

```
static void updateAxFields(int part)
{
      int size = 0;
      char* answer;

      switch (part) {
      case -1:
            sprintf(color_transfer_buffer, "RGB");
            break;
      case COLOR_RED:
            sprintf(color_transfer_buffer, "RED");
            break;
      case COLOR_GREEN:
            sprintf(color_transfer_buffer, "GREEN");
            break;
      case COLOR_BLUE:
            sprintf(color_transfer_buffer, "BLUE");
            break;
      }

      size = strlen(color_transfer_buffer);
      answer = axctrl_transfer_data(h_ax_color, AXCTRL_RECEIVE, &size);

      switch (part) {
      case -1:
            sscanf(answer, "%d %d %d", &m_color.red, &m_color.green, &m_color.blue);
```

```
            break;
      case COLOR_RED:
            sscanf(answer, "%d", &m_color.red);
            break;
      case COLOR_GREEN:
            sscanf(answer, "%d", &m_color.green);
            break;
      case COLOR_BLUE:
            sscanf(answer, "%d", &m_color.blue);
            break;
      }

      m_wrktoinfo(COLOR_mskp, TRUE);
}
```

verwendet. Sie sendet als Request "RGB" oder einen Farbnamen an die FIX/Win Seite und bekommt daraufhin die Werte der RGB-Komponenten oder die entsprechende Farbe zurück. Für komplexere Beispiele wäre es denkbar, binäre Daten zu verwenden (Zahlen/Strukturen). In diesem Fall müssen die Funktionen zur Konvertierung der Byteorder (z.B. ntohl()/htonl()) verwendet werden.

Die Funktion updateAxFields() wird beim Verlassen des Feldes COLOR_COLINFO, die dem ActiveX Control zugeordnet ist, aufgerufen. Sie wird also immer dann aufgerufen, wenn das ActiveX Control verlassen wird.

```
case L_PRVFIELD:
      ...
      if (prg_elemnr == COLOR_COLINFO) {
            updateAxFields(-1);
      }
      ...

case L_NXTFIELD:
      ...
      if (prg_elemnr == COLOR_COLINFO) {
            updateAxFields(-1);
      }
      ...
```

Um das ActiveX Control bei einem Mausklick zu aktivieren wird einfach das Event BT_LEFT_AX abgefangen. Auf das Ermitteln der ID mittels axctrl_get_clicked_id() wird verzichtet, da es nur ein ActiveX Control in der Maske gibt.

```
case BT_LEFT_AX:
      F_GOTO(COLOR_COLINFO);
      return L_STAY;
```

3.2 Schnittstelle zur Benutzerbibliothek

Die Benutzerbibliothek ist für die Erzeugung und für den Datenaustausch des ActiveX Controls verantwortlich. Die einfachste Möglichkeit zur Erzeugung von ActiveX Controls ist der Zugriff über ATL.

Verwendung von ATL

Für die mit FIX/Win ausgelieferte Bibliothek im Verzeichnis nsigrp/fwown ist dazu wie folgt vorzugehen:

(1) Includedateien einbinden

```
#include <comsvcs.h>
#include <atlbase.h>
#include <atlcom.h>
```

```
#include <atlctl.h>
```

(2) Dummy Modul erzeugen

```
class CDummyModule : public CAtlDllModuleT<CDummyModule> {};
CDummyModule _Module;
```

```
stdafx.h
stdafx.cpp
```

(3) ATL initialisieren

```
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD reason, LPVOID reserved)
{
      switch (reason) {
      ...
      case DLL_PROCESS_ATTACH:
            AtlAxWinInit();
            ...
            break;
      case DLL_PROCESS_DETACH:
            AtlAxWinTerm();
            ...
            break;
      ...
      }
      return TRUE;
}
```

Auf ähnliche Weise kann die eigene Benutzerbibliothek umgestellt.

Deklarationen für ActiveX

Um die notwendigen Deklarationen für die ActiveX Controls zu erzeugen, kann einfach die ocx-Datei importiert werden.

Dadurch erzeugt der Compiler eine Headerdatei (Endung .thi) und bindet diese automatisch ein.

Erzeugen des Controls

FIX/Win ruft als eine der Reaktionen auf axctrl_create() die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownCreateAxctrl(CallBackFct p_callBack,
      void* p_callID, long p_ax_id , HWND*  p_axctrl,
      int p_x, int p_y, int p_width, int p_height, HWND p_parent)
```

auf, sobald das ActiveX Controls das erste Mal angezeigt werden soll. Sie hat die Aufgabe, das ActiveX Control zu erzeugen. Der Parameter p_ax_id definiert, um welches Control es sich handelt. Er wird von axctrl_create() durchgereicht. p_x, p_y, p_width und p_height definiert und Position und Größe des Controls. p_parent enthält das Handle des Fensters, in das das Control eingebettet werden soll.

Als Rückgabewert muss die Funktion NULL oder einen Fehlertext liefern. Das Fensterhandle des erzeugten Controls ist in die Variable p_axctrl zu schreiben.

Zum Erzeugen eines ActiveX Controls mittels ATL ist ein Fenster der Klasse "AtlAxWin" mittels der normalen CreateWindow() Funktion zu erzeugen. Da sich der Name der Klasse von Version zu Version ändert und nur in den ersten Versionen exakt "AtlAxWin" lautete, sollte er durch Aufruf von CAxWindow2::GetWndClassName() jedesmal ermittelt werden. Die CLSID des Controls ist als Fenstertext an CreateWindow() zu übergeben. Alle anderen Parameter werden wie üblich gesetzt.

Zeichnen eines Rahmens

Einige ActiveX Controls zeichnen ihren Rahmen selbst, andere nicht. Aus diesem Grund besteht die Möglichkeit, den Rahmen von der Benutzerbibliohtek zeichnen zu lassen. Dazu ist die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownDrawAxctrlFrame(CallBackFct p_callBack,
     void* p_callID,     long p_ax_id, RECT* p_frame, HDC p_hDC,
     int p_width, int p_height) {
```

zu implementieren. Sie wird von FIX/Win vor der Erzeugung eines ActiveX Controls aufgerufen. p_hDC ist das Handle zu der Zeichenfläche, in die der Rahmen zu zeichnen ist. p_width und p_height sind die Dimensionen der Zeichenfläche. Anhand des Wertes von p_ax_id, der wie bei FwownCreateAxctrl() definiert um welches Control es sich handelt, kann entschieden werden, ob und in welcher Weise ein Rahmen gezeichnet werden soll. Wenn ein Rahmen gezeichnet wird, dann sind die Breiten für oben, unten, links und rechts in p_frame einzutragen. Die Größe des ActiveX Controls wird um diese Beträge verringert.

Datenaustausch

Als Reaktion auf axctrl_transfer_data() ruft FIX/Win die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownProcessAxData(CallBackFct p_callBack,
     void* p_callID, int p_direction, long p_ax_id, HWND p_axctrl,
     char* p_data, long* p_realsize, long maxdatasize);
```

auf. Der Parameter p_direction bestimmt, ab Daten an das ActiveX Control übertragen werden sollen (AXCTRL_SEND), oder ob Daten von ihm gelesen werden sollen (AXCTRL_RECEIVE). Um welches Control es sich handelt steht in ax_id. Zusätzlich wird in p_axctrl das Fentsterhandle des Controls mitgegeben. Die Daten stehen in p_data und deren Anzahl Bytes steht in *p_realsize. Die Daten in diesem Puffer werden auch wieder zurückgesendet. Sie können also während der Abarbeitung verändert oder ersetzt werden. Dabei sollte die maximale Größe von maxdatasize nicht überschritten werden.

Wenn Daten gesendet werden, dann stehen die Daten in p_data/p_realsize. Als Rückgabe sollte eine mindestens ein Byte große Bestätigung in den Puffer geschrieben werden.

Wenn Daten gelesen werden, dann steht in p_data/p_realsize die Art der Daten, die gelesen werden sollen (also eine Beschreibung des Request). Als Rückgabe werden die vom Control gelesenen Daten nach p_data/p_realsize geschrieben.

Das Format der Daten kann frei durch die Anwendung definiert werden. FIX und FIX/Win reichen die Daten lediglich weiter.

Wenn für den Parameter p_direction der Wert AXCTRL_SENDNOWAIT angegeben wird, dann werden keine Daten zurückgesendet. Dies bringt einige Performacevorteile (siehe Beschreibung von axctrl_transfer_data()). In diesem Fall muss also nach p_data/p_realsize keine Bestätigung geschrieben werden. Allerdings sollte man in diesem Fall keine Dialoge oder Messageboxen innerhalb von FwownProcessAxData() starten. Dadurch wird eine zweite Nachrichtenschleife gestartet, die zu Störungen im Protokol von FIX und FIX/Win führen kann.

Tastenbelegung

Wenn ein ActiveX Control aktiv ist, dann wird bei jedem Tastendruck bestimmt, ob

- die Taste an das ActiveX Control gesendet werden soll.
- die Taste als Event an FIX gesendet werden soll.
- die Taste ignoriert werden soll.

Dazu wird die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownHandleAxKey(
     CallBackFct p_callBack, void* p_callID, long p_ax_id, HWND p_axctrl,
     WPARAM p_key, bool p_keyFShift, bool p_keyFCtrl, bool p_keyFAlt,
     int* action);
```

aufgerufen. Die Taste wird durch p_key (virtueller Tastencode), p_keyFShift (Zustand von Shift), p_keyFCtrl (Zustand von Control) und p_keyFAlt (Zustand von Alt) bestimmt. Das ActiveX Control wird duch die ax_id definiert. p_axctrl enthält das Fensterhandle zu dem Control.

In action ist ein Code für das weitere Vorgehen zu hinterlegen. Bei AXCTRL_KEYACTION_IGNORE wird die Taste ignoriert. Dieser Wert kann verwendet werden, wenn in FwownHandleAxKey() die Taste behandlt wird und bestimmte Methoden an dem ActiveX Control aufgrund einer Taste aufgerufen werden.

Bei AXCTRL_KEYACTION_SENDTOAX wird die Taste an das ActiveX Controls gesendet, und das Control ist für die Verarbeitung zustängig.

Wenn AXCTRL_KEYACTION_SENDTOFIX nach der Rückkehr in action steht, dann wird geprüft, ob zu der Taste ein FIX-Event in der Tastentabelle hinterlegt ist. In diesem Fall wird das ActiveX Controll deaktiviert und das Event an FIX gesendet. Ansonsten wird die Taste ignoriert.

Wenn innerhalb von FwownHandleAxKey() ein Fehler auftritt, dann kann eine Fehlermeldung zurückgegeben werden, die von FIX/Win ausgegeben wird.

Freigabe

Wenn von FIX/Win die Datenstruktur eines ActiveX Controls freigegeben wurde, dann wird die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownDeleteAxctrl(CallBackFct p_callBack,
     void* p_callID, long p_ax_id, HWND p_axctrl)
```

zur Freigabe des Controls aufgerufen.

Beispiel

Das folgende Beispiel zeigt den Code zu dem oben begonnen Beispiel auf Seite der Benutzerbibliothek.

Zum Erzeugen des ActiveX Controls wird folgende Funktion implementiert.

```
#import "FxColor.ocx"
using namespace FxColorLib;

#define AX_COLOR 10

FWOWN_API _TCHAR* CALLBACK FwownCreateAxctrl(CallBackFct p_callBack,
     void* p_callID, long p_ax_id, HWND* p_axctrl,
     int p_x, int p_y, int p_width, int p_height, HWND p_parent) {

     switch(p_ax_id) {
     case AX_COLOR:
            *p_axctrl = ::CreateWindow(CAxWindow2::GetWndClassName(),
                  _T("{cbe80dd8-3883-44da-8724-795afbc80ed4}"),
                  WS_CHILD, p_x, p_y, p_width, p_height,
                  (HWND)p_parent, NULL, hInst_lib, NULL);
            break;
     default:
            _stprintf_s(errorbuf, ERRORBUFLEN,
            _T("FwownCreateAxctrl(): Control with id %d ist not supported\n"), p_ax_id);
            return errorbuf;
     }
     if (*p_axctrl == NULL) {
            _stprintf_s(errorbuf, ERRORBUFLEN,
```

```
            _T("FwownCreateAxctrl(): CreateWindow() failed with %ld\n"),
GetLastError());
            return errorbuf;
      }
      return NULL;
}
```

FxColor.ocx ist das ActiveX Control, das eingebunden werden soll. Durch die #import-Anweisung wird eine Headerdatei mit allen notwendigen Deklarationen erzeugt und eingebunden. Der Namespace (FxColorLib) kann in dieser Datei (fxcolor.tlh) nachgesehen werden. Die zum Erzeugen des Fensters benötigte CLSID (cbe80dd8-3883-44da-8724-795afbc80ed4) steht ebenfalls in dieser Datei.

Das Control wird als Kindfenster (WS_CHILD) zum FIX/Win Fenster (p_parent) erzeugt. Damit liegt es über diesem und wird auch mit verschoben.

Zum Datenaustausch wird die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownProcessAxData(CallBackFct p_callBack,
      void* p_callID, int  p_direction, long p_ax_id, HWND p_axctrl,
      char* p_data, long* p_realsize, long maxdatasize) {

      switch(p_ax_id) {
      case AX_COLOR:
            return ProcessFxColorData(p_axctrl, p_direction, p_data,
                  p_realsize, maxdatasize);
            break;
      }
      _stprintf_s(errorbuf, ERRORBUFLEN,
            _T("FwownProcessAxData(): Control with id %d ist not supported\n"),
            p_ax_id);
      return errorbuf;
}
```

implementiert. Sie reicht die Parameter im Falle AX_COLOR an die Funktion

```
static _TCHAR* ProcessFxColorData(HWND p_axctrl, int p_direction,
      char* p_data, long* p_realsize, long maxdatasize)
{
      IUnknown *pUnk = NULL;
      HRESULT hr;

      if (p_axctrl == NULL) {
            _stprintf_s(errorbuf, ERRORBUFLEN,
            _T("ProcessFxColorData(): Control does not exist\n"));
            return errorbuf;
      }

      hr = AtlAxGetControl(p_axctrl, &pUnk);
      if (FAILED(hr)) {
            _stprintf_s(errorbuf, ERRORBUFLEN,
            _T("ProcessFxColorData(): AtlAxGetControl failed with %ld\n"), hr);
            return errorbuf;
      }

      CComPtr<_DFxColor> spColor;
      hr = pUnk->QueryInterface(__uuidof(spColor), (void**)&spColor);
      if (FAILED(hr)) {
            _stprintf_s(errorbuf, ERRORBUFLEN,
            _T("ProcessFxColorData(): QueryInterface failed with %ld\n"), hr);
            return errorbuf;
      }

      if (p_direction == AXCTRL_SEND) {
            if (strcmp(p_data, "RGB") == 0) {
                  sprintf_s(p_data, maxdatasize, "%d %d %d",
                        spColor->GetRed(),
                        spColor->GetGreen(),
                        spColor->GetBlue());
            }
            else if (strcmp(p_data, "RED") == 0) {
                  sprintf_s(p_data, maxdatasize, "%d", spColor->GetRed());
```

```
            }
            else if (strcmp(p_data, "GREEN") == 0) {
                  sprintf_s(p_data, maxdatasize, "%d", spColor->GetGreen());
            }
            else if (strcmp(p_data, "BLUE") == 0) {
                  sprintf_s(p_data, maxdatasize, "%d", spColor->GetBlue());
            }
            *p_realsize = strlen(p_data) + 1;
      }
      else {
            int red, green, blue;

            if (strncmp(p_data, "RGB", 3) == 0) {
                  sscanf_s(p_data + 3, "%d %d %d", &red, &green, &blue);
                  spColor->SetRed(red);
                  spColor->SetGreen(green);
                  spColor->SetBlue(blue);
            }
            else if (strncmp(p_data, "RED", 3) == 0) {
                  sscanf_s(p_data + 3, "%d", &red);
                  spColor->SetRed(red);
            }
            else if (strncmp(p_data, "GREEN", 5) == 0) {
                  sscanf_s(p_data + 5, "%d", &green);
                  spColor->SetGreen(green);
            }
            else if (strncmp(p_data, "BLUE", 4) == 0) {
                  sscanf_s(p_data + 4, "%d", &blue);
                  spColor->SetBlue(blue);
            }
            sprintf_s(p_data, maxdatasize, "OK");
            *p_realsize = strlen(p_data) + 1;
      }
      return NULL;
}
```

weiter. Im Falle p_direction==AXCTRL_RECEIVE liest sie die Werte für Rot, Grün, Blau vom ActiveX Control und schreibt sie in den Puffer. Im Fall p_direction==AXCTRL_SEND werden die Werte in Variablen gelesen und dann an dem ActiveX Control gesetzt. Als Antwort wird das Wort "OK" verwendet. Je nach Request werden alle Farben oder nur einzelne Komponenten behandelt.

Zur Definition der möglichen Tasten wird die Funktion

```
FWOWN_API bool CALLBACK FwownHandleAxKey(CallBackFct p_callBack, void* p_callID,
      long p_ax_id, HWND p_axctrl,
      WPARAM p_key, bool p_keyFShift, bool p_keyFCtrl, bool p_keyFAlt, int *action) {

      *action = AXCTRL_KEYACTION_SENDTOAX;
      switch(p_ax_id) {
      case AX_COLOR:
            switch (p_key) {
            case VK_TAB:
            case VK_HOME:
            case VK_END:
            case VK_ESCAPE:
                  *action = AXCTRL_KEYACTION_SENDTOFIX;
            }
      }
      return NULL;
}
```

implementiert. Sie liefert für die Tasten Tab, Pos1, Ende und Escape unabhängig von dem Modifier den Wert AXCTRL_KEYACTION_SENDTOFIX. Diese Tasten werden deshalb an FIX weitergereicht. Für anderen Tasten liefert sie AXCTRL_KEYACTION_SENDTOAX. Sie werden deshalb an das ActiveX Controls weitergegeben.

