Applikationsgenerator *FIX*

Release 3.1.0 Unicode

Vorabdokumentation

Dokumentversion: Draft 1, 06.03.2008

Nonne & Schneider Informationssysteme GmbH
Friedrich-List-Str. 31
35398 Gießen
Tel.: 0641/97477-0
Fax: 0641/97477-77

Im Zuge der Weiterentwicklung des Produkts *FIX* können Leistungsmerkmale hinzukommen, verändert werden oder entfallen.

1 Überblick

Ab der Version 3.1.0 ist *FIX* in der Lage, alternativ Zeichen aus dem Unicode-Zeichensatz zu verarbeiten. Es existieren zwei verschiedene Versionen von *FIX*, die unabhängig voneinander installiert werden können:

- *FIX 3.1.0* - *FIX* ohne Unicode-Unterstützung
- *FIX 3.1.0 Unicode* - *FIX* mit Unicode-Unterstützung

Die im Folgenden beschriebenen Änderungen beziehen sich vorwiegend auf die Version *FIX 3.1.0 Unicode*. Wenn sich eine Änderung auch auf die Version *FIX 3.1.0* bezieht, wird dies entweder im Text explizit erwähnt oder der Text ist mit dem Zeichen ▲ markiert.

Allgemein gelten für *FIX 3.1.0 Unicode* folgende Punkte:

- *FIX* 3.*1.0 Unicode* unterstützt Zeichen mit Codes zwischen 0 und 65535 entsprechend ISO/IEC 10646-1:1993 oder Unicode 1.1 (Basic Multilingual Plane).

  Zu den Bezeichnungen vgl. http://www.unicode.org/versions/Unicode4.0.0/appC.pdf.

- Bei *FIX 3.1.0 Unicode* ist als Zeichensatzeinstellung UTF-8 zu verwenden.

  Anmerkung:

  UTF-8 (*8*-bit *U*nicode *T*ransformation *F*ormat) ist eigentlich die Bezeichnung für ein in RFC3629/STD 63 (2003) definiertes Multibyte-Kodierungsverfahren, mit dem die Zeichen des durch den Standard ISO/IEC 10646-1 definierten *U*niversal *C*haracter *S*ets (UCS) repräsentiert werden können.

  Wenn bei *FIX* vom "Zeichensatz" UTF-8 die Rede ist (vgl. Seite 14), meint dies "Zeichen aus ISO/IEC 10646-1:1993 (entspricht der *B*asic *M*ultilingual *P*lane des derzeit aktuellen Standards ISO/IEC 10646-3:2003) in der durch RFC3629/STD 63 (2003) definierten Repräsentation".

- *FIX 3.1.0 Unicode* bietet keine Unterstützung zur Konvertierung von Zeichen bei der Eingabe per Tastatur (→ fxkeymap) oder der Ausgabe am Bildschirm (→ fxcharmap). Das benutzte Terminal bzw. die Terminalemulation muss UTF-8-Zeichen produzieren und ausgeben können.
- *FIX 3.1.0 Unicode* wird ohne REP ausgeliefert.
- *FIX 3.1.0 Unicode* arbeitet - wie auch *FIX 3.1.0* - nur mit *FIX/Win 3.1.0* zusammen.

UTF-8 Locale

Programme, die mit *FIX 3.1.0 Unicode* entwickelt wurden, und viele Skripte und Hilfsprogramme müssen in einem UTF-8-basierten Locale ablaufen.

Tipps:

- Das Kommando **locale** zeigt die aktuellen Locale-Einstellungen an:

  ```
  LANG=…
  LC…=…
  ```

- Um die Bezeichnungen aller auf der Plattform verfügbaren (öffentlichen) Locales nachzuschlagen, kann i.d.R. das Kommando **locale -a** benutzt werden.

Für *FIX* selbst ist vor allem der Wert LC_CTYPE wichtig. Er muss die Form *language_territory*.UTF-8 (z.B. AIX) oder *language_territory*.utf8 (z.B. Linux) haben. Für andere Programme kann es jedoch wichtig sein, auch die anderen Bestandteile passend zu setzen.

So tritt beispielsweise unter Solaris ein Fehler auf, wenn das Kommando

```
ls -l x.mfo | awk '{print $3}'
```

ausgeführt wird und nur LC_CTYPE auf ein UTF-8-basiertes Locale gesetzt ist. Das Kommando soll den Eigentümer der Datei x.mfo ermitteln. Wenn das übrige Locale ISO-basiert ist, dann enthält die Ausgabe von ls -l ISO-Zeichen. awk erwartet jedoch aufgrund von LC_CTYPE UTF-8-kodierte Zeichen und produziert Fehler, sobald die Ausgabe ein Nicht-ASCII-Zeichen enthält. Dies kommt z.B. dann vor, wenn die Datei als Zeitangabe "März" enthält.

Terminalanpassung

Damit Zeichen korrekt eingegeben werden können und korrekt dargestellt werden, ist es notwendig, dass die verwendete Terminalemulation als Kodierung UTF-8 unterstützt. Da das Verfahren zur Einstellung uneinheitlich ist, können hier nur Hinweise gegeben werden, wie die Konsole von Windows zu konfigurieren ist:

- Als Zeichensatz ist "Lucida Console" einzustellen.
- Mittels

  ```
  mode con cp select=65001
  ```

  ist die Codepage der Console auf UTF-8 einzustellen.

Zusätzlich zur Konfiguration der Terminalemulation ist durch Setzen von FXTERM eine Terminalbeschreibung zu definieren, die über den Eintrag SG die Semigrafikzeichen für die Rahmen auf einfache ASCII-Zeichen abbildet. Für Windows wird dazu die Datei `nt.console.ascii` mitgeliefert. Unter UNIX kann ggf. die Datei `xterm.ascii` verwendet werden.

Mit einer so konfigurierten Terminalemulation ist die Bedienung von **fxm**, **led** und einer *FIX*-Anwendung grundsätzlich möglich. Es wird jedoch nicht garantiert, dass alle Zeichen aus UNICODE korrekt dargestellt und eingegeben werden können. Dies betrifft vor allem Fullwidth-Zeichen.

2 Installation

1 UNIX-Plattform

Besonderheiten

Die Shell (**sh**), die `installfix` ausführt, muss in einem UTF-8-basierten Locale ablaufen.[1]

`installfix` unterstützt die folgenden Installationsalternativen:

1. FIX-Runtime
2. FIX-Development
3. FIX-Development + FIX-Demo-Software

Die Frage nach dem “Zeichensatz fuer die FIX-Installation” unterbleibt.

Unterschiede im Umfang des Runtime-Systems

Paket T_run

- Die Lizenzerfassung (**fxlicense**) benutzt versionsspezifisch angepasste Dateien.
- Das Skript `cmd/lpcat` fehlt.
- Die Verzeichnisse `etc/ISO-15`, `etc/ISO-2`, `etc/IBM437`, `etc/GERMAN7` fehlen.
- Die Skripte `install/cs_extract` und `install/cs_showsets` und die Verzeichnisse `install/T_ISO-15`, `install/T_ISO-2`, `install/T_IBM437`, `install/T_GERMAN7` fehlen.
- Das Verzeichnis `de/hpd` enthält die Hilfetexte zu **fxlicense**.

Paket T_run.sys

- Das Programm `bin/lpcat` fehlt.
- Die Programme `bin/jahr`, `bin/monat` und `bin/tag` sind Kopien von und keine Links auf `bin/datum`.
- Das Programm `bin/tgoto` ist eine Kopie von und kein Link auf `bin/tcout`.

Unterschiede im Umfang des Entwicklungssystems

Paket T_dev

- Die Skripte `cmd/al`, `cmd/choice`, `cmd/edcmap`, `cmd/edkmap`, `cmd/fxindex`, `cmd/fxman`, `cmd/lconv`, `cmd/ml`, `cmd/mlylow`, `cmd/mlynorm`, `cmd/selo` und `cmd/vl` fehlen.
- Die Verzeichnisse `cho`, `hpd`, `men`, `mfo`, `pan`, `runtime/hpd`[2] und `sel` liegen jetzt unterhalb des neuen Verzeichnisses `de`.▲
- Die Dateien `fix_msg` und `messages` liegen jetzt im Verzeichnis `de` und im Verzeichnis `us`.▲

1. wegen der Aufrufe von **msgprep**.
2. von *FIX*-Tools benutzte Standard-Hilfetexte.

- Das Verzeichnis `tool_us` fehlt. Die Inhalte liegen jetzt im Verzeichnis `us`.▲
- Das Verzeichnis `doc` und die Datei `ERRATA` fehlen.
- Das Verzeichnis `old` fehlt.▲
- Es gibt ein zusätzliches Verzeichnis `extra` mit nützlichen Skripten für das UTF-8-Umfeld.

Paket T_dev.sys

- Die Programme `bin/ftlow`, `bin/ftmly`, `bin/ftnorm`, `bin/ftr`, `bin/mly2pan`[1], `bin/mlycheck` und `bin/pan2mly`[2] fehlen.

Binden der Tools (nur Entwicklungssystem)

Der **make**-Aufruf muss in einem UTF-8-basierten Locale ablaufen.

Lizenzerfassung

Die Shell (**sh**), die `licensefix` ausführt, muss in einem UTF-8-basierten Locale ablaufen. Damit Umlaute korrekt angezeigt werden, muss die Terminalemulation so konfiguriert sein (Kodierung, Schriftart etc.), dass sie UTF-8-Zeichen korrekt anzeigt.

Für die Felder “Name des Lizenznehmers” und “Ort der Installation” sind ausschließlich ASCII-Zeichen zugelassen.

2 Windows™-Plattform

Besonderheiten

Der Zeichensatz kann bei der Installation nicht ausgewählt werden.

Unterschiede im Umfang des Runtime-Systems

- Das Programm `bin/lpcat.exe` fehlt.
- Das zum Zeichensatz gehörige Unterverzeichnis (`ISO-15`, `ISO-2`, `IBM437` oder `GERMAN7`) von `etc` fehlt.

Unterschiede im Umfang des Entwicklungssystems

- Die Skripte `cmd/choice`, `cmd/edcmap`, `cmd/edkmap`, `cmd/fxindex`, `cmd/fxman`, `cmd/ml`, `cmd/selo` und `cmd/vl` fehlen.
- Die Verzeichnisse `cho`, `hpd`, `men`, `mfo`, `pan`, `runtime/hpd`[3] und `sel` liegen jetzt unterhalb des neuen Verzeichnisses `de`.▲
- Die Dateien `fix_msg` und `messages` liegen jetzt im Verzeichnis `de`.▲
- Das Verzeichnis `tool_us` fehlt. Die Inhalte liegen jetzt im Verzeichnis `us`.▲
- Es gibt ein zusätzliches Verzeichnis `extra` mit im UTF-8-Umfeld nützlichen Skripten.

1. vgl. aber **mlyconv** auf Seite 38.
2. vgl. aber **panconv** auf Seite 37.
3. von *FIX*-Tools benutzte Standard-Hilfetexte.

- Die Programme `bin/mly2pan.exe`[1], `bin/mlycheck.exe`, und `bin/pan2mly.exe`[2] fehlen.

Lizenzierung

Zur Ausführung von *FIX*-Anwendungen, die mit der Version 3.1.0 erstellt wurden, ist kein neues Lizenz-Paket erforderlich. Die Version 3.1.0 kann zusammen mit dem *FIX-Lizenz-Paket* der Version 2.9.4 oder höher betrieben werden.

1. vgl. aber **mlyconv** auf Seite 38.
2. vgl. aber **panconv** auf Seite 37.

3 Grundlagen von *FIX 3.1.0 Unicode*

1 Locale-Unterbau

Beim Einsatz der Version *FIX 3.1.0 Unicode* muss die Umgebungsvariable FXCHARSET auf den Wert UTF-8 gesetzt sein (→ `/etc/stdprofile`). Wenn ein *FIX*-Programm bei der Initialisierung diesen Wert vorfindet, wird der globalen Variable B_CodesetRequired (vgl. Seite 26) der Wert 1 zugewiesen. Unter UNIX wird anschließend das Locale aktiviert (→ setlocale (3)) und heuristisch geprüft, ob es UTF-8-konform ist. Besitzt die Umgebungsvariable nicht den Wert UTF-8 oder ist - unter UNIX - das Locale nicht aktivierbar oder ungeeignet, wird das Programm abgebrochen.

Aus externen Quellen gelesene oder dorthin geschriebene Zeichen(folgen) müssen, sofern sie überhaupt Zeichen außerhalb des ASCII-Bereichs enthalten dürfen, bis auf wenige Ausnahmen UTF-8-kodiert sein. Dabei werden Zeichen mit Codes oberhalb von 65535, d.h. solche außerhalb der Basic Multilingual Plane, nicht unterstützt.

Da *FIX* sich zur Wandlung zwischen Multibyte- und wchar-Darstellung von Zeichen der Funktionen

mbrtowc (3)	- convert a multibyte sequence to a wide character
mbsrtowcs (3)	- convert a multibyte string to a wide character string
wcrtomb (3)	- convert a wide character to a multibyte sequence
wcsrtombs (3)	- convert a wide character string to a multibyte string

sowie

MB_CUR_MAX (3)	- maximum length of a multibyte character in the current locale
mbrlen (3)	- determine number of bytes in next multibyte character

und zur Klassifizierung/Konvertierung von Zeichen der Funktionen

iswalnum (3)	- test for alphanumeric wide character
iswalpha (3)	- test for alphabetic wide character
iswblank (3)	- test for whitespace wide character[1]
iswcntrl (3)	- test for control wide character
iswdigit (3)	- test for decimal digit wide character
iswgraph (3)	- test for graphic wide character
iswlower (3)	- test for lowercase wide character
iswprint (3)	- test for printing wide character
iswpunct (3)	- test for punctuation or symbolic wide character
iswspace (3)	- test for whitespace wide character
iswupper (3)	- test for uppercase wide character
iswxdigit (3)	- test for hexadecimal digit wide character
towlower (3)	- convert a wide character to lowercase
towupper (3)	- convert a wide character to uppercase

bedient, muss das Locale, in dem ein *FIX*-Programm abläuft, korrekt definiert und UTF-8-konform sein.

Unter Windows™ benutzt *FIX* Nachbildungen dieser Funktionen.

1. nicht auf allen Plattformen verfügbar.

