Styleguide für proNet Anwendungssysteme in Unique 4GL

Stand: 23.August 2000

proNet Informationssysteme

proNet Informationssysteme GmbH
Tel. 040/323104-70 Fax 040/323104-71
E-Mail: info@pronet-systeme.de
www.pronet-systeme.de

Inhaltsverzeichnis

1. Einleitung

Diese Broschüre dient als verbindliche Anweisung für die Entwicklung von Anwendungssystemen bei der proNet Informationssysteme GmbH, sie richtet sich aber auch an andere Entwickler, die mit Unique CONCEPT in Unique 4GL Datenbankanwendungen schreiben. Für Entwickler außerhalb der proNet Informationssysteme GmbH werden Vorschläge unterbreitet, die das Design der Datenbank und der Unique 4GL-Programme betreffen. Diese Vorschläge sind auch als solche zu verstehen und sind nicht bindend. Die Gründe für diese Vorschläge sind sehr unterschiedlich - auch was ihre Gewichtung betrifft.

Der Styleguide basiert auf dem Unique 4GL Style Guide der Unique (Deutschland) GmbH vom 3. Juni 1997 orientiert sich aber an der aktuellen Version 7.00 von Unique CONCEPT.

Diese Broschüre erhebt nicht den Anspruch auf Vollständigkeit! Wir können auch nicht garantieren, daß diese Vorschläge für künftige Versionen optimal sind.

Lesbarkeit und Wartbarkeit der Programme

Programme, die in einem einheitlichen Stil geschrieben worden sind, erleichtern es dem Entwickler selbst zu einem späteren Zeitpunkt oder natürlich auch anderen Entwicklern, die Programme zu lesen und zu verstehen. Dies spart Zeit und ärgerlichen Mehraufwand. Dieser Styleguide soll nicht nur zu einer einheitlichen Programmierweise, sondern auch zur Erstellung von gut strukturierten Programmen beitragen.

Erleichterung bei der Programmentwicklung

Bei Systemen, die von mehreren Programmierern entwickelt werden, erleichtet es sehr die Arbeit, wenn sie einen ähnlichen Programmierstil verwenden. Auch können so Programmteile von einem Programm in ein anderes kopiert werden, ohne daß aufwendige Umformatierungen oder Umbenennungen notwendig werden.

Kompabilität zu künftigen Versionen von Unique 4GL

Der Unique 4GL-Kompiler ist weitestgehend abwärtskompatibel, d.h. daß auch Programme, die mit älteren oder sehr alten Versionen von Unique 4GL entwickelt wurden, mit den neuesten Versionen von Unique 4GL bis auf wenige Ausnahmen kompiliert werden können. Der Kompiler versteht also noch die alten Programme, obwohl sich die - im Referenzhandbuch definierte - Syntax der Programmiersprache erweitert und verändert hat. Bereits jetzt gibt es aber einige Teile der Unique II-Syntax, die nicht mehr kompiliert werden können oder von denen man heute bereits sagen kann: "typisch Unique II". Diese Tendenz wird sich künftig noch verstärken. Damit die Programme also auch in künftigen Unique 4GL-Versionen ohne Änderungen kompiliert werden können, sollte die aktuelle Syntax verwendet werden.

Verwendbarkeit von Unique DOC

Soll Unique DOC zur Dokumentation eines Programmpaketes verwendet werden, sollten einige Regeln befolgt werden, um eine möglichst vollständige und lesbare Dokumentation zu erhalten. Beispiel: nur wenn für Menu-Entries Purpose-Texte definiert sind, können auch vernünftige Texte zu einem Menü-Punkt in der Dokumentation stehen. Bei einigen Punkten werden Limitationen von Unique DOC berücksichtigt. Unique DOC ist nur bis Unique CONCEPT 4.02 enthalten. Für spätere Versionen von Unique CONCEPT können Hinweise zu Unique DOC ignoriert werden.