Zum Entfernen des ActiveX Controls wird die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownDeleteAxctrl(CallBackFct p_callBack,
      void* p_callID, long p_ax_id, HWND p_axctrl)
{
```

```
      switch(p_ax_id) {
      case AX_COLOR:
            if (p_axctrl != NULL)
                  DestroyWindow(p_axctrl);
            break;
      }
      return NULL;
}
```

implementiert. Da das Fenster für ein ActiveX Controls erst beim ersten Anzeigen erzeugt wird, wird geprüft, ob ein Handle für das Fenster vorhanden ist. Nur in diesem Fall wird das Fenster entfernt.

3.3 FIX/Win

FIX/Win übernimmt die Daten von FIX und reicht sie an die Benutzerbibliothek weiter. Die Hauptaufgabe von FIX/Win besteht jedoch darin, das ActiveX Control zum richtigen Zeitpunkt zu aktivieren und zu deaktivieren und eine interne Darstellung im Speicher zu erzeugen, die immer dann angezeigt wird, wenn das ActiveX Control nicht aktiv ist. Damit das funktioniert, muss das ActiveX Control das Interface

```
IViewObject
```

implementieren. Bei Controls, die mit MFC erstellt wurden, ist das immer der Fall, da sie von COleControl abgeleitet wurden. Allerdings ist es auch wichtig, dass bei einer Anforderung über IViewObject das Control richtig gezeichnet wird. Ob ein ActiveX Control mit FIX/Win richtig funktioniert, sollte daher in einem Versuch getestet werden.

Ansonsten ist es sinnvoll, dass das ActiveX Control visualisiert, dass es den Fokus besitzt. Bei selbst erstellten Controls ist das durch Abfangen der Fokusanforderung leicht machbar.

Eine weitere Aufgabe von FIX/Win besteht in der Tastensteuerung. FIX/Win fängt sämtliche Tasten ab und leitet nur bestimmte Tasten an das ActiveX Control weiter. Andere Tasten führen zur Beendigung des von FIX gestarteten Vorgangs der Felderfassung. Damit wird auch das ActiveX Control deaktiviert.

Damit die Tastensteuerung korrekt funktioniert, ist es wichtig, dass das ActiveX Control als ein Kindfenster des Fixwin-Fensters erzeugt wird. Dabei muss es sich nicht um ein direktes Kind handeln. Es können beliebig viele Fenster dazwischen eingefügt werden.

3.4 Probleme bei Mehrfachverwendung

Bei ActiveX Controls wird davon ausgegangen, dass es sich um größere Objekte handelt, die nicht häufig in einem Programm auftauchen. Deshalb wird auch nicht davon ausgegangen, dass ein und das selbe Control mehrfach in einer Anwendung an verschiedenen Stellen benutzt wird. Dies funktioniert jedoch solange, wie das (oder die) jeweils andere Control vor dem Erzeugen des anderen freigegeben wird. Sollte der Fall eintreten, bei dem es notwendig wird gleichzeitig mit mehreren Instanzen des gleichen Controls zu arbeiten, dann ist für jede Instanz eine eigene eigene ax_id zu vergeben.

Aus Sicht der Benutzerbibliothek kann es auch auf andere Art zu einer Mehrfachverwendung kommen. Nämlich dann, wenn in einer FIX/Win Instanz mehrere Verbindungen gestartet werden (also ohne den Modus /single). Um in diesem Fall Problemen vorzubeugen, ist es wichtig, dass ein ActiveX Control nicht nur über die ax_id referenziert wird. Die ax_id definiert in diesem Fall so etwas wie den Typ des Controls. Um zu einem Controls bestimmte Daten in der Benutzerbibliothek zu verwalten, sollte das Fensterhandle (was allen Funktionen als p_axctrl mitgegeben wird) als eindeutiger Schlüssel zum Zugriff auf diese Daten verwendet werden. Aus diesem Grund ist es auch nicht ratsam, die Handles zu den ActiveX Controls in Variablen der Benutzerbibliothek zu halten. Die Variablen werden dann beim Erzeugen des Controls von dem Fenster der zweiten Verbindung überschrieben.

3.5 Windemo

Die Anwendung windemo wurde überarbeitet und um 4 ActiveX Controls erweitert:

- Hilfetexte werden mit Hilfe des Internetexplorer ActiveX Controls dargestellt.
- Im Modul "Farben" kann eine Farbe über ein ActiveX Control verändert werden.
- Das Modul "SQL" enthält einen Editor mit Syntaxhervorhebung und ein ActiveX Grid zur Darstellung der Ergebnismenge eines SQL Statements.

Um die notwendigen Controls zu registrieren, ist im Unterverzeichnis nsigrp das Programm regax.bat aufzurufen.

Abgesehen von diesen Erweiterungen werden jetzt vier verschiedene Stile mit ausgeliefert:

- XP Blau
- XP Silber
- Windows 7
- Terminal

Diese Stile können als Basis für eigene Stile verwendet werden.

Version 4.1.0

Allgemeine Änderungen

1 Kompatibilität

FIX/Win und FIX der Versionen 4.1.0, 4.0.0 und 3.1.0 sind zueinander kompatibel. Das bedeutet, dass mit FIX/Win 4.1.0 (4.0.0) eine Anwendung gestartet werden kann, die mit FIX 3.1.0 (4.0.0) erstellt wurde und dass mit FIX/Win 3.1.0 (4.0.0) eine Anwendung gestartet werden kann, die mit FIX 4.1.0 (4.0.0) erstellt wurde. Allerdings stehen dann die mit der Version 4.0.0 (4.1.0) eingeführten Features nicht vollständig zur Verfügung. Das bedeutet beispielsweise, dass eine Anwendung, die Menüs und Felder mit Proportionalschrift verwendet, mit FIX/Win 3.1.0 gestartet werden kann. Es wird jedoch für Menüs und Felder weiterhin die nicht proportionale Schrift verwendet. Solange in der Anwendung keine Anpassungen von Menü- und Feldbreiten vorgenommen wurden, funktioniert die Anwendung ohne Probleme.

2 C-Runtime

Aufgrund von immer wieder auftretenden Problemen wird FIX/Win ab der Version 4.1.0 statisch gebunden. Damit ist die Installation eines C-Runtimes nicht mehr notwendig.

3 Neue FIX-Funktionen für eigene Funktionen zum Einlesen von Events

Um eigene Funktionen zum Einlesen von Events zu unterstützen (Ersetzen von S_ReadEvent) wurden folgende Funktionen in FIX implementiert.

```
BOOLEAN is_within_frontend_faccept(void);
```

Diese Funktion liefert TRUE, wenn sich FIX/Win in einer Felderfassung mit Edit Controls befindet und FIX auf das Ergebnis wartet. Dieser Fall ist in einer eigenen Funktion zum Lesen von Events gesondert zu behandeln. FIX wartet in diesem Fall auf das Event K_SFE und ignoriert alle anderen Events. Die eigene Funktion sollte deshalb genauso arbeiten, wie fxReadEvent().

- Setzen des Events, das an die Anwendung ausgeliefert werden soll.
- Rückgabe von K_SFE

Zum Setzen des Events für die Anwendung, ist die Funktion

```
void set_frontend_field_leaving_event(long ev);
```

zu verwenden. Bei der Rückgabe von K_SFE verfährt FIX so, als ob die Felderfassung ohne Änderung des Wertes beendet wurde. Das mit set_frontend_field_leaving_event() wird an die FIX-interne Felderfassung ausgeliefert und wenn es sich dabei um ein Event handelt, das die Felderfassung beendet (also keine feldinterne Editiertaste), dann wird das Event auch an Maskenlogik und die Anwendung ausgeliefert. Der vom Benutzer im Frontend eingegebene Feldwert wird dabei verworfen, da er noch nicht an FIX übertragen wurde.

4 Laden des Latin-2 Fonts

Wenn in lookAndFeel.frc der Eintrag

Font: FIXWIN*

definiert wird, dann werden zur Darstellung von nicht proportionalen Zeichen die beiden Fonts FW-[M|L|H]-L15.FNT und FW-[M|L|H]-L2.FNT geladen. Beim Aufbau der Verbindung wird dananch anhand des Zeichensatzes der Anwendung (FXCHARSET) entschieden, mit welchem der beiden Fonts gearbeitet wird.

5 Konvertierung bei exec_ms_command

Beim Aufruf von exec_ms_command werden für $1 ... $n die Info-Komponenten der Felder eingesetzt. Beim Einsatz einer Konvertierungsfunktion (S_ConvertFieldValue) ist es oft erforderlich, dass die unkonvertierten Werte verwendet werden. Um dies zu erreichen kann ab der Version 4.1.0 die Ressource

```
NoConvertOnExecCmd
```

auf TRUE gesetzt werden. In diesem Fall wird die Konvertierungsfunktion vor dem Erzeugen des Kommandos ausgeschaltet und die Werte werden durch m_wrktoinfo() erneut in die info-Komponenten geschrieben. Nach dem Erzeugen des Kommandos wird die Konvertierungsfunktion wieder aktiviert und die Werte werden erneut in die info-Anteile geschrieben.

6 Zugriff auf Ressourcen

Zum Zugriff auf Ressourcen in fix.rc können die folgenden Funktionen benutzt werden:

fxres_getstring - Ressource als Zeichenkette lesen

Definition

char *fxres_getstring(resname, defval, newmem) char *resname; char *defval; BOOLEAN newmem;

Beschreibung

Ließt den Wert zur Ressource resname als Zeichenkette. Wenn für resname kein Wert definiert wurde, dann wird defval als Wert zurückgeliefert. newmem bestimmt mit TRUE, dass für einen gelesenen Wert Speicher mittels strsave() angelegt wird. Ist newmem == FALSE, dann wird einer der drei internen Puffer benutzt. In diesem Fall muss davon ausgegangen werden, dass der zurückgelieferte Wert beim drittnächsten Aufruf überschrieben wird. Es wird jedoch in jedem Fall ein Puffer verwendet, auch wenn die Ressource nicht definiert wird. In diesem Fall wird defval in den bereitgestellten Puffer kopiert.

fxres_getlong - Ressource als long-Wert lesen

Definition

long fxres_getlong(resname,defval) char* resname; long defval;

Beschreibung

Ließt den Wert zur Ressource resname als long-Wert. Wenn sich der Wert nicht konvertieren lässt, wird eine Fehlermeldung ausgegeben und defval wird zurückgeliefert. Wenn die Ressource nicht definiert ist, dann wird ebenfalls defval zurückgegeben.

fxres_getdouble - Ressource als double-Wert lesen

Definition

double fxres_getdouble(resname,defval) char* resname; double defval;

Beschreibung

Ließt den Wert zur Ressource resname als double-Wert. Wenn sich der Wert nicht konvertieren lässt, wird eine Fehlermeldung ausgegeben und defval wird zurückgeliefert. Wenn die Ressource nicht definiert ist, dann wird ebenfalls defval zurückgegeben.

fxres_getboolean - Ressource als BOOLEAN-Wert lesen

Definition

BOOLEAN fxres_getboolean(resname,defval) char* resname; BOOLEAN defval;

Beschreibung

Ließt den Wert zur Ressource resname als BOOLEAN-Wert. Wenn sich der Wert nicht konvertieren lässt, wird eine Fehlermeldung ausgegeben und defval wird zurückgeliefert. Wenn die Ressource nicht definiert ist, dann wird ebenfalls defval zurückgegeben. Als Werte sind TRUE und FALSE in der Ressourcedatei erlaubt.

7 Zugriff auf lookAndFeel.frc

Fix/Win besitzt eine neue Funktion zum Zugriff auf die Datei lookAndFeel.frc aus der Benutzerbibliothek heraus.

GetLFRes - Ressource aus lookAndFeel.frc lesen

Definition

void* GetLFRes(void* p_callID, _TCHAR* resname)

Makros

FW_GETLFRES, FWFCT_GETLFRES(callback, varname)

Beschreibung

Die Ressource wird nicht unmittelbar aus der Datei lookAndFeel.frc gelesen. Statt dessen wird die Datei beim Laden eines Look&Feels gelesen und im Speicher abgelegt. Der Aufruf von GetLFRes() greift dann auf den Wert im Speicher zu.

Als Parameter resname kann eines der Makros aus fwown.h angegeben werden, die mit LFRES_ beginnen. Für diese Werte ist ein Defaultwert definiert, der zurückgegeben wird, wenn die Ressource nicht definiert ist. FIX/Win verwendet den gleichen Defaultwert, bei nicht definierter Ressource. Je nach Art der Ressource unterscheidet sich der Rückgabewert. Er kann entweder vom Typ long sein oder vom Typ _TCHAR*. Das Ergebnis ist in den entsprechenden Typ zu casten. Die Typen der Rückgabewerte und die Defaultwerte sind in fwown.h zur jeder Ressource hinter dem Makro dokumentiert.

Zusätzlich ist es möglich, selbst definierte Ressourcen in lookAndFeel.frc zu hinterlegen und mittels GetLFRes() auszulesen. Als resname ist der Name der Ressource als _TCHAR* anzugeben. Bei den Ressourcewerten findet keine Umwandlung statt. Das Ergebnis ist immer vom Typ _TCHAR*. Wenn Zahlenwerte oder boolesche Werte gewünscht werden, dann ist eine Umwandlung in den entsprechenden Datentyp in der Benutzerbibliothek vorzunehmen.

Obwohl die Ressource beim Zugriff bereits im Speicher steht, kostet der Aufruf von GetLFRes() relativ viel Performance. Der Grund liegt darin, dass der Schlüssel (resname) eine Zeichenkette ist, die über binäre Suche und Stringvergleich gefunden werden muss. Bei häufig aufgerufenen Funktionen (z.B. FwownDrawPaintArea()) lohnt es sich deshalb beim ersten Zugriff den Wert in eine Variable zu lesen und von da an diese zu verwenden. Beim Wechsel des LookAndFeels (FwownHook(HOOK_CHANGESTYLE)), ist die Variable zu invalidieren, so dass beim nächsten mal wieder die Ressource gelesen wird.

Beispiel

Innerhalb von FwownDrawPaintArea() wird der Name des Fonts und die Größe ermittelt.

```
GetResFct GetLFRes = NULL;
_TCHAR* propFontName = NULL;
long propFontSize = -1;

FWOWN_API _TCHAR* CALLBACK FwownDrawPaintArea(
     CallBackFct CallBack, void* p_callID,
     HDC p_hDCPA, int p_width, int p_height,
     unsigned long p_pa_id, short p_pa_type,
     _TCHAR* p_text, unsigned short p_pa_attr, short p_pa_align,
     long p_longval1, long p_longval2,
     long p_longval3, long p_longval4,
     _TCHAR* p_ms_name, short p_objclass,
     int p_size, int p_style) {

     ...

     if (propFontName == NULL) {
```

```
            /* Font unbekannt -> neu ermitteln */

            FWFCT_GETLFRES(CallBack, GetLFRes);     /* Funktionszeiger belegen */
            propFontName = (_TCHAR*)GetLFRes(p_callID, LFRES_PROPFONT);
            propFontSize = (long)GetLFRes(p_callID, LFRES_PROPFONTSIZE_M);
      }

      ...
}

FWOWN_API _TCHAR* CALLBACK FwownHook(CallBackFct CallBack, void* p_callID,
      int id) {

      if (id == HOOK_CHANGESTYLE ) {
            propFontName = NULL; /* Font als unbekannt definieren */
            propFontSize = -1;
}
```

8 Neue Ressourcen in lookAndFeel.frc

Über die Ressourcen

```
FieldFontYOffset_M
FieldFontYOffset_L
FieldFontYOffset_H
```

kann eine Verschiebung in Y-Richtung für die Schrift von Fieldareas angegeben werden. Diese Änderung ist notwendig, weil bei bestimmten Schriftarten in bestimmten Größen von Windows ein unsichtbarer Rahmen als Abstand innerhalb eines Edit Controls verwendet wird. Der Defaultwert für die ersten beiden ist 0 und für die letzte 1.

Über die Ressourcen

```
PropFontYOffset_M
PropFontYOffset_L
PropFontYOffset_H
```

kann eine Verschiebung für die Schrift von Paintareas definiert werden. Der Defaultwert für alle drei Resourcen ist 0.

Die Ressourcen

```
FieldEditYOffset_M
FieldEditYOffset_L
FieldEditYOffset_H
```

definieren eine Verschiebung des Edit Controls nach unten. Der Defaultwert für alle drei Ressourcen ist 0. Das Festlegen einer Verschiebung ist sinnvoll, wenn eine kleinere Schriftart für ein Feld gewählt wird. Windows richtet den Feldinhalt immer am oberen Rand in einem Edit Control aus. Eine vertikale Zentrierung ist nicht möglich. Durch die Verschiebung des gesamten Edit Controls ist jedoch eine vertikale Zentrierung gegenüber des von FIX/Win gezeichneten Feldrahmens möglich.

9 Erfassung von Datetime- und Interval-Feldern

Datetime- und Interval-Felder werden bei der Verwendung von Fieldareas durch ein Edit Control, dass eine spezielle Eingabelogik verwendet, von FIX/Win erfasst. Der Wert wird komponentenweise erfasst.

Erfassen einer Komponente

Das Navigieren innerhalb einer Komponente oder das Entfernen einzelner Ziffern ist nicht möglich. Eine Komponente muss immer komplett erfasst werden, was bei bis zu 4 Ziffern kein Problem darstellt. Beim Betreten einer Komponente wird sie komplett selektiert. Das Tippen einer Ziffer löscht deshalb alle Ziffern und fügt die eingegebene Ziffer an der letzten Position ein. Die übrigen Stellen werden mit dem Zeichen "_" zur Darstellung einer Leerstelle gefüllt. Das Eingeben von weiteren Ziffern führt dazu, dass der aktuelle Wert nach links geschoben wird und die Ziffer an der letzten Position eingefügt wird. An dieser Position steht auch die Schreibmarke.

Navigieren zwischen Komponenten

Mittels der Pfeiltasten links und rechts kann die aktuelle Komponente gewechselt werden. Die Taste Pos1 setzt die Schreibmarke auf die erste Komponente, die Taste Ende setzt sie auf die letzte. Eine neu betretene Komponente wird dabei immer komplett selektiert.

Wenn die linke Pfeiltaste zusammen mit Shift gedrückt wird, dann wird die aktuelle Komponente selektiert. Beim Drücken der rechten Pfeiltaste zusammen mit Shift wird die nächste Komponente betreten und selektiert. Hier besteht kein Unterschied zum Drücken der Taste ohne Shift.

Eine andere Möglichkeit auf eine andere Komponente zu navigieren, besteht in der Eingabe eines Trennzeichens. Je nach Konfiguration wird zum nächsten Vorkommen des Zeichens oder zur nächsten Komponente gesprungen.

Beim Verlassen einer Komponente werden die fehlenden Ziffern (die durch das Füllzeichen "_" dargestellt werden) ergänzt. Zusätzlich findet eine Plausibilitätsprüfung (z.B. Monat <= 12) statt. Findet diese einen Fehler, dann wird die Komponente nicht verlassen.

Beim Verlassen des Feldes werden in allen Komponenten die Füllzeichen ersetzt. Das Prüfen des kompletten Feldwertes wird FIX überlassen.

Konfiguration

Durch Einträge in config/fixwin.frc kann das Verhalten bei der Eingabe von Datetime- und Interval-Feldern konfiguriert werden. Bei allen Werten handelt es sich um Boolean Werte, deren Default FALSE ist.

DtAutoNext

Wird dieser Wert auf TRUE gesetzt, dann wird die nächste Komponente automatisch betreten, wenn die aktuelle keine Füllzeichen mehr enthält. Die Eingabe von "10122010" führt somit beispielsweise zu dem Datum "10.12.2010". Wenn eine Komponente automatisch betreten wurde, dann wird die Eingabe des nächsten Zeichens ignoriert, wenn es sich nicht um eine Ziffer handelt. Die Eingabe von "10.12.2010" führt deshalb ebenfalls zu dem Wert "10.12.2010".

DtSearchByChar

Der Wechsel auf die nächste Komponente kann normalerweise durch die Eingabe eines beliebigen Zeichens erfolgen, dass keine Ziffer ist. Wenn dieser Wert auf TRUE gesetzt wird, dann wird nach dem eingegeben

Zeichen gesucht und die Scheibmarke in die Komponente hinter diesem Zeichen positioniert.

DtCheckComplete

Bei der Eingabe einer Ziffer werden alle anderen Ziffern um eine Stelle nach links verschoben. Wenn auf der ersten Stelle eine Ziffer steht, dann wird diese entfernt. Wird diese Ressource auf TRUE gesetzt, dann ist die Eingabe einer weiteren Ziffer nur möglich, wenn die Komponente noch Platzhalter ("_") enthält. Damit können keine bestehenden Ziffern gelöscht werden. Um eine bestehende Komponente unmittelbar nach der Eingabe zu überschreiben, ist die Komponente (durch Shift Pfeiltaste links) zu selektieren.

DtGuessYear

Das Setzen dieser Ressource auf TRUE bewirkt das Berechnen des Jahres, wenn bei einer vierstelligen Jahreszahl weniger als vier Stellen angegeben werden. Bei der Angabe von bis zu zwei Stellen wird 20 in die ersten beiden Stellen geschrieben, wenn der Wert unter 40 liegt. Ansonsten wird 19 in die ersten beiden Stellen geschrieben. Bei der Angabe von drei Stellen wird die erste Stelle mit 1 belegt.

10 Promptzeile in Proportionalschrift

Durch das Setzen der Ressource

```
PromptAsPaintarea
```

auf TRUE, wird zur Darstellung der Promptzeile eine Paintarea verwendet. Damit verwendet auch die Promptzeile Proportionalschrift.

Zur Darstellung wird der neue Paintarea-Typ PA_PROMPT verwendet. Wenn Paintareas über die Funktion FwownDrawPaintArea() in der Benutzerbibliothek von FIX/Win gezeichnet werden, dann ist diese Funktion so anzupassen, dass sie entweder "" für diesen Typ zurückliefert oder die Paintarea selbst zeichnet. Die Rückgabe von "" bewirkt, dass FIX/Win die Paintarea zeichnet. Dazu wird die übergebene Zeichenkette analysiert. Die Zeichen mit den Codes 1-5 (=^A-^E) schalten das für die Ausgabe verwendete Videoattribut um. Die beiden Codes 16, 20 markieren den Anfang eines Tastenlabels. Die Codes 25, 21 markieren das Ende des Labels. FIX/Win interpretiert den Text zwischen diesen beiden Marken als Tastenlabel und stellt die entsprechende Grafik dar. Die pixelgenauen Positionen der Tastenlabels werden von FIX/Win gespeichert und zum Abarbeiten von Klick-Events genutzt. Dadurch sind die Tastenlabels wie bisher anklickbar, obwohl sie nicht mehr exakt über Bildschirmzellen liegen.

Als Schriftart wird die durch PropFont, PropFontSize, PropFontQuality und PropFontWeight definierte Schriftart verwendet. Die Farben für die Videoattributte werden aus den Einträgen PA_TEXT_NORMAL, PA_TEXT_INVERS, PA_TEXT_LOW, PA_TEXT_UNDERLINE und PA_TEXT_BLINK der Farbzuordnungstabelle gelesen.

Zusätzlich wurden für die Promptzeile die Ressourcen:

```
PromptWindowXPos
```

und

```
PromptWindowYPosOffset
```

eingeführt. Diese verhalten sich analog zu den Positionierungs-Resoucen der Meldungszeile.

11 Darstellung von Tasten in Hilfetexten

In Hilfetexten können Tastenbeschriftungen in der Form

```
Event
```

verwendet werden. Diese Beschriftungen beziehen sich nicht auf das aktive Objekt - also den Hilfetext. Stattdessen wird das Objekt, das vor dem Aufblenden des Hilfetextes aktiv war, zur Ermittlung der Beschriftung herangezogen. Wenn beispielsweise im Hilfetext das Event EN mit der Taste "Ende" belegt ist und in der Maske mit der Taste "Escape", dann wird im Hilfetext die Taste "Escape" dargestellt.

Es gibt aber auch Fälle, in denen die Tastenbeschriftungen sich auf den Hilfetexte beziehen müssen (Beispielsweise: "Verlassen Sie die Hilfe mit <Ende>"). In diesen Fällen kann die Tastenbeschriftung in der Form

```
Event
```

angegeben werden.

12 Ausrichtung von Inhalten einer Choice-Spalte mit Hilfe spezieller Separatoren

Diese Funktionalität dient zur Ausrichtung von Werten, einzelner Spalten der Choice-Treffermenge mit Hilfe der Displayfunktion innerhalb einer Choice-Spalte. Durch die Verwendung von Proportionalschrift, ist das bisherige Verfahren zur Ausrichtung von Werten Mithilfe von vor- oder nachgestellen Leerzeichen nicht mehr verwendbar. Die Breite in einer solchen Werte-Spalte wird weiterhin durch die Anzahl Zeichen der Spalte festgelegt. Um die Spalten abzugrenzen, kann man jetzt spezielle Separatoren zwischen den Spalten einfügen, die damit eine Breitenberechnung der Spalten ermöglichen und für die Darstellung in FIX/Win das Alignement festlegen. Dazu wurden 2 Separatoren definiert: ':<:' für Linksbündig und ':>:' für Rechtsbündig. Der entsprechende Separator muss jeweils vor der zugehörigen Spalte stehen. Für die FIX-interne Displayfunktion ch_display_columns ist dieses Verfahren über die FIX-Ressource:

```
FormattedChoiceDisplay
```

durch Setzen auf TRUE einschaltbar. Standard-Wert ist FALSE (ausgeschaltet).

13 Neue Events

Die Events u0-u9 sind neu. Sinn und Zweck dieser Events ist die Möglichkeit in einer Anwendung benutzerdefinierte Events zu verwenden. Sie dienen somit als teilweiser Ersatz für die unbenannten Events. Das Problem bei unbenannten Events ist, dass sie zwar unter FIX/Win einer Windows-Taste zugeordnet werden können, jedoch nicht in der Promptzeile oder in einem Hilfetext angegeben werden können. Damit bietet sich keine Möglichkeit, ein Tastenlabel für die Taste darzustellen.

Prinzipiell könnten für diesen Einsatz auch die Events h0-h9 verwendet werden. Da diese jedoch oft schon von einer Anwendung verwendet werden und auch schon in der Standard Tastenbelegung von FIX/Win definiert werden, wurden deshalb die neuen Events u0-u9 eingeführt.

14 Verschiedene Tastenbelegungen

Durch Angabe des Schalters

```
/keytab <name>
```

kann der Name der Tatenbelegungsdatei definiert werden. Somit ist es möglich, verschiedene Tastenbelegungen innerhalb eines Gruppenverzeichnisses zu verwenden. Der Wert kann auch als Ressource in die Datei config/fixwin.frc geschrieben werden.

15 Neue Objektklasse für Tastenbelegung

Bei der Tastenbelegung kann jetzt die Objektklasse

```
SELOFIELD
```

angegeben werden. Diese Klasse ist aktiv, wenn ein Feld zur Startwerterfassung für Selos betreten wird. Die Tastenbelegung ist damit für diesen Fall getrennt definierbar.

16 Belegung von "Alt"

In der Tastenbelegung sind jetzt drei zusätzliche Events für eine Taste anzugeben, um die Alt-Taste auszunutzen. Damit ergibt sich folgender Aufbau einer Zeile:

```
<Tastenname> <Normal> <Shift> <Strg> <Shift+Strg> <Alt> <Shift+Alt> <Strg+Alt>
```

Als neue Modifier für die Tastenlabels sind

A - für Alt Z - für Shift+Alt Y - für Strg+Alt

in den Namen der Bitmaps zu verwenden.

17 Verwerfen von Events

Durch Aufruf der Funktion

```
clear_undc_buffer();
```

können alle Events verworfen werden, die mittel undcgetch() in den Tastaturpuffer gestellt wurden.

Durch das Setzen der Variablen

```
S_clear_undc_buffer
```

kann gesteuert werden, wann FIX die Funktion aufruft. Dazu sind die in keys.h definierten Werte zusammen zu addieren.

- CLEAR_UNDC_ON_FLASH - Aufruf bei der Ausgabe von flash()
- CLEAR_UNDC_ON_MSG - Aufruf bei der Ausgabe von msg()
- CLEAR_UNDC_ON_RTMSG - Aufruf bei der Ausgabe von rt_msg()
- CLEAR_UNDC_ON_JNMSG - Aufruf bei der Ausgabe von jn_msg()
- CLEAR_UNDC_ON_ERRMSG - Aufruf bei der Ausgabe von err_msg()
- CLEAR_UNDC_ON_ALLMSG - Aufruf in allen obigen Fällen

Mit der Formulierung "Aufruf bei der Ausgabe von xxx()" ist gemeint, dass clear_undc_buffer() nicht immer aufgerufen wird, wenn die Funktion xxx() aufgerufen wird, sondern nur, wenn die Funktion xxx() eine Ausgabe macht. Wenn in S_ShowMessage eine Funktion definiert wird, die die Ausgabe verhindert, dann wird auch nicht clear_undc_buffer() aufgerufen. Ist das Leeren des Puffers trotzdem gewünscht, dann ist clear_undc_buffer() in der eigenen Funktion aufzurufen.

Version 4.0.0

Allgemeine Änderungen

1 Kompatibilität

FIX/Win und FIX der Versionen 4.0.0 und 3.1.0 sind zueinander kompatibel. Das bedeutet, dass mit FIX/Win 4.0.0 eine Anwendung gestartet werden kann, die mit FIX 3.1.0 erstellt wurde und dass mit FIX/Win 3.1.0 eine Anwendung gestartet werden kann, die mit FIX 4.0.0 erstellt wurde. Allerdings stehen dann die mit der Version 4.0.0 eingeführten Features nicht vollständig zur Verfügung. Das bedeutet beispielsweise, dass eine Anwendung, die Menüs und Felder mit Proportionalschrift verwendet, mit FIX/Win 3.1.0 gestartet werden kann. Es wird jedoch für Menüs und Felder weiterhin die nicht proportionale Schrift verwendet. Solange in der Anwendung keine Anpassungen von Menü- und Feldbreiten vorgenommen wurden, funktioniert die Anwendung ohne Probleme.

Zur Erstellung von FIX/Win wird ab der Version 4.0.0 Visual Studio 2008 verwendet. Die entsprechende C-Laufzeitumgebung kann durch Ausführen von

```
cruntime/vcredist_x86.exe
```

installiert werden. Da sich die Umgebung nicht unter Windows 2000 installieren lässt, wird diese Plattform ab der Version FIX/Win 4.0.0 nicht mehr unterstützt.

2 Unterstützung von Datenbanken

FIX ab der Version 4.0.0 unterstützt nicht mehr Informix vor 4.1 und Oracle vor 8.0

3 Unterstützung von Zeichensätzen

Ab FIX 4.0.0 wird der Zeichensatz GERMAN7 nicht mehr unterstützt.

Bei der Verwendung des Zeichensatzes ISO-2 wird die benötigte Schriftart nicht mehr automatisch von FIX/Win geladen. Wenn dieser Zeichensatz verwendet wird, dann ist die Ressource "Font" in der entsprechenden lookAndFeel.frc entweder auf FIXWIN-L2 oder auf eine Systemschriftart, die Unicode unterstützt, zu setzen.

4 Unterstützung von Plattformen

Die Plattformen

- SunOS 4.x
- Sinix / Reliant Unix

werden nicht mehr unterstützt. Weiterhin werden alle Plattformen, die nicht X/OPEN-konform sind, nicht mehr unterstützt.

5 Neue Hooks

```
define HOOK_BEFORECHANGESIZE       13
define HOOK_BEFORECHANGESTYLE      14
```

werden aufgerufen, bevor sich ein Look&Feel ändert.

6 Neue Windows-Ressourcen

Folgende Ressourcen sind neu und müssen in eigenen Übersetzungen der Windows-Ressourcen für FIX/Win ergänzt werden:

ResFixwinCore.dll:

```
ERR_PR160
ERR_OB155
IDD_CONNECTBOX_NOEDIT
```

ResFixwin.dll:

```
MSG_FX900
```

7 Neue Ressourcen

Abgesehen von den im folgenden Text beschriebenen Ressourcen, sind diese Ressourcen neu in 4.0.0:

FontQuality (lookAndFeel.frc)

Diese Ressource bestimmt die Qualität der nicht proportionalen Schriftart. Mögliche Werte: siehe FieldFontQuality.

Die Einstellung wird nur ausgewertet, wenn FONT nicht auf FIXWIN oder FIXWIN-L2 gesetzt wird.

FontWeight (lookAndFeel.frc)

Diese Ressource bestimmt die Dicke der nicht proportionalen Schriftart. Mögliche Werte: siehe FieldFontWeight.

Die Einstellung wird nur ausgewertet, wenn FONT nicht auf FIXWIN oder FIXWIN-L2 gesetzt wird.

FontBoldWeight (lookAndFeel.frc)

Diese Ressource bestimmt die Dicke der nicht proportionalen Schriftart mit dem Attribut BOLD. Mögliche Werte: siehe FieldFontBoldWeight.

Die Einstellung wird nur ausgewertet, wenn FONT nicht auf FIXWIN oder FIXWIN-L2 gesetzt wird.

PropFontQuality (lookAndFeel.frc)

Diese Ressource bestimmt die Qualität der Proportionalschriftart. Mögliche Werte: siehe FieldFontQuality.

PropFontWeight (lookAndFeel.frc)

Diese Ressource bestimmt die Dicke der Proportionalschriftart. Mögliche Werte: siehe FieldFontWeight.

PropFontBoldWeight (lookAndFeel.frc)

Diese Ressource bestimmt die Dicke der Proportionalschriftart mit dem Attribut BOLD. Mögliche Werte: siehe FieldFontBoldWeight.

8 Änderung der Farbzuordnungstabelle

Der Aufbau der Farbzuordnungstabelle hat sich geändert. Grund für diese Änderung war die Anforderung für einige Elemente mehr als eine Linie zur Umrandung definieren zu können. Für Felder in Darstellung "Normal" und in "Invers" war dies bereits in der Version 3.1.0 möglich. Da eine Ausweitung auf andere Attribute eine erhebliche Erhöhung der Elemente der Tabelle bedeutet hätte, wurde das Format grundlegend geändert. Für jedes Element können jetzt bis zu 4 Linienfarben definiert werden. Die Elemente, die früher diese Linienfarben definiert haben (..._LINE1, ..._LINE2), fallen damit weg:

```
FIELD_LINE1
FIELD_LINE2
FIELD_LINE1_INVERS
FIELD_LINE2_INVERS
TABLE_LINE1
TABLE_LINE2
SPECIAL_LINE
MENU_LINE
CHOICE_LINE
SELO_LINE1
SELO_LINE2
LABEL_LINE
HEAD_LINE
PA_HEADER_LINE1
PA_HEADER_LINE2
PA_BUTTON_LINE1
PA_BUTTON_LINE2
PA_VARBUTTON_LINE1
PA_VARBUTTON_LINE2
```

Zusätzlich wurde die Beschreibung um ein Schriftattribut erweitert. Bisher wurde bei den Schriften nur zwischen "Normal" und "Underline" unterschieden. Mit dem Einsatz von Systemschriftarten stehen jedoch wesentlich mehr Attribute zur Verfügung. Diese Attribute können jetzt den von FIX vergebenen Attributen zugeordnet werden.

Jeder Zeile der Farbzuordnungstabelle hat folgendes Format:

```
<Element> <Textfarbe> <Hintergrund> <Attribut> <Line1> ... <Line4>
```

<Element> <Textfarbe> und <Hintergrund> entsprechen den bisherigen Einträgen der Farbzuordnungstabelle.

Für den neuen Eintrag <Attribut>, der das Schriftattribut definiert, können folgende Werte verwendet werden:

```
NORMAL
UNDERLINE
```

```
ITALIC
BOLD
```

Die von FIX/Win für einige Elemente implizit getroffene Zuordnung von UNDERLINE entfällt. Für diese Elemente ist das Attribut UNDERLINE explizit in der Farbzuordnungstabelle aufzuführen, wenn es verwendet werden soll.

Die Werte <Line1>, <Line2>, <Line3>, <Line4> definieren vier Farben, die zur Darstellung von Linien verwendet werden. Als Werte für diese Elemente können die gleichen Bezeichner verwendet werden, die auch für Text- und Hintergrundfarbe verwendet werden. Ob und für welche Linien die Farben verwendet werden, hängt vom jeweiligen Element ab. Es gibt auch Elemente, die gar keine Linienfarben oder nur die ersten beiden verwenden. Deshalb können diese Farben auch weggelassen werden.

So können beispielsweise für das Element TEXT_NORMAL die Attribute <Line1> ... <Line4> weggelassen werden, da zur Darstellung dieses Elements keine Linien gezeichnet werden. Bei dem Element FIELD_NORMAL sollten diese Werte jedoch unbedingt angegeben werden, da ansonsten die Umrandung des Feldes aus roten und grünen Linien (hier werden auffällige Defaultfarben verwendet, um undefinierte Farben sofort zu erkennen) besteht.

Wenn Farben für Linien in einer Definition weggelassen werden, dann sind sie von hinten nach vorne wegzulassen. Die Angabe von <Attribut> kann ebenfalls weggelassen werden, wenn keine Farbe für eine Linie definiert wird. In diesem Fall wird NORMAL verwendet.

Zeichnen von Linien

Folgende Aussage aus dem Handbuch gilt mit dieser Änderung nicht mehr:

```
Bei Elementen, die Linien besitzen, wird versucht diese dreidimensional zu
zeichnen, indem die obere Linie in der Vordergrundfarbe des Linienelements
und die untere Linie in der Hintergrundfarbe des Linienelements gezeichnet
wird. Wenn dies nicht gewünscht wird, dann besteht die Möglichkeit, die
beiden Begrenzungslinien in der gleichen Farbe zeichnen zu lassen, wie das
Element selbst. Dazu sind die beiden Farbangaben der Linie auf die gleiche
Farbe zu setzen, wie die Hintergrundfarbe des Elements beim Attribut
"normal".
```

Statt dessen gilt folgendes:

```
Wenn die Angabe für die Farbe aller Linien der Hintergrundfarbe des
Elementes entspricht, dann werden nicht nur die Linien in der
Hintergrundfarbe gezeichnet, sondern der komplette vom Element belegte
Platz.

Dieser Platz wird ansonsten in der durch das Element BACKGROUND definierten
Farbe gezeichnet.

Da durch die vier Linien oft kein weiterer Platz mehr zur Verfügung steht,
ist eine Auswirkung dieser Regel meistens nur in der größten Bildschirmgröße
(Huge) von FIX/Win zu sehen.

Es gibt jedoch auch Elemente, die nur zwei Linien verwenden. In diesem Fall
sind trotzdem alle vier Linien zu definieren, damit die Regel zum Einsatz
kommt.
```

Im folgenden wird beschrieben, wie die Linien in den einzelnen Elementen gezeichnet werden. Die Angabe "Standardverfahren mit 4 Linien" bezieht sich auf folgendes Verfahren:

```
<Line1> wird für die obere äußere Linie verwendet.
<Line2> wird für die obere innere Linie verwendet.
<Line3> wird für die untere innere Linie verwendet.
<Line4> wird für die untere äußere Linie verwendet.
```

Damit werden die Linien von oben nach unten in der angegeben Reihenfolge gezeichnet. Wenn eine äußere Linie nicht gezeichnet werden soll, dann kann diese nicht einfach in der Definition weggelassen werden (dann

würde als Farbe rot oder grün verwendet). Statt dessen ist die allgemeine Hintergrundfarbe zu verwenden (BACKGROUND). Wenn eine innere Linie nicht gezeichnet werden soll, dann ist sie mit dem gleichen Wert wie <Hintergrund> zu belegen. (Tatsächlich werden die Linien natürlich gezeichnet. Durch die Farbwahl entsprechen sie jedoch dem Hintergrund des Elements.)

Neben diesem Verfahren gibt es ein zweites Standardverfahren, dass nur zwei Linien verwendet:

```
<Line1> wird für die obere Linie verwendet.
<Line2> wird für die untere Linie verwendet.
```

Dieses Verfahren wird im folgenden Text als "Standardverfahren mit 2 Linien" bezeichnet.

Bei der Angabe der folgenden Elemente wurde der Attributanteil des Elementnamens durch * ersetzt (z.B. FIELD_* satt FIELD_NORMAL), weil das jeweilige Verfahren nicht vom Attribut abhängig ist. Das Attribut bestimmt lediglich den Eintrag, der die Farben für die Linien definiert.

*FIELD_**

Die Linien dieser Elemente werden nach dem Standardverfahren mit 4 Linien gezeichnet.

*TABLE_**

Die Linien dieser Elemente werden nach dem Standardverfahren mit 4 Linien gezeichnet.

*SPECIAL_**

Die Linien dieser Elemente werden nach dem Standardverfahren mit 4 Linien gezeichnet.

*MENU_**

Die Linien dieser Elemente werden nach dem Standardverfahren mit 2 Linien gezeichnet.

*CHOICE_**

Die Linien dieser Elemente werden nach dem Standardverfahren mit 2 Linien gezeichnet.

*SELO_**

Wenn das Selo keine Tabellendarstellung verwendet, wird das Standardverfahren mit 2 Linien verwendet. Ansonsten wird das Standardverfahren mit 4 Linien verwendet.

*LABEL_**

Die Linien dieser Elemente werden nach dem Standardverfahren mit 2 Linien gezeichnet.

*HEAD_**

Die Linien dieser Elemente werden nach dem Standardverfahren mit 2 Linien gezeichnet.

*TEXT_**

Für diese Elemente werden keine Linien gezeichnet.

*PA_TEXT_**

Für diese Elemente werden keine Linien gezeichnet.

*PA_HEADER_**

<Line1> wird für die obere und untere äußere Linie verwendet. <Line2> wird für die obere innere Linie und für die linke innere Linie verwendet. <Line3> wird für die obere innere und die linke äußere Linie verwendet.

Bei allen Tableheadern bis auf den letzten werden rechts keine Linien gezeichnet. Bei dem letzten Tableheader wird für die rechte innere Linie <Line3> verwendet und für die äußere <Line2>.

*PA_BUTTON_**

Ein Button wird von 4 Linien umrahmt. Für die linke Linie wird <Line1> verwendet. Für die obere Linie wird <Line2> verwendet. Für die rechte Linie wird <Line3> verwendet und für die untere Linie <Line4>.

*PA_MENUITEM_**

Diese Elemente verwenden keine Linien.

Auslesen der Farben in der Benutzerbibliothek

Mit dieser Änderung hat sich die Funktion GetColor() geändert. Statt des Parameters foreground bekommt sie jetzt den Parameter colpart. Sie hat somit folgenden Prototyp:

```
COLORREF (* GetColorFct)(void* p_callID, ColtabIdx idx, ColtabPart colpart);
```

Die neue Enumeration ColtabPart definiert die möglichen Farben eines Elements: TEXT, BACK, LINE1, LINE2, LINE3, LINE4. Damit kann auf alle in der Farbzuordnungstabelle definierten Farben zugegriffen werden.

9 Steuerung des Verbindungsaufbaus

Folgende Angaben können zur Steuerung des Verbindungsaufbaus auf der Kommandozeile angegeben werden oder als globale Einstellung in config/fixwin.frc angegeben werden. Bei der Angabe als globale Einstellung ist der Wert true oder false (= Default) zu vergeben. Die Angabe auf der Kommandozeile entspricht dem Wert true und überschreibt die globale Einstellung in config/fixwin.frc. Der Angabe ist ein / voranzustellen.

noeditlogin

Wenn diese Option auf true gesetzt wird, dann ist das Verändern von Benutzername und Passwort nicht mehr möglich. Die Felder werden in der Dialogbox zum Verbindungsaufbau ausgeblendet.

Die Werte für Benutzer und Passwort müssen deshalb entweder auf der Kommandozeile definiert werden oder über die folgende Option bestimmt werden.

systemlogin

Wenn diese Option gesetzt ist, dann wird als Benutzername der aktuelle am System angemeldete Benutzername verwendet und als Passwort die Zeichenkette