2 Objektbeschreibungsdateien (.men, .mfo, .sel, .cho)

Die Syntax bleibt erhalten, aber die Tokens dürfen Multibyte-Zeichen enthalten. Als Multibyte-Kodierung wird ausschließlich UTF-8 unterstützt. Zeichenketten, in denen *FIX* die Umschaltung in den Grafikzeichensatz erlaubt, dürfen zwischen ^N (\116) und ^O (\117) die Kodierung von Codes zwischen 32 und 65535 enthalten.

Besonderheit: Wie bisher darf für Bytes mit Werten außerhalb von 0-127 deren oktale Ersatzdarstellung benutzt werden. Innerhalb von *FIX* geschieht dies für Nicht-ASCII-Zeichen immer, um die Umsetzbarkeit der Datei mit Hilfe von Skripts im POSIX/C-Umfeld zu garantieren.

Beispiel:

Die UTF-8-Kodierung des Zeichens “Ä” (U+00C4), die aus zwei Bytes mit Werten 195 (**110**00011) und 132 (**10**000100) besteht, kann als \303\204 geschrieben werden.

Die nachfolgenden Abschnitte verfeinern die Aussagen des bisherigen Handbuchs. Nur an den grün hervorgehobenen Stellen sind Tokens mit Multibyte-Zeichen zugelassen. Die Token werden beim Einlesen von Objektbeschreibungsdateien auf Zeichensatzkonformität überprüft. Alle nicht markierten Tokens dürfen nur ASCII-Zeichen enthalten.

2.1 .men-Dateien

Die Beschreibungsdatei ist *zeilenorientiert*. Sie beginnt mit einem zweizeiligen Kopf, der die Menüattribute enthält.

menue <objektname> <titel> <layoutdatei> [**noframe**]
[**function** <funktionsname>] \
    <hoehe> <breite> <ypos> <xpos> [**set** <daten>] [**prompt** <unsigned>] [**ignorecase**] [**auto**]

wobei

<objektname> ::= <identifier>
<titel> ::= <string> | #<unsigned>
<hoehe> ::= <unsigned>
<breite> ::= <unsigned>
<ypos> ::= <unsigned>
<xpos> ::= <unsigned>
<daten> ::= bis zu 8 durch : getrennte <unsigned> Werte mit anwendungsspezifischen Informationen

Auf den Kopf folgen die Menüelemente. Jeder Menüpunkt belegt genau eine Zeile und muss in folgender Form angegeben sein:

<name> <y> <x> <markierung> <text> <aktion> [**set** <daten>] [**prompt** <unsigned>] \
    [**iconset** <unsigned>] [**nobackground**] [**steady**]

wobei

<name> ::= <identifier>
<y> ::= <unsigned>
<x> ::= <unsigned>
<markierung> ::= <unsigned>
<text> ::= <string> | #<unsigned>
<aktion> ::= **sh** <string>
    | **shclear** <string>
    | **call** <funktionsname>
    | **perform** <objektdatei>
    | **perfix** <maskendatei>
<daten> ::= bis zu 8 durch : getrennte <unsigned> Werte mit anwendungsspezifischen Informationen

Abgeschlossen wird die Menübeschreibung durch eine Zeile, die mit dem Symbol

END_OF_MFO

beginnt. Darauf noch folgende Zeilen behandelt *FIX* als Kommentar.

2.2 `.mfo`-Dateien

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

(**mask** | **submask** | **rollmask** | **table**) <objektname> <titel> <layoutdatei> [**noframe** | **emptyframe**]
<hoehe> <breite> <ypos> <xpos> [**set** <daten>] [**prompt** <unsigned>] [<start-element> [<fetch-element>]]

wobei

<objektname> ::= <identifier>
<titel> ::= <string> | #<unsigned>
<hoehe> ::= <unsigned>
<breite> ::= <unsigned>
<ypos> ::= <unsigned>
<xpos> ::= <unsigned>
<daten> ::= bis zu 8 durch : getrennte <unsigned> Werte mit anwendungsspezifischen Informationen
<start-element> ::= <unsigned>
<fetch-element> ::= <unsigned>

Auf den Kopf folgen optionale Eigenschaften und Elemente der Maske. Zu den Eigenschaften zählen

die Art der Maus-Unterstützung:

(**disable** | **enable**) **mouse**

Varianten der Maske:

zoom (<maskendatei>)+ [**restart**]

Zeilentypen (nur bei Tabellenmasken):

use (<maskendatei>)+

eine Aktion, die bei Verlassen der Maske ausgeführt wird:

sh <string> | **shclear** <string> | **call** <funktionsname>

das Verhalten bei Verwendung als Submaske:

disable [**rowcount**] [**delete**]
with (<zuordnung>)+
where <string>

wobei

<zuordnung> ::= <column>=#<feldname> (*ein* Symbol !)

Jede der Angaben darf in der Beschreibung *höchstens* einmal vorkommen.

Jedes Element - dies ist entweder ein Feld oder eine eingebettete Maske - belegt genau eine Zeile und muss in folgender Form angegeben sein:

Feld:

field <name> <y> <x> <form_spec> <styp> <dtyp> [**NULL** | <variable> | <table>**.**<column>]

wonach in beliebiger Abfolge weitere optionale Angaben folgen können:

set <daten>
prompt <unsigned>
iconset <unsigned>
link #[<maskenname>].<feldname> (*ein* Symbol !)
selo <selodatei> | (**use** | **offer** | **choice**) <choicedatei> [**steady** | **pasting**]
(**values** <string> | **regex** <string>)
check <funktionsname>
def <string>

wobei

<name> ::= <identifier>
<y> ::= <unsigned>
<x> ::= <unsigned>
<form_spec> ::= <formatstring>
| #<unsigned>
| <feldlaenge>[**.**<nachkommastellen>|**:**<display_laenge>]
(*ein* Symbol !)
<formatstring> ::= <string>
<feldlaenge> ::= <unsigned>
<nachkommastellen> ::= <unsigned>
<display-laenge> ::= <unsigned>
<styp> ::= <unsigned> (Kodierung für Feldeigenschaften)
<dtyp> ::= <unsigned> (Kodierung für Datentyp)
| <unsigned>**:null**
<variable> ::= <identifier>
<table> ::= <identifier>
<column> ::= <identifier>
<daten> ::= bis zu 8 durch : getrennte <unsigned> Werte mit anwendungsspezifischen Informationen
<maskenname> ::= <identifier>
<feldname> ::= <identifier>
<funktionsname> ::= <identifier>

eingebettete Maske:

(**mask** | **submask** | **rollmask** | **table**) <maskendatei> [**steady** | **pasting**]

Abgeschlossen wird die Maskenbeschreibung durch eine Zeile, die mit dem Symbol

END_OF_MFO

beginnt.

Darauf noch folgende Zeilen behandelt *FIX* als Kommentar.

2.3 `.sel`-Dateien (SQL)

Die Beschreibungsdatei eines Selos ist im Gegensatz zu der der meisten anderen Objekte nicht zeilenorientiert und hat folgenden Aufbau:

selo [<objektname>] [**at** <ypos> <xpos>] [**size** <anz>] [**set** <daten>]
(<query> [**restrict** <conditions>] | <procedure_call>)
<item>
(, <item>)*
end

wobei

<objektname> ::= <identifier>
<ypos> ::= <unsigned>
<xpos> ::= <unsigned>
<anz> ::= <unsigned>
<daten> ::= bis zu 8 durch : getrennte <unsigned> Werte mit anwendungsspezifischen Informationen
<query> ::= Liste von Symbolen, die mit dem Symbol select beginnt und mit einem Semikolon abgeschlossen ist;
optional mit einer into-Klausel wie unten
<conditions> ::= Liste von Symbolen, die mit einem Semikolon abgeschlossen ist

```
<procedure_call> ::= execute procedure <proc_name> ( <proc_arg_list> )
                     [ <into_clause> ] ;
<proc_name>      ::= SPL-Prozedur-Name
<proc_arg_list>  ::= /* leer */
                   | <identifier> = <value> [ , <identifier> = <value> ] *
                   | <value> [ , <value> ] *
<value>          ::= Zeichenketten-Konstante                (in ' eingeschlossen)
                   | numerische Konstante
                   | TODAY
                   | Datetime-Literal
                   | CURRENT … TO …
                   | Intervall-Literal
                   | <integer> UNITS …
                   | NULL
                   | #
                   | #<maskname>.<fieldname>
                   | #+<unsigned>
                   | #-<unsigned>
                   | ?<maskname>.<fieldname>
                   | ?+<unsigned>
                   | ?-<unsigned>

<into_clause>    ::= into <column_spec> [ , <column_spec> ] *
<column_spec>    ::= [ <column_name> ] <column_type>
<column_name>    ::= <identifier>
<column_type>    ::= CHAR(<len>)                            → FXCHARTYPE
                   | CHARACTER(<len>)                       → FXCHARTYPE
                   | GRAPHICS(<len>)                        → FXGRAPHICSTYPE
                   | TRUTH                                  → FXTRUTHTYPE
                   | SHORT                                  → FXSHORTTYPE
                   | LONG                                   → FXLONGTYPE
                   | FLOAT                                  → FXFLOATTYPE
                   | DOUBLE                                 → FXDOUBLETYPE
                   | DECIMAL [ (<prec>,<scale>) ]           → FXDECIMALTYPE
                   | DATE                                   → FXDATETYPE
                   | DATETIME … TO …                        → FXDTIMETYPE
                   | INTERVAL … TO …                        → FXINVTYPE
<len>            ::= <unsigned>
<prec>           ::= <unsigned>
<scale>          ::= <unsigned>

<item>           ::= <column_name> [ < heading> [ <format> [ <target> ] ] ]
<heading>        ::= <string> | #<msgnr>
<format>         ::= <string> | #<msgnr>
<target>         ::= #<maskname>.<fieldname>
                   | #+<unsigned>
                   | #-<unsigned>
```

2.4 .sel-Dateien (C-ISAM)

Hier hat die Beschreibungsdatei folgenden Aufbau:

selo [<objectname>] [**at** <ypos> <xpos>] [**size** <anz>] [**set** <daten>]
from <selection>
[**join** <column> **with** <selection>]*
end

wobei

```
<objectname>     ::= <identifier>
```

<ypos> ::= <integer>
<xpos> ::= <integer>
<anz> ::= <unsigned>
<daten> ::= <integer> | <integer>:<integer>
<selection> ::= <table> **key** <column> <item> [, <item>]* [**where** <condition>]
<table> ::= <identifier>
<item> ::= <column> [< heading> [<format> [<target>]]]
<heading> ::= <string>
<format> ::= <string>
<target> ::= #<maskname>.<fieldname>
<condition> ::= <comparison> [**and** <condition>]
<comparison> ::= <column> <op> [<column> | #<field> | constant]
<column> ::= <identifier>
<field> ::= <identifier>
<op> ::= (**!=** | **<** | **<=** | **=** | **>=** | **>** | **matches** | **in**)
<constant> ::= <string>

2.5 .cho-Dateien

Die Beschreibungsdatei einer Choice - nach Konvention cho/<objektname>.cho - ist zeilenorientiert. Sie beginnt mit einem zweizeiligen Kopf, der allgemeine und objekttypische Attribute enthält.

choice <objektname> <titel> <layoutdatei> [**noframe** | **emptyframe**]
<hoehe> <breite> <ypos> <xpos> [**set** <daten>] [**prompt** <unsigned>] [**ignorecase**]

wobei

<objektname> ::= <identifier>
<titel> ::= <string> | #<msgnr>
<hoehe> ::= <unsigned>
<breite> ::= <unsigned>
<ypos> ::= <unsigned>
<xpos> ::= <unsigned>
<daten> ::= bis zu 8 durch : getrennte <unsigned> Werte mit anwendungsspezifischen Informationen.

Daran schließt sich eine Zeile mit der Definition zur Darstellungsmatrix an:

<zeilen> <spalten> <abstand>

wobei

<zeilen>: ::= <unsigned> | -<unsigned>
<spalten>: ::= <unsigned>
<abstand> ::= <unsigned>

Auf den Kopf kann eine - optionale - Zeile mit der Angabe zur Datenbeschaffung (*data-Klausel*) folgen:

[**static**] **data from** <data_stmt>

wobei

<data_stmt> ::= **file** <datendatei> [<into_clause> ;]
| **sh** <cmd> [<into_clause> ;]
| <query>
| <procedure_call>
<cmd> ::= <string>

Zur Syntax von <query> und <procedure_call> siehe Seite 11. Bei beiden unterscheidet *FIX* - außer in geschützten Bezeichnern (von doppelten Hochkommata umgeben) und Zeichenketten-Konstanten (von einfachen Hochkommata umgeben) - nicht zwischen Groß- und Kleinschrift.

In Verbindung mit **static data from** <query> funktionieren relative Feldreferenzen (#+0 bzw. #feldname) nicht, da hier zum Zeitpunkt der Datenbeschaffung (beim Ladens der Choice) noch keine logische Verknüpfung zwischen der Choice und der sie verwendenden Maske besteht. Auch eine absolute Referenz auf diese Maske macht in Allgemeinen keinen Sinn, da deren Felder zu diesem Zeitpunkt stets noch ihre Defaultwerte besitzen.

Zeilenumbrüche innerhalb von <into_clause>, <query> und <procedure_call> werden von *FIX* wie Leerzeichen behandelt, d.h. beenden die data-Klausel nicht.

Verpflichtend ist eine Zeile, die Kriterien zur Auswahl beinhaltet:

[**range** <min> <max> [**auto**]] [**restart**]

wobei

<min> ::= <unsigned> (Mindestanzahl ausgewählter Optionen bei Verlassen)
<max> ::= <unsigned> (Höchstanzahl ausgewählter Optionen bei Verlassen)

Die Angabe von **restart** ist nur in Verbindung mit **static data** zulässig.

Daran können sich weitere Zeilen anschließen:

display <funktionsname>
import [<funktionsname>]
export [<funktionsname>]

Abgeschlossen wird die Choicebeschreibung durch eine Zeile, die mit dem Symbol

END_OF_MFO

beginnt. Nachfolgende Symbole werden ignoriert.

3 Layoutdateien

3.1 .pan-Dateien

Die Layoutbeschreibung unterstützt eine zusätzliche Art der Zeichenkodierung. Die Syntax bleibt erhalten, aber die Tokens dürfen Multibyte-Zeichen enthalten. Als Multibyte-Kodierung wird ausschließlich UTF-8 unterstützt.

Nur an den grün hervorgehobenen Stellen sind Tokens mit Multibyte-Zeichen zugelassen. Alle nicht markierten Tokens dürfen nur (noch) ASCII-Zeichen enthalten.