Kompatibilität der Programme zwischen verschiedenen Datenbanksystemen

Datenbankdesign generell ist nicht Thema dieser Broschüre. Dennoch soll zumindest auf einige Punkte eingegangen werden, die berücksichtigt werden sollten, damit nicht bereits die Wahl von Feldnamen dazu führen, daß Datenbankschemata und Programme nicht zwischen den Datenbanksystemen kompatibel sind.

2. Design der Datenbank

Die **maximale Länge** der Tabellen-, Feld- und Indexnamen variiert von DBMS zu DBMS. Um die Portabilität der Datenbankstruktur zu gewährleisten, sollte die minimale maximale Länge nicht überschritten werden, wobei DB-NAME-PREFIX und ggf. automatisch hinzugefügte Zusätze (Informix Dynamic Server/SE bei Indizes) zu berücksichtigen sind.

DBMS	Max. Bezeichnerlänge
Informix C-ISAM (DOS/Windows)	8
Informix Dynamic Server	18
Informix SE	8
Microsoft SQL Server 6.5	30
Microsoft SQL Server 7.0	128
Sybase Adaptive Server	30
Sybase Adaptive Server Anywhere	128
Techra	16 (Techra ist vom Hersteller eingestellt worden und wird hier nicht mehr berücksichtigt.)
Unique ISAM	30

Auch die Menge der **erlaubten Zeichen** unterscheidet sich von DBMS zu DBMS. Zulässig sind aber bei allen Datenbanksystemen: 'A'-'Z', '0'-'9' und das Zeichen '_', wobei alle Namen mit einem Buchstaben beginnen sollten.

2.1 Tabellennamen

- Es sollten sprechende Namen verwendet werden
- Nicht länger als 8 Zeichen bei Datenbanken, die noch unter. Informix C-ISAM oder SE laufen müssen (VERTRIEB und TBOE)
- Nicht länger als 16 Zeichen bei allen anderen Datenbanken
- Bezeichnung der Tabelle als Einzahl (Beispiel: KUNDE statt KUNDEN)
- Keine Sonderzeichen in Tabellennamen
- Beginnend mit einem Buchstaben

2.2 Feldnamen

- Es sollten sprechende Namen verwendet werden
- Inklusive DB-NAME-PREFIX nicht länger als 16 Zeichen (wg. Informix OnLine und Techra)
- Keine Sonderzeichen im Namen
- Beginnend mit einem Buchstaben
- Die 16 Zeichen sollten ausgenutzt werden

2.3 Indexnamen (Gruppenfelder)

- Inklusive DB-NAME-PREFIX nicht länger als 15 Zeichen.
  Hinweis: bei Informix Online und SE wird an Indexnamen ein Suffix ('_q<n>') angehängt, wenn der Schlüsselname in der gesamten Datenbankdefinition mehrfach vorkommt. Bei Informix Online darf der komplette Name (DB-NAME-PREFIX + Name + Suffix) nicht mehr als 18 Zeichen sein.
- Keine Sonderzeichen im Namen außer Unterstrich
- Beginnend mit einem Buchstaben
- Die 15 Zeichen sollten ausgenutzt werden
- Der Name sollte aus Abkürzungen der Namen der Gruppenelementen hergeleitet sein, wobei für jedes Element dieselbe Anzahl von Zeichen verwendet werden sollte. Ein führendes Mandanten-Feld sollte nur mit einem M im Indexnamen dargestellt werden. Als Name des Hauptindexes kann auch der Tabellenname benutzt werden.

