PRINT.PAGE

Release 4.0
December, 2000

David L. Rotman
rotmand@cedarville.edu
Rotman & Howder
Cedarville University
251 N. Main Street
Cedarville, OH 45314

Preface

PRINT.PAGE is a forms generator, or page based reporting system, that works with Unidata files. The usefulness of PRINT.PAGE comes from its ability to use multiple Unidata file dictionaries as well as the ability to combine these fields with internal PRINT.PAGE fields to describe the data that will be used on the form.

This software was written at Cedarville University by Doug Sjoquist and modified by Dave Rotman. You may freely distribute this software, but this software is not to be sold by itself nor as part of any other software package. A current version of the software may be obtained via anonymous ftp from:

ftp.cedarville.edu

This software is made available on an "as-is" basis, with no warranty of any kind.

Currently, the software is being maintained by Dave Rotman. Suggestions for new features or bug fixes should be sent to rotmand@cedarville.edu. Since Dave cannot devote a lot of time to support, you be able to get usage questions answered more quickly by posting an item to e-mail list serves which focus on Unidata software and Datatel products (moderated-subscription list; contact Datatel directly).

Chapter 1. PRINT.PAGE Quick Start

a. Highlights

PRINT.PAGE is a forms generator, or page based reporting system, that works with Unidata files. The usefulness of PRINT.PAGE comes from its ability to use multiple Unidata file dictionaries as well as combine these fields with internal PRINT.PAGE fields to describe the data that will be used on the form. Some outstanding features of PRINT.PAGE include:

- Multiple Unidata file dictionaries can be used without creating I-descriptors between them.

- Any I-descriptor or data field in the dictionary may be used on the form.

- PRINT.PAGE can prompt for the value of a field at run time.

- The value for a field can be computed based on dictionary fields from multiple files as well as internal PRINT.PAGE fields.

- The value for a field can be returned from a subroutine, and the subroutine is able to call ITYPE() if needed.

- PRINT.PAGE handles multi-valued fields and subvalues intelligently, allowing choices in how they are printed.

- PRINT.PAGE will work with expanded select lists created with the "BY.EXP" option.

- Data can be printed in fixed length format for columnar reports, in variable length format for use in word processing applications, or a mixture of both in the same document.

- The form description file is a text file, so the form can be designed or "painted" using any text editor, including full screen text editors.

- PRINT.PAGE can e-mail the resulting document(s) rather than sending them to the system printer.

- PRINT.PAGE may be run as a stand-alone utility, or the PRINT.PAGE routines may be called from other UniBasic programs.

b. Designing the Form

Use a text editor such as Unidata “AE” or Unix “vi” to create a template file (called a form description file) as in the following example. This example is designed to generate a letter to an applicant for a mortgage, informing the recipient that certain documents are required before the mortgage application can be processed.

```
:AE PP.FORMS APP.LETTER
    New record.
     ----:  I
     0001=  &DEFINITION
     0002=  FIELD CUSTOMER CUSTOMER.ID
     0003=  FIELD REF REFERENCE.NUMBER
     0004=  FIELD LOC LOCATION
     0005=      INSERTING
     0006=  FIELD NAME CUSTOMER.NAME
     0007=  FIELD CITY
     0008=      INSERTING
     0009=  FIELD ST STATE
     0010=  FIELD SALUTATION
     0011=      INSERTING
     0012=  &BODY
     0013=  Hometown Mortgage Company         Customer: %CUSTOMER%
     0014=  234 North Main Street                Agent: %AGENT%
     0015=  P.O. Box 567                          Date: %#DATE()%
     0016=  Springfield, KY 45555-0567       Reference: %REF%
     0017=
     0018=
     0019=  %NAME%
     0020=  %STREET%
     0021=  %CITY%, %ST%%ZIP%
     0022=
     0023=  Dear %SALUTATION%:
     0024=
     0025=  We have received your mortgage application for property
     0026=  located at %LOC%.
     0027=
     0028=  However, we need the following items in order to process
     0029=  your application:
     0030=
     0031=       %MISSING.ITEMS%
     0032=
     0033=  Your application will receive prompt attention as soon
     0034=  as we have received the missing items.
     0035=
```

```
 0036=  Sincerely,
 0037=
 0038=
 0039=
 0040=  John K. Hartsell
 0041=  Loan Officer
 0042=
Bottom at line 41.
----: FILE
"APP.LETTER" filed in file "PP.FORMS".
```

In this example, "PP.FORMS" is the name of the type-1 file (Unix directory or subdirectory) which will contain the form description file "APP.LETTER".