```
Single\nSign\tOn
```

Über die Kommandozeile definierte oder in der Dialogbox eingegebene Benutzernamen und Passwörter werden ignoriert. Eine Kombination mit der Option noeditlogin ist deshalb sehr sinnvoll.

Anhand des Passwortes kann der rexec-Daemon oder fxsrv erkennen, dass der Prozess unter einem bestimmten Konto gestartet werden soll. Dazu ist eine Anpassung über die entsprechenden Schnittstellen notwendig, die zu dem Benutzernamen, der von FIX/Win übergeben wird, einen Benutzernamen mit dazugehörigem Passwort ermittelt, unter dem die FIX-Anwendung gestartet wird.

Damit nicht jeder, der einen Benutzernamen kennt, ein FIX-Programm unter diesem Konto starten kann, besitzt das Passwort verschiedene Sicherheiten:

- Es ist dem Benutzer nicht bekannt.
- Es kann - wegen Tab und Return - nicht in einer Dialogbox eingegeben werden - auch nicht über Copy&Paste.

- Als zusätzliche Sicherheit verweigern alle zukünftigen Versionen von FIX/Win das Senden dieses Passwortes, wenn es in der Dialogbox eingegeben wurde. (Meldung MSG_FX900: Passwort enthält unzulässige Zeichen !)

Trotzdem entstehen durch die Verwendung dieses Verfahrens Sicherheitslücken, die durch Personen mit entsprechenden Kenntnissen leicht ausgenutzt werden können:

- Das Passwort wird ohne Verschlüsselung übertragen und ist im Binary fest eingebrannt.
- Jeder der das Passwort kennt, kann unter jedem zulässigen Benutzernamen Prozesse über einen rexec-Client (der die Eingabe des Passwortes erlaubt) starten.

10 Verhalten bei /auto

Das Verhalten bei der Angabe des Schalters /auto wurde verbessert. Zum einen wurde ein Fehler behoben, der dazu geführt hat, dass bei der Angabe von /auto und /single FIX/Win beendet wurde, obwohl eine Messagebox einen Fehler anzeigt. FIX/Win 4.0.0 beendet sich in diesem Fall erst, wenn die Messagebox mit OK bestätigt wurde.

Zusätzlich kann über die Ressource

autoexitwithmsg

in der Datei config/fixwin.frc global eingestellt werden, wie sich FIX/Win verhalten soll, wenn der Schalter /auto angegeben wurde und Meldungen im Meldungsfenster ausgegeben wurden. Wenn die Ressource den Wert true besitzt - das ist der Default - dann wird FIX/Win beim automatischen Start auch automatisch beendet, obwohl Meldungen im Meldungsfenster ausgegeben wurden. Wenn die Ressource den Wert false besitzt, dann wird FIX/Win beim Beenden der Verbindung nur dann automatisch beendet, wenn seit dem Start keine Meldungen ausgegeben wurden. Meldungen, die aus dem Meldungsfenster gelöscht wurden, gelten in diesem Fall als nicht ausgegeben.

11 Laden von Bitmaps für Tastenlabels

Bitmaps für Tastenlabels werden ab Version 4.0.0 auch dann nachgeladen, wenn der Menüpunkt Entwickler/Nachladen/Tastenbelegung ausgelöst wird.

12 Zeichnen von Tabellenüberschriften

Beim Zeichnen von Paintareas vom Typ PA_TABLEHEADER beginnt der Text um eine Zellenbreite nach rechts verschoben. Er ist damit mit dem Text im darunter stehenden Feld bündig. Diese Änderung betrifft nur die von FIX/Win dargestellten Tabellenüberschriften. In der Benutzerbibliothek gezeichnete Tabellenüberschriften sind nicht davon betroffen.

13 Feldlinks

Links auf Felder in der selben Maske funktionieren jetzt. Allerdings muss sich das Link-Feld hinter dem Root-Feld befinden.

Verwendung von Proportionalschrift

1 Fieldareas

Zur Darstellung von Proportionalschrift in Feldern, Selospalten und Choicespalten werden Fieldareas verwendet.

Fieldareas arbeiten ähnlich wie Paintareas. Folgende Unterschiede bestehen:

- Eine Fieldarea beinhaltet nur den inneren Teil des Feldes, der vom Text belegt wird. Die Feldbegrenzer und die oberen, sowie die unteren Trennlinien gehören nicht dazu. Das ist genau der Bereich, den ein Edit Control zur Erfassung des Feldwertes belegt.
- Die Unterscheidung von Text und Token ist nicht notwendig.
- Eine Fieldarea kann mehrere Texte und mehrere Tooltiptexte besitzen. Die unterschiedlichen Texte werden zur Darstellung von Tabellen verwendet.
- Da sich der Text einer Fieldarea häufig ändert, existieren zum Aktualisieren spezielle Funktionen, die nur den Text zum Frontend übertragen und die anderen Parameter außer Acht lassen.

1.1 Verwendung von Fieldareas

Durch das Setzen der Ressource

```
UseFieldAreas: TRUE
```

werden Fieldareas zur Darstellung von Feldinhalten, Selospalten und Choices verwendet, wenn die Anwendung mit FIX/Win bedient wird. Davon ausgenommen werden Felder, die den Datentyp FXGRAPHICS.

1.2 Tooltips auf Fieldareas

Wenn für eine Fieldarea kein Tooltip definiert wurde und bei der Darstellung auf dem Frontend der Text abgeschnitten wurde, dann wird als Tooltip der Text der Fieldarea verwendet.

Zur Definition eines Tooltips für eine Fieldarea ist eine Funktion in der Anwendung zu implementieren, die den Text festlegt. Dabei kann für Felder, Selospalten und Choicespalten jeweils eine eigene Funktion verwendet werden. Die Adresse der Funktion(en) ist in den globalen Variablen

```
void (*S_SetFieldTooltipHook)(field *f);        /* fuer Felder */
void (*S_SetSeloTooltipHook)(field *f);         /* fuer Selospalten */
void (*S_SetChoiceTooltipHook)(choice_item *f); /* fuer Choicespalten */
```

zu hinterlegen.

Die Funktion(en) wird dann aufgerufen, wenn die Fieldarea neu gezeichnet wird. Durch Aufruf der Funktion fset_tooltiptext() kann ein Tooltip zu einer Fieldarea gesetzt werden. Als Kriterium zur Bestimmung des Tooltips kann das als Parameter f übergebene Feld verwendet werden. Bei Choicespalten handelt es sich hierbei allerdings um einen Zeiger auf eine Struktur des Typs choice_item. (Dieser Zeiger kann auch fset_tooltiptext() mitgegeben werden). Da Selos keine Felder besitzen, werden zur Verwendung von Fieldareas intern Pseudofelder erzeugt, bei denen nicht alle Komponenten belegt sind. Deshalb darf bei Selospalten nur auf folgende Komponenten des Parameters f zugegriffen werden:

```
f->z
f->s
f->ob_parent
f->feldname
f->displen
f->infolen
f->info
f->styp
f->ob_props
f->tooltiptext
```

1.3 Konfiguration des Fieldarea-Caches

Fieldareas arbeiten - genau wie Paintareas - mit einem Cache, um den Netzwerkverkehr zwischen FIX und FIX/Win zu reduzieren.

Fieldareas werden unmittelbar vor dem Anzeigen (map) eines Objektes angelegt und an FIX/Win übertragen. Beim Entfernen (unmap) wird der Eintrag als frei markiert. Bei nochmaligem Anzeigen wird versucht, den eventuell noch vorhandenen Eintrag zu nutzen. Beim Neubelegen eines Eintrags werden zuerst die noch nie belegten Einträge verwendet. Erst wenn von diesen keine mehr vorhanden sind, wird einer der als frei markierten Einträge verwendet. Dazu wird ein Zähler mitgeführt, der bei jeder Benutzung erhöht wird. Der am wenigsten benutzte und noch freie Eintrag wird verwendet.

Zur Wiederverwendung enthält das Feld eine Referenz auf die fieldarea. Dabei wird geprüft, ob die Fieldarea noch zu dem entsprechenden Feld gehört und nicht von einem anderen Feld verwendet wurde.

Das Freigeben einer Maske führt zum Freigeben der Felder. Beim erneuten Laden und Anzeigen der Maske können die im Cache vorhandenen Werte nicht verwendet werden, da es sich um eine neue Maske mit neuen Felder aus Sicht der Fieldarea-Verwaltung handelt.

Der Cache kann über zwei globale Variablen konfiguriert werden.

```
int S_fa_max_tabs
```

definiert die Anzahl Einträge im Cache, wobei für jedes FIX-Objekt ein Eintrag belegt wird. Je größer S_fa_max_tabs ist, desto mehr Speicher wird verwendet und desto größer ist die Wahrscheinlichkeit, dass ein bereits vorhandener Eintrag wieder verwendet werden kann. Die Mindestgrenze für S_fa_max_tabs ergibt sich aus der maximalen Anzahl der gleichzeitig angezeigten Objekte einer Anwendung. Der Default für S_fa_max_tabs ist 20.

Die maximale Anzahl Felder pro Objekt kann über

```
int S_fa_tab_size
```

eingestellt werden. Diese Anzahl wird für jedes Objekt allokiert. Wenn ein Objekt mehr Felder besitzt, dann wird ein Vielfaches von S_fa_tab_size für dieses Objekt allokiert. Der Default für S_fa_tab_size ist 20.

1.4 Konfiguration der Schriftart

Die Konfiguration der Schriftart erfolgt über Einträge in der Datei lookAndFeel.frc.

FieldFont

legt die Schriftart fest, die für Fieldareas und für die Eingabe über Edit Controls verwendet wird. Der Default für diese Ressource hängt vom Windows-Gebietsschema ab. In einer japanischen Umgebung wird die Schriftart MS PGothic verwendet. In allen anderen Umgebungen wird die Schriftart Arial verwendet.

FieldFontSize_M, FieldFontSize_H, FieldFontSize_L

Diese Ressourcen legen die Größen für die proportionalen Schriftarten in den Bildgrößen Medium (_M), Large (_L) und Huge (_H) fest. Als Default werden die Werte 10, 16, 22 verwendet, wenn der entsprechende Wert nicht als Ressource definiert wird.

FieldFontQuality

Diese Ressource bestimmt die Qualität der Schriftart. Sie muss eine der Konstanten aus dem Windows-API enthalten:

```
DEFAULT_QUALITY                0
DRAFT_QUALITY                  1
PROOF_QUALITY                  2
NONANTIALIASED_QUALITY         3
ANTIALIASED_QUALITY            4
CLEARTYPE_QUALITY              5
CLEARTYPE_NATURAL_QUALITY      6
```

Die Beschreibung dieser Konstanten ist in der Microsoft Dokumentation zur Funktion CreateFont() nachzulesen.

Wenn die Ressource nicht definiert wird, dann wird der Wert 0 verwendet.

FieldFontWeight, FFieldFontBoldWeight

Diese Ressourcen bestimmt die Dicke der Schriftart. Sie muss eine der Konstanten aus dem Windows-API enthalten:

```
FW_DONTCARE      0
FW_THIN        100
FW_EXTRALIGHT  200
FW_LIGHT       300
FW_NORMAL      400
FW_MEDIUM      500
FW_SEMIBOLD    600
FW_BOLD        700
FW_EXTRABOLD   800
FW_BLACK       900
```

Die Beschreibung dieser Konstanten ist in der Microsoft Dokumentation zur Funktion CreateFont() nachzulesen.

FieldFontWeight definiert die Dicke der normalen Schrift. FieldFontBoldWeight definiert die Dicke der Schrift mit dem Attribut BOLD. Wenn ein Wert nicht angegeben wird, dann wird für FieldFontWeight der Wert 0 verwendet und für FieldFontBoldWeight der Wert FW_BOLD.

Nicht Proportionale Schrift

Wenn die (neue) Feldeigenschaft FIXEDFONT gesetzt wird, dann wird die oben beschriebene Schriftart nicht verwendet. Statt dessen wird die über die Einträge FONT und FONTSIZE_M/L/H definierte Schriftart verwendet.

2 Felder

Proportionalschrift in Feldern wird durch zwei Konzepte realisiert:

- Zur Darstellung werden Fieldareas verwendet
- Zur Erfassung werden Edit Controls verwendet

2.1 Felderfassung durch Edit Controls

Wenn ein Feld durch eine Fieldarea dargestellt wird, dann erfolgt die Erfassung durch ein Edit Control.

Durch die Verwendung von Edit Controls verändert sich die Erfassung von Feldern grundlegend. Die von Windows bekannten Tasten können zur Erfassung des Feldwertes verwendet werden. Hierbei entsteht ein Konflikt der Tastenbelegung von FIX mit der Tastenbelegung von Windows, der über die Konfiguration der Tastenbelegung gelöst werden kann.

Wenn ein Feld betreten wird, dann wird der komplette Text selektiert. Das Drücken einer Taste, die nicht zum Editieren dient, löscht deshalb den kompletten Feldwert. Neu ist auch, dass der Mauszeiger zum Positionieren des Carets und zum Selektieren von Text verwendet werden kann. Außerdem sind folgende Tasten belegt, wenn sie nicht durch eine Konfiguration anderes belegt werden:

```
Pfeil links/rechts                Caret bewegen
Pfeil links/rechts mit Strg       Caret wortweise bewegen
Pfeil links/rechts mit Shift      Text selektieren
Pos1                              Caret zum Anfang
Ende                              Caret zum Ende
Entf                              Zeichen rechts löschen
Backspace                         Zeichen links löschen
Strg-Entf                         löscht bis zum Ende
Strg-C                            in Zwischenablage kopieren
Strg-X                            in Zwischenablage kopieren, löschen
Strg-V                            fügt Text aus Zwischenablage ein
Strg-Z                            letzte Änderung rückgängig machen
```

Die Eingabe eines Zeichens führt zum Einfügen des Zeichens an der aktuellen Caret-Position. Dabei wird ein selektierter Text gelöscht.

Je nach Feldtyp verwendet FIX/Win eine andere Logik bei der Eingabe. Diese Logik ersetzt die Eingabelogik von FIX. Sie wird in zwei Schritte unterteilt, die auch von Funktionen in der Benutzerbibliothek übernommen werden können:

Schritt1: Vorbereiten des Edit Controls. Das Edit Control wird über das Feld gelegt und entsprechend der Feldeigenschaften konfiguriert. Dieser Schritt wird jedes mal ausgeführt, wenn FIX einen Text zur Erfassung an FIX/Win übermittelt.

Schritt2: Prüfung von Eingabezeichen. Ein Zeichen wird vor dem Einfügen geprüft. Dieser Schritt wird jedes mal ausgeführt, wenn ein Zeichen eingegeben wurde.

2.2 Vorbereiten des Edit Controls

Nach dem Positionieren des Edit Controls über dem Feld wird es konfiguriert:

- Die Länge der Eingabe wird auf die Feldlänge begrenzt (f->len).
- Der durch FieldFont und FieldFontSize_<x> definierte Font wird eingestellt (siehe ...).
- Die durch den Eintrag FIELD_INVERS definierte Hintergrundfarbe wird eingestellt.
- Wenn die Feldeigenschaft LOWER gesetzt ist, dann wird der Fensterstil ES_LOWERCASE für das Edit Control gesetzt.
- Wenn die Feldeigenschaft UPPER gesetzt ist, dann wird der Fensterstil ES_UPPERCASE für das Edit Control gesetzt.
- Wenn die Feldeigenschaft INVISIBLE gesetzt ist, dann wird der Fensterstil ES_PASSWORD für das Edit Control gesetzt. Als Eingabezeichen wird durch Senden der Message EM_SETPASSWORDCHAR das Zeichen "*" definiert.
- Wenn die Feldeigenschaft RGTADJ gesetzt ist, dann wird der Fensterstil ES_RIGHT für das Edit Control gesetzt.
- Wenn die Feldeigenschaft NOMOD gesetzt ist, dann wird durch Senden der Message EM_SETREADONLY das Feld gegen Veränderungen geschützt.

- Bei numerischen Feldern (FXSHORTTYPE, FXLONGTYPE, FXDOUBLETYPE, FXFLOATTYPE, FXDECIMALTYPE, FXMONEYTYPE) werden alle Leerzeichen am Anfang entfernt. Dieser Schritt ist notwendig, weil die Eingabelänge begrenzt ist und damit zusätzliche Ziffern eingegeben werden können.
- Bei Feldern, die ein Format zur Erfassung von Brüchen besitzen, werden die Leerzeichen von allen Bestandteilen (Ganzzahl, Zähler, Nenner) entfernt. Außerdem werden fehlende Bestandteile ergänzt.
- Wenn das Feld nicht die Eigenschaft NOMOD besitzt und es sich nicht um ein Feld mit values-Angabe handelt, dann wird der komplette Feldwert selektiert.

Nach dieser Konfiguration wird die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownAcceptControlPrepare(
     CallBackFct p_callBack,   void* p_callID,
     HWND p_hWNDAccept, long orgWndstyle, int p_dtyp, int p_styp,
     int p_editflags, const _TCHAR* p_text, const _TCHAR* p_format);
```

aufgerufen, wenn sie in der Benutzerbibliothek definiert wurde. Damit besteht die Möglichkeit zusätzliche Schritte zur Vorbereitung des Edit Control zu programmieren. Die Parameter haben folgende Bedeutung:

- p_hWNDAccept: Fenster-Handle des Edit Controls.
- p_orgWndstyle: Fensterstil beim Anlegen des Edit Controls. Durch Verknüpfung mit ES_UPPERCASE, ES_RIGHT, ... können die entsprechenden Eigenschaften gesetzt werden.
- p_dtyp: FIX-Datentyp des Feldes
- p_styp: FIX-Feldeigenschaften
- p_editflags: Allgemeine Flags: EDITFLAG_FRACFORMAT - Feld hat Bruchformat, EDITFLAG_DECIMALPOINT - Formatiertes Feld verwendet Dezimalpunkt und nicht Dezimalkomma, EDITFLAG_VALUES - Feld hat values-Angabe.
- p_text: Text des Feldes
- p_format: Format des Feldes oder "", wenn kein Format vorhanden.

Als Rückgabewert muss die Funktion NULL oder "" liefern, wenn kein Fehler aufgetreten ist. Im Fehlerfall muss sie den Fehlertext zurück liefern.

2.3 Prüfung von Eingabezeichen

Die Prüfung von Zeichen für Felder vom Typ FXCHARTYPE lässt grundsätzlich alle Zeichen zu. Wenn die Eigenschaft ASCIIONLY gesetzt ist, werden jedoch nur ASCII-Zeichen zugelassen. Wenn die Eigenschaft FIRSTUP gesetzt ist, dann wird bei der Eingabe eines Zeichens an Position 0 dieses Zeichen in einen Großbuchstaben umgewandelt.

Bei Feldern vom Typ FXSHORTTYPE oder FXLONGTYPE dürfen nur die Zeichen '0'-'9' und '-' eingegeben werden. Alle anderen Zeichen werden abgewiesen. Das Zeichen '-' erfährt eine Sonderbehandlung. Es schaltet das Vorzeichen um, das unabhängig von der Schreibmarke immer am Anfang des Feldes eingefügt wird.

Bei Feldern vom Typ FXDOUBLETYPE, FXFLOATTYPE, FXDECIMALTYPE, FXMONEYTYPE dürfen nur die Zeichen '0'-'9', '-', '.' und ',' eingegeben werden. Die Zeichen '.' und ',' werden immer in den Dezimaltrenner umgesetzt, um die Eingabe mit dem Ziffernblock zu ermöglichen. Das ist in der Regel das Zeichen '.'. Für Felder mit Format lässt sich der Dezimaltrenner umdefinieren (->FXRADIXCHAR). Tausendertrenner lassen sich nicht eingeben. Sie können jedoch im vorhandenen Feldwert vorkommen. Deshalb werden sie bei der Übernahme durch FIX ignoriert und bei der Neudarstellung durch das Format rekonstruiert. Der Benutzer kann Tausendertrenner bei der Eingabe ignorieren. Das Zeichen '-' kann nur eingegeben werden, wenn es durch das Format oder durch die Feldeigenschaften (UPPER) erlaubt wird. In diesem Fall reagiert es genauso wie bei Feldern vom Typ FXSHORTTYPE oder FXLONGTYPE.