2.4 db-name-prefix

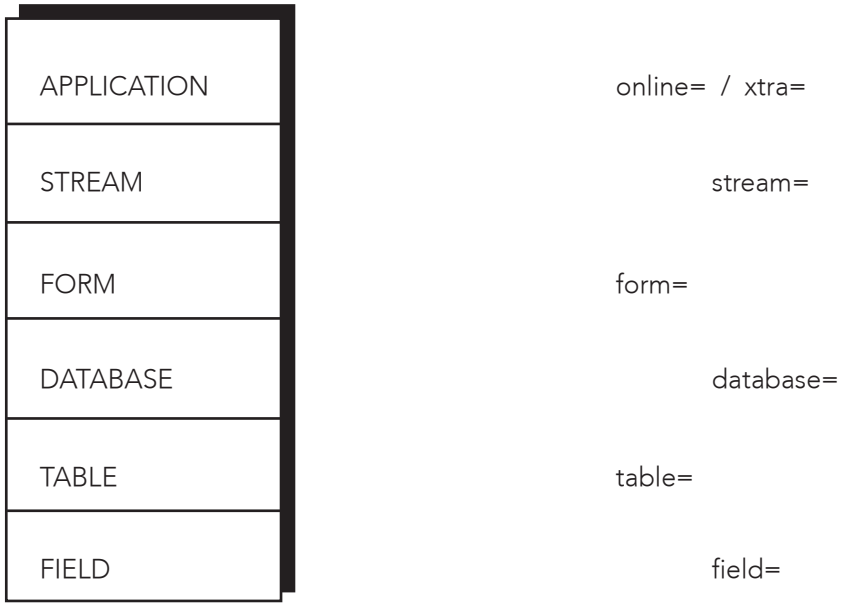
Die Verwendung eines db-name-prefix ist notwendig, wenn als Tabellen-, Feld- oder Index-Namen Begriffe verwendet werden, die Schlüsselworte in der SQL-Syntax sind und eine SQL-Datenbank verwendet wird. Die Verwendung eines db-name-prefix macht es aber notwendig, daß beim Zugriff auf die Datenbank mit einem anderen Programm für alle Datenbankobjekte dieser Prefix vorangestellt werden muß.

- Aus diesem Grunde sollte kein db-name-prefix verwendet werden. Deswegen sollten als Tabellen-, Feld- oder Indexnamen keine SQL-Schlüsselbegriffe verwendet werden (z.B. database, table, column, index, unique, create, ...).
- Wird ein db-name-prefix verwendet, sollte er aus zwei Zeichen bestehen, einem Buchstaben und dem Unterstrich. Beispiel: 'u_'

3. Design der Unique 4GL-Programme

3.1 Positionierung von Funktionen und Deklarationen

Allgemeiner Aufbau eines Unique 4GL-Programmes



- Deklarationen und Eigenschaften (speziell on-Deklarationen) müssen auf Ebene der Komponente definiert werden, zu der sie gehören. Beispiel: on (@START) wird nur auf für die Applikationskomponente (d.h. nach der online-/xtra-Definition und vor der ersten STREAM-Definition) definiert und nicht z.B. auf Ebene der database-Komponente.

Reihenfolge von Eigenschaften, Deklarationen, on-Deklarationen und Funktionen auf einer Ebene

- Die Reihenfolge sollte sein:

  1. Eigenschaften/Deklarationen (deren Reihenfolge sollte in allen Programmen einheitlich gewählt werden)
  2. on-Deklarationen
  3. Funktionen (deren Reihenfolge bestimmt die Ausführungszeitpunkte der Funktionen)

  Beispiel:

```
field="Feld"                    display(X(10))
                                       join(table("Tabelle"))
```

  statt

```
field="Feld"                    join(table("Tabelle"))
                                       display(X(10))
```

- Zur optischen Trennung kann zwischen Deklarationen und Funktionen eine Leerzeile eingefügt werden.

Verwendung von declare/enddeclare

- on DECLARE/ENDDECLARE darf nicht mehr verwendet werden.

Predeklaration von gejointen Tabellen

- Alle gejointen Tabellen sollten inklusive der zugehörigen Tabellendeklarationen vor der ersten Haupttabelle definiert werden (Predeklaration). Dies gilt auch, wenn dies nicht unbedingt notwendig ist. Diese Deklarationen beinhalten zumindest den key-field-Paramter.

- **join(table=..., key-field=..)** ist Unique II.