<CharacterSet '*zeichenfolge*'>
<Write <Yx *y x*> '*zeichenfolge*'>
<Put <Yx *y x*> 'zeichenfolge'>
<Draw <Yx *y x*> '*bytefolge*'>

Als Argument in <CharacterSet …> muss in diesem Fall 'UTF-8' verwendet werden.

Wegen der erweiterten Syntax muss auch die Versionsangabe angepasst werden:

<Version 310>

Es ist aber auch möglich, einen der Werte ISO-15, ISO-2, IBM437 oder GERMAN7 für die Angabe <CharacterSet> zu verwenden und die Zeichen in dem jeweiligen Zeichensatz zu kodieren. In diesem Fall wandelt *FIX* beim Einlesen die Zeichen in den Unicode-Zeichensatz um.

3.2 `.mly`-Dateien

Das bisherige Binärformat kann bei *FIX 3.1.0 Unicode* nicht verwendet werden, da es nicht in der Lage ist, einen Zeichencode größer 255 zu speichern.

Daher wurde ein zusätzliches Binärformat eingeführt.

Hinweise zum alten Binärformat

Die Datei beinhaltet eine Folge von Objekthöhe × Objektbreite × 2 Bytes.

Byte (y × Objektbreite + x) × 2

ist als vorzeichenlose 1-Byte-Zahl aufzufassen

Die Bits bestimmen die Zeichenart (Text oder Semigrafik) und das Videoattribut (NORMAL, INVERS, UNDERLINE, LOW) der Position (y, x).

Byte (y × Objektbreite + x) × 2 + 1

ist als vorzeichenlose 1-Byte-Zahl aufzufassen

Zeichencode (gemäß FXCHARSET) bzw. Semigrafikcode für die Position (y, x)

Neues Binärformat

Die Datei beinhaltet einen Kopf plus eine Folge von Objekthöhe × Objektbreite × 4 Bytes.

Kopf

Bytes 0 bis 3

Magic Number

Bytes 4 bis 15

Layout-Formatversion (benutzt nur ASCII-Zeichen)

Bytes 16 bis 35

Layout-Zeichensatz (benutzt nur ASCII-Zeichen)

Bytes 36 bis 37

Höhe (vorzeichenlose 2-Byte-Zahl im Big-Endian-Format[1])

Bytes 38 bis 39

Breite (vorzeichenlose 2-Byte-Zahl im Big-Endian-Format)

Daten

Bytes 40 + (y × Objektbreite + x) × 4 bis 40 + (y × Objektbreite + x) × 4 + 1

sind als vorzeichenlose 2-Byte-Zahl im Big-Endian-Format aufzufassen

Die Bits bestimmen die Zeichenart (Text oder Semigrafik) und das Videoattribut (NORMAL, INVERS, UNDERLINE, LOW) der Position (y, x)

zusätzliche Bits bei Fullwidth-Zeichen

Bytes 40 + (y × Objektbreite + x) × 4 + 2 bis 40 + (y × Objektbreite + x) × 4 + 3

sind als vorzeichenlose 2-Byte-Zahl im Big-Endian-Format aufzufassen[2]

1. auch als Network Byte Order bekannt
2. vgl. auch UCS-2BE

Zeichencode (gemäß FXCHARSET; ISO 10646-1:1993 beim Zeichensatz “UTF-8”) bzw. Semigrafikcode für die Position (y, x)

Anmerkung:

Bei Fullwidth-Zeichen, d.h. solchen, die mehr als eine Ausgabeposition beanspruchen, müssen alle zugehörigen Positionen den gleichen Code beinhalten und die Attribute hinsichtlich Zeichenart und Videoattribut übereinstimmen.

4 Hilfetexte

Die Syntax bleibt erhalten, aber die Texte dürfen Multibyte-Zeichen enthalten. Als Multibyte-Kodierung wird ausschließlich UTF-8 unterstützt.

5 Meldungsdatei-Vorlage (`fix_msg`, `fx_texte`)

Die Syntax bleibt erhalten, aber die Texte dürfen Multibyte-Zeichen enthalten. Als Multibyte-Kodierung wird ausschließlich UTF-8 unterstützt.

Die Texte mit den folgenden Nummern sind neu:▲

10077-10079, 10560-10561, 10570-10576, 10580-10589, 10970-10975 sowie
12000-12003, 12011-12013, 12020-12021, 12510, 12520, 12530

Die Texte mit den folgenden Nummern sind entfallen:▲

10080 sowie
13100-13105

Die Texte mit den folgenden Nummern haben sich geändert:▲

10089, 10098-10099, 10132, 10345, 10672 sowie
14006, 14060-14065, 14068, 14072-14073

4 Allgemeine Verbesserungen

1 ASCII-Felder

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

Felder mit Typ FXCHARTYPE und der neuen Feldeigenschaft ASCIIONLY erlauben nur noch die Eingabe von ASCII-Zeichen. Diese Feldeigenschaft verhindert allerdings nicht, dass auf anderem Wege (z.B. über die Hostvariable) Werte in das Feld geschrieben werden, die Nicht-ASCII-Zeichen enthalten. Die Eigenschaft wirkt sich - wie auch die Feldeigenschaften UPPER, LOWER und FIRSTUP - nur auf die Felderfassung aus.

Das Setzen der Eigenschaft ASCIIONLY ist nur bei FXCHARTYPE-Feldern sinnvoll, da andere als FXCHARTYPE- und FXGRAPHICSTYPE-Felder nur ASCII-Eingabe akzeptieren.

2 Insert-Modus

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

FIX unterstützt bei der Felderfassung einen Insert-Modus, der sich allerdings nur auf Felder vom Typ FXCHARTYPE auswirkt. Wenn der Modus aktiv ist, wird das Zeichen an der aktuellen Position nicht mehr überschrieben; stattdessen wird das eingegebene Zeichen an der aktuellen Position eingefügt, wenn genügend Platz im Eingabepuffer des Feldes vorhanden ist. Ist kein Platz im Eingabepuffer vorhanden, ertönt ein akustisches Signal und der Feldinhalt bleibt unverändert.

Der Insert-Modus wird dreistufig gesteuert. Die oberste Stufe bildet die Ressource

```
UseFieldInsertMode
```

Damit kann der Insert-Modus global für die ganze Anwendung abgeschaltet werden. Enthält diese Ressource den Wert FALSE (das ist der Default), dann wird der Insert-Modus nie verwendet.

Die zweite Stufe bildet die Variable

```
BOOLEAN S_field_insert_mode
```

Sie bestimmt mit dem Wert TRUE, ob der Insert-Modus für ein Feld verwendet wird. Durch das Setzen auf FALSE unmittelbar vor der Felderfassung kann der Insert-Modus für ein bestimmtes Feld abgeschaltet werden. *FIX* initialisiert die Variable mit TRUE und wertet sie nur aus. Die Anwendung ist bei einer Veränderung dafür verantwortlich, die Varaiable nach der Felderfassung oder spätestens vor der Erfassung des nächsten Feldes wieder auf den Ursprungswert zu setzen, wenn dies notwendig ist.

Das Setzen von S_field_insert_mode hat nur einen Einfluss, wenn die Ressource UseFieldInsertMode den Wert TRUE besitzt.

Die dritte Stufe bildet das Event CI, das typischerweise durch die Taste "Einfügen" erzeugt wird. Wenn UseFieldInsertMode und S_field_insert_mode beide den Wert TRUE besitzen, dann wird der Insert-Modus beim Betreten des Feldes aktiviert. Mit Hilfe dieses Events kann der Modus während der Felderfassung umgeschaltet werden. Im ausgeschalteten Zustand unterscheidet sich die Felderfassung von dem bisherigen Verfahren jedoch dadurch, dass ein weiteres Event CI nicht zum Einfügen eines Leerzeichens führt. Stattdessen wird der Insert-Modus wieder eingeschaltet. Damit wird ein Verhalten implementiert, das von vielen anderen Felderfassungen verwendet wird. Beim Betreten eines anderen Feldes oder beim erneuten Betreten des gleichen Feldes wird der durch das Event CI während der Felderfassung hergestellte Zustand des Insert-Modus nicht übernommen.

Besitzt UseFieldInsertMode oder S_field_insert_mode den Wert FALSE, dann wird das bisher von *FIX* verwendete Verfahren benutzt, bei dem das Event CI zum Einfügen eines Leerzeichens führt.

Da die Umschaltmöglichkeit durch eine Taste bei Benutzern, die das alte Verfahren gewohnt sind, zu Verwirrungen führen kann, kann sie durch das Setzen der Ressource

AllowToggleInsertMode

auf FALSE abgeschaltet werden. In diesem Fall bewirkt das Event CI auch im Insert-Modus das Einfügen eines Leerzeichens.

3 Editieren von Textdateien

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

Durch den Aufruf von editor() können Dateien mit einem Editor bearbeitet werden.[1] Ab der Version *FIX 3.1.0* is t diese Funktion auch in Verbindung mit *FIX/Win* nutzbar. Dazu werden die Dateien auf die *FIX/Win*-Seite kopiert und dort ein Editor gestartet. Dabei finden mehrere Konvertierungen statt.

Auf der *FIX*-Seite wird die Datei in dem Zeichensatz gelesen und geschrieben, der über FXCHARSET eingestellt ist. Als Zeilenende wird die unter dem Betriebssystem typische Markierung (\r\n unter Windows und \n unter UNIX) beim Schreiben verwendet. Beim Lesen sind unter Windows beide Markierungen erlaubt, unter UNIX muss \n an den Zeilenenden stehen.

Nachdem die Daten an *FIX/Win* gesendet wurden, werden sie in den UNICODE-Zeichensatz konvertiert und in eine temporäre Datei (Verzeichnis $TMP) geschrieben. Zum Schreiben wird einer der Zeichensätze ANSI, UNICODE oder UTF-8 verwendet. Windows konvertiert dabei alle Zeichen in diesen Zeichensatz. Der Zeichensatz ist in die anwendungsunabhängige Ressource

FixEditorCP

einzutragen (`fw_usr.frc`). Fehlt die Angabe, dann wird ANSI verwendet. Danach wird ein Editor gestartet, der in der Lage sein muss, diesen Zeichensatz zu lesen und zu schreiben. Die Datei beinhaltet zur Erkennung des Formats die unter Windows übliche Byteordermark am Anfang. Der Pfad des Editors ist in die anwendungsunabhängige Ressource

FixEditor

einzutragen (`fw_usr.frc`).

Während der Editor die Datei bearbeitet, wird das *FIX/Win*-Fenster durch einen Dialog gesperrt, der die Namen der Datei (auf *FIX*- und auf *FIX/Win*-Seite) anzeigt. Durch das Anklicken der Schaltfläche “Anzeigen” kann der Editor in den Vordergrund gebracht werden. Mit Hilfe der Schaltfläche “Abbrechen” kann der Editor abgebrochen werden.

Wenn der Editor beendet wird und die Datei verändert wurde, wird die Datei wieder in *FIX/Win* eingelesen. Die Datei muss dabei den in FixEditorCP vereinbarten Zeichensatz besitzen. Nach dem Einlesen werden die Zeichen, nach UTF-8 gewandelt, zu *FIX* gesendet und dort in den durch FXCHARSET definierten Zeichensatz umgewandelt. *FIX* schreibt die Daten dann in die Datei zurück.

Wurde die Datei nicht verändert, dann wird *FIX* lediglich über das Ende des Editiervorgangs informiert und der Aufruf von editor() kehrt zurück.

In beiden Fällen wird der Dialog über dem *FIX/Win* Fenster entfernt.

1. Auch der Layouteditor **led** nutzt diese Möglichkeit.

4 Speicher für die textuelle Darstellung eines Feldes

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

Die Größe des Speicherbereichs, in dem die textuelle Darstellung eines Feldes *f* gehalten wird, wird in der neuen Komponente *f*->infolen hinterlegt. Für Felder vom Typ FXCHARTYPE und FXGRAPHICSTYPE verwendet *FIX* den Wert *f*->len ∗ S_MB_CUR_MAX + 1.

5 Aufbau des *FIX*-Verzeichnisses

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

Die Dateien, die für die Tools von *FIX* benötigt werden, liegen jetzt nicht mehr direkt in Unterverzeichnissen von FXDIR, sondern im Verzeichnisbaum `de`.

Analog dazu existiert ein Verzeichnisbaum `us`, der die Dateien der Tools von *FIX* in englischer Sprache enthält. Zur Benutzung dieses Verzeichnisses ist in der Umgebungsvariablen FXTOOLHOME der Pfad zu diesem Verzeichnis einzutragen. Zusätzlich ist die Variable FXTOOLLANG auf den Wert `us` zu setzen (Wenn die Variable FXSTRERRLANG mit dem Wert `us` bereits englische Meldungen für die Anwendung definiert, kann das Setzen von FXTOOLLANG unterbleiben).

6 Das Entwicklermenü fxm

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

fxm weist eine Reihe von - in der Praxis unbedeutenden -Einschränkungen auf, die die Stabilität verbessern.

Die in den Masken von **fxm** benutzten Felder verweigern generell die Eingabe von Nicht-ASCII-Zeichen. Einzige Ausnahmen sind

- das Feld zur Eingabe der Argumente in der durch “Source→Programm ausführen” aufgeblendeten Maske

und

- das Feld zur Eingabe des Kommandos in der durch “Usw.→Kommando ausführen” aufgeblendeten Maske.

Alle von **fxm** zur Beschaffung von Datei- oder Verzeichnisvorschlägen verwendeten Choices in `de/cho` benutzen ein neues Hilfsskript `cmd/do/getdirentries.sh` , anstatt unmittelbar das Programm **ls** aufzurufen. Das Skript dient dazu, die brauchbaren Vorschläge herausfiltern; insbesondere werden nur solche Verzeichniseinträge berücksichtigt, die ausschließlich aus den ASCII-Zeichen ! (0x21) bis ~ (0x7E) gebildet sind.

Die durch “FIX-Masken→generieren Maske” aufgeblendete Maske enthält ein Feld zur Eingabe einer Datenbanktabelle. Die auf diesem Feld angebotene Auswahl (`cho/relations.cho`) benutzt zur Beschaffung der Tabellenvorschläge das Hilfsskript `cmd/do/getrelations`. Bei Versionen für IBM Informix benutzt dieses Skript statt **dbaccess** ein neues Hilfsprogramm **getdb_tables** und berücksichtigt nur solche Tabellen, deren tabid größer-gleich 100 ist und die einen Namen besitzen, der ausschließlich aus den ASCII-Zeichen ' ' (0x20) bis ~ (0x7E) gebildet ist.