Wenn das Feld ein Format zur Erfassung von Brüchen besitzt, dann können zusätzlich die Zeichen '/' und T verwendet werden. Mit T wird das Zeichen bezeichnet, das im Format als Trenner zwischen Ganzzahl und Bruch definiert wird. Diese Zeichen und die Zeichen '.' und ',' werden jedoch nicht in den Text eingefügt. Stattdessen werden sie zur Positionierung und Selektion der Teilkomponenten verwendet. Die Eingabe von '/' springt hinter den Nenner und selektiert diesen, damit er durch die Eingabe von weiteren Ziffern ersetzt werden kann. Die Eingabe von T springt hinter den Zähler und selektiert diesen. Die Eingabe von '.' oder ',' springt

jeweils eine Komponente weiter. Auf diese Weise ist es leicht, Brüche mit dem Ziffernblock einzugeben. Da es vorkommen kann, dass nicht alle Komponenten vorhanden sind, werden vor dem Springen die fehlenden Teile ergänzt.

Bei Feldern vom Typ FXDATETYPE werden alle Zeichen außer '0'-'9', '.' und ',' abgewiesen. Auch hier wird das Zeichen ',' auf das Zeichen '.' abgebildet, um eine Eingabe mit dem Ziffernblock zu ermöglichen. Da nach dem Übermitteln des Wertes an FIX die übliche Felderfassung den Wert weiterverarbeitet, ist es auch bei der Verwendung von Edit Controls möglich nur Teile eines Datums einzugeben. Die restlichen Teile werden dann - genauso wie bei der Erfassung ohne Edit Controls - von FIX ergänzt.

Felder mit den Datentypen FXDTIMETYPE und FXINVTYPE werden zurzeit nur rudimentär unterstützt. Zur Eingabe sind die Zeichen '0'-'9', '-', ':', ' ' und '.' erlaubt. Der Wert muss genauso eingegeben werden, wie er vom Format definiert wird. Eine Ergänzung von fehlenden Komponenten findet nicht statt.

Zur Änderung oder Erweiterung dieser Prüfverfahren ist es möglich, eine Funktion in der Benutzerbibliothek zu hinterlegen, die die Eingabe eines Zeichens prüft. Dazu ist die Funktion

```
FWOWN_API _TCHAR* CALLBACK FwownAcceptControlInsert(CallBackFct p_callBack,
     void* p_callID, _TCHAR* p_char, bool* p_doDefault);
```

zu implementieren und zu exportieren. Wenn sie vorhanden ist, dann wird sie vor der FIX/Win internen Funktion aufgerufen. Über den Rückgabewert steuert sie den weiteren Verlauf:

- NULL: FIX/Win interne Prüfung wird aufgerufen.
- "": Kein Fehler. FIX/Win interne Prüfung wird nicht aufgerufen.
- alle anderen Texte: Fehler. Rückgabewert enthält Fehlermeldung, die ausgegeben wird.

Der Parameter p_char enthält das eingegebene Zeichen. Um es einzufügen, kann die Funktion entweder den Wert, auf den p_doDefault zeigt, auf true setzen oder das Zeichen selbst in das Edit Control schreiben und *p_doDefault auf false setzen. Das letzte Verfahren ist anzuwenden, wenn das Zeichen nicht an der aktuellen Position eingefügt werden soll. Wenn das Zeichen ignoriert werden soll, dann ist *p_doDefault ebenfalls auf false zu setzen. Wenn durch die Rückgabe von NULL die FIX/Win interne Prüfung aufgerufen wird, dann wird der Wert in p_doDefault ignoriert und neu bestimmt. Zusätzliche Parameter, die zur Prüfung des Zeichens benötigt werden (z.B. der Feldtyp), müssen beim Aufruf von FwownAcceptControlPrepare() in entsprechende Datenstrukturen gespeichert werden.

2.4 Konfiguration von Tasten

FIX/Win verwendet für das Edit Control eine eigene Fenster-Prozedur um das Verhalten bei der Tastatureingabe zu steuern. Dazu wird folgendes Verfahren verwendet:

- Wenn es sich um eine Taste zum Editieren handelt, dann wird die Taste an das Edit Control weitergeleitet und löst somit die von Windows definierte Standardreaktion aus. Diese Tasten werden Infield-Editing-Tasten genannt. Sie können über eine Datei (infieldkey.fix) definiert werden.
- Wenn es sich nicht um eine Infield-Editing-Taste handelt, dann wird geprüft, ob für die Taste ein FIX-Event in der Tastenbelegung (stdkey.fix) definiert ist. Wenn ja, dann wird die Erfassung beendet und das Event zusammen mit dem Text des Edit Controls an FIX gesendet.
- Wenn der Taste kein FIX-Event zugeordnet wurde und es sich um eine Zeichentaste handelt (WM_CHAR), dann wird FwownAcceptControlInsert() b.z.w. die interne Prüfung von Zeichen aufgerufen. Je nach Ergebnis wird die Taste an das Edit Control weitergeleitet oder nicht.

Damit kann die Belegung von zweideutigen Tasten wie z.B. Pos1 leicht gesteuert werden. Wenn die Taste benutzt werden soll, um an den Anfang des Feldes zu gelangen, kann die Taste entweder als Infield-Editing-Taste definiert werden oder das Event kann aus der Tastenbelegung entfernt werden. Der Unterschied zwischen den beiden Methoden besteht darin, dass das Entfernen aus der Tastenbelegung sich global (z.B. auch bei Menüs) auswirkt.

Eine Abweichung dieses Verfahrens findet bei Feldern mit values-Angabe statt. Bei diesen Feldern führt die Eingabe eines Zeichens zum Beenden der Erfassung durch FIX/Win. Das Zeichen wird als Event an FIX gesendet, was zur Auswahl des zugeordneten Wertes führt. Außerdem werden alle Tasten zum Löschen von Text ignoriert.

Format der Datei infieldkey.fix

Die Datei enthält für jede Taste, die als Infield-Editing-Taste gelten soll, eine Zeile mit folgenden Angaben:

```
<Tastenname> <Shift> <Ctrl> <Alt> [<smart>]
```

Der Wert für <Tastenname> kann aus dem Kapitel "4 Erstellen einer eigenen Tastenbelegung" aus FIX/Win Handbuch ermittelt werden. Die Werte für <Shift>, <Ctrl>, <Alt> können entweder 0 oder 1 sein, je nach dem ob die entsprechende Taste gedrückt ist oder nicht.

Der Wert für <smart> ist optional. Er kann ebenfalls 0 oder 1 sein, wobei das Weglassen der Angabe dem Wert 0 entspricht. Wenn er mit 1 angegeben wird, dann wird die Taste als "smart" behandelt. Das bedeutet, dass die Taste nur in bestimmten Situationen als Infield-Editing-Taste verwendet wird. Da dieses Verhalten für jede Taste programmiert werden muss, ist die Angabe nur für bestimmte Tasten sinnvoll. In dieser Version können die Tasten HOME und END ohne Zusatztasten (Shift, Ctrl und Alt) als "smart" definiert werden. In diesem Fall gilt die Taste HOME nur als Infield-Editing-Taste, wenn sich die Schreibmarke nicht am Anfang des Edit Controls befindet. Entsprechendes gilt für die Taste END. Sie gilt nur als Infield-Editing-Taste, wenn sich die Schreibmarke nicht am Ende befindet.

Um Kommentare zu schreiben, ist am Anfang der Zeile das Zeichen '#' aufzuführen.

2.5 Behandlung von Maustasten

Wenn eine Maustaste in dem aktiven Feld gedrückt wird, dann wird diese an die Fensterprozedur des Edit Controls ausgeliefert. Da FIX/Win für das Edit Control eine eigene Prozedur besitzt, werden die Maustasten zunächst von FIX/Win behandelt. FIX/Win leitet die linke Maustaste an die Prozedur des Edit Controls weiter. Sie wird damit zur Positionierung der Schreibmarke innerhalb des Feldes benutzt. Die rechte Maustaste wird an das Hauptfenster von FIX/Win weitergeleitet. Je nach Konfiguration sendet FIX/Win dann das Event BT_RIGHT oder das in der Tastaturtabelle definierte Event an die Anwendung. Damit wird die rechte Maustaste nicht für das (von Windows zugeordnete) Kontextmenü des Edit Controls genutzt. Statt dessen kann sie für FIX/Win Kontextmenüs verwendet werden.

Wenn mit eine der Maustasten außerhalb des Edit Controls gedrückt wird, dann wird die Erfassung mit dem Edit Control beendet und das der Maustaste zugeordnete Event (in der Regel BT_...) wird an FIX gesendet (siehe folgender Abschnitt).

2.6 Beenden des Editvorgangs

FIX startet auch bei der Erfassung über Edit Controls zunächst die normale Felderfassung. An der Stelle, an der FIX auf eine Eingabetaste wartet, wird der aktuelle Text des Feldes (f->info) an FIX/Win gesendet. FIX/Win positioniert dann das Edit Control im Inneren des Feldes und schreibt den Text in das Control. Der Benutzer kann dann den Text verändern.

Der Editvorgang wird in folgenden Situationen beendet:

- Es wurde eine Taste gedrückt, die keine Infield-Editing-Taste ist.
- Es wurde eine Aktion ausgelöst, die zum Senden eines FIX-Events führt. Das kann zum Beispiel das Anklicken eines Icons, eines anderen Feldes oder einer Paintarea sein.
- Das letzte Zeichen eines Feldes wurde eingegeben und das Feld besitzt die Eigenschaft AUNXT.

Der geänderte Text wird zusammen mit dem Event, dass für die Beendigung verantwortlich ist, zurück an FIX gesendet. Im Falle "AUNXT" wird K_RT als Event verwendet. FIX formatiert den geänderten Text ja nach Datentyp um und schreibt ihn in das Feld zurück. Danach wird das Event nach dem herkömmlichen Verfahren verarbeitet.

2.7 Copy And Paste

Das Verhalten von Copy und Paste ist bei der Verwendung von Fieldareas und Edit Controls anders als bei der Verwendung der bisherigen FIX-Felder.

Bisher wurde beim Kopieren immer der sichtbare Teil des aktuellen Feldes kopiert. Beim Einfügen wurden die Zeichen in der Zwischenablage in Events umgewandelt und an FIX gesendet. Mit den Edit Controls erfolgt die Felderfassung auf Windows Seite. Da sie bereits einen Copy&Paste Mechanismus mitbringen, wird dieser auch genutzt. Er arbeitet in der gewohnten Weise, wobei Teile des Textes markiert werden können, über Strg-C kopiert werden und über Strg-V durch den aktuellen Inhalt der Zwischenablage ersetzt werden können. Der alte Mechanismus ist jedoch immer noch vorhanden und kann bei Bedarf über spezielle Funktionen angesteuert werden.

Copy

Wenn im Kontextmenü die Copy-Funktion ausgelöst wird (CTXMEN_FCT_COPY), dann wird nicht mehr (wie in 3.1.0 und 3.0.0) der Inhalt des aktuellen Feldes kopiert. Statt dessen wird der Inhalt des Feldes kopiert, über dem die Maustaste zum Öffnen des Kontextmenüs gedrückt wurde. Diese Änderung ist sowohl bei der alten Erfassung mit FIX-Feldern als auch bei der Verwendung von Fieldareas wirksam.

Bei FIX-Feldern wird allerdings nur der sichtbare Teil kopiert (Keine Veränderung gegenüber FIX/Win 3.1.0). Bei der Verwendung von Fieldareas wird der komplette Feldinhalt als Text kopiert.

Verwendung von Tasten

Bei der Verwendung von FIX-Feldern wird durch das Drücken von Strg-C der sichtbare Teil des aktuellen Feldes in die Zwischenablage kopiert (Es sei denn, der Taste wurde ein FIX-Event in der Tastenbelegung zugewiesen).

Bei der Verwendung von Fieldareas und Edit Controls wird durch das Drücken von Strg-C der markierte Teil des Feldes (Edit Controls) kopiert.

Dabei ist zu beachten, dass Strg-C über die oben beschriebenen Verfahren (Abschnitt: Konfiguration von Tasten) umkonfiguriert werden kann.

Paste

Wenn im Kontextmenü die Paste-Funktion ausgelöst wird (CTXMEN_FCT_PASTE), dann wird der Inhalt der Zwischenablage als Events an die Anwendung gesendet. Wenn die Felderfassung über Edit Controls aktiv ist, dann wird diese vorher beendet und erst wieder eingeschaltet, nachdem das letzte Event gesendet wurde. Das bedeutet, dass die gesendeten Events von der FIX-Felderfassung verarbeitet werden.

Verwendung von Tasten

Das Drücken von Strg-V sendet den Inhalt der Zwischenablage als Tastenevents an die Anwendung, wenn es sich bei dem aktuellen Feld um ein FIX-Feld handelt oder zu diesem Zeitpunkt kein Feld aktiv ist.

Bei Edit Controls fügt Strg-V den Inhalt der Zwischenablage an der aktuellen Position ein (Windows-Standardverhalten). Wenn mit Edit Controls gearbeitet wird, sollte die Tastenbeschreibung Strg-V nicht im Kontextmenü verwendet werden, weil in diesem Fall die Funktion anders arbeitet als die Taste.

Dabei ist zu beachten, dass Strg-V über die oben beschriebenen Verfahren (Abschnitt: Konfiguration von Tasten) umkonfiguriert werden kann.

Auslösen in der Anwendung

Durch Aufruf von

```
fx_send_clipboard()
```

kann FIX/Win veranlasst werden, den Inhalt der Zwischenablage als Events zu senden. Dazu wird die gleiche Funktion verwendet, die auch über das Kontextmenü aufgerufen werden kann.

Füllen von mehreren Feldern

Das Senden von Events kann (und konnte auch schon in den Versionen 2.9.5 - 3.1.0) zum Füllen von mehreren Feldern verwendet werden, wenn der Text in der Zwischenablage ein entsprechendes Format hat. Dazu wird ein Zeilenumbruch in das Event K_RT umgesetzt.

In der Version bis einschließlich 3.1.0 bestand das Problem, dass dadurch der Feldinhalt nur soweit überschrieben wurde, wie Zeichen in der Zwischenablage vorhanden waren.

Beispiel

In drei Feldern stehen die Texte

```
"eins"
"zwei"
"drei"
```

In der Zwischenablage steht:

```
A
B
C
```

Durch das Einfügen über fx_send_clipboard() - oder das Kontextmenü steht danach in den Feldern:

```
"Ains"
"Bwei"
"Crei"
```

oder wenn der mit 3.1.0 eingeführte Insert-Modus eingeschaltet ist:

```
"Aeins"
"Bzwei"
"Cdrei"
```

vorausgesetzt am Ende des Feldes ist noch genügend Platz vorhanden.

Man würde jedoch erwarten, dass die Feldinhalte komplett durch die Inhalte der Zwischenablage ersetzt werden.

Um dies zu erreichen, kann der Feldinhalt beim Betreten des Feldes (L_ENTERFIELD) gelöscht werden. Um das Löschen beim normalen Betreten zu verhindern, kann abgefragt werden, ob gerade Events von FIX/Win aus der Zwischenablage gesendet werden. Dazu ist die Funktion

```
BOOLEAN is_within_pasting()
```

zu verwenden. Sie liefert TRUE, wenn FIX/Win begonnen hat, Events aus der Zwischenablage zu senden und noch nicht alle Events gesendet hat. Da zu Beginn dieses Vorgangs die aktuelle Felderfassung beendet wird und das Feld nochmals betreten wird, genügt folgender Code, um alle Felder vor dem Versorgen mit Events aus der Zwischenablage zu leeren:

```
...
case L_ENTERFIELD:
     if (is_within_pasting()) {
          ferase(_f, FALSE);
     }
     break;
...
```

Alternativ zu diesem Verfahren, ist es möglich, das Event L_PASTING abzufangen. Dieses Event wird vor und

nach dem Senden von Events aus der Zwischenablage gesendet. Beginn und Ende können durch Aufruf von is_within_pasting() unterschieden werden. Wurden alle Events gesendet, dann liefert die Funktion FALSE. Wurden noch keine oder einige Events gesendet, liefert sie TRUE.

```
...
case L_PASTING:
     if (is_within_pasting()) {
           ferase(fx_mfo_f(mskp, MASK_FELD1), FALSE);
           ferase(fx_mfo_f(mskp, MASK_FELD2), FALSE);
           ferase(fx_mfo_f(mskp, MASK_FELD3), FALSE);
     }
     break;
...
```

Dieses Beispiel löscht zu Beginn des Einfügevorgangs die Felder MASK__FELD1, MASK__FELD2, MASK__FELD3. Es zeigt, den typischen Einsatz des Verfahren. Hier wird immer eine feste Anzahl Felder gelöscht - unabhängig davon, wie viele Daten sich in der Zwischenablage befinden.

Trotz dieser Verbesserung in FIX 4.0.0 bleibt das Einfügen von mehreren Feldwerten aus der Zwischenablage kompliziert. Es gibt viele Fälle, in denen es nicht korrekt funktioniert:

- Ein Feldwert enthält unzulässige Zeichen.
- Ein Feldwert enthält mehr Zeichen als für das Feld zugelassen.
- Die Anwendung - oder FIX - meldet einen Fehler, der eine Eingabe erwartet.
- Durch die Zeichen in der Zwischenablage wird ein unzulässiger Feldwert erzeugt.

... um nur einige zu nennen. Deshalb sollte diese Möglichkeit nur an wenigen Stellen mit Vorsicht eingesetzt werden.

2.8 Verkürzen von Feldern

Da durch die Verwendung von Proportionalschrift gerade bei numerischen Feldern und Datumsfelder weniger Platz benötigt wird, ist es möglich, die angezeigte Länge (Displaylänge) auch bei Feldern dieser Typen kleiner anzugeben als die tatsächliche Länge in Zeichen. Diese Möglichkeit war bisher Feldern mit dem Datentyp "char" vorbehalten. Wenn durch diese Angabe das Feld zu kurz wird, dann kann der Wert bei der Eingabe verschoben (gescrollt) werden.

Bei der Anzeige werden längere Werte abgeschnitten. Bei Feldern, die rechts ausgerichtet sind, wird der vordere Teil abgeschnitten. Da es dadurch zu Missverständnissen kommen kann (Von dem Wert 1.234.200 ist nur 234.200 sichtbar), sollte die Darstellung von wichtigen Feldern beim Ändern der angezeigten Länge und nach jeder Änderung der Schriftart überprüft werden.

Bei weniger wichtigen Feldern kann ein Abschneiden akzeptiert werden. Durch automatische Tooltips kann der komplette Feldwert dieser Felder zur Anzeige gebracht werden, ohne das Feld zu betreten.

Wenn eine Maske geladen wird, die kürzere Displaylängen für Felder definiert und es wird nicht mit Fieldareas gearbeitet, dann korrigiert FIX beim Laden der Maske die Displaylängen auf die Feldlänge. Wenn das Programm mit dem Schalter -dev gestartet wird, wird dabei eine Meldung ausgegeben.

2.9 Probleme

Feldeigenschaft BEHIND

Die Feldeigenschaft BEHIND wird von Edit Controls anders interpretiert. Dadurch, dass der Feldwert beim Betreten komplett selektiert wird, steht die Schreibmarke immer am Ende des Textes. Das Setzen von BEHIND bewirkt deshalb, dass die Schreibmarke zwar am Ende steht, der Text jedoch nicht selektiert wird.

Meldungen mit fflash()

Meldungen, die mit fflash() ausgegeben werden (z.B. "Maske kann nur im Ersten Feld verlassen werden" beim Drücken von Ende) verschwinden bei der FIX-Felderfassung mit der nächsten Taste. Bei der Verwendung von Edit Controls wird die nächste Taste jedoch erst beim Verlassen des Feldes gesendet.

Verwendung von undcgetch

Mit undcgetch() in den Puffer gestellte Zeichen werden von der FIX-Felderfassung interpretiert. Das Edit Control ist nicht davon betroffen. Das bedeutet, dass sich diese Tasten nur auswirken, wenn sie den Feldwert ändern.

Beispiele

```
undcgetch(K_CE); /* Feld löschen */
...
faccept(); /* Felderfassung */
```

K_CE löscht das Feld, dann wird der Wert zum Editieren an Windows weitergegeben. Dieses Beispiel funktioniert auch mit Edit Controls.

```
undcgetch(K_kr); /* Cursor nach rechts */
undcgetch(K_kr); /* Cursor nach rechts */
undcgetch(K_kr); /* Cursor nach rechts */
undcgetch(K_kr); /* Cursor nach rechts */
undcgetch('T');
undcgetch('S');
undcgetch('E');
undcgetch('T');
...
faccept(); /* Felderfassung */
```

Der Wert "TEST" wird in das Feld geschrieben. Danach wird der Cursor um vier Stellen nach rechts bewegt. Diese Positionierung findet nur im FIX-Feld und nicht im Edit Control statt. Der Wert wird jedoch vom Edit Control übernommen.