Verwendung von Include-Dateien

- Include-Dateien sollten auf der Ebene verwendet werden, dem der Inhalt zugehört.

3.2 Schreibweise von Funktionen, Deklarationen und deren Parametern

Abkürzungen

Unique 4GL gestattet es, Schlüsselworte abzukürzen, solange das Schlüsselwort eindeutig ist. Dies fördert nicht unbedingt die Lesbarkeit der Programme. Deswegen empfehlen wir:

- Schlüsselworte sollten grundsätzlich gemäß Syntaxbeschreibung in dem aktuellen Referenzhandbuch geschrieben und nicht abgekürzt werden. Beispiel:

  initial-value statt **init** oder **init-val** oder **initial**
  dialogue-box statt **dialog** oder **dia** oder **dialogue**

  Hinweis: unvollständige Schreibweisen können dazu führen, daß in künftigen Versionen ein Schlüsselwort nicht mehr eindeutig ist!

- In älteren Versionen sind häufig noch andere Schreibweisen üblich. Richtlinie sollte die Syntaxbeschreibung des aktuellen Referenzhandbuches sein (Vorsicht: die Beispiele im Referenzhandbuch verwenden häufig noch die alte Schreibweise!). Beispiele:

  field statt **field-name**
  database statt **data-base-name**
  table statt **table-name**
  join statt **join-key**
  column statt **column-name**

- Die Speicherformate als Parameter der storage-Deklaration sollten abgekürzt werden. Beispiele:

  storage(A(10)) statt **storage(ALPHANUMERIC(10))**
  storage(I4) statt **storage(INTEGER4)**
  storage(P D(3,2)) statt **storage(PACKED DECIMAL(3,2))**

Verwendung von Klammern, Gleichheitszeichen und Leerzeichen

Unique 4GL läßt es häufig zu, alternativ Klammern, Gleichheitszeichen oder Leerzeichen zu verwenden.

- Gleichheitszeichen sollten nur bei den Deklarationen der Hauptebenen verwendet werden. Beispiel:

```
xtra=...
online=...
stream=...
form=...
database=...
table=...
field=...
```

- Bei allen anderen Deklarationen und Funktionen sollten die Parameter in Klammern eingeschlossen werden. Beispiele:

```
scroll(source(...), destination(...))
join(table(...))
dialogue-box(text(...), buttons(...), result(...))
access(...)
column(...)
if (...)
on (...)
lb()
```

- Einige Funktionen und Deklarationen benötigen keine Parameter. Trotzdem kann man nach dem Schlüsselwort Klammern schreiben. Diese Klammern sind notwendig, wenn diesem Schlüsselwort weitere Funktionen oder Deklarationen in derselben Zeile folgen sollen. Beispiele:

key()		statt	key
static()	statt	static	
mandatory()		statt	mandatory

Groß-/Kleinschreibung

- Die Namen von Komponenten und Funktionen sowie deren Eigenschaften bzw. Parameter sollten nur in Kleinbuchstaben geschrieben werden.

- Die Werte dieser Parameter, die Schlüsselworte in Unique 4GL sind, sollten in Großbuchstaben geschrieben werden. Beispiel:

```
type(PUSH-BUTTON)
lock(ON)
colour(TEXT)
storage(I4)
display(X(10))
on (ZOOM)
access(QUERY)
```

- Die Namen von internen Feldern, Operatoren, Konstanten und Datenbank-Elementen sollten in Großbuchstaben geschrieben werden. Beispiel:

```
"@APPLICATION_PATH"
"@STREAM-1"
DATE('YYYYMMDD')
UPPERCASE("Feld")
$STREAMFARBE
db-table(KUNDE)
column(FELDNAME)
```

- Namen der online/xtra/stream/form/database/table/field-Deklarationen sollten so geschrieben werden, wie es der deutschen Sprache entspricht. Beispiele:

```
xtra="Kundenliste erstellen"
stream="Kundenliste"
form="Vor Kunden"
field="KundenNr"
field="KundName"
```

Speicherformate und Anzeigeformate

Speicherformate und Anzeigeformate werden immer groß geschrieben. Nur konstante Texte in Anzeigeformaten können im gewünschten Format geschrieben werden.