Das Hilfsprogramm **obgen** akzeptiert als zweites (optionales) Argument neben old-txt und old-bin auch 310-txt (Voreinstellung bei der Version mit Unicode-Unterstützung) und 310-bin. Bei *FIX 3.1.0 Unicode* ruft **fxm** bei Anwahl des Menüpunkts “FIX-Masken→Layout-Editor” oder “Menü→Layout-Editor” **obgen** mit zweitem Argument 310-txt auf.

Für $1 gleich xxx.mfo (und $2 gleich 310-txt) generiert **obgen** die Dateien `mfo/xxx.mfo`

```
mask XXX “XXX Maske” pan/xxx.pan
22 80 0 0
```

END_OF_MFO

und `pan/xxx.pan`

<Version 293>
<Geometry 22 80>
<CharacterSet 'UTF-8'>

Die durch "Source→Programm ausführen" aufgeblendete Maske enthält ein Feld zur Eingabe eines Programmnamens. Die auf diesem Feld angebotene Auswahl (`cho/bins.cho`) benutzt zur Beschaffung der Programmvorschläge das Hilfsskript `cmd/do/getprograms`. Dieses Skript berücksichtigt nur noch solche Dateien, die für den Benutzer ausführbar sind (nur Versionen für Unix) und die einen Namen besitzen, der ausschließlich aus den ASCII-Zeichen ' ' (0x20) bis ~ (0x7E) gebildet ist.

Das von "Usw.→DB-Tool aufrufen" aufgerufene Skript `cmd/do/ex-dbtool` verwendet bei Versionen für IBM Informix nun **dbaccess** (statt **isql**), wenn die Umgebungsvariable FXDBTOOL nicht gesetzt ist.

fxm entfernt die Umgebungsvariable CDPATH aus der Umgebung, um die störende Ausgabe des Verzeichnisnamens zu vermeiden, wenn ausgeführte Shellkommandos mittels **cd** das Verzeichnis wechseln.

7 Source-Generierung

FIX 3.1.0 verwendet standardmäßig einen neuen Stil "V3.1.0" für Sourcen (vgl. Ressource SourceStyle).

7.1 Source-Generierung für Masken

Der neue Stil "V3.1.0" unterscheidet sich von dem zuvor verwendeten Stil "V2.9.4" durch die Art, wie Hostvariablen für FXCHARTYPE- und FXGRAPHICSTYPE-Felder definiert werden:

	V2.9.4	V3.1.0
FXCHARTYPE-Feld der Länge n	char[n'], wobei n' gleich n + 1	char[n * factor + 1]
FXGRAPHICSTYPE-Feld der Länge n	fixchar[n'], wobei n' gleich n	fixchar[n * factor]

Hierbei ist *factor* wahlweise eine numerische Konstante oder das Makro BPC, je nachdem, ob die neu eingeführte Ressource fgmask.BytesPerCharMethod den Wert "constant" (Voreinstellung) oder "macro" besitzt.

Der Wert der Konstante wird von der *FIX*-Version bestimmt (→ Variable S_MB_CUR_MAX).[1]

Um die Einstellung "macro" nutzen zu können, wurde auch die makefile-Datei modifiziert. Durch

- eine zusätzliche Vereinbarung
  ESQLCFLAGS =
  # Achtung: je nach Variante der Generierung muss -EDBPC=<n> hinzugefuegt werden.
  (vgl.`cmd/mkfixenv`, `gen/all/MakeHeader.pat`)
- einen Hinweis bei der Definition von CFLAGS
  CFLAGS = $(FXCFLAGS) -I$(SRC)
  # Achtung: je nach Variante der Generierung muss -DBPC=<n> hinzugefuegt werden.

1. Hinweis zu *FIX 3.1.0*:Im Falle "constant" entfällt die (oben unterstrichene) Multiplikation, wenn die numerische Konstante den Wert 1 besitzt.

(vgl.`cmd/mkfixenv`, `gen/all/MakeHeader.pat`)

und

- eine Erweiterung der Regel zum Aufruf des Precompilers
  $(ESQLC) $(ESQLCFLAGS) *source*.ec
  (vgl.`gen/mask/GnrcRule1.pat`, `gen/mask/GnrcRule4.pat`, `gen/mask/GnrcRule4c.pat`)

wurden die Voraussetzung dafür geschaffen, dem Precompiler und dem C-Compiler einen Wert für das Makro BPC mitzuteilen.

Generierung im Stil der Vorversion

Wird die Ressource [fgmask.]SourceStyle explizit auf den Wert “V2.9.4” gesetzt, werden die Hostvariablen und die makefile-Datei wie bei *FIX* 3.0.0 generiert. *FIX* verwendet in diesem Fall spezielle Musterdateien mit dem Namensbestandteil `_29X` aus `gen`.

7.2 Source-Generierung für Menüs

Der neue Stil “V3.1.0” unterscheidet sich von dem zuvor verwendeten Stil “V2.9.4” nur darin, dass beim Anlegen einer makefile-Datei der gleiche Kopf wie bei der Source-Generierung für Masken verwendet wird (vgl. Abschnitt 7.1).

Generierung im Stil der Vorversion

Wird die Ressource [fgmenu.]SourceStyle explizit auf den Wert “V2.9.4” gesetzt, wird die makefile-Datei wie bei *FIX* 3.0.0 generiert. *FIX* verwendet in diesem Fall spezielle Musterdateien mit dem Namensbestandteil `_29X` aus `gen`.

5 Besonderheiten von *FIX 3.1.0 Unicode*

1 Kodierung von Events

Mit der Einführung von Unicode tritt das Problem auf, dass die von *FIX* bisher verwendeten Kodierungen für Tasten- und logische Events sich mit den Codes von Unicode-Zeichen überschneiden. Deshalb werden die Events ab der Version 3.1.0 relativ zu einer Konstanten K_BASE definiert. Diese Konstante besitzt im Falle UCS_1993 den Wert 100000 gesetzt und sonst 0. Damit liegen die Codes für Events in der Version *FIX 3.1.0 Unicode* außerhalb des Bereichs der Unicode-Zeichen (siehe `keys.h`).

2 Semigrafik

FIX 3.1.0 Unicode unterstützt im Semigrafik-Zeichensatz Codes zwischen 32 und 65535.[1]

Hinweise:

Da *FIX/Win 3.1.0* jedoch (derzeit) nicht in der Lage ist, Bitmaps für die Codes oberhalb von 255 einzulesen, weist es Semigrafikzeichen oberhalb von 255 mit einer Fehlermeldung ab.

Beim Einlesen von Semigrafikzeichen aus einer Layoutdatei, die nicht den Zeichensatz UTF-8 verwendet, werden Semigrafikzeichen, im Gegensatz zu den anderen Zeichen in dieser Datei, nicht in den Unicode-Zeichensatz umgewandelt. Es können also keine Semigrafikzeichen oberhalb von 255 entstehen.

3 API

3.1 Header-Dateien

Zur Differenzierung zwischen Code, der für die Version mit Unicode-Unterstützung bestimmt ist, und Code, der ausschließlich für die klassische Version gilt, benutzt *FIX* das Makro UCS_1993:

```
#ifdef UCS_1993
/* Code fuer die Version mit Unicode-Unterstuetzung */
#else
/* Code fuer die klassische Version */
#endif /* UCS_1993 */
```

(vgl. z.B. `fix/keys.h`).

1. Allerdings ist die Capability SG der Bildschirmbeschreibung nicht leistungsfähig genug, dieses Feature zu nutzen.

Dieses wird derzeit bei der Definition des Make-Makros CC in `spec/m.default` berücksichtigt.

fix/accept.h

definiert die neue Feldeigenschaft ASCIIONLY.▲

fix/fixwin.h

deklariert Konstanten zur Steuerung des “Input Method Editors” (IME).

fix/fxtypes.h

deklariert die globale Variable S_MB_CUR_MAX.▲

Definition der Makros FXCHARSIZE(*l*) und FXGRAPHICSSIZE(*l*) geändert.▲

fix/keys.h

enthält versionsabhängigen Code: in der Version mit Unicode-Unterstützung

- besitzen alle Nicht-Zeichen-Events Werte oberhalb von 65535,
- liefert ISCHAR(*c*) 1 für Events *c* zwischen 0 und 65535.

fix/obj.h

Die Datenstruktur menue besitzt eine weitere, nur zur internen Verwendung vorgesehene Komponente, um die Einrückung zu verwalten.▲

Es werden zusätzliche, nur zur internen Verwendung vorgesehene Konstanten definiert.▲

Die Komponentenliste FIELD_2_COMPONENTS enthält zusätzliche und modifizierte Komponenten. Für die Anwendungsentwicklung interessant ist lediglich die Komponente infolen, die die Größe des in info hinterlegten Speicherbereichs enthält.▲

fix/release.h

deklariert das Makro FIX_VERSION mit dem WERT 310.▲

fix/libfix.h

schließt zwei neue Header-Dateien ein▲:

- `proto/char_pto.h`
- `proto/mb_pto.h` (nur, wenn WCHAR_MIN definiert ist, d.h. zuvor `wchar.h` eingeschlossen wurde)

codeset.h

neue Header-Datei:

- deklariert die neue Variable B_CodesetRequired▲
- vereinbart verschiedene intern benutzte Konstanten, Variablen und Funktionen

fxctype.h

neue Header-Datei, die Konstanten, Variablen und Funktionen vereinbart, um Bereiche von Zeichenklassen zu definieren.

testflags.h

definiert den neuen Wert CODESET_C.▲

video.h

definiert zusätzliche Konstanten im Zusammenhang mit Fullwidth-Zeichen.

proto/char_pto.h

neue Header-Datei, die zwei Funktionen fx_char_collen_to_len() und fx_char_copy()▲ deklariert.

proto/fixwin_pto.h

deklariert neue Funktionen fx_get_ime_mode() und fx_set_ime_mode().

proto/fxctype_pto.h

enthält nicht mehr die Funktionen makeupper() und makelower().

deklariert die neue Funktion fxisblank().▲

proto/fxstring_pto.h

enthält nicht mehr die Funktion strnswap().▲

proto/keys_pto.h

deklariert die neue Funktion get_last_input().▲

proto/mask_pto.h

defineButtonSet() und (∗S_GetTableFieldCodes)() sind versionsabhängig vereinbart.

proto/mb_pto.h

neue Header-Datei, die Konstanten, Makros und Funktionen vereinbart, die im Zusammenhang mit der Wandlung zwischen der Multibyte- und der wchar_t-Darstellung von Zeichen stehen.

proto/obj_pto.h

deklariert die neuen Funktionen layout_getchar(), layout_setchar(), layout_getattr(), layout_setattr(), layout_getcharset() und layout_getvideo().▲

proto/output_pto.h

deklariert die neue Funktion sync_refresh().▲

proto/selo_pto.h

enableSeloAsTable() ist versionsabhängig vereinbart.

3.2 Datenstrukturen

Folgende Komponenten werden als Zeiger auf UTF-8-kodierte Zeichenketten (oder Funktionen, die solche zurückgeben) interpretiert:

Struktur	Strukturkomponente	Bemerkungen
box_cb	bx_headline	ob_box-Kopmponente bei Menü, Maske, Choice, Selo
struct cmd_dcr	cmd_action	ob_command-Komponente bei Maske, falls cmd_type == EXEC_CMD oder cmd_type == EXEC_CLCMD
choice	inputarg	sofern inputtype == BY_PIPE
	(*mr_display)(…)	
menue_entry	txt	
	action	sofern type == EXEC_CMD oder type == EXEC_CLCMD (→ fx_docmd())
field	wrkp	bei Feldtyp FXCHARTYPE oder FXGRAPHICSTYPE
	f_default	bei Feldtyp FXCHARTYPE oder FXGRAPHICSTYPE
	info	bei Feldtyp FXCHARTYPE oder FXGRAPHICSTYPE
	lastinfo	bei Feldtyp FXCHARTYPE oder FXGRAPHICSTYPE
	tooltiptext	
struct prtu	head	
struct fxse_item	heading	wg. prp->head
context_menu_entry	text	
	bmp_fname	
paintarea	pa_token	
	pa_text	
	pa_tooltiptext	
varbutton	token	
	text	
	tooltiptext	

3.3 *FIX*-Library

3.3.1 Modifizierte Prototypen

Folgende Funktionen bzw. Funktionszeiger besitzen einen versionabhängigen Prototyp:

- defineButtonSet()
- (*S_GetTableFieldCodes)()
- enableSeloAsTable()

3.3.2 Entfallene Funktionen

Folgende Funktionen sind bei *FIX 3.1.0 Unicode* nicht mehr in der *FIX*-Library enthalten:

- void makeupper() - ehemals deklariert in `proto/fxctype_pto.h`
- void makelower() - ehemals deklariert in `proto/fxctype_pto.h`
- void strnswap() - ehemals deklariert in `proto/fstring_pto.h`▲

3.3.3 Neue Variablen und Funktionen

int B_CodesetRequired: readonly[1]

wird bei der Initialisierung in Abhängigkeit vom Wert der Umgebungsvariable FXCHARSET gesetzt und dient dazu zu entscheiden, ob *FIX* sich bei der Behandlung bestimmter Zeichenfolgen (tatsächlich oder - bei Windows™ - simuliert) auf das Locale stützt, in dem das Programm abläuft (vgl. Seite 8).

int S_MB_CUR_MAX: readonly[2]

dient - nach erfolgreicher Initialisierung des Locales für das Programm - dazu zu erkennen, ob das Locale eine Multibyte-Kodierung benutzt und wie lang die Bytesequenz für ein Zeichen maximal sein kann (vgl. MB_CUR_MAX (3)). Da *FIX 3.1.0 Unicode* als einzige (Multibyte-)Kodierung UTF-8 mit auf eingeschränktem ISO/IEC 10646-1:1993 Zeichenvorrat unterstützt, ergibt sich in diesem Fall als Wert 3.