Performance-Probleme

Auf langsamen Rechnern kommt es zu folgenden Problemen, denen mit einer Einstellung in config/fixwin.rc entgegen gewirkt werden kann.

Das Edit Control wird zur Verbesserung der Performance nicht immer ausgeblendet, verschoben und dann wieder eingeblendet. Stattdessen wird es nur verschoben. Dabei kann es vorkommen, dass die Maske vom Schirm genommen wird, das Edit Control aber noch stehen bleibt und erst verschoben wird, wenn die nächste Maske eingeblendet wird.

Durch das Setzen der Einstellung

```
HideEditControl: true
```

wird bewirkt, dass das Edit Control nach jedem Editiervorgang ausgeblendet wird.

Windows optimiert das Neuzeichnen des Fensterinhalts. Wenn das System ausgelastet ist, wird der Fensterinhalt nicht direkt neu gezeichnet. Das kann dazu führen, dass nicht alle Zwischenzustände gezeichnet werden. Da mit dem Edit Control ein weiteres Fenster ins Spiel kommt, kann es vorkommen, dass die Inhalte der beiden Fenster nicht synchron sind. Das äußert sich beispielsweise in einer Tabelle, in der mit der Pfeil-nach-unten Taste nach unten geblättert wird. Man erkennt, dass in der Tabelle noch die alten Inhalte angezeigt werden, während der Wert im Edit Control aktualisiert wird. Erst beim Loslassen der Taste wird der Inhalt der Tabelle aktualisiert.

Das Setzen von

```
ForceUpdateScreen: true
```

bewirkt, dass ein geänderter Bildschirminhalt immer sofort gezeichnet wird. Damit wird die Optimierung von Windows umgangen und das oben beschriebene Verhalten tritt nicht mehr auf.

3 Menüs

Proportionalschrift in Menüs wird über Paintareas abgebildet. Dazu wurde ein neuer Paintarea-Typ PA_MENUITEM geschaffen. Damit die Paintareas im Menü gezeichnet werden können, ist es wichtig, dass das Menü ein (wenn auch leeres) Layout besitzt. Wenn das nicht der Fall ist, dann wird beim Laden eines Menüs, das Paintareas verwendet, ein leeres Layout angelegt.

Die Paintarea des Menüpunktes legt sich über den gesamten Bereich des Menüpunktes inklusive des Platzes des linken und rechten Begrenzers. Dadurch kann in der Frontend-Library fwown.dll der komplette Menüpunkt in einer Paintarea gezeichnet werden.

Um Paintareas für Menüpunkte zu benutzen, muss in der FIX-Ressource-Datei fix.rc die Ressource

```
UseMenuItemPaintarea
```

auf TRUE gesetzt werden.

Als Token für die Paintarea wird der im Feld "Darstellung" (led) definierte Text verwendet, wobei er vorne und hinten um ein Leerzeichen für den Begrenzer erweitert wird. Wenn eine Anzahl hervorzuhebender Zeichen angegeben wurde, dann wird dieser Wert als Länge des Tokens interpretiert. Im Gegensatz zur Darstellung von Menüpunkten mit nicht-Proportionalschrift, kann dadurch der für das Token verwendete Text und damit die Breite des Menüpunkts gekürzt werden. Wenn der Wert größer ist, als die Länge des Tokens plus der beiden Leerzeichen für die Begrenzer, dann wird das Token am Ende um die entsprechende Anzahl Leerzeichen ergänzt.

Als Text für die Paintarea wird ebenfalls der Text des Menüpunktes verwendet. Er wird nicht gekürzt oder um Leerzeichen am Ende erweitert, wenn eine andere Länge angegeben wurde. Am Anfang wird jedoch ein Leerzeichen eingefügt, so dass der Text in der Darstellung nicht direkt am Anfang der Paintarea beginnt. Wenn der Text der Paintarea über eine Übersetzungsfunktion (S_pa_translate_token) definiert werden soll, dann ist die Ressource

```
UseMenuItemText
```

auf FALSE zu setzen. In diesem Fall wird NULL als Text übergeben und FIX ruft die Übersetzungsfunktion zur Bestimmung des Textes auf. Wenn das Menü im led bearbeitet wird, dann wird diese Ressource nicht ausgewertet. Statt dessen wird immer der Text des Menüpunktes (um ein Leerzeichen am Anfang erweitert) verwendet.

Eine weitere Ressource

```
MenuItemTooltip
```

bestimmt den Text für den Tooltip. Sie kann einen der Werte 1-8 enthalten. Dieser Wert definiert die Nummer des Longval-Wertes, der den Tooltip bestimmt. Enthält die Ressource beispielsweise den Wert 4, dann definiert longval4 die Nummer der Meldung aus der Meldungsdatei, die den Tooltip definiert. Zusätzlich ist es möglich, den Wert PROMPT oder den Wert ITEM der Ressource zuzuweisen. In diesem Fall wird die Meldungsnummer aus der Promptnummer gebildet oder der Tooltip wird aus dem Text des Menüpunktes gebildet. Alle anderen Werte der Ressource führen dazu, dass keine Tooltips für Menüpunkte verwendet werden.

Damit die Anklickbarkeit von Menüpunkten auch außerhalb des aktuellen Menüs gewährleistet ist, wird dem Menü nach dem Laden eine ActiveObject-List zugewiesen. Diese Liste enthält alle Menüs oberhalb des aktuellen Menüs im Menü-Baum, soweit sie mittels perform 'Menüdatei' vom Menü gestartet wurden.

Ist das nicht ausreichend, so muss die ActiveObject-List des Menüs geändert werden. Dabei ist zu beachten, dass die am Menü-Objekt hängende Liste per trycalloc allokiert wurde und nach dem Beenden des Menüs, wenn das Menüobjekt nicht aufgehoben wird, freigegeben wird.

Ist das Menü so konfiguriert, das es nicht freigegeben wird, so muss der Speicher für die ActiveObject-List mittels tryfree selbst freigegeben werden. Anderenfalls darf der Speicher nicht freigegeben werden, da er beim Löschen des Menüobjektes auch freigegeben wird.

4 Selos

4.1 Probleme bei der Ausrichtung

Für eine Spalte eines Selos konnte bisher keine Ausrichtung definiert werden. Zur Ausrichtung der Spalten untereinander wurden Leerzeichen verwendet, die durch das entsprechende Format erzeugt wurden. Da bei einer proportionalen Schriftart die Leerzeichen eine andere Breite haben als andere Zeichen, ist eine Ausrichtung durch Leerzeichen nicht mehr möglich. Anderseits werden die Leerzeichen bei der Übertragung an FIX/Win aus Optimierungsgründen entfernt. Damit die Spaltenwerte untereinander ausgerichtet werden, ist die Definition der Ausrichtung (rechts oder links ausgerichtet) pro Spalte notwendig.

FIX bestimmt deshalb die Ausrichtung der Spalte anhand des Datentyps. Alle numerischen Spalten sowie Datums-, Interval- und Zeitangaben werden rechts ausgerichtet (INTTYPE, SERIALTYPE, LONGTYPE, FLOATTYPE ,DOUBLETYPE MONEYTYPE, DECIMALTYPE, DATETYPE, DTIMETYPE, INVTYPE). Alle anderen Spalten werden links ausgerichtet (CHARTYPE, GRAPHICSTYPE, TRUTHTYPE).

4.2 Definition der Überschriften

Die Überschriften werden schon seit der Version 3.0.0 als Paintareas vom Typ Tableheader dargestellt. Ab der Version 4.0.0 ist es möglich, zu dieser Paintarea weitere Attribute zu definieren. Dazu ist hinter der Überschrift das Schlüsselwort TA (für TableheaderAttributes) anzugeben. Danach sind der Text, der Tooltiptext, die Ausrichtung und ein Wert für Longval2 anzugeben. Text und Tooltip können auch als undefiniert ("") angegeben werden. Im Falle eines leeren Textes wird der Text über die Übersetzungsfunktion bestimmt oder wenn nicht vorhanden aus der Überschrift gebildet.

Wenn ein Text abgegeben wird, dann hat die Überschrift nur noch die Aufgabe, die Breite der Spalte zu bestimmen. Damit ist es möglich, die Breite von Spalten an die Proportionalschrift anzupassen. Bei numerischen Spalten ist das ohne Gefahr möglich, da alle Ziffern gleich breit sind und somit auch alle Feldwerte. Die Feldwerte benötigen weniger Platz als bei der Darstellung durch nicht proportionale Schrift. Deshalb könnte die Überschrift eigentlich durch die entsprechende Anzahl Leerzeichen ersetzen werden. Es ist jedoch sinnvoll, weiterhin eine gekürzte textuelle Bezeichnung zur wählen. Zum einen ist dieser Text beim Start der Anwendung über ein Terminal sichtbar (allerdings muss man hier damit rechnen, dass die Werte abgeschnitten werden) und zum anderen steht der Text im virtuellen Textbildschirm und ist bei der Fehlersuche hilfreich (z.B. bei Diagnose/Bildschirmanalyse).

Als Ausrichtung kann einer der Werte "Left", "Center" oder "Right" verwendet werden. Alle anderen Werte führen dazu, dass die Ausrichtung undefiniert ist. Die Ausrichtung wirkt sich zum einen auf die Überschrift und zum anderen auf die angezeigten Werte aus. Bei der Angabe "Left" oder "Right" übersteuert sie die durch den Datentyp definierte Ausrichtung. Umgekehrt richtet sich die Ausrichtung der Überschriften nach dem Datentyp, wenn keine explizite Ausrichtung angegeben wurde. Damit sind Werte und Überschrift gleich ausgerichtet. Wenn dieses Verhalten nicht gewünscht wird, dann lässt es sich durch Setzen der Ressource

```
AlignSeloHeadings
```

auf FALSE abschalten. Damit werden alle Überschriften links ausgerichtet und die Angabe wirkt sich nur auf die Darstellung der Werte aus.

Beispiel einer Spaltendefinition:

```
datum "Datum  " TA "Auf.Datum" "Datum des Auftrags" "Right" 0 "%10s",
```

Die Spalte datum verwendet als Überschrift eine Paintarea vom Typ Tableheader. Als Token wird "Datum " verwendet. Damit besitzt die Spalte eine Breite von 7 Zeichenzellen. Die Überschrift wird um 2 Zeichenzellen breiter, da links und rechts von der Spalte je ein Feldbegrenzer steht. Als Text wird "Auf.Datum" und als Tooltip wird "Datum des Auftrags" verwendet. Sowohl die Überschrift als auch die Werte der Spalte werden rechts ausgerichtet. Die Angabe "Right" kann auch weggelassen werden, da der Datentyp bereits eine Ausrichtung nach rechts definiert. Das Format "%10s" erzeugt Werte, die 10 Zeichen lang sind. Durch die Verwendung von

Proportionalschrift genügen 7 Zeichenzellen zur Darstellung dieser Werte.

5 Choices

Bei Choices gibt es keine Möglichkeit, bei der Definition der Fieldareas festzustellen, wie lang die Texte werden, die die Displayfunktion erzeugen werden. Bisher wurden zu lange Texte einfach abgeschnitten. Bei der Verwendung von Proportionalschrift bietet es sich jedoch an, Texte zuzulassen, die länger sind als die Spaltenbreite in Zeichenzellen. Fieldareas benötigen jedoch eine maximale Länge für den Text. (Bei Feldern wird f->info verwendet). Deshalb wird die maximale Textlänge auf das Doppelte der Spaltenbreite in Zeichen definiert. Wenn eine Choicespalte beispielsweise 20 Zeichenzellen breit ist, können in dieser Spalte Texte mit einer Maximallänge von 40 Zeichen ausgegeben werden. Wird diese Länge überschritten, dann kürzt FIX den Text. Bei Angabe von -dev wird eine entsprechende Meldung auf stderr (und damit im Meldungsfenster) ausgegeben.

Abgesehen davon sollte man beachten, dass Texte von FIX/Win abgeschnitten dargestellt werden, wenn die Darstellung die Breite der Spalte überschreitet. Wenn also in dem obigen Beispiel ein Text zwar weniger als 40 Zeichen lang ist aber bei der Darstellung mehr als 20 Zeichenzellen beansprucht, wird er ebenfalls abgeschnitten dargestellt.

Zur Ausrichtung des Spaltenwertes wurde die Syntax von Choices erweitert. Hinter dem Schlüsselwort display kann optional das Schlüsselwort RGTADJ angegeben werden. In diesem Fall werden die Werte rechts ausgerichtet. Eine Ausrichtung durch die Displayfunktion unter Verwendung von Leerzeichen ist wegen der Verwendung von Proportionalschrift nicht mehr möglich. Dies betrifft insbesondere Displayfunktionen, die aus mehreren Spaltenwerten den Wert für eine Spalte der Choice erzeugt haben und dabei die Spaltenwerte durch Leerzeichen untereinander ausgerichtet haben.

6 Meldungszeile

Durch das Setzen der Ressource

```
MsgAsPaintarea
```

auf TRUE, wird zur Darstellung einer Meldung in der Meldungszeile eine Paintarea verwendet. Damit verwendet auch die Meldungszeile Proportionalschrift.

Zur Darstellung wird der neue Paintarea-Typ PA_MSGLINE verwendet. Wenn Paintareas über die Funktion FwownDrawPaintArea() in der Benutzerbibliothek von FIX/Win gezeichnet werden, dann ist diese Funktion so anzupassen, dass sie entweder "" für diesen Typ zurückliefert oder die Paintarea selbst zeichnet. Die Rückgabe von "" bewirkt, dass FIX/Win die Paintarea zeichnet. Dazu wird die übergebene Zeichenkette analysiert. Die Zeichen mit den Codes 1-5 (=^A-^E) schalten das für die Ausgabe verwendete Videoattribut um. Die beiden Codes 16, 20 markieren den Anfang eines Tastenlabels. Die Codes 25, 21 markieren das Ende des Labels. FIX/Win interpretiert den Text zwischen diesen beiden Marken als Tastenlabel und stellt die entsprechende Grafik dar.

Als Schriftart wird die durch PropFont, PropFontSize, PropFontQuality und PropFontWeight definierte Schriftart verwendet. Die Farben für die Videoattribute werden aus den Einträgen PA_TEXT_NORMAL, PA_TEXT_INVERS, PA_TEXT_LOW, PA_TEXT_UNDERLINE und PA_TEXT_BLINK der Farbzuordnungstabelle gelesen.

7 Definition von Paintareas

Es ist ab der Version 4.0.0 möglich Paintareas direkt in der Layoutbeschreibungsdatei zu definieren. Dazu wurde die Syntax dieser Dateien geändert.

Die Beschreibungsdatei ist zeilenorientiert. Sie beginnt mit einem Kopf, der allgemeine Angaben enthält.

```
<Version <versionsnummer>>
<Geometry <ypos> <xpos>>
<CharacterSet <characterset>>
```

wobei

```
<versionsnummer>     ::= <unsigned>
<ypos>               ::= <unsigned>
<xpos>               ::= <unsigned>
<characterset>       ::= <string>
```

Auf den Kopf folgt die Beschreibung der einzelnen Layoutelemente, von Attributen und Paintareas.

Layoutelemente:

```
<(Draw|Put|Write) <Yx <ypos> <xpos>> <string>>
```

wobei

```
<ypos>               ::= <unsigned>
<xpos>               ::= <unsigned>
```

Bei Write ist der <string> der Text, der in das Layout geschrieben wird. Bei Draw werden die Zeichen aus <string> mit dem Attribut Grafik (CHARSET2) ins Layout geschrieben. Put schreibt Special-Chars mit dem Attribut SINGLE ins Layout.

Attribute:

```
<Set <Yx <ypos_von> <xpos_von>> <Yx <ypos_bis> <xpos_bis>> <Attr <attribut>>>
```

wobei

```
<ypos_von>           ::= <unsigned>
<xpos_von>           ::= <unsigned>
<ypos_bis>           ::= <unsigned>
<xpos_bis>           ::= <unsigned>
<attribut>           ::= <unsigned>|<attribut_list>
<attribut_list>      ::= ((Normal|Invers|Underline|Low|Blink|Bold|Notvisible|Grayed) )+
```

Die Set-Anweisung definiert das für einen Layoutbereich von <ypos_von> und <xpos_von> bis <ypos_bis> und <xpos_bis> gehenden Bereich gültige Attribut.

Paintareas:

Es können verschiedene Typen von Paintareas in der Layoutbeschreibungsdatei definiert werden. Dazu stehen folgende Anweisungen zur Verfügung:

```
<TextLabel <Yx <ypos> <xpos>> <labeltoken>
      [<Text <labeltext>>]
      [<Tooltip <tooltiptext>>]
      [<Attr <attribut>>]
      [<Align <align>>]
      [<MouseActionOn <linke_taste> <mittlere_taste> <rechte_taste>>]
      [<Longvals <longval1> <longval2> <longval3> <longval4>>]
>
```

wobei

```
<ypos>               ::= <unsigned>
<xpos>               ::= <unsigned>
<labeltoken>         ::= <string>
<labeltext>          ::= <string>
<tooltiptext>        ::= <string>
<attribut>           ::= <unsigned>|<attribut_list>
<attribut_list>      ::= ((Normal|Invers|Underline|Low|Blink|Bold|Notvisible|Grayed) )+
<align>              ::= <align_string>
<align_string>       ::= (Left|Center|Right)
<linke_taste>        ::= (0|1)
<mittlere_taste>     ::= (0|1)
<rechte_taste>       ::= (0|1)
<longval1>           ::= <unsigned>
<longval2>           ::= <unsigned>
<longval3>           ::= <unsigned>
<longval4>           ::= <unsigned>
```

Die TextLabel-Anweisung erzeugt an der Position <ypos> <xpos> eine Paintarea vom Typ TextLabel. <labeltoken> wird als Token verwendet. Soll der angezeigte Text abweichen, so ist die optionale Text-Anweisung anzugeben. Mit Tooltip kann der Paintarea ein Tooltip zugewiesen werden. Ein Zeilenumbruch innerhalb eines Tooltips muss oktal ('\012') angegeben werden. Weitere Eigenschaften, die optional zugewiesen werden können sind das Attribut, das Alignment, die Anklickbarkeit mit einer Maustaste und die vier an die Paintarea übertragenen Longvals. Bei der Anklickbarkeit bedeutet eine 1 das bei Klick der jeweiligen Taste ein entsprechendes pa - Event generiert wird.

```
<Button <Yx <ypos> <xpos>> <labeltoken>
      [<Text <labeltext>>]
      [<Tooltip <tooltiptext>>]
      [<Attr <attribut>>]
      [<Align <align>>]
      [<MouseActionOn <linke_taste> <mittlere_taste> <rechte_taste>>]
      [<Longvals <longval1> <longval2> <longval3> <longval4>>]
>
```

wobei

```
<ypos>               ::= <unsigned>
<xpos>               ::= <unsigned>
<labeltoken>         ::= <string>
<labeltext>          ::= <string>
<tooltiptext>        ::= <string>
<attribut>           ::= <unsigned>|<attribut_list>
<attribut_list>      ::= ((Normal|Underline|Low|Blink|Notvisible|Grayed) )+
<align>              ::= <align_string>
<align_string>       ::= (Left|Center|Right)
<linke_taste>        ::= (0|1)
<mittlere_taste>     ::= (0|1)
<rechte_taste>       ::= (0|1)
<longval1>           ::= <unsigned>
<longval2>           ::= <unsigned>
<longval3>           ::= <unsigned>
<longval4>           ::= <unsigned>
```

Die Button-Anweisung erzeugt an der Position <ypos> <xpos> eine Paintarea vom Typ Button. <labeltoken> wird als Token verwendet. Soll der angezeigte Text abweichen, so ist die optionale Text-Anweisung anzugeben. Mit Tooltip kann der Paintarea ein Tooltip zugewiesen werden. Ein Zeilenumbruch innerhalb eines

Tooltips muss oktal ('\012') angegeben werden. Weitere Eigenschaften, die optional zugewiesen werden können sind das Attribut, das Alignment, die Anklickbarkeit mit einer Maustaste und die vier an die Paintarea übertragenen Longvals. Bei der Anklickbarkeit bedeutet eine 1 das bei Klick der jeweiligen Taste ein entsprechendes pa - Event generiert wird. Bei der Button-Anweisung ist zu beachten, dass die Attribute Bold und Invers wegen der Einschränkungen der Paintarea vom Typ Button nicht verwendet werden dürfen. Sollten diese Attribute angegeben werden, so werden sie durch Normal ersetzt.

```
<TableHeader <Yx <ypos> <xpos>> (First|Middle|Last) <labeltoken>
     [<Text <labeltext>>]
     [<Tooltip <tooltiptext>>]
     [<Attr <attribut>>]
     [<Align <align>>]
     [<MouseActionOn <linke_taste> <mittlere_taste> <rechte_taste>>]
     [<Longvals <longval2> <longval3> <longval4>>]
>
```

wobei

```
<ypos>              ::= <unsigned>
<xpos>              ::= <unsigned>
<labeltoken>        ::= <string>
<labeltext>         ::= <string>
<tooltiptext>       ::= <string>
<attribut>          ::= <unsigned>|<attribut_list>
<attribut_list>     ::= ((Normal|Invers|Underline|Low|Blink|Bold|Notvisible|Grayed) )+
<align>             ::= <align_string>
<align_string>      ::= (Left|Center|Right)
<linke_taste>       ::= (0|1)
<mittlere_taste>    ::= (0|1)
<rechte_taste>      ::= (0|1)
<longval2>          ::= <unsigned>
<longval3>          ::= <unsigned>
<longval4>          ::= <unsigned>
```

Die TableHeader-Anweisung erzeugt an der Position <ypos> <xpos> eine Paintarea vom Typ TableHeader. Die Paintarea muss als Erste, Mittlere oder Letzte Spalte gekennzeichnet werden, da diese Positionen unterschiedlich gezeichnet werden. <labeltoken> wird als Token verwendet. Soll der angezeigte Text abweichen, so ist die optionale Text-Anweisung anzugeben. Mit Tooltip kann der Paintarea ein Tooltip zugewiesen werden. Ein Zeilenumbruch innerhalb eines Tooltips muss oktal ('\012') angegeben werden. Weitere Eigenschaften, die optional zugewiesen werden können sind das Attribut, das Alignment, die Anklickbarkeit mit einer Maustaste und die vier an die Paintarea übertragenen Longvals. Bei der Anklickbarkeit bedeutet eine 1 das bei Klick der jeweiligen Taste ein entsprechendes pa - Event generiert wird. Bei den Longvals dürfen nur drei Werte angegeben werden, da der Longval1 intern für die Kennzeichnung des TableHeaders als Erste, Mittlere oder Letzte Spalte verwendet wird.