Beispiele:

```
storage(I4)
storage(P D(3,2))
display(X(10))
display(ZZZ9.99'DM')
display(+ZZ9.9'km')
```

Konstanten

- Konstanten dürfen keine Ziffern im Namen enthalten, da eine Ziffer am Ende einer Konstante die Länge des Konstantenwertes angibt.

3.3 Benennungen

Applikation

- Der Name sollte die Hauptaufgabe des Programmes beschreiben. Die Namen sollten nicht länger als 20 Zeichen sein. Beispiele:

```
xtra="Kundenliste"
online="Kundenstammdaten"
```

Stream

- Sollte beschreiben, um was es geht, und in Online-Programmen denselben Namen wie das Programm haben. Beispiele:

```
stream="Kundenliste"
stream="Kundenstammdaten"
```

Form

- Der Name sollte sich auf den Ausführungszeitpunkt der Form beziehen oder den Inhalt der Form beschreiben. Der Name sollte der TITLE-Deklaration der Form entsprechen. Beispiele:

```
form="Vor Kunden"
form="Jeder Kunde"
form="Nach Kunden"
form="Seitenkopf"
form="Übertrag"
form="Funktionen"
```

Field

- Es sollten sprechende Namen verwendet werden.

Umbenennungen von Tabellen und Datenbankfeldern

- Werden Tabellen mehrfach verwendet, so sollten alle Felder einer logischen Tabelle mit demselben Prefix umbenannt werden. Beispiel:

```
table="KunKunde"                    db-table(KUNDE)
  field="KunKundenNr"          column(KUNDENNR)
  field="KunKundName"          column(KUNDNAME)
```

Verwendung von "

- Sollte grundsätzlich für alle Namen verwendet werden, die mit online/xtra/stream/form/database/table/field definiert werden, auch wenn keine Sonderzeichen im Namen verwendet werden. Datenbankobjekte sollten in db-table und column-Deklarationen nicht in " eingeschlossen werden. Beispiele:

```
table="Kundentabelle"db-table(KUNDE)
field="Kunden-Nummer" column(KUNDENNR)
```

3.4 Einrückungen

- online/xtra/stream/form/database/table-Definitionen beginnen mit der ersten Spalte.
- field-Definitionen beginnen mit der dritten Spalte (sind es versteckte Felder steht die öffnende Klammer auf der zweiten Spalte).
- Deklarationen und Funktionen auf Form-, Tabellen- oder Feldebene beginnen auf einer Tabulator-Position um Spalte 30 herum. Zu empfehlen ist Spalte 33, weil dies eine Standard-Tabulatorposition seit Version 5.00 ist.
- Deklarationen auf Applikations- oder Streamebene beginnen mit Spalte 3.
- Funktionen innerhalb von on/if/loop/scroll/group-by-Strukturen werden um 2 Spalten eingerückt.

3.5 Kommentare

Programmkopf

- Im Programmkopf sollte der Name der Applikation, eine kurze Beschreibung und eine Liste der durchgeführten Änderungen mit Datum und Autor (Standardname für Benutzer- und E-Mail-Konto) stehen. Am einfachsten sollte die Datei $UNIQUE_HOME\setup\filehead.tpl entsprechend angepasst werden, so dass neue Programme automatisch diesen Programmkopf erhalten (seit Version 6.01-02).

```
************************************************************************ (80 Zeichen)
* Programm     : ...
* Modul        : ...
* Beschreibung : ...
*                ...
* Erstellt am  : ...
* Autor        : ...
************************************************************************
* Änderungen
*
* Datum      Autor      Beschreibung
* 15.08.2000 mschoebel  ........  ..............
* .......... ..........  ..............
************************************************************************
```

Innerhalb des Programmes

- Kommentare, die sich nur auf eine Programmzeile beziehen, sollten an das Ende der zu kommentierenden Programmzeile geschrieben werden, sofern der Platz reicht. Ansonsten in eine Zeile zuvor.