Der Wert spielt insbesondere für die Allokation von Speicherflächen (z.B. info, wrkp von FXCHARTYPE- und FXGRAPHICSTYPE-Feldern) und für die Umrechnung von Spaltenlängen der Datenbank in Zeichen (→ fx_char_collen_to_len()) eine Rolle.

int B_utf8_db: readonly, Defaultwert 0

wird von fx_database_connect() bei erfolgreichem Verbindungsaufbau an eine IBM Informix-Datenbank bestimmt. Die Variable erhält den Wert 1, wenn *FIX* Multibyte-Kodierung unterstützt und der Datenbank*server* augenscheinlich ein UTF-8-basiertes Locale benutzt, sonst auf 0.

Im Falle 1 geht *FIX* davon aus, dass CHARACTER-Spalten der Länge *n* zur Ablage von bis zu $n/3$ signifikanten Zeichen dienen, und weist der globalen Variable S_chlen_to_dblen_factor den Wert 3 zu.

int S_chlen_to_dblen_factor: readonly

hält den Multiplikator (≥ 1), mit dem *FIX* eine Zeichenanzahl in eine Byteanzahl umrechnet (→ fx_database_connect()). Zum Setzen der Variable vgl. B_utf8_db.

→ fx_char_collen_to_len()

BOOLEAN S_field_insert_mode▲

Bestimmt den Insert-Modus für ein Feld (siehe *Insert-Modus* auf Seite 17).

1. Eine analoge Variable wird auch von den Tools **msgprep** und **panconv** verwendet.
2. dto. Fußnote 1.

int fx_char_collen_to_len(collen)
int collen;

bestimmt zur übergebenen *collen* (in Bytes) die Mindestanzahl möglicher Zeichen bei vollständiger Ausnutzung des Platzes.

int fx_char_copy(dest, destsize, src, srcsize, trimright, maxcnt, padchar, addzero)
char *dest; int destsize; char *src; int srcsize; BOOLEAN trimright; int maxcnt; int padchar; BOOLEAN addzero;▲

kopiert die Zeichenfolge an der Adresse *src* (partiell) an die Adresse *dest*. *srcsize* gibt an, wieviele Bytes ab *src* maximal gelesen werden dürfen, *destsize* bestimmt, wieviele Bytes ab *dest* maximal geschrieben werden dürfen. Ist *maxcnt* größer-gleich 0, werden höchstens *maxcnt* Zeichen kopiert.

trimright wird in dieser Version nicht berücksichtigt; als Argument sollte TRUE übergeben werden.

Ist *padchar* ungleich '\0' und *src* ungleich dem NULL-Wert, werden an die kopierte Zeichenfolge Bytes mit Wert *padchar* angefügt, bis die Zeichenfolge an der Adresse *dest destsize* Bytes umfasst; ist *maxcnt* größer-gleich 0, allerdings höchstens soviele, dass sie nicht mehr als *maxcnt* Zeichen enthält.

Ist *padchar* gleich '\0' und *src* ungleich dem NULL-Wert, wird an die Zeichenfolge an der Adresse *dest* '\0' angefügt.

Ist *addzero* ungleich FALSE und *src* ungleich dem NULL-Wert, wird die Zeichenfolge '\0'-terminiert; in diesem Fall sollte *dest*[*destsize*] geschrieben werden dürfen.

int fxisblank(code)
int code;

testet die Zugehörigkeit des Zeichens *code* zur Zeichenklasse BLANK.

Die Funktion liefert für int-Argumente größer-gleich 0 und kleiner-gleich 65535 das Ergebnis von fxisspace(*code*) && ! fxisgraph(*code*) und sonst 0.

Diese häufig anzutreffende Implementierung wurde gewählt, da die Funktion

iswblank (3) - test for whitespace wide character

nur von wenigen Plattformen angeboten wird.

size_t fx_mbrlen(mb, n, ps)
char *mb; size_t n; mbstate_t *ps;

Windows™: Nachbildung der Funktion mbrlen (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

size_t fx_mbrtowc(wc, mb, n, ps)
wchar_t *wc; char *mb; size_t n; mbstate_t *ps;

Windows™: Nachbildung der Funktion mbrtowc (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

int fx_mbslen(mbs)
char *mbs;

liefert die Anzahl der in *mbs* kodierten Zeichen oder -1, wenn *mbs* eine ungültige Multibyte-Kodierung oder ein Zeichen außerhalb des Zeichenvorrats enthält.

int fx_mbswidth(mbs, n)
char *mbs; size_t n;

liefert die Breite der in *mbs* kodierten Zeichen oder -1, wenn *mbs* eine ungültige Multibyte-Kodierung oder ein Zeichen außerhalb des Zeichenvorrats enthält. Die Funktion ruft für jedes Zeichen die Funktion fx_wcwidth() auf und liefert die Summe der Ergebnisse zurück. Wenn der Wert *n* kleiner 0 ist, werden alle Zeichen in *mbs* berücksichtigt. Ansonsten gibt *n* die Anzahl Zeichen (nicht die Anzahl Bytes) an, deren Breiten addiert werden.

size_t fx_mbsrtowcs(wcs, mbsaddr, n, ps)
wchar_t *wcs; char **mbsaddr; size_t n; mbstate_t *ps;

Windows™: Nachbildung der Funktion mbsrtowcs (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

size_t fx_mbstowcs(wcs, mbs, n)
wchar_t *wcs; char *mbs; size_t n;

Windows™: Nachbildung der Funktion mbstowcs (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

Hinweis: Diese Funktion wird *FIX*-intern nicht verwendet, da sie einen anonymen mb_state-Puffer benutzt.

int fx_mbtowc(wc, mb, n)
wchar_t *wc; char *mb; size_t n;

Windows™: Nachbildung der Funktion mbtowc (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

Hinweis: Diese Funktion wird *FIX*-intern nicht verwendet, da sie einen anonymen mb_state-Puffer benutzt.

size_t fx_wcrtomb(mb, wc, ps)
char *mb; wchar_t wc; mbstate_t *ps;

Windows™: Nachbildung der Funktion wcrtomb (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

size_t fx_wcsrtombs(mbs, wcsaddr, n, ps)
char *mbs; wchar_t **wcsaddr; size_t n; mbstate_t *ps;

Windows™: Nachbildung der Funktion wcsrtombs (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

size_t fx_wcstombs(mbs, wcs, n)
char *mbs; wchar_t *wcs; size_t n;

Windows™: Nachbildung der Funktion wcstombs (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

Hinweis: Diese Funktion wird *FIX*-intern nicht verwendet, da sie einen anonymen mb_state-Puffer benutzt.

int fx_wctomb(mb, wc)
char *mb; wchar_t wc;

Windows™: Nachbildung der Funktion wctomb (3) für UTF-8.

UNIX: Makro aus `proto/mb_pto.h`.

Hinweis: Diese Funktion wird *FIX*-intern nicht verwendet, da sie einen anonymen mb_state-Puffer benutzt.

int fx_wcwidth(wc)
wchar_t wc;

ähnlich der Funktion wcwidth (3) (siehe *Fullwidth-Zeichen* auf Seite 38).

void fx_get_ime_mode()

fragt einen internen Zustand ab, der bei Benutzung eines grafischen Frontends mit beeinflusst, ob diesem übermittelt wird, wann der “Input Method Editor” aktiviert sein soll.

void fx_set_ime_mode(mode)
int mode;

setzt einen internen Zustand, der bei Benutzung eines grafischen Frontends mit beeinflusst, ob diesem übermittelt wird, wann der “Input Method Editor” aktiviert sein soll.

int layout_getattr(objp, y, x)▲
obj *objp; int y; int x;

liefert das Attribut in Zeile *y*, Spalte *x* des Layouts des Objekts *objp*.

int layout_setattr(objp, y, x, attr)▲
obj *objp; int y; int x; int attr;

trägt als Attribut in Zeile *y*, Spalte *x* des Layouts des Objekts *objp* den Wert *attr* ein.

Siehe auch *Fullwidth-Zeichen* auf Seite 38.

int layout_setchar(objp, y, x, c)▲
obj *objp; int y; int x; int c;

trägt als Zeichencode in Zeile *y*, Spalte *x* des Layouts des Objekts *objp* den Wert *c* ein.

Siehe auch *Fullwidth-Zeichen* auf Seite 38.

int putstrwidth(s)▲
char *s;

liefert den Platzbedarf (in Bildschirmspalten) zur Darstellung der Zeichen im String *s*.

void define_coderange (cat, range)
int cat; struct coderange *range;

definiert eine Ausnahmetabelle für eine Funktion zur Bestimmung der Zeichenklasse (siehe *Zeichenklassifizierung* auf Seite 32).

Event get_last_input()▲

ermittelt das zuletzt von dcgetch() gelieferte Event.

Die Funktion kann verwendet werden, um bei der Event-Behandlung zu entscheiden, ob ein Event von *FIX* generiert wurde oder ob es eingegeben bzw. mit undcgetch() in den Input-Puffer gestellt wurde.

Das Event L_GETTOOLTIP wird nicht berücksichtigt, es sei denn, es wird explizit mit undcgetch() in den Input-Puffer gestellt.

3.3.4 Variablen und Funktionen mit stark veränderter Funktionalität / Implementierung

char *S_months[12]: readonly

S_months ist nicht mehr als zweidimensionales char-Array, sondern als Array von 12 char-Zeigern definiert. Die Elemente zeigen auf Multibyte-kodierte Zeichenfolgen aus exakt drei Zeichen.

char *S_weekdays[7]: readonly

S_weekdays ist nicht mehr als zweidimensionales char-Array, sondern als Array von 7 char-Zeigern definiert. Die Elemente zeigen auf Multibyte-kodierte Zeichenfolgen aus exakt drei Zeichen.

int fxisalnum(code)*
int fxisalpha(code)*
int fxisdigit(code)
int fxisgraph(code)
int fxislower(code)
int fxisprint(code)*
int fxispunct(code)*
int fxisspace(code)
int fxisupper(code)
int fxisxdigit(code)

greifen für int-Argumente größer-gleich 0 und kleiner-gleich 65535 auf die entsprechende Funktion int isw…(wint_t) (3) (vgl. Seite 8) zurück und liefern deren Ergebnis, anderenfalls 0.

Hinweis: *FIX* für Windows greift statt auf isw…() auf die Funktion GetStringTypeW() zurück.

Die mit * markierten Funktionen berücksichtigen die in *Zeichenklassifizierung* auf Seite 32 beschriebenen Ausnahmetabellen.

int fxtolower(code)
int fxtoupper(code)

greifen für int-Argumente größer-gleich 0 und kleiner-gleich 65535 auf die entsprechende Funktion int tow…(wint_t) (3) (vgl. Seite 8) zurück und liefern deren Ergebnis, sofern es im unterstützten Zeichenvorrat ist, anderenfalls *code*.

Hinweis: *FIX* für Windows greift statt auf tow…() auf die Funktionen CharLowerW()/CharUpperW() zurück.

char *stddisplay(chp, tp, n)
choice *chp; TupelType tp; int n;

unterstützt bei FXCHARTYPE-Spalten Multibyte-kodierte Werte.

Klarstellung: Die Darstellung ist auf eine Zeichenfolge mit maximal 512 Bytes beschränkt (vormals: 255 Zeichen).

char *ch_display_columns(chp, tp, n)
choice *chp; TupelType tp; int n;

unterstützt bei FXCHARTYPE- und FXGRAPHICSTYPE-Spalten Multibyte-kodierte Werte.

Klarstellung: Die Darstellung ist auf eine Zeichenfolge mit maximal 512 Bytes beschränkt (vormals: 255 Zeichen).

BOOLEAN fis_empty(f)
field *f;

unterstützt bei FXCHARTYPE- und FXGRAPHICSTYPE-Spalten Multibyte-kodierte Werte und die Einstellung der Ressource TrailingNonASCIIBlanksMode.

char *fxlapptxt(n)
long n;

liefert eine Multibyte-kodierte Zeichenfolge.

char *fxlsystxt(n)
long n;

liefert eine Multibyte-kodierte Zeichenfolge.

int layout_getchar(objp, y, x)
obj *objp; int y; int x;

liefert einen Zeichen- oder Semigrafikcode aus dem Zeichenvorrat.

BOOLEAN m_init_varbuttons(mskp, icnt, placement, z, s_from, s_to)
mask *mskp; int cnt; long placement; int z; int s_from; int s_to;

liefert FALSE, wenn der Bereich *s_from* bis *s_to* ein Fullwidth-Zeichen zerschneidet.

HPAINTAREA pa_declare(pa)
paintarea *pa;

liefert HPAINTAREA_INVALID, wenn pa->pa_token, pa->pa_text oder pa->pa_tooltiptext fehlerhafte Multibyte-Sequenzen enthalten.

Der Platz, der für eine Paintarea verwendet wird, ergibt sich nicht mehr aus der Länge des Tokens. Vielmehr ist der Platzbedarf der Zeichen relevant (siehe fx_wcwidth()).

HPAINTAREA pa_put_type(type, objp, pos_y, pos_x, token, text, tooltiptext, attr, align, bt_mask, longval1, longval2, longval3, longval4, ptrval1, ptrval2);
short type; obj *objp; int pos_y; int pos_x; char *token; char *text; char *tooltiptext; int attr; int align; int bt_mask; long longval1; long longval2; long longval3; long longval4; char *ptrval1; char *ptrval2;

liefert HPAINTAREA_INVALID, wenn *token*, *text* oder *tooltiptext* fehlerhafte Multibyte-Sequenzen enthalten.

Der Platz, der für eine Paintarea verwendet wird, ergibt sich nicht mehr aus der Länge des Tokens. Vielmehr ist die Platzbedarf der Zeichen relevant (siehe fx_wcwidth()).

BOOLEAN pa_update(h_pa, pa, validmask)
HPAINTAREA h_pa; paintarea *pa; unsigned long validmask;

liefert FALSE, wenn pa->pa_token, pa->pa_text oder pa->pa_tooltiptext fehlerhafte Multibyte-Sequenzen enthalten.

Die Länge des neuen Tokens darf sich von der des alten Tokens unterscheiden. Jedoch müssen beide den gleichen Platzbedarf aufweisen (siehe fx_wcwidth()).

int undcgetstr(str)
char *str;

erlaubt im Argument *str* nur ASCII-Zeichen.