Here is the rationale for some of the entries in the definition section of form description file:

The FIELD statements (such as "FIELD CUSTOMER CUSTOMER.ID") indicate that the Unidata field "CUSTOMER.ID" is to be identified by a different name within the form description file (the PRINT.PAGE field name "CUSTOMER").

INSERTING tells PRINT.PAGE that the exact amount of room needed to print this field should be used. If the value of the "SALUTATION" field is shorter or longer than the space taken up by the field name on the form, the text to the right of the field name will adjusted to the left or right.

The fields AGENT, STREET, ZIP, DATE(), and MISSING.ITEMS are not included in the definition section because no changes from the default usage needed to be made.

Fields are included in the form section by enclosing the PRINT.PAGE field name with the character "%".

MISSING.ITEMS is a multi-valued field. By default, PRINT.PAGE prints all of the values and inserts print lines to make room for subsequent values.

The character "#" in front of the DATE() field marks this field as being a "predefined" field whose value is determined by PRINT.PAGE.

c. Running PRINT.PAGE

Now that the form has been designed and entered, run PRINT.PAGE:

```
:SELECT MORTGAGE.APPS WITH MISSING.ITEMS BY CUSTOMER.NAME
:PRINT.PAGE MORTGAGE.APPS PP.FORMS APP.LETTER -FORM MTGAPP
```

Note that the command to run PRINT.PAGE specifies the file to be used "MORTGAGE.APPS", the type-1 file containing the form description file "PP.FORMS", and the actual form description file to use "APP.LETTER". Output is spooled with attribute "MTGAPP".

Output for the example presented above would look like this:

```
Hometown Mortgage Company          Customer: 3824
234 North Main Street                 Agent: 48
P.O. Box 567                           Date: 05/18/97
Springfield, KY 45555-0567        Reference: 38-3829


Janice K. Wills
1978 Cherryblossom Way
West Jefferson, KY  45528

Dear Ms. Wills:

We have received your mortgage application for property located
at 38 North Wayne Avenue.

However, we need the following items in order to process
your application:

     Employment verification
     Termite inspection report
     Occupancy permit

Your application will receive prompt attention as soon
as we have received the missing items.

Sincerely,



John K. Hartsell
Loan Officer
```

Chapter 2. Overview of PRINT.PAGE

a. Introduction

i. What is PRINT.PAGE?

PRINT.PAGE is a forms generator, or page based reporting system, that works with Unidata files. For each record processed, PRINT.PAGE will generate a single logical form, which may be a standard page or it may contain multiple physical pages. The form length and width are completely under the user's control. PRINT.PAGE uses the fields already defined in the Unidata dictionary as well as defining its own fields. Multiple files and their dictionaries may be used within a single PRINT.PAGE form.

PRINT.PAGE is designed to be run from the PERFORM colon prompt with an active select list from an Unidata file. The form itself is based on a file called a form description file, which can be created and edited using any word processor or text editor (such as PRIMOS ED or Unidata ED) that does not use special storage formats.

ii. What can PRINT.PAGE be used for?

PRINT.PAGE can be used to generate any form that requires data from individual records in an Unidata file. Any custom form that needs a "fill in the blanks" approach from the data base is a good candidate for using PRINT.PAGE. PRINT.PAGE can also produce almost any non-columnar report that does not include summaries across records.

Some specific examples are:

1. Customer profiles using customer demographics and transactions
2. Invoices and statements
3. Sales leads data sheets
4. Reply cards
5. Mail-merge applications
6. Sales receipts

iii. Key PRINT.PAGE terms

Form description file	This is a standard text file within an Unidata type-1 file (a Unix directory or subdirectory).
Initialization section	This section may contain Unidata commands to be executed before the main processing takes place (initializing commons used in subroutines, etc.)
Termination section	This section may contain Unidata commands to be executed after the main processing takes place.
Definition section	Portion of the form description file that sets up special features of PRINT.PAGE and overrides the default settings for the fields to be printed.
Form section	This section shows the actual layout of the document and is made up of the &HEADING, &BODY, and &FOOTING subsections.
Print directives	These are commands embedded in form section of the form description file that cause nonstandard printing to be done.
PRINT.PAGE field name	The name that PRINT.PAGE identifies every field in the form by. It may or may not be the same name as the dictionary field name.

b. Defining files to be used

Most PRINT.PAGE forms do not require the use of more than one file and so do not need to define secondary files. By default, PRINT.PAGE defines the primary file to be the Unidata file that is entered on the command line when PRINT.PAGE is run, and that is the file it will use to define the fields within the form as well.