Erzeugung der Paintareas

Aus den in der Layoutbeschreibungsdatei definierten Paintareas wird eine Datenstruktur zu einem Objekt erzeugt. Die Ressource

```
MaxPaintareasOnMask
```

definiert die maximale Anzahl Paintareas, die diese Datenstruktur aufnehmen kann (fix.rc). Als Default wird 256 verwendet.

Wenn ein Objekt angezeigt wird (o_appear()), dann wird die Datenstruktur ausgelesen und es werden die entsprechenden Paintareas erzeugt. Wenn das Objekt wieder vom Schirm genommen wird (o_disappear()), dann werden die Paintareas wieder entfernt.

Beim nächsten Anzeigen des Objektes werden die in der Datenstruktur aufgebauten Paintareas wieder erzeugt. Da mit einem Cache gearbeitet wird, entstehen nur geringe Performanceverluste. Allerdings gehen Paintareas, die programmatisch nach dem Anzeigen der Maske hinzugefügt wurden, verloren. Andererseits werden Paintareas, die nach den Anzeigen der Maske gelöscht wurden, wieder erzeugt. Eine Möglichkeit wäre es, die programmatisch ausgeführten Schritte zum Anlegen und Löschen von Paintareas nach jedem Anzeigen einer Maske zu wiederholen. Oft ist das jedoch nur schwer möglich, weil der dafür verantwortliche Code in einer bestehenden Anwendung über mehrere Stellen verteilt ist und nur schwer zu finden ist. In

diesem Fall kann die globale Variable

```
BOOLEAN B_save_pa;
```

vor dem Aufruf von o_disappear() auf TRUE gesetzt werden. FIX löscht dann die zu einem Objekt gemerkte Datenstruktur mit den Informationen aus der Layoutbeschreibungsdatei und baut sie anhand der aktuell vorhandenen Paintareas neu auf. Das bedeutet, das beim erneuten Anzeigen des Objektes nicht mehr die in der Layoutdatei beschriebenen Paintareas erzeugt werden. Stattdessen werden die Paintareas erzeugt, die vor dem Aufruf von o_disappear() vorhanden waren. Das betrifft nicht nur die Menge der Paintareas, sondern auch die Eigenschaften. Wird beispielsweise der Text einer in der Layoutbeschreibungsdatei definierten Paintarea geändert, dann ist diese Änderung auch bei der zweiten Anzeige sichtbar.

Folgende Punkte sind außerdem zu beachten:

- Die Anzahl der Paintareas kann durch programmatisch hinzugefügte Paintareas steigen. Deshalb ist MaxPaintareasOnMask hinreichend groß zu dimensionieren.
- Der Text einer Paintarea wird ebenfalls übernommen. Dabei kann nicht festgestellt werden, ob dieser Text von einer Übersetzungsfunktion stammt oder beim Anlegen der Paintarea definiert wurde. Das hat zur Folge, dass die Übersetzungsfunktion beim zweiten Anlegen der Paintarea nicht nochmals aufgerufen wird.
- Die als ptrval1 und ptrval2 angegebenen Werte gehen verloren, da die Datenstruktur zur Layoutbeschreibungsdatei keine Speicherflächen für diese Werte besitzt.

8 Objektüberschriften

Zur Darstellung von Objektüberschriften in Proportionalschrift werden Paintareas verwendet. Dazu ist an der Adresse

```
void (*S_GetHeadlinePaintareas)(obj *objp, unsigned long *h_pa);
```

die Adresse einer Funktion zu hinterlegen. Die Funktion wird dann beim Erzeugen von Paintareas für ein Objekt aufgerufen. Sie hat die Aufgabe, zwei Paintareas zu erzeugen und deren Handles in h_pa[0] und h_pa[1] zu hinterlegen. In h_pa[0] wird die Überschrift dargestellt und in h_pa[1] wird der Maskenzustand (Holen, Korrektur, Neuanlage, ...) oder der Satzanzeiger dargestellt. Wenn eine der Paintareas nicht definiert wird, dann stellt FIX die entsprechende Überschrift wie bisher (ohne Proportionalschrift) dar.

Abgesehen von der Schrift bestehen folgende Unterschiede:

- Die Paintarea ist im Gegensatz zum Text in nicht-Proportionalschrift immer sichtbar.
- Die Darstellung der Paintarea kann frei (durch Programmierung in der fwown.dll) definiert werden.
- Die Position kann selbst bestimmt werden.

Bei Choices ist es notwendig, dass die Choice ein (leeres) Layout besitzt. Ansonsten können keine Paintareas gezeichnet werden.

8.1 Beispiel für eine Funktion:

```
void getHeadlinePaintareas(obj *objp, unsigned long *h_pa)
{
      int cols_rowcount;


      if (objp->ob_box.bx_headline && *objp->ob_box.bx_headline) {
            h_pa[0] = pa_put_textlabel(objp, 0, 8, objp->ob_box.bx_headline,
                  objp->ob_box.bx_headline, NULL,
                  UNDERLINE, PA_ALGN_CENTER, PA_BT_NONE,
```

```
                0L, 0L, 0L, 0L, NULL, NULL);
    }
    cols_rowcount = 0;

    switch(objp->ob_class) {
    case SUBMASK:
    case ROLLMASK:
    case TABLE:
        if (((submask*)objp)->sm_state & SM_ROWCOUNT) {
            cols_rowcount = 10;
        }
        break;
    case MASK:
    case CHOICE:
        cols_rowcount = 16;
        break;
    }

    if (cols_rowcount > 0) {
        int indent, x_offset;
        char token[20];

        indent = MIN(objp->ob_box.bx_cols, cols_rowcount);
        x_offset = objp->ob_box.bx_cols - indent - 4;
        sprintf(token, "ROWCOUNT          ");
        token[indent] = '\0';
        h_pa[1] = pa_put_textlabel(objp, 0, x_offset, token,
            "", NULL,
            UNDERLINE, PA_ALGN_CENTER, PA_BT_NONE,
            0L, 0L, 0L, 0L, NULL, NULL);
    }

}
```

Die Funktion legt nur eine Paintarea für die Überschrift an, wenn das Objekt eine Überschrift besitzt. Ansonsten wäre eine leere Paintarea sichtbar, die nicht gefüllt wird.

Für die Paintarea der Satzanzeige b.z.w. für die des Objektzustandes wird zuerst eine Länge berechnet. Entsprechend dieser Länge wird die Paintarea erzeugt. Wenn die Länge 0 ist (z.B. bei einem Selo oder einer Tabelle ohne Satzanzeige) dann unterbleibt das Erzeugen der Paintarea.

Objektspezifische Tastenbelegung

Durch die Angabe einer oder mehrerer Objektklassen zu einer Taste der Tastenbelegung (stdkey.fix) können Tasten in verschiedenen Objekten unterschiedlich belegt werden. Wird die Angabe weggelassen, dann gilt die Belegung für alle Objektklassen.

Folgende Bezeichner sind als Objektklasse zulässig:

```
MASK
SUBMASK
ROLLMASK
TABLE
SELO
CHOICE
MENUE
HELPTEXT
```

Wenn mehrere Objektklassen für eine Taste definiert werden, dann sind diese durch Leerraum (Space, Tab) zu trennen.

Beispiel:

```
F9       f9       g9     --      h9
F6       f9       g9     --      h9    MASK TABLE
```

Für die Objektklassen MASK und TABLE muss die Taste F6 gedrückt werden, um das Event f9 (Auswahl, Selo) zu senden. Für alle anderen Objekte ist die Taste F9 zu verwenden.

Wie bei dem alten Format gilt hier, dass weiter unten stehende Definitionen die oberen Definitionen überschreiben. Das betrifft sowohl die Belegung einer Taste als auch die Festlegung des Bitmaps für das Tastenlabel. Das kann zu Problemen führen, wie das folgende Beispiel zeigt (die Nummern vorne stehen nicht in der Datei. Sie enthalten nur die Zeilennummern. Die Belegung für die Modifier Shift, Ctrl und Shift-Ctrl wurde weggelassen).

```
1  F6       f6
2  F6       f9  MASK TABLE
3  F9       f9
4  F9       f6    MASK TABLE
```

Die Belegung der Tasten funktioniert in diesem Beispiel. Jedoch beim Bestimmen der Bitmaps für die Tasten-Labels kommt es zu einem Problem. Zeile 2 definiert für die Objektklassen MASK und TABLE das Bitmap F6_N.bmp. Diese Definition wird von Zeile 3 jedoch durch F9_N.bmp überschrieben, obwohl die Taste für das Event f9 in einer Maske oder Tabelle F6 ist.

Um das Problem zu lösen, könnten alle Definitionen mit Objektklassen an das Ende der Datei geschrieben werden. Da es jedoch übersichtlich ist, die Definitionen zu einer Taste untereinander zu schreiben, führt FIX/Win diese Sortierung durch. Dazu wird die Datei in zwei Pässen gelesen. Der erste Pass ließt nur Angaben ohne Objektklasse, der zweite Pass ließt nur Angaben mit Objektklasse. Dadurch überschreiben die spezielleren Angaben (mit Objektklasse) die allgemeinen Angaben. Ansonsten bleibt die in der Datei definierte Reihenfolge erhalten.

Zum Test kann FIX/Win mit dem Schalter /diag 16 gestartet werden. Neben anderen Daten des LookAndFeels werden dann in der Datei die Tastenbelegungen und die Tasten-Labels für jedes Objekt gesondert ausgegeben. Dabei wird auch getestet, ob das entsprechende Bitmap für das Tasten-Label vorhanden ist.

Damit ist bis auf eine Ausnahme eine Windows-konforme Tastenbelegung möglich. Da die Taste SPACE (Leertaste) nicht in die Tastenbelegung eingetragen werden kann, ist es nicht möglich, mit der Leertaste einen Choice-Eintrag zu markieren. Das Erweitern der Tastentabelle um SPACE ist zwar prinzipiell möglich, jedoch ist damit zum einen das Belegen von SPACE auch in anderen Fällen möglich und zum anderen wird bei jeder Texteingabe die Tastaturbelegung durchsucht. Deshalb wird ist das Belegen der Leertaste durch den Eintrag

```
MarkChoiceEntryWithSpace: TRUE
```

in der Datei fix.rc zu konfigurieren. Damit ist das Markieren eines Eintrags einer Choice mit der Leertaste möglich. Für die Textsuche steht die Leertaste jedoch nicht mehr zur Verfügung. Die Taste Return kann weiterhin zum Markieren verwendet werden, wenn sie nicht, wie unter Windows üblich, mit EN zum

bestätigenden Verlassen der Choice belegt wird.

Damit lassen sich Choices bis auf eine Ausnahme ähnlich bedienen wie Selos. Die Ausnahme besteht darin, dass bei einem Selo immer der aktuelle Satz markiert wird. Bei Choices, bei denen mehr als ein Eintrag markiert werden kann, ist das unsinnig. Bei Choices, bei denen genau ein Satz markiert werden kann (range 1 1), ist das jedoch sinnvoll. Durch das Setzen der Ressource

```
AutoMarkSingleChoice: TRUE
```

kann diese Art der Markierung eingeschaltet werden. Die Angabe auto wird in diesem Fall ignoriert, da ansonsten bei dem ersten Verändern des aktuellen Satzes die Choice verlassen würde. Das Umschalten der Markierung mit Return oder der Leertaste wird ebenfalls unterbunden.

0.1 Beispiel einer Windows-ähnlichen Tastenbelegung

```
# Taste  Normal  Shift  Ctrl   Ctrl
#                             Shift

END      EN       --     --     --
END      PD       --     --     --    MENUE
END      f0       --      --        --       SELO
END      f0       --      --        --       CHOICE
HOME     kh       --      --        --
HOME     f8       --      --        -- SELO
HOME     f8       --      --        -- CHOICE
ESCAPE   EN       --     --     --
ESCAPE   PL       --     --     --  CHOICE
INSERT   CI       --      --        --
DELETE   CD       --      --        --
LEFT     kl       PL      --        --
RIGHT    kr       PR     --         --
UP       ku       --     --         --
DOWN     kd       --     --         --
PRIOR    PU       --     --         --
NEXT     PD       --     --         --
BACK     DL       --     --     --
RETURN   RT       --     --     --
RETURN   EN       --     --     --  CHOICE
TAB      TB       BT     --     --
TAB      RT       PL     --     --  MASK ROLLMASK SUBMASK TABLE
```

END sendet in allen Masken weiterhin das Event EN. Diese Taste wird zwar von den Eingabefeldern als Infield-Editing-Taste verwendet. Doch durch die Definition als "smart"-Taste ist sie immer noch für die Bedienung von Masken verwendbar. Ansonsten kann ESCAPE zum Verlassen eines Objektes verwendet werden. Eine Ausnahme bildet die Choice. Hier ist EN ein bestätigendes Verlassen, bei dem der ausgewählte Wert in das Feld übernommen wird. Deshalb wird für eine Choice beim Drücken von Escape das Event PL und beim Drücken von Return das Event EN gesendet.

Da HOME ebenfalls eine "smart"-Taste ist, kann sie ebenfalls für das Navigieren in Masken weiterverwendet werden.

Für Menüs, Selos und Choices wurden die Tasten HOME und END umdefiniert. Sie springen jetzt - wie unter Windows erwartet - zum ersten (Event f8) b.z.w letzten (Event f0) Element. Da das Event kh in Menüs zum ersten Punkt springt, ist eine Umdefinition von HOME für Menüs nicht unbedingt notwendig.

Um in Masken mit TAB navigieren zu können, wurden dieser Taste für MASK ROLLMASK SUBMASK und TABLE die Events RT und PL zugeordnet. In allen anderen Objekten wird weiterhin TB und BT gesendet.

Alle anderen Belegungen wurden beibehalten. Dies betrifft auch die Funktionstasten, die hier nicht aufgeführt sind, weil sie meistens anwendungsspezifisch belegt sind.

Streng genommen müsste das Verlassen einer Maske mit END (Senden von EN) unterbunden werden. Dass durch das Drücken von HOME der Eingabefokus in das erste Feld zurückkehrt, kann ein Windowsbenutzer leicht nachvollziehen. Jedoch erwartet er, dass beim Drücken von END, das letzte Feld der Maske besucht wird, wofür es jedoch kein entsprechendes Event in FIX gibt. In dieser Belegung wird mit Rücksicht auf die

FIX-Benutzer, die diese Belegung gewohnt sind, END weiterhin zum Verlassen einer Maske erlaubt.

1 Umsetzen von Tasten in FIX

Da mit einer objektspezifischen Tastaturbelegung nicht alle Probleme gelöst werden können, besteht die Möglichkeit, Tasten unmittelbar nach dem Lesen in FIX umzusetzen. Der Vorteil dieser Möglichkeit gegenüber einer Umsetzung in FIX/Win besteht darin, dass auf FIX-Seite wesentlich mehr Informationen zur Verfügung stehen, die ausgewertet werden können.

Um eine Umsetzung zu implementieren ist an der Adresse

```
int   (*S_map_key)(int ev, obj *objp, char **reason);
```

eine Funktion zu hinterlegen. Diese Funktion wird unmittelbar nach dem Lesen eines Events, dass über Tastatur eingegeben wurde, aufgerufen. Sie bekommt als Parameter das eingelesene Event ev, das aktuelle Objekt objp (oder NULL, wenn es keines gibt) und einen Zeiger auf eine char* Variable.

Wenn die Anwendung, das Event umdefinieren möchte, dann kann sie ein anderes Event zurückgeben. An der Adresse reason sollte in diesem Fall ein Zeiger auf einen kurzen Text hinterlegt werden, der den Grund für das Umsetzen des Events beschreibt.

Wenn die Anwendung mit dem Schalter -test 65536 gestartet wird, dann werden sämtliche Umsetzungen zusammen mit den in reason abgelegten Texten ausgegeben. Man hat somit die Möglichkeit, nachzuvollziehen, warum sich die Tastenbedienung an der einen Stelle anderes verhält als an der anderen.

1.1 Beispiel

Die Taste Esc soll nicht immer das gleiche bewirken wie Ende. In eingebetteten Masken soll sie ohne Wirkung sein. Nur in der Hauptmaske soll sie K_EN senden.

```
int map_key(ev, objp, reason)
int ev; obj* objp; char **reason;
{
      int new_ev;

      new_ev = ev;
      if (objp == NULL) {
            return(new_ev);
      }
      switch (objp->ob_class) {
            case MASK:
            case SUBMASK:
            case ROLLMASK:
            case TABLE:
                  switch (ev) {
                  case K_EC:
                        if (objp->ob_parent == NULL) {
                              *reason = "Esc in mask with no parent";
                              new_ev= K_EN;
                        }
                  }
                  break;
            }
            break;
      }
      return new_ev;
}
S_map_key = map_key;
```

Die Funktion prüft, ob es ein aktuelles Objekt gibt. Wenn nicht, dann findet keine Umsetzung des Events statt. Wenn ja, dann wird geprüft, ob es sich um eine Maske handelt und ob das Event K_EV eingelesen wurde. Wenn die Maske dann kein Vaterobjekt besitzt, dann wird K_EN als neues Event zurückgeliefert. Das Beispiel setzt voraus, dass Esc auf FIX/Win Seite statt auf K_EN auf K_EC gesetzt wurde.

Erweiterung des led

Durch die Möglichkeit, Paintareas in der pan-Datei zu definieren, war es notwendig den led zu überarbeiten. Dabei wurden einige Erweiterungen vorgenommen:

- Anlegen und Bearbeiten von Paintareas im WYSIWYG-Modus
- Bearbeiten von Feldeigenschaften im WYSIWYG-Modus
- Suchfunktion für Felder und Paintareas
- Suchfunktion für Beschreibungsdateien
- UNDO-Funktionalität

1 Anlegen und Bearbeiten von Paintareas

Bearbeiten/Neuanlegen

Mittels

```
f6
```

im WYSIWYG-Modus wird eine Maske zur Bearbeitung einer Paintarea gestartet. Wenn sich unter der Schreibmarke eine Paintarea befindet, dann wird vorher diese Paintarea als aktuelle Paintarea markiert und die Daten können in der Maske bearbeitet werden. Befindet sich unter der Schreibmarke keine Paintarea, dann wird eine neue Paintarea angelegt. In der Maske zur Bearbeitung der Paintarea kann mit den Tasten PU und PD zu anderen Paintareas geblättert werden. Die Änderungen der aktuellen Paintarea werden dabei übernommen.