U.a. folgende Funktionen unterstützen in Verbindung mit dem Datentyp FXCHARTYPE oder FXGRAPHICSTYPE ebenfalls Multibyte-kodierte (Literale für) Werte:

- char *fgetstring(field *f)
  und allgemein alle Funktionen, die den Wert eines FXCHARTYPE- oder FXGRAPHICSTYPE-Feldes auslesen
- BOOLEAN fputstring(field *f, char *str)
  und allgemein alle Funktionen, die den Wert eines FXCHARTYPE- oder FXGRAPHICSTYPE-Feldes setzen
- interne Funktion get_record()

Bei folgenden Funktionen sollten die grün hervorgehobenen Parameter Multibyte-kodierte Argumente bzw. Resultate akzeptieren, i.d.R. jedoch, ohne deren Korrektheit zu prüfen:

- void arginspect(argcp, argv)
  int *argcp; char *argv[];
  außer argv[0].
- int ch_fill_by_args(chp, argc, argv, empty)
  choice *chp; int argc; char **argv; BOOLEAN empty;
- int execute_cmd(cmdstring, flags)
  char *cmdstring; int flags;
- void fastexit(status, txt)
  int status; char *txt;
- void fflash(type, fmt, …)
  int type; char *fmt;
- int flash(type, txt, rzeile, rspalte)
  int type; char *txt; int rzeile; int rspalte;
- void fset_tooltiptext(f, text)
  field *f; char *text;
- int fxExecProgram(cmdstring)
  char *cmdstring;
- void fx_sql_error(errcode, cursornr, fct, txt)
  long errcode; int cursornr; int fct; char *txt;
- void fx_sql_test(txt)
  char *txt;
- void fx_sql_undef(cursornr, fct, txt)
  int cursornr; int fct; char *txt;
- void fxinit(argc, argv)
  int argc; char **argv;
  außer argv[0].
- BOOLEAN jn_msg(type, fmt, …)
  int type; char *fmt;
- void l_msg(type, fmt, …)
  int type; char *fmt;
- BOOLEAN layout_putstr(mskp, y, x, s, display)
  mask *mskp; int y; int x; char *s; BOOLEAN display;
- BOOLEAN m_put_varbutton(mskp, posnr, variant, token, text, tooltiptext, attr, align, bt_mask)
  mask *mskp; int posnr; int variant; char *token; char *text; char *tooltiptext; unsigned char attr; unsigned char align; unsigned char bt_mask;

- Event msg(type, fmt, …)
  int type; char *fmt;
- int notrailcmp(s1, s2)
  char *s1; char *s2;
- void oflash(objp, type, fmt, …)
  obj *objp; int type; char *fmt;
- HPAINTAREA pa_declare_type(type, objp, pos_y, pos_x, token, text, tooltiptext, attr, align, bt_mask, longval1, longval2, longval3, longval4, ptrval1, ptrval2)
  short type; obj *objp; int pos_y; int pos_x; char *token; char *text; char *tooltiptext; int attr; int align; int bt_mask; long longval1; long longval2; long longval3: long longval4; char *ptrval1; char *ptrval2;
- void putstr(str)
  char *str;
- void writetty(y, x, str, clflg)
  int y; int x; char *str; int clflg;

3.3.5 Namen von Monaten und Wochentagen

Die Nachbildungen der ESQL/C-Funktionen zur Wandlung zwischen Datumswerten und Datumsliteralen unterstützen Multibyte-kodierte Abkürzungen für Monate und Wochentage (→ S_months, S_weekdays).

3.3.6 Zeichenklassifizierung

Die Funktionen zur Zeichenklassifizierung (fxisprint(), fxisalpha(), …) basieren auf den Funktionen des Betriebssystems. Die Erfahrung hat gezeigt, dass diese Funktionen nicht für alle Zeichen korrekt arbeiten. Dabei ist es meist so, dass ein Zeichen, das einer bestimmten Klasse zugeordnet ist, von der Funktion des Betriebssystems nicht als solches erkannt wird.

Zur Lösung dieses Problems ist es möglich, Ausnahmetabellen zu definieren, die die Codes der Zeichen beinhalten, die zusätzlich zu denen, die das Betriebssystem erkennt, zu einer Klasse gehören sollen. Die notwendigen Funktionen und Strukturen sind in der Datei `fxctype.h` definiert. Eine Ausnahmetabelle besteht aus einem Array der Struktur `struct coderange`. Jedes Element definiert einen von-bis-Bereich von Zeichencodes, die zusätzlich zu einer Klasse gehören sollen. Die Elemente müssen **aufsteigend sortiert** sein und dürfen keine Überlappungen definieren. Das Ende des Arrays muss durch ein Element markiert sein, das die Werte 0, 0 enthält. Um die Tabelle für eine Klasse zu definieren, ist die Funktion

```
void define_coderange (int cat, struct coderange *range)
```

aufzurufen. Sie bekommt neben dem Zeiger auf die Ausnahmetabelle eine Konstante für die Zeichenklasse als Parameter.

Die folgende Tabelle zeigt die möglichen Konstanten und die Funktionen, die die damit definierten Ausnahmetabellen verwenden:

Klasse	Funktion
CODERANGE_ALPHA	fxisalpha(), fxisalnum()
CODERANGE_PUNCT	fxispunct()
CODERANGE_PRINT	fxisprint()

Der Speicherbereich der Ausnahmetabelle, auf den *range* zeigt, darf nach dem Aufruf der Funktion nicht freigegeben werden.

Um festzustellen, ob ein Zeichen nicht zu einer Zeichenklasse gehört, kann ein *FIX*-Programm mit dem Schalter -dev gestartet werden. In diesem Fall wird für jeden Code, für den die Funktion zur Klassifizierung aufgerufen wird und FALSE liefert, die Meldung

is_in_range(): <code> not found in table <klasse>\n

ausgegeben.

3.3.7 Umgang mit unzulässigen Zeichenfolgen

An einigen Stellen ist es notwendig, Zeichen in UTF-8-Kodierung in die entsprechenden Unicode-Werte (Datentyp wchar_t) umzuwandeln und zu testen, ob die Zeichen bestimmten Anforderungen entsprechen (z.B. durch fxisprint()). Dabei können unzulässige Bytesequenzen entdeckt werden oder es kann vorkommen, dass die umgewandelten Zeichen nicht den Anforderungen entsprechen. Um für Konversionsfehler verantwortliche Zeichenfolgen aufzuspüren, kann die Anwendung mit dem Testschalter CODESET_C (-dev -test 8) aufzurufen werden. Dann werden detaillierte Fehlermeldungen ausgegeben, die nähere Hinweise (z.B. den Feldnamen) enthalten.

Beim Speichern von Zeichenfolgen mit unzulässigen Multibyte-Sequenzen in CHARTYPE- oder GRAPHICSTYPE-Spalten von Memory Relations werden diese Werte nicht übernommen. Es bleiben die bisherigen Werte erhalten.

3.4 Neue Ressourcen

▲ UseFieldInsertMode
AllowToggleInsertMode

Anwendung: Siehe *Insert-Modus* auf Seite 17.

▲ TrailingNonASCIIBlanksMode

steuert die Behandlung nachgestellter Leerzeichen.

- 0 : Nur das Zeichen mit Code 32 gilt als Leerzeichen.
- andere Werte: alle Zeichen, für die fxisblank() gilt, gelten als Leerzeichen.

Der Wert wirkt sich u.a. aus:

- bei der Eingabe von Werten in FXCHARTYPE-Felder,
- bei Defaultwerten von FXCHARTYPE-Feldern beim Laden von Masken,
- beim Export von Feldwerten aus Selos und Choices,
- in den Funktionen fputstring(), fputval(), fis_empty().

IMEEnabled

Anwendung: Siehe *Steuerung des frontendseitigen "Input Method Editors"* auf Seite 39.

IoopOpt
NoFullwidthAttr
SkipSyncRefresh

Optimierung des Protokolls: Siehe *FIX/Win*-Handbuch.

▲ Mfodirs
StartWithElementTable
ActiveEntryAttr
ActiveFieldAttr
IncludeFieldDelimiter
InitNewSpace
SwapAfterLastField
BeforeLoadCmd
AfterSaveMfoCmd
AfterSavePanCmd
AfterSaveMlyCmd
AfterEditCmd

led: siehe *Der Layouteditor led* auf Seite 34.

▲ SourceStyle
BytesPerCharMethod

Sourcegenerierung: siehe *Source-Generierung* auf Seite 20.

4 Der Layouteditor led

Der Layouteditor **led** wurde in der Version 3.1.0 überarbeitet. Insbesondere wurde er so modifiziert, dass er (auch) mit *FIX/Win* gestartet werden kann. Hauptgrund dafür war neben der Umstellung auf Unicode die Möglichkeit, (japanische) Unicode-Zeichen über den unter Windows vorhandenen IME einzugeben.

4.1 led unter *FIX/Win*

led unter *FIX/Win* bietet folgende Vorteile, die teilweise auch in der (Nicht-Unicode-)Version *FIX 3.1.0* von Bedeutung sind:

- Die Anzeige von Grafikzeichen stellt im Zusammenhang mit UTF-8 für die Terminalausgabe ein Problem dar, da dieser Zeichensatz keine Semigrafikzeichen im 8-Bit-Bereich beinhaltet. Zeichen im 16-Bit-Bereich werden durch das Format der Termcap-Beschreibung nicht unterstützt (Eintrag SG).

  Unter *FIX/Win* werden die Bitmaps für die Grafikzeichen dargestellt, die später auch in der Anwendung zu sehen sind.▲
- Nur unter *FIX/Win* kann **led** mit der Maus bedient werden.▲
- Der IME kann zur Eingabe von japanischen Zeichen genutzt werden.

Wenn diese Vorteile nicht von Bedeutung sind, kann **led** auch in der Version 3.1.0 weiterhin in der Terminalemulation gestartet werden.

4.2 Start von led unter *FIX/Win*

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

Auf der Seite von *FIX* muss ein Startskript erstellt werden, das die Umgebung der Anwendung aufbaut und das das Programm $FXDIRSYS/bin/led mit dem Schalter -service startet. Die Angabe einer mfo- oder men-Datei ist nicht notwendig, da die Datei über einen Dialog ausgewählt werden kann.

Dazu erscheint als erstes eine Auswahlmaske, in der ein Verzeichnis und eine Datei in diesem Verzeichnis ausgewählt werden können. Beim Verlassen des Datei-Feldes wird die Datei zur Bearbeitung geladen. Nach der Bearbeitung kehrt **led** wieder in die Dateiauswahl zurück. Dieses Verhalten ist nicht auf *FIX/Win* beschränkt. Beim Start in der Terminalemulation verhält sich **led** ab der Version 3.1.0 genauso, wenn keine Beschreibungsdatei angegeben wird.

Die möglichen Verzeichnisse können in der Ressource

Mfodirs

hinterlegt werden. Als Trennzeichen zwischen den Verzeichnissen ist das Zeichen '|' zu verwenden. Die Angabe ist relativ zu `FXHOME` zu sehen. Wird die Ressource nicht angegeben, dann werden die Verzeichnisse `mfo` und `men` verwendet.

4.3 Änderungen

Maskeneingaben

- Das Layout der Eingabemasken wurde überarbeitet. Dabei wurden die Masken zur Menübearbeitung den Masken zur Maskenbearbeitung angepasst. Bei der Maskenbearbeitung wurde die Feldreihenfolge beibehalten.▲
- Die Logik der Masken wurde überarbeitet, so dass die Felder mit der Maus angeklickt werden können.▲
- In Feldern, die Daten zugeordnet sind, die Multibyte-Zeichen zulassen, können auch Multibyte-Zeichen eingegeben werden. Dabei werden auch Fullwidth-Zeichen unterstützt. Ansonsten sind nur ASCII-Zeichen erlaubt.
- In *FIX 3.1.0* sind an den Stellen, an denen die Version *FIX 3.1.0 Unicode* nur ACSII-Zeichen erlaubt, ebenfalls nur ASCII-Zeichen erlaubt.▲
- Zum Wechsel zwischen Maskendefinition, Felddefinition und Layoutbearbeitungen können Buttons verwendet werden.▲
- Das Speichern und das Abbrechen der Abarbeitung erfolgt ebenfalls über Buttons. Hier entfällt die Ja/Nein-Abfrage.▲
- Die Feldreihenfolge kann in der Tabellenansicht über Buttons nach Zeilen oder Spalten sortiert werden.▲
- In dem Feld "Eigenschaften-Klartext", das die Eigenschaften klartextlich darstellt, können die Eigenschaften über #f9 ausgewählt werden. Dazu wird das vorherige Feld besucht und die Choice gestartet. Danach befindet sich der Eingabefokus wieder im Feld "Eigenschaften-Klartext".▲

Layoutbearbeitung

- Im Layout können Unicode-Zeichen eingegeben werden. Dabei werden auch Fullwidth-Zeichen unterstützt.
- Das aktuelle Feld wird in der Layoutbearbeitung hervorgehoben.▲
- Der Hintergrund wird unter *FIX/Win* in der Layoutbearbeitung anders gezeichnet, so dass auch bei Objekten ohne Rahmen der Bereich des Objekts optisch erkennbar ist.▲
- Bei der Eingabe eines Zeichens auf der letzten Spalte findet kein Zeilen- und Spaltenwechsel statt.▲
- Die aktuelle Position kann durch Anklicken mit der Maus bestimmt werden.▲
- Das aktuelle Feld kann durch Anklicken mit der Maus bestimmt werden.▲
- Die Zeile mit den Tastenlabels und die Ausgabe der Statuszeile in der Layoutbearbeitung wurde übersichtlicher gestaltet.▲
- Im Positionierungsmodus kann eine der vier Ecken der Maske angeklickt werden. Mit einem weiteren Mausklick wird diese Ecke und damit die Maske verschoben.▲
- Im Positionierungsmodus kann auf den (neuen) SIZE-Modus umgeschaltet werden. In SIZE-Modus kann mit Hilfe der Pfeiltasten oder durch Anklicken mit der Maus die Größe der Maske bestimmt werden. Dabei wird immer die untere linke Ecke verändert.▲
- Viele Funktionen der Layoutbearbeitung sind auch über ein Kontextmenü zugänglich. So kann beispielsweise das aktuelle Feld an eine bestimmte Mausposition verschoben werden.▲
- Während der Layoutbearbeitung zeigen Tooltips auf Feldern die wichtigsten Daten an.▲

4.4 Aufruf von Programmen zur Vor- und Nachbearbeitung

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

Durch Einträge in `fix.rc` lassen sich Programme (Shell-Skripte) definieren, die vor dem Laden und nach dem Speichern aufgerufen werden. Für diese Programme sind folgende Ressourcen mit dem Pfad des Programms zu belegen:

- BeforeLoadCmd: Vor dem Laden eines Objekts (mfo+Layout)

- AfterSaveMfoCmd: Nach dem Speichern einer mfo-Datei
- AfterSavePanCmd: Nach dem Speichern einer pan-Datei
- AfterSaveMlyCmd: Nach dem Speichern einer mly-Datei
- AfterEditCmd: Nach dem Bearbeiten eines Objekts (immer)

Vor dem Ladevorgangs wird nur ein Programm aufgerufen, weil es nicht möglich ist, nur die mly- oder mfo-Datei einzeln zu laden. Beim Speichern ist es jedoch möglich, jede Datei getrennt zu speichern. Deshalb können hier auch drei verschiedene Programme definiert werden. Das unter AfterEditCmd definierte Programm wird unabhängig davon aufgerufen, ob die Bearbeitung mit oder ohne Speichern verlassen wurde.