- Längere Kommentare (über eine Zeile) sollten vor den zu kommentierenden Programmzeilen stehen und auf der Spalte beginnen, auf der auch die folgende Prgrammzeile beginnt.

- Kommentare sollten mit zwei Sternen beginnen, um bei der Textsuche von Mulitplikation zu unterscheiden.

- Alle Programmänderungen nach der Freigabe der ersten Version müssen nicht nur im Kopf, sondern auch innerhalb des Programmes mit Datum, Autor und Beschreibung kenntlich gemacht werden.

  Beispiel:

```
** 15.08.2000 jhjacke update in insert geändert
insert("Kunde")
```

3.6 Besonderheiten

Verwendung von Kompiler-Konstanten

- Kompiler-Konstanten sind sprechender als deren numerische Werte. Deshalb sollten sie verwendet werden, wo es möglich ist. Beispiel:

  status("Kunde")=$NOT_FOUND statt status("Kunde")=1

Verwendung von "" (Referenz auf das aktuelle Feld)

- Kann verwendet werden, wenn möglich.

repeat/endrepeat

- repeat x times muß vor den zugehörigen Table-Deklarationen stehen (Spalte 1). Beispiel:

```
repeat 5 times
table="Kunde"
  field="KundenNr"
  ...
endrepeat
```

Display-Code in der Form-Beschreibung

- Nicht verwenden. Auch Unterstriche können zu Problemen führen.

Verwendung von +

- Nicht verwenden. Stattdessen die Funktion auf Tabellenebene definieren. Beispiel:

```
table="Kunde" join(table("Ort"))
                                ...
  field="PLZ"  join(table("ORT"))
```

statt

```
table="Kunde" ...
  field="PLZ"  +join(table("ORT"))
```

Verwendung von compute()/assign()

- Wertzuweisungen können sowohl über compute() wie assign() ausgeführt werden. compute() soll nur benutzt werden, wenn auch die Feldfunktionen auf dem Zielfeld ausgeführt werden sollen.

3.7 Für Unique DOC

Falls die Applikationen mit Unique DOC dokumentiert werden sollen, sind u.a. folgende Punkte zu beachten:

- Bei Datenbank-Feldern sollte als Text in der Maske das Heading des Feldes benutzt werden.
- Der Feldname von sichtbaren Arbeitsfeldern sollte gleich dem Text in der Maske sein. Der Feldname von menu-entries sollte gleich dem Text des Menüpunktes sein.
- Zu sichtbaren Arbeitsfeldern immer purpose angeben (auch menu-entries, push-buttons, ...)
- Der Applikationsname (online="..." / xtra="...") darf nicht länger als 20 Zeichen sein.
- Der Applikationsname darf keinen Schrägstrich (/) enthalten.
- Purpose-Texte, die sich nicht in einer Help-Datei befinden, dürfen höchstens 500 Zeichen lang sein.
- Es sollte höchstens eine komplexe Applikation in einer Source-Datei gespeichert werden. Die zugehörige Applikationshilfe sollte in der Help-Datei abgelegt werden. Dann darf der Hilfstext auch länger als 500 Zeichen werden.
- Join von statischen Feldern vermeiden.