Die Funktion zum Editieren kann auch über das Kontextmenü ausgelöst werden. In diesem Fall wird die Schreibmarke an die aktuelle Mausposition positioniert und die Paintarea an dieser Position wird zur aktuellen Paintarea.

nderungen in der Maske werden übernommen, wenn diese nach untern verlassen wird. Wird die Maske nach oben verlassen, dann werden die Änderungen verworfen. Beim Blättern auf einen anderen Satz (K_PU, K_PD) werden die Änderungen des aktuellen Satzes übernommen.

Statt f6 zu verwenden, kann eine Paintarea auch mit der linken Maustaste zur aktuellen Paintarea gemacht werden und mit einem zweiten Klick (Doppelklick) zur Beabeitung geladen werden.

Löschen

Mittels

```
f7
```

im WYSIWYG-Modus wird die Paintarea unter der Schreibmarke zur aktuellen Paintarea. Danach wird die aktuelle Paintarea nach einer Sicherheitsabfrage gelöscht. Befindet sich unter der Schreibmarke keine Paintarea (und auch kein Feld - siehe nächster Abschnitt), bleibt das Event ohne Wirkung.

Wenn die Funktion über das Kontextmenü ausgelöst wird, dann wird die Schreibmarke an die aktuelle Mausposition verschoben. Somit wird die Paintarea unter der aktuellen Mausposition gelöscht.

Verschieben

Durch Anklicken mit der linken Maustaste wird eine Paintarea zur aktuellen Paintarea. Sie wird dann durch das Attribut, das in der Ressource

```
ActivePaintareaAttr
```

definiert wird, hervorgehoben. Die Ressource muss einen der numerischen Werte aus video.h enthalten. Wird die Ressource nicht angegeben, dann wird der Wert für INVERS verwendet.

Mittels

```
PT
```

wird die aktuelle Paintarea an die Position der Schreibmarke verschoben. Wenn diese Funktion durch das Kontextmenü ausgelöst wird, dann wird die Schreibmarke vorher an die aktuelle Mausposition verschoben.

Die alte Funktion der Taste PT (Erstellen einer Hardcopy) entfällt.

Paintarea aus-/einblenden

Mittels

```
f0
```

können Paintareas aus- und auch wieder eingeblendet werden. Das kann sinvoll sein, um Überlappungen mit Feldern zu erkennen.

2 Bearbeiten von Feldeigenschaften

hnlich wie Paintareas können auch Felder im WYSIWYG-Modus bearbeitet werden. Zur Bearbeitung der Feldeigenschaften wird eine ähnliche Maske verwendet, wie sie auch in der Detailansicht der Feldbearbeitung verwendet wird. Die Daten werden in dieser Maske nur etwas komprimierter dargestellt.

Zur Bearbeitung von Feldern werden teilweise die gleichen Events verwendet. Das bedeutet, dass mit einem Doppelklick oder mit f6 ein Feld bearbeitet werden kann, wenn statt einer Paintarea ein Feld an der entsprechenden Position liegt. Analog dazu löscht f7 ein Feld, wenn ein Feld an der Position liegt.

Da es ein aktuelles Feld und eine aktuelle Paintarea geben kann, ist zum Verschieben allerdings ein anderes Event notwendig: ST (keine Änderung gegenüber vorherigen Versionen).

Zum Ein- und Ausblenden der Felder ist wie auch schon in vorherigen Versionen von FIX f8 zu verwenden.

3 Suchfunktion für Felder und Paintareas

Durch Drücken der Taste

```
h9
```

kann eine Maske zum Suchen von Feldern und Paintareas eingeblendet werden. Die Maske enthält eine

Tabelle mit allen Paintareas und Feldern des aktuellen Objekts. Durch Doppelklick oder Return auf einer Tabellenzeile kann das entsprechende Element zum aktuellen gemacht werden. Durch die Eingabe eines Wertes im Feld "Suchen" kann ein Ausdruck angegeben werden, der im Namen des Elements enthalten sein muss. Damit wird die Menge der Zeilen der Tabelle eingeschränkt. Hier sind auch folgende Wildcards erlaubt:

- "?" steht für ein beliebiges Zeichen
- "*" steht für eine beliebige Anzahl Zeichen
- "^" steht für den Zeilenanfang
- "$" steht für das Zeilenende

Zwischen Groß- und Kleinschreibung wird bei der Suche nicht unterschieden.

Zusätzlich kann die Menge der Treffer über die Auswahl des Typs ("Field" oder "Paintarea") eingeschränkt werden.

Die Suchmaske kann nicht nur im WYSIWYG-Modus benutzt werden. Sie kann auch über h9 während der Bearbeitung in der Maske "Feld-Daten" oder in der Übersicht aufgerufen werden. In diesem Fall ist es jedoch nicht möglich (und auch nicht sinnvoll) eine Paintarea auszuwählen.

4 Suchfunktion für Beschreibungsdateien

hnlich wie die Suche nach Feldern und Paintareas funktioniert die Suche nach einer Datei, die mit led bearbeitet werden soll. Diese Art der Suche ersetzt den bisherigen Mechanismus der Dateiauswahl. Wenn er verwendet werden soll, dann ist die Ressource

```
led.load2: TRUE
```

In diesem Fall kann statt des Dateinamens im Feld "Datei" ein Wert angegeben werden, der ein Suchmuster darstellt. Alle Dateien, die dieses Muster enthalten werden in der unteren Tabelle angezeigt. Wie bei den Feldern darf das Muster auch die Wildcards "?", "*", "^" und "$" enthalten (siehe oben). Durch Doppelklick auf eine Tabellenzeile oder durch das Drücken von RT wird die entsprechende Datei zur Bearbeitung geladen.

5 Änderung der Tastenbelegung

Wenn led über FIX/Win gestartet wird - und nur das ist beim Arbeiten mit Paintareas sinnvoll, dann arbeitet FIX/Win in der Regel mit dem Gruppenverzeichnis der Anwendung, damit Bitmaps im led auch so dargestellt werden, wie sie später in der Anwendung aussehen. Das Gruppenverzeichnis definiert jedoch auch die Tastenbelegung. Sollte diese passend zur Eingabe über Edit Controls ausgelegt worden sein, dann sollte auch led Edit Controls verwenden. Dazu ist in der Datei fix.rc der Eintrag

```
led.UseFieldAreas: TRUE
```

Auf die Verwendung von Proportionalschrift wurde in led vorerst verzichtet. Alle Felder besitzen das Attribut FIXEDFONT. Grund hierfür ist, dass bei der Verwendung von Proportionalschrift auch die Beschriftungen auf Proportionalschrift umgestellt werden müssen und dadurch viel Leerraum in den Masken entsteht. Das erfordert eine Überarbeitung des Maskenlayouts und ein Verkürzen der Felder und Bezeichner. Dadurch wird jedoch led in der Terminal-Version unbenutzbar.

6 UNDO-Funktionalität

Die UNDO-Funktionalität beschränkt sich auf den WYSIWYG-Modus. Sie funktioniert auf folgende Art und Weise:

Vor bestimmten Operationen werden die Daten in Backupdateien gesichert. Dabei wird jeweils eine mfo-Datei mit der dazugehörigen pan-Datei gesichert. Intern wird ein Zähler mitgeführt, der einen Teil des Dateinamens bildet und vor jedem Schreiben hochgezählt wird.

Durch das Tasten-Event

```
K_AB
```

kann ein UNDO-Vorgang ausgelöst werden. Dazu wird die zu dem Zähler passende Datei wieder eingelesen und der Zähler wird erniedrigt.

Zum Aktivieren der UNDO-Funktionalität muss ein Verzeichnis definiert werden, in dem die Dateien abgelegt werden. Dazu ist in der Umgebungsvariablen

```
LED_UNDODIR
```

der Pfad zu einem Verzeichnis zu hinterlegen. Dabei sollten folgende Punkte beachtet werden:

- Das Verzeichnis muss existieren und für den Benutzer schreibbar sein.
- Wenn in dem Verzeichnis Dateien vorhanden sind, dann werden diese ohne Vorwarnung bei der Bearbeitung eines anderen Objektes überschrieben.
- led löscht keine Dateien. Der Anwender ist selbst dafür verantwortlich, die Dateien bei Bedarf zu löschen. Die Dateien können auch als Sicherheitskopie verwendet werden. Die Dateien XXX@nnnn@1.mfo/pan enthalten beispielsweise den Stand bei Beginn des Editiervorgangs. Durch das Umbenennen und das Anpassen der pan-Referenz in der mfo-Datei kann dieser Stand (oder ein späterer) auch noch nach dem Bearbeiten restauriert werden.

Als Format für die Dateien wird

```
<Basisname>@<Prozessnummer>@<Undolevel>.pan/mfo
```

verwendet.

Beim Arbeiten mit der UNDO-Funktionalität ist zu beachten, dass nur Daten gesichert werden und nicht der Zustand des led. So bleiben die aktuellen Bildschirmattributte und die Position der Schreibmarke auch nach einem UNDO erhalten und werden nicht in einen vorherigen Zustand zurückgesetzt.

led erzeugt vor folgenden Operationen eine UNDO-Datei:

- Vor dem Eingeben des ersten Zeichens einer Zeichenkette. Ein Zeichen gilt dann als zugehörig zu der Zeichenkette, wenn vor der Eingabe dessen nicht die Schreibmarke bewegt wurde oder eine andere Operation ausgelöst wurde. Beispiel: Bei der Eingabe von "Das<Pfeil nach unten>ist<Pfeil nach unten>ein<Pfeil nach unten>Test" wird jeweils beim ersten Buchstaben jedes Wortes eine UNDO-Datei erzeugt. Wird ein UNDO über K_AB ausgelöst, dann wird jeweils das letzte Wort entfernt.
- Vor dem Löschen des ersten Zeichens von mehreren Zeichen. Wird unmittelbar nach dem Löschen eines Zeichens ein weiteres gelöscht, dann wird keine neue UNDO-Datei geschrieben.
- Vor dem Einfügen des ersten Leerzeichens. Werden unmittelbar danach weitere Leerzeichen eingefügt, dann wird keine neue UNDO-Datei geschrieben.
- Vor dem Löschen einer Zeile.
- Vor dem Einfügen einer Zeile.
- Vor der ersten Operation, die das Objekt in der Größe oder der Position ändert. Folgen dieser Operation weitere Operationen zur Änderung der Größe oder der Positionen, wird keine neue UNDO-Datei geschrieben. Erst das Verlassen und erneute Betreten des POSIT-Modes führt zum erneuten Erzeugen einer UNDO-Datei, bei einer Größen-/Positionsänderung.
- Vor jedem Ändern von Feld- oder Paintarea-Daten, über eine der Masken, die aus dem WYSIWYG-Modus (K_f6, Doppelklick) gestartet werden können. Wird in diesen Masken auf ein anderes Feld (eine andere Paintarea) geblättert, wird vor dessen Änderung ebenfalls eine UNDO-Datei erzeugt.
- Vor dem Löschen eines Feldes oder einer Paintareas.
- Vor dem Umpositionieren eines Feldes oder einer Paintareas.

6.1 Achtung:

Beim Auslösen der UNDO-Funktionalität werden auch Änderungen rückgängig gemacht, die nach dem Erzeugen der letzten UNDO-Datei außerhalb des WYSIWYG-Modes gemacht wurden.

Beispiel

Es wird eine Maske bearbeitet. Im WYSIWYG-Modus wird ein Text eingegeben. Dadurch wird eine UNDO-Datei erzeugt. Danach wird der WYSIWYG-Modus verlassen und in den Maskendaten wird der Titel verändert. Beim nächsten Betreten des WYSIWYG-Modus wird die UNDO-Funktionalität durch das Drücken der Taste K_AB ausgelöst. Dadurch wird in diesem Fall nicht nur die Änderung des Textes, sondern auch die Änderung des Maskentitels rückgängig gemacht, da diese Änderung nicht in der UNDO-Datei steht.

Limitierung des Zeichenvorrats für FXCHARTYPE-Werte

Wird eine Anwendung, die mit "FIX mit Unicode-Unterstützung" erstellt ist, mit einer Nicht-Unicode-Datenbank eingesetzt, können CHARACTER-Werte mit Zeichen auftreten, die nicht im Zeichensatz der Datenbank enthalten sind. Während "fehlende" Zeichen im SQL-Anweisungstext - etwa einem String-Literal - bei der Ausführung der Anweisung zu einem SQL-Fehler

```
x202: An illegal character has been found in the statement.
```

führen, ist bei Hostvariablen höchste Vorsicht geboten.

Nehmen wir an, die Datenbank benutzt das Locale en_us.8859-15 und die Anwendung verwendet für DB_LOCALE "en_us.8859-15" und für CLIENT_LOCALE "en_us.utf8".

FXCHARTYPE-Werte können beliebige Zeichen der Basic Multilingual Plane von Unicode enthalten, sofern sie darstellbar sind, also neben "exotischen" Zeichen auch solche wie "Currency Sign" (U+A4), "Broken Bar" (U+A6), "Diaeresis" (U+A8), "Acute Accent" (U+B4), "Cedilla" (U+B8), "Fraction One Quarter" (U+BC), "Fraction One Half" (U+BD) und "Fraction Three Quarters" (U+BE) aus dem "Latin-1"-Block oder die in Osteuropa gebräuchlichen Zeichen des Blocks "Latin Extended-A".

Wird nun einer INSERT- oder UPDATE-Anweisung ein derartiger Wert übergeben, so ersetzt das Datenbank-API aus Datenbanksicht ungültige Zeichen durch ein Ersatzzeichen - hier das (ASCII-)Zeichen U+1A - und übergibt den so entstehenden Wert an die Datenbank. Diese Modifikation bewirkt seitens der Datenbankschnittstelle weder einen Fehler noch eine Warnung. Dasselbe gilt bei der Verwendung von Hostvariablen in where-Klauseln.

Um diese Problematik zu entschärfen, erlaubt es FIX, den Zeichenvorrat für FXCHARTYPE-Werte zu limitieren. Dazu muss in der Umgebungsvariable FXDATACHARFILE der Pfad einer Datei hinterlegt werden, die die zulässige Untermenge der BMP-Zeichen definiert. Sie hat folgenden Aufbau:

```
<definitions> ::= <definition>*

<definition> ::= ( <code range> | <singular code> ) ','{0,1}

<code range> ::= <singular code> '-' <singular code>

<singular code> ::= <dec. digit>+ | 0<octal digit>+ | 0x<hex. digit>+
```

Zwischen den Token kann beliebig viel Leerraum stehen und mittels "#" oder "//" können sich bis zum Zeilenende erstreckende Kommentare angegeben werden. Der numerische Wert eines <singular code> darf 0xFFFF nicht übersteigen.

Für die Untermenge gelten folgende Bedingungen:

- Sie muss die Codes U+20 bis U+7E umfassen; diese brauchen nicht explizit angegeben zu werden, da FIX sie beim Einlesen der Datei automatisch aufnimmt.
- Sie darf höchstens 256 Codes enthalten.

Eine weitere sinnvolle Einschränkung wird nicht validiert:

- Mit einem Großbuchstaben sollte auch der entsprechende Kleinbuchstabe enthalten sein.

FIX enthält im Verzeichnis etc/unicode drei exemplarische "Datachar Definition Files":

- codes.CP437: spezifiziert die Codes jener 224 Zeichen, die in der Code Page 437 enthalten sind (und keine Steuerzeichen sind),
- codes.ISO8859-15: spezifiziert die Codes jener 191 Zeichen, die in ISO 8859-15 enthalten sind,
- codes.ISO8859-2: spezifiziert die Codes jener 191 Zeichen, die in ISO/IEC 8859-2 enthalten sind.

FIX wertet die Umgebungsvariable FXDATACHARFILE bei der Programminitialisierung aus und liest ggf. die angegebene Datei ein. Aus den Angaben wird eine Datenbasis für die Funktion

```
int fxisdatachar(int code)
```

erzeugt. Diese Funktion wird überall dort (zusätzlich) aufgerufen, wo Zeichen von FXCHARTYPE-Werten oder für sie bestimmte Zeichen auf Gültigkeit, d.h. bislang Darstellbarkeit, geprüft wurden. Solche Situationen sind

insbesondere

- das Ablegen von Werten in FXCHARTYPE-Feldern und FXCHARTYPE-Spalten von Memory Relations,
- die Bestimmung des Literals zu einem FXCHARTYPE-Wert,
- die Bestimmung der formatierten Darstellung eines FXCHARTYPE-Werts,
- die Eingabe von Zeichen in FXCHARTYPE-Felder.

FIX verhält sich für Zeichen c, die fxisdatachar(c) verletzen, in der gleichen Weise wie bei Verletzung von fxisprint(c).

FIX ersetzte bislang unzulässige UTF-8-Sequenzen durch das Zeichen U+FFFD. Bei FXCHARTYPE-Werten wird nun, wenn U+FFFD von fxisdatachar() nicht akzeptiert wird, '?' verwendet.

Bei Einsatz eines grafischen Frontends werden die akzeptierten Zeichen an das Frontend übermittelt, damit es die Limitierung bei der Feldeingabe berücksichtigen kann.

Möglichkeit zur Reaktion bei Nichttätigkeit

Durch Setzen der Umgebungsvariablen

FXUSERIDLEFACTOR

besteht die Möglichkeit, nach einer bestimmten Zeit ohne Benutzereingaben eine Funktion der Anwendung aufzurufen, deren Adresse in der Variablen

S_user_is_idle

hinterlegt wird. Die Funktion hat somit die Möglichkeit, die Anwendung abzubrechen. Wird in der Variablen keine Adresse hinterlegt, dann wird keine Funktion aufgerufen und der Mechanismus bleibt wirkungslos.

FXUSERIDLEFACTOR ist aus technischen Gründen nicht als Zeit in Sekunden anzugeben. Stattdessen ist ein Faktor zu dem Wert FXREADTIMEOUT0 anzugeben.

FXREADTIMEOUT0 definiert die Zeit in Sekunden, nach der die Anwendung ein Lebenszeichen vom Frontend anfordert. Nach dieser Zeit wird die Leseroutine unterbrochen und ein Lebenszeichen angefordert - es sei denn die Leseroutine beendet sich vorher, weil ein Zeichen gelesen wurde.

Beispiel

Startscript:

```
...

FXREADTIMEOUT0=60
FXREADTIMEOUT1=120
FXUSERIDLEFACTOR=5

export FXREADTIMEOUT0 FXREADTIMEOUT1 FXUSERIDLEFACTOR

# starte Anwendung

...
```

anwendung.c

```
void my_user_idle()
{
      fprintf(stderr, "User is idle\n");
}

main()
{
      ...

      S_user_is_idle = my_user_idle;

      ...
}
```

Nach FXREADTIMEOUT0 - also nach 60 Sekunden - wird ein Lebenszeichen vom Frontend angefordert. Dieses muss innerhalb von FXREADTIMEOUT1 - also nach 120 Sekunden - eintreffen. Sonst wird die Anwendung abgebrochen.

Wenn 5 mal (FXUSERIDLEFACTOR) unmittelbar hintereinander ein Lebenszeichen angeforder wurde - also nach FXUSERIDLEFACTOR * FXREADTIMEOUT0 = 5 * 60 = 5 min - wird die Funktion my_user_idle() aufgerufen. In diesem Beispiel wird von dieser Funktion nur eine Meldung ausgegeben.

Wenn innerhalb dieser Zeit eine Taste gedrückt wird, dann wird der interne Zähler zurückgesetzt und die Zeit beginnt von neuem. my_user_idle() wird also 5 min nach dem letzten Tastendruck aufgerufen.

Probleme

Die Eingaben in Edit Controls der FIX Version ab 4.0.0 werden nicht sofort zu FIX gesendet. Erst wenn das

Feld verlassen wird, werden Events zu FIX gesendet. Wenn also ein Benutzer für die Zeit von FXREADTIMEOUT0 in ein und dem selben Feld verweilt, dann wird von FIX ein Lebenszeichen angefordert. Das ist nicht problematisch, da FIX/Win das Lebenszeichen in der Regel sendet. Wenn der Benutzer jedoch für die Zeit FXUSERIDLEFACTOR * FXREADTIMEOUT0 im gleichen Feld verweilt, dann wird die Funktion S_user_is_idle aufgerufen - unabhängig davon, ob der Benutzer im Feld tippt oder untätig ist.

Aus diesem Grund sollte der Wert FXUSERIDLEFACTOR * FXREADTIMEOUT0 so groß sein, dass der Benutzer innerhalb dieser Zeit ohne Probleme einen Feldwert erfassen kann.