Wird eine Ressource nicht definiert, dann findet auch kein Aufruf statt.

Alle Programme bekommen als Parameter den Dateinamen der mfo-, mly- oder pan-Datei.

Endet der Name eines Programms mit .cmd, .bat oder .exe, dann wird für den Dateinamen das Zeichen \ als Trenner verwendet. Außerdem wird die Variable FxExecShell auf 1 gesetzt, was zur Folge hat, dass das Programm über die Funktion system() gestartet wird. (siehe auch Addendum - *FIX* für Windows).

Hinweis:

Die alte (bisher nicht dokumentierte) Umgebungsvariable LED_AFTER_PAN_SAVE wird zwar weiterhin noch unterstützt. Stattdessen sollte jedoch die neue Ressource AfterSavePanCmd verwendet werden.

4.5 Weitere Konfiguration über Ressouren

▲ Dieser Abschnitt gilt auch für *FIX 3.1.0*.

SwapAfterLastField

entscheidet, ob bei der Bearbeitung der Elemente nach dem Verlassen des letzten Feldes der Detailansicht zur tabellarischen Übersicht zurückgekehrt wird (TRUE) oder nicht (FALSE). Der Default ist TRUE, so dass das bisherige Verhalten verwendet wird.

StartWithElementTable

entscheidet, ob zu Beginn die tabellarische Übersicht (TRUE) oder die Detailansicht (FALSE) zur Bearbeitung der Elemente angezeigt wird. Der Default ist TRUE, so dass das bisherige Verhalten verwendet wird.

InitNewSpace

entscheidet, ob beim Vergrößern der Maske im Position Mode durch das Anlegen neuer Zeilen oder Spalten diese initialisiert werden (TRUE) oder ob der vom Entfernen der Zeilen (Spalten) vorhandene Speicher erhalten bleibt (FALSE). Der Default ist TRUE, so dass das bisherige Verhalten verwendet wird. Wenn die Ressource auf FALSE gesetzt wird, kann die Maske im Position Mode verkleinert und wieder vergrößert werden, ohne dass die im Layout vorhandenen Zeichen zerstört werden.

ActiveFieldAttr / ActiveEntryAttr

legt das Attribut fest, das zur Darstellung des aktuellen Feldes bzw. Menüpunkts im WYSIWYG-Modus verwendet wird. Der Wert für ein bestimmtes Attribut kann in `video.h` nachgesehen werden. Als Default wird INVERS verwendet.

IncludeFieldDelimiter

entscheidet, ob die Begrenzer zum Feld gehören (TRUE) oder nicht (FALSE). Je nachdem wird beim Positionieren und Anklicken im WYSIWYG-Modus der linke Begrenzer als erste Position angesehen oder nicht. Der Default ist FALSE, so dass das bisherige Verhalten verwendet wird.

4.6 Formate und Zeichensätze

led verwendet zum Abspeichern von Nicht-ASCII-Zeichen in Layoutbeschreibungen immer die oktale Ersatzdarstellung . Zum Abspeichern von Nicht-ASCII-Zeichen in der mfo-Datei werden je nach Angabe des Kommandozeilenschalter -8 oder -7 8-Bit-Zeichen oder die Ersatzdarstellungen verwendet.

In *FIX 3.1.0 Unicode* wirken sich die Schalter -7 und -8 beim Starten von **led** auch auf die pan-Datei aus. Der Schalter -7 bewirkt die Verwendung der Ersatzdarstellung in mfo- und pan-Datei. Die Angabe von -8 oder keine Angabe bewirkt die Verwendung von 8-Bit-Zeichen.

Die Angabe des Zeichensatzes der pan- und der mly-Datei (neues Format) wird von **led** in *FIX 3.1.0 Unicode* berücksichtigt. Dazu werden intern die Zeichen in den Unicode-Zeichensatz konvertiert. Grafikzeichen werden dabei nicht konvertiert, sondern nur auf 16-Bit erweitert. Grund dafür ist, dass durch die Konvertierung Codes entstehen würden, die größer als 255 sind und von *FIX/Win* derzeit nicht unterstützt werden. Außerdem wären bei einer Umsetzung die Codes der Grafikzeichen zeichensatzabhängig. Beim Schreiben werden die intern verwendeten Unicode-Zeichen in den Zeichensatz umgesetzt, der beim Lesen verwendet wurde. Wurden während des Editierens Zeichen erzeugt, die nicht in diesen Zeichensatz konvertiert werden können, dann wird eine Fehlermeldung ausgegeben und das Zeichen wird durch ein ? ersetzt.

5 Tools zur Layoutbehandlung

Folgende von *FIX* 3.0.0 gewohnte Tools sind in *FIX 3.1.0* und *FIX 3.1.0 Unicode* nicht mehr enthalten:

- **pan2mly**: diente bis *FIX* 3.0.0 zum Konvertieren von Layoutbeschreibungen in die alte binäre Form.
- **mlycheck**: diente bis *FIX* 3.0.0 zum Validieren von Layouts in der alten binären Form.
- **mly2pan**: diente bis *FIX* 3.0.0 zum Konvertieren von Layouts in der (alten) binären Form in Layoutbeschreibungen.

Ihre Funktionalität wird durch neue Tools abgedeckt.

5.1 panconv

panconv dient zum Konvertieren von Layoutbeschreibungen in die (neue oder alte) binäre Form.

panconv [-t *outformat*] [-ocs *outcharset*] [-o *outfile*] [*panfile*]

panconv erwartet, dass die Datei *panfile* in dem bei der Installation von *FIX* gewählten Zeichensatz (→ FXCHARSET) abgefasst ist. Die Angabe in der Datei selbst wird - analog zum Runtime von *FIX* - derzeit nicht berücksichtigt.

Wird *panfile* weggelassen oder ist *panfile* gleich -, so wird von stdin gelesen.

Wird der Schalter -o nicht angegeben oder ist *outfile* gleich -, so wird das Ergebnis (binär) nach stdout geschrieben.

Mittels des Schalters -t kann bestimmt werden, ob das bisherige binäre Format (-t old) oder das neue binäre Format (-t 310) erzeugt werden soll. Fehlt die Angabe, wird -t 310 angenommen.

Mittels des Schalters -ocs kann ein vom Quellzeichensatz abweichender Zielzeichensatz festgelegt werden. *outcharset* muss einer der Werte ISO-15, ISO-2, IBM437, GERMAN7 oder – nur in Verbindung mit dem neuen binären Format – UCS-2 sein (Statt UCS-2 kann auch der Wert UTF-8 angegeben werden). Ohne die Angabe wird als Zielzeichensatz der für die Quelldatei angenommene Zeichensatz verwendet.

Hinweise:

- In der Version ohne Unicode-Unterstützung wird **panconv** weder das Lesen UTF-8-kodierter Quelldateien noch das Schreiben UCS-kodierter Zieldateien beherrschen.
- In der Version mit Unicode-Unterstützung kann **panconv** derzeit nur UTF-8-kodierte Quelldateien lesen.

5.2 mlyconv

mlyconv dient zum Konvertieren von Layouts in alter oder neuer binärer Form in Layoutbeschreibungen. Das Format wird mit Hilfe der Magic Number automatisch erkannt.

mlyconv [-t pan] [-7 | -8] [-ocs *outcharset*] [-o *outfile*] [-cs *charset*] -h *height* -w *width mlyfile*

mlyconv verlangt eine Datei als Eingabe, d.h. das Lesen des Quelllayouts aus einer Pipe ist nicht möglich. **mlyconv** erwartet, dass die Datei *mlyfile* in dem bei der Installation von *FIX* gewählten Zeichensatz (→ FXCHARSET) abgefasst ist; siehe aber Schalter -cs. Die Angabe in der Datei selbst (beim neuen binären Format) wird - analog zum Runtime von *FIX* - derzeit nicht berücksichtigt.

Wird der Schalter -o nicht angegeben oder ist *outfile* gleich -, so wird das Ergebnis nach stdout geschrieben.

Der Schalter -t ist für künftige Erweiterungen gedacht.

Neben der Breite muss auch die Höhe angegeben werden; **mlyconv** ergänzt für die Berechnung fehlende bzw. ignoriert überzählige Positionen im Layout.

Mittels des Schalters -cs kann ein - von dem bei der Installation von *FIX* gewählten Zeichensatz (→ FXCHARSET) abweichender - Zeichensatz festgelegt werden, den **mlyconv** beim Einlesen von *mlyfile* zu Grunde legen soll. *charset* muss einer der Werte ISO-15, ISO-2, IBM437, GERMAN7 oder UCS-2 sein. (Letzterer macht i.d.R. nur in Verbindung mit dem neuen binären Format Sinn. Statt UCS-2 kann auch der Wert UTF-8 angegeben werden.)

Mittels des Schalters -ocs kann ein vom Quellzeichensatz abweichender Zielzeichensatz festgelegt werden. *outcharset* muss einer der Werte ISO-15, ISO-2, IBM437, GERMAN7 oder UTF-8 sein.[1] Ohne die Angabe wird als Zielzeichensatz der für die Quelldatei angenommene Zeichensatz verwendet.

Standardmäßig werden – analog zu dem früheren **mly2pan** - das (die) Byte(s) eines Zeichens mit einem Code oberhalb von 127 in oktaler Ersatzdarstellung geschrieben:

Beispiel: (UTF-8) Ä → \303\204 (8 Bytes), Ö → \303\226, …

Mittels des Schalters -8 kann eingestellt werden, dass stattdessen Bytes mit gesetztem höchstwertigem Bit ausgegeben werden:

Beispiel: (UTF-8) Ä → 2 Bytes mit Oktalwerten 303 und 204, Ö → 2 Bytes mit Oktalwerten 303 und 226, …

Beim Zielzeichensatz UTF-8 wird in die Layoutbeschreibung als Versionsnummer 310 (statt 293) eingetragen. **mlyconv** schreibt statt “unspecified” den Namen des Zielzeichensatzes in die Datei. Diese Angabe dient allerdings nur zur Dokumentation.[2]

Hinweise:

- In der Version *ohne* Unicode-Unterstützung wird **panconv** weder das Lesen UTF-8-kodierter Quelldateien noch das Schreiben UTF-8-kodierter Zieldateien beherrschen.
- Der Kompatibilität zu **mly2pan** wegen validiert **mlyconv** die gefundenen Zeichencodes nicht, wenn der Zielzeichensatz mit dem Quellzeichensatz übereinstimmt.

6 Fullwidth-Zeichen

In ostasiatischen Sprachen existieren Zeichen, die doppelt so breit sind wie ein normales Zeichen. Diese Zeichen werden Fullwidth-Zeichen genannt. Die normalen Zeichen werden im Gegensatz dazu Halfwidth-Zeichen genannt. Für Fullwidth-Zeichen reserviert *FIX* im internen Bildschirm zwei Zellen und schreibt den Code des Zeichens in beide Zellen.

Um die Breite eines Zeichens festzustellen, wird die Funktion

1. Derzeit kann als Zielzeichensatz nur UTF-8 oder der Quellzeichensatz gewählt werden.
2. Das Runtime von *FIX 3.1.0* wertet die Zeichensatzangabe in der Datei beim Einlesen nicht aus, sondern erwartet, dass die Datei in dem bei der Installation von *FIX* gewählten Zeichensatz abgefasst ist (→ FXCHARSET).

```
fx_wcwidth()
```

verwendet. Erfahrungen haben gezeigt, dass die Betriebssystemfunktion wcwidth() nicht korrekt arbeitet. Wenn die Schriftarten von Windows untersucht werden, ist zu erkennen, dass ein und dasselbe Zeichen in manchen Schriftarten doppeltbreit ist und in manchen halbbreit. Aus diesem Grund verwendet *FIX* nicht die Funktion wcwidth(). Stattdessen wird eine Nachbildung verwendet, die auf den Daten einer Schriftart basiert. Die Daten stehen in Form einer Fontinfo-Datei bereit. Diese Art von Dateien kann mit dem Tool **FontInfo**, das zusammen mit *FIX/Win* ausgeliefert wird, erzeugt werden. Zur Verwendung einer Fontinfo-Datei unter *FIX* ist sie auf dem entsprechenden Rechner zu installieren. Einige oft benötigte Dateien werden zusammen mit *FIX 3.1.0 Unicode* ausgeliefert (`etc/unicode/*.ffi`). Damit *FIX* die Datei findet, ist der Pfad der Datei in der Umgebungsvariablen

```
FXFONTINFO
```

zu hinterlegen. Wenn die Datei fehlerhaft ist oder nicht gelesen werden kann, wird eine Meldung ausgegeben und die Anwendung wird beendet. Wenn die Variable FXFONTINFO nicht gesetzt ist, gelten alle Zeichen als halbbreit.

Beim Schreiben von Zeichen in den virtuellen Bildschirm tritt das Problem auf, dass dadurch eine Zeichenzelle eines aus zwei Zeichenzellen bestehenden Fullwidth-Zeichens überschrieben werden kann und somit ein unvollständiges Fullwidth-Zeichen zurückbleibt. Dieses Problem tritt häufig dann auf, wenn *FIX*-Objekte übereinander gelegt werden und wieder entfernt werden. *FIX* prüft deshalb alle Zeichenzellen vor der Ausgabe des virtuellen Bildschirms auf den realen Bildschirm und behebt diese Probleme im virtuellen Bildschirm. Dabei werden folgende Situationen behandelt:

- Zu einem Fullwidth-Zeichen fehlt der hintere oder der vordere Teil. In diesem Fall wird das unvollständige Fullwidth-Zeichen durch ein Leerzeichen (Code 32) ersetzt.
- Der rechte Teil eines Fullwidth-Zeichens hat einen anderen Zeichencode als der linke Teil. In diesem Fall werden beide Teile durch den Code 32 (Leerzeichen) ersetzt.
- Der rechte Teil eines Fullwidth-Zeichens hat ein anderes Attribut als der linke Teil. In diesem Fall wird das Attribut des linken Teils in den rechten Teil geschrieben.