- Die Schreibweise einiger Namen von Funktionen und Deklarationen hat sich geändert. Beispiele:

```
field                          statt    field-name
database                               statt    data-base-name
table                          statt    table-name
column                                 statt    column-name
join                           statt    join-key
```

- Einige Deklarationen der key und der join-Funktion sollten nicht mehr als Parameter der Funktion, sondern als Tabellendeklarationen stehen. Beispiel:

```
table="Tabelle"                statt    join(table("Tabelle"),key-field("Key-Field"),
  key-field("Key-Field")                      key-column("Key-Column"),store-after-key(ON),
  key-column(KEY-COLUMN)                      access(UPDATE))
  store-after-key(ON)
  access(UPDATE)
```

- Innerhalb der Form-Beschreibung hat sich die Syntax für Include-Dateien geändert. Bei Include-Dateien nicht die Dateierweiterung anfügen. Beispiel:

```
$include source                statt    @in,source.inc;
```

- Die Schreibweise von einigen internen Variablen hat sich geändert. Beispiele:

```
"@COMPANY"                     statt    "^company"
"@REPEAT_LINE"                         statt    "^repeat_line"
DATE('YYYYMMDD')               statt    "^DATE: YYYYMMDD"
^pn                            statt    ^page
$OK                            statt    ^OK
```

3.9 Element-Syntax

Element-Namen

- Elemente sollten grundsätzlich passende Namen erhalten, die automatisch vom Forms Painter vergebenen Namen wie @ELEM-1 sind zu vermeiden.

- Element, die auf Felder verweisen, sollten den gleichen Namen wie das Feld erhalten.

- Text-Elemente, die eine Beschreibung für ein anderes Element enthalten, erhalten als Namen den Element-Namen des beschreibenden Elementes mit dem Zusatz "Text".

Beispiele:

```
element="KundenNameText"
      type(TEXT)

element="KundenName"
      connect("KundenName")

field="KundenName"
```

Deklarationen und Funktionen für Elemente

- Es sollten immer die Deklarationen und Funktionen für Elemente benutzt werden, statt die Eigenschaften noch für Felder anzugeben, d.h. change-element() statt change(field()). Ausnahme bildet nur die Funktion change(field(<Feldname"), values(<Ausdruck>) , für die es keine alternative Schreibweise gibt.

3.10 Dateiformate

- Bei der Erstellung von neuen Programmen ist das UCS-Format zu benutzen. In diesem Zeichensatz sind in Version 4.x bis 6.x Semigrafik-Zeichen enthalten. Seit Version 7.0 basiert der UCS-Zeichensatz auf der Codepage 1252, die bisherigen Semigrafikzeichen zwischen den Positionen 128 und 160 sind damit entfallen. Damit sollten auch in Anwendungen, die bisher noch nicht in Version 7.0 oder höher betrieben werden, keine Semigrafikzeichen mehr benutzt werden, die einen späteren Umstieg erschweren. Einzige Ausnahme vom UCS-Format ist ein System, das noch in Unique CONCEPT 3.12 implementiert werden muß, hier sollte der Zeichensatz CP850 benutzt werden.

Umstellung von Unique II oder Unique 3-Programmen

- Die Umstellung auf den UCS-Zeichensatz kann automatisiert werden, in dem per Report die Programmdateien im alten Zeichensatz eingelesen und im UCS-Zeichensatz geschrieben werden.

3.11 Namen von Programmdateien

Syntax:

```
[<Programmtyp>][<Modulkürzel>]<Objekt>[<Methode>].<Dateierweiterung>
```

- Sprechende Namen verwenden.
- Programmdateien sollten keine Sonderzeichen außer Unterstrich im Namen haben.
- Zoom-Programme beginnen mit 'z'.
  Reports beginnen mit 'x'.
  Eingabedialogue für Reports beginnen mit 'i'.
- Ist das Programmsystem in Module unterteilt, sollten die Dateinamen einen Prefix aus 2 oder 3 Buchstaben haben, der die Zugehörigkeit zu einem Modul kennzeichnet.
- Die Namen sollten als Einzahl beschrieben werden.
- Als Dateierweiterungen sollte verwendet werden: uni, inc, imp, sys.
- Die Namen von Programmdateien sollten nicht länger als 16 Zeichen sein zzgl. Dateierweiterung. Beim Basismodul darf der Name nicht länger als 8 Zeichen sein, solange die 16bit-Version noch unterstützt werden muss.

Beispiele:

auftrag.uni
xrechdru.uni
irechdru.uni