Dieses Verfahren führt dazu, dass Fullwidth-Zeichen entfernt werden, wenn sie nicht vollständig sind. Wenn sich beispielsweise ein Menü über eine Maske legt und dadurch nur noch der erste Teil eines Fullwidth-Zeichens vorhanden ist, dann wird dieser Teil durch ein Leerzeichen entfernt. Damit sind keine halben Fullwidth-Zeichen zu sehen.[1] Da beim Entfernen eines *FIX*-Objektes alle anderen Objekte nochmals ausgegeben werden, wird das Fullwidth-Zeichen dann wieder restauriert.

Ähnliche Probleme treten beim Schreiben von Zeichen und Attributen in ein Layout auf. Deshalb sind die Funktionen layout_setattr() und layout_setchar() mit Vorsicht einzusetzen. Beim Schreiben eines Fullwidth-Zeichens müssen immer zwei Zeichenzellen mit dem gleichen Code und dem gleichen Videoattribut beschrieben werden. Zusätzlich muss im Attribut gekennzeichnet werden, ob es sich um den ersten oder um den letzten Teil eines Fullwidth-Zeichens handelt. Dazu sind die Werte FIRSTFULLWIDTHCOL und LASTFULLWIDTHCOL (aus `video.h`) mit dem Attribut zu verknüpfen. Damit durch das Schreiben eines Zeichens kein unvollständiges Fullwidth-Zeichen entsteht, sollte außerdem vor dem Schreiben (egal ob Fullwidth oder Halfwidth) geprüft werden, ob sich an der betreffenden Stelle ein Fullwidth-Zeichen befindet. Wenn ja, dann ist dieses komplett zu überschreiben. Beim Einsatz von layout_putstr() wird das Schreiben von Fullwidth-Zeichen mit den richtigen Attributen von *FIX* übernommen. Jedoch ist auch hier darauf zu achten, dass am Anfang und am Ende des Bereichs kein unvollständiges Fullwidth-Zeichen entsteht.

7 Steuerung des frontendseitigen “Input Method Editors”

FIX macht es vom Wert eines internen Zustands und von der Einstellung der Ressource IMEEnabled abhängig, ob bei Benutzung eines grafischen Frontends diesem zusätzlich übermittelt wird, wann der “Input Method Editor” aktiviert sein soll. Der interne Zustand besitzt die Vorbelegung FE_IME_FIX_HANDLED[2] und kann mittels der Funktionen

```
void fx_set_ime_mode(mode)
int mode;
```

und

1. Es sei denn, die Texte werden durch Paintareas dargestellt.
2. Der Input Method Editor soll nur aktiv sein, wenn ein veränderbares FXCHARTYPE-Feld besucht wird.

int fx_get_ime_mode()

gesetzt und abgefragt werden. Mögliche Werte neben FE_IME_FIX_HANDLED sind FE_IME_OFF und FE_IME_ON (vgl. `fix/fixwin.h`).

8 Bekannte Einschränkungen

FIX 3.1.0 Unicode besitzt die im Folgenden beschriebenen Einschränkungen.

Nur ASCII-Zeichen

- Einträge für die Tastaturtabelle dürfen nur ASCII-Zeichen enthalten.
- Das Programm **keyinit** kann nur korrekt mit ASCII-Zeichen umgehen.
- Die Angaben <query> bzw. <procedure_call> in Choice- und Selobeschreibungen dürfen nur ASCII-Zeichen enthalten. Beim Einlesen ersetzt *FIX* Nicht-ASCII-Zeichen durch '?', was dann i.d.R. zu einem inkorrekten Statement führt.
- Beim "Scannen" oder Aufbereiten eines Datums oder DATETIME-Wertes gemäß eines Formats werden nur Formate unterstützt, die ausschließlich ASCII-Zeichen enthalten (Dies gilt auch für die Nachbildungen der entsprechenden ESQL/C-Funktionen).
- Bei der Bruch-Darstellung von Zahlen werden nur Formate unterstützt, die ausschließlich ASCII-Zeichen enthalten.
- Für den Inhalt der Umgebungsvariablen PANEL_BREAK_BEFORE sind nur ASCII-Zeichen zugelassen. *FIX* prüft den Inhalt der Variablen nicht.
- Reguläre Ausdrücke funktionieren nur, wenn sowohl im regulären Ausdruck als auch im Eingabewert ausschließlich ASCII-Zeichen angegeben werden.
- Nicht-ASCII-Zeichen im Kommandotext behandelt *FIX* bei der Ausführung des Kommandos nicht als Metazeichen.

Fullwidth-Zeichen

- Die Funktion layout_putstr() prüft nicht, ob die Schreiboperation ein Fullwidth-Zeichen zerteilt.
- layout_setchar() und layout_setattr() überprüfen weder zu schreibende Argumente noch, ob die Schreiboperation ein Fullwidth-Zeichen zerteilt.
- Die Breitenberechnung für Hilfetext-/Meldungsfenster ist fehlerhaft, wenn der Text Tabulator- und Fullwidth-Zeichen auf einer Zeile enthält.
- Die Verwendung "positionierender" Codes wie '\b' beim Schreiben in ein Layout funktioniert nicht, wenn links von ihnen ein Fullwidth-Zeichen auftritt.

Prüfung auf Zeichensatzkonformität

- Die Funktionen fxvalcmp() und typecmp() vgl. FXCHARTYPE- bzw. FXGRAPHICSTYPE-Werte byteweise, d.h. ohne sie auf Zeichensatzkonformität zu prüfen.
- Die Funktion strapp() konkateniert ihre Argumente, ohne sie auf Zeichensatzkonformität zu prüfen.
- Die Funktionen zur Ausführungen von Kommandos prüfen den Kommandotext nicht auf Zeichensatzkonformität.
- Die Funktion putstr() schreibt ihr Argument, ohne es auf Zeichensatzkonformität zu prüfen.

Sonstige

- Die Angabe von -trace funktioniert nicht, da die Eingabe nicht in die spezifizierte Datei protokolliert wird.
- Es fehlt eine Multibyte-fähige Entsprechung zur Funktion instring(int, char *).
- Die Anwahl von Menüpunkten durch Zeichen ist nur mit (alphanumerischen) Zeichen mit Codes zwischen U+0000 und U+017F (d.h. Latin Extended A) möglich.
- Von einigen internen Funktionen (fx_fillblanks(), db_fillblanks(), fx_truncate_asc()) werden nachgestellte Nicht-ASCII-Leerzeichen nicht durch ASCII-Leerzeichen ersetzt oder abgeschnitten.

6 Umstellung einer *FIX*-Anwendung auf Unicode

Dieses Kapitel behandelt Gesichtspunkte, die bei der Umstellung einer *FIX*-Anwendung, die ursprünglich für den Zeichensatz ISO-15 entwickelt wurde, zur Nutzung mit UTF-8 zu beachten sind. Es basiert auf Erfahrungen, die bei der Umstellung der *FIX*-Demoanwendung gemacht wurden.

Datenbank, Verzeichnis `sql`

Die Datenbank sollte ein UTF-8-basiertes DB_LOCALE verwenden. Damit CHARACTER-Spalten weiter die gleiche Anzahl Zeichen aufnehmen können, sind die Spalten um den Faktor 3 zu vergrößern, da die UTF-8-Darstellung von Zeichen der Basic Multilingual Plane maximal 3 Bytes benötigt.

Das *FIX*-Programm muss in einem UTF-8-basierten CLIENT_LOCALE ablaufen.

Verzeichnis `unl`

Sofern die Datendateien im Format “ISO Code Page 8859-15” (IBM Informix GLS: e02f) vorliegen und umgesetzt werden sollen - was nicht unbedingt erforderlich ist, da beim Importieren auch das Client Locale geeignet gesetzt werden kann -, kann hierfür u.U. das Programm **iconv** nützlich sein:

```
iconv -f ISO8859-15 -t UTF-8 …
```

Die Bezeichnungen von Ausgangs- und -Ziel-Locale sind allerdings plattformspezifisch (oben: AIX 4.3). Auch müssen die zugehörigen Betriebssystemkomponenten installiert sein.

Datei `.profile`, FIX/Win-Startskripte

Hier muss - auf UNIX-Plattformen - darauf geachtet werden, dass beim Start von *FIX*-Programmen die LC_CTYPE-Einstellung ein UTF-8-konformes Locale bestimmt, z.B.

```
LC_CTYPE=DE_DE.UTF-8
```

Die Bezeichnungen für Locales sind allerdings plattformspezifisch (oben: AIX 4.3). Bei getrennten Verzeichnissen für textuelle und binäre Anteile und unterschiedlichen Plattformen kann es sich anbieten, eine im Verzeichnis FXHOMESYS abgelegte Datei einzuschleusen.

Verzeichnisse `men`, `mfo`, `cho`, `sel`, `hpd`, `chp`

Hier sind die Objektbeschreibungen und Hilfetexte, sofern sie nicht ausschließlich ASCII-Zeichen enthalten, nach UTF-8 umzusetzen. Sofern die Zeichen oberhalb des ASCII-Bereichs darin in oktaler Ersatzdarstellung angegeben sind (was bei den *FIX*-eigenen Dateien stets der Fall ist), kann hierzu das sed-Skript `fix/extra/ISO-15-_pan_to_UTF-8_pan.sed` (oder eine vereinfachte Kopie davon) verwendet werden. Anderenfalls kommt wieder **iconv** in Betracht.

In den Beschreibungsdateien von Choices und Selos müssen bei FXCHARTYPE-/FXGRAPHICSTYPE-Spalten die Längenangaben in **into**-Klauseln angepasst werden.

Verzeichnis pan

Hier ist zunächst ähnlich wie beim vorigen Punkt vorzugehen. Zusätzlich muss allerdings noch die Formatversion und die Zeichensatzangabe angepasst werden:

<Version 29*n*> → <Version 310>

<CharacterSet 'ISO-15'> → <CharacterSet 'UTF-8'>

Sofern für die Umsetzung das sed-Skript `fix/extra/ISO-15_pan_to_UTF-8_pan.sed` verwendet wird, erledigt es diese Aufgable gleich mit.

Verzeichnis `mly`

Binäre Layout-Dateien können nicht einfach weiterverwendet werden. Der unten abgedruckte Code generiert ein sh-Skript[1], das allerdings kontrolliert und ggf. nachbearbeitet werden sollte. Zu allen Objektbeschreibungen in `men`, `mfo` und `cho` bestimmt er, ob sie und welche binäre Layoutdatei sie verwenden. Ggf.

- bildet er zwei Dateinamen für eine Layoutbeschreibung und eine binäre Layoutdatei,
- entnimmt er der Objektbeschreibung die Höhe und die Breite des Objekts,
- generiert er eine Anweisung, die die erwartete Größe und die tatsächliche Größe des Originallayouts nach stderr ausgibt
- generiert er eine Anweisung, deren Ausführung zu dem Originallayout eine UTF-8-kodierte Layoutbeschreibung generiert, die mit der Dateiendung `.pan.UTF-8` in dem mittels dir= eingestellten Verzeichnis abgelegt wird (muss vor Ausführung des Skriptes angelegt werden),
- generiert er eine Anweisung, deren Ausführung zur UTF-8-kodierten Layoutbeschreibung ein binäres Layout im neuen Format generiert, das mit der Dateiendung `.mly.UTF-8` in dem eingestellten Verzeichnis abgelegt wird.

```
for f in men/*.men mfo/*.mfo cho/*.cho ; do
   awk '
NR == 1 {
     if (0 < match($0, "mly/.*[.]mly")) {
        mlyfile = substr($0, RSTART, RLENGTH)
        gen_panfile = dir "/" substr($0, RSTART + 4, RLENGTH - 8) ".pan.UTF-8"
        gen_mlyfile = dir "/" substr($0, RSTART + 4, RLENGTH - 8) ".mly.UTF-8"
     } else {
          exit 1
     }
}
NR == 2 {
     if ($1 == "function") {
        h = $3
        w = $4
     }
     else {
        h = $1
        w = $2
     }
     printf "echo \"%s: expected: %d found: `ls -l %s | awk %c{ print $5 }%c`\"
                                    >&2\n", mlyfile, h * w * 2, mlyfile, 39, 39
     printf "${FXDIRSYS}/bin/mlyconv -t pan -o\"%s\" -7 -ocsUTF-8 -csISO-15 -h%s
                                    -w%s \"%s\"\n", gen_panfile, h, w, mlyfile
```

1. Beide sollten unter AIX 4.3 ablauffähig sein.

```
      printf "env FXCHARSET=UTF-8 LC_ALL=EN_US.UTF-8 ${FXDIRSYS}/bin/panconv -t 310
                          -ocsUTF-8 -o\"%s\" \"%s\"\n", gen_mlyfile, gen_panfile
   exit 0
}
   ' dir=generated "$f"
done
```

Die unterstrichenen Stellen können/müssen ggf. angepasst werden.

Mit dem Skript können also sowohl Layoutbeschreibungen als auch binäre Layouts erstellt werden, die zu dem Originallayout äquivalent sind.

Datei `fx_texte`

Für die Klartext-Vorlage für `messages` gilt das Gleiche wie für Objektbeschreibungsdateien.

Datei `fix.rc`

Hier ist ggf. die Ressource TrailingNonASCIIBlanksMode einzustellen, die regelt, wie Nicht-ASCII-Blank-Zeichen behandelt werden sollen.

C-Quellen

Mindestens erforderlich ist:

- das Vergrößern der Datenbank- und/oder Feldhostvariablen zu FXCHARTYPE- und FXGRAPHICSTYPE-Feldern:

  char[n] → char[(n-1)∗3 + 1]

- die Berücksichtigung der in *API* auf Seite 22 beschriebenen Änderungen, insb. Anpassungen an die veränderten Header-Dateien. Empfohlen wird außerdem, in für *FIX 3.1.0* modifizierten Sourcen die Definition des Makros FIXRELEASE nach 310 zu ändern.
- die Eliminierung von nicht Zeichensatz-konformen Darstellungen von Nicht-ASCII-Zeichen (Umlaute etc.)

Sonstiges:

- Die Werte anwendungsdefinierter Events dürfen nicht mit Zeichen (0 bis 65535) oder von *FIX* vordefinierten Events (100000 +/- 400) kollidieren.