When a complex form that uses multiple files needs to be prepared, PRINT.PAGE handles the related files by declaring one as the primary file and the related files as secondary files.

i. Primary file versus secondary files

The primary file in a PRINT.PAGE form description file is simply the file and dictionary that the form is designed to use. PRINT.PAGE uses the file name specified on the command line as the primary file whenever a file name or dictionary is needed. This is the file that the records have been selected from, and it is the dictionary that PRINT.PAGE will use to define and verify the dictionary fields in the form.

Secondary files are files that are related to the primary file through some field associated with the primary file. There can be multiple secondary files based on the same or different keys. Secondary files are used to link together two different files without the need to create I-descriptors in the primary file's dictionary. If the field that is the record key to the secondary file is a multi-valued field, then PRINT.PAGE will return a record from the secondary file for each of the values in the field. Using this along with other PRINT.PAGE features allows faster setup by avoiding the need to write I-descriptors for every field in the secondary file that will be needed on the form.

ii. Using fields from secondary files

To use a field from a secondary file in a PRINT.PAGE form, the secondary file first needs to be defined with the SECONDARY.FILE clause in the definition section. Fields from the secondary file may then be defined using the FIELD clause, or they may simply be used within the form section of the form as described in the next section.

c. Defining fields to be used

A PRINT.PAGE form can be set up and used without any definition section at all simply by creating the form to be printed as desired and including the appropriate field names from the dictionary of the Unidata file being used. Usually, though, some fields need some of the PRINT.PAGE options making it necessary to explicitly define them in the definition section. Even in this case, only those fields that are to be used in other than the default method need to be defined, the rest of the fields being used may still be used in the form section without any explicit definition.

i. PRINT.PAGE field types

There are five different types of fields that may be defined: dictionary fields, prompt fields, evaluate fields, subroutine fields, and predefined fields. Because of the nature of the prompt and evaluate fields, they must always be explicitly defined in the definition section.

Dictionary fields are defined using the dictionary of an Unidata file. The PRINT.PAGE field name usually is the same name as the field name in the dictionary, but a different name may be used if desired (eg. to use an abbreviated name in the PRINT.PAGE form).

Prompt fields request their value from the user at the time PRINT.PAGE is run and may be single-valued or multi-valued.

Evaluate fields are calculated at the time each individual form is printed. The calculations that may be performed are a subset of the calculations available in I-descriptors, but will work with any PRINT.PAGE field in the form from any of the files being used, not just dictionary fields from a single file.

Subroutine fields have a value returned from an UniBasic subroutine that has a single argument. The subroutine is allowed to do whatever it needs to do to generate the return value (passed back through the one argument.) This type of field will be most useful when used to do something an ordinary I-descriptor would not be able to do, such as calling a subroutine that called the function ITYPE.

Predefined fields are from a fixed list of fields whose values are determined by PRINT.PAGE when it is run. Predefined fields include values such as the current date or time, as well as page and record counts.

ii. PRINT.PAGE field names

When a PRINT.PAGE field is defined using the FIELD clause, PRINT.PAGE uses that name to identify the field throughout the form description file.

Each PRINT.PAGE field name must be unique, with one exception: fields defined within a SECONDARY.FILE may have the same name as a field from the PRIMARY.FILE or a different SECONDARY.FILE. But, every time these fields are referenced anywhere within the form description file, they must be identified by both the file name and the field name to avoid ambiguity. In any case, the use of the same PRINT.PAGE field name for two different fields should be avoided whenever possible for the sake of clarity.

When a PRINT.PAGE dictionary field is defined, it may be named the same as the

field in the Unidata file dictionary or it may be named something entirely different. Whatever it is named, to use this field and its associated definitions from the definition section, the PRINT.PAGE field name must be used to identify it and not the field name from the Unidata file dictionary.

If the PRINT.PAGE field name for a dictionary field is something other than the field name from the Unidata file dictionary, and then later, the field name from the dictionary is used instead of the PRINT.PAGE field name, PRINT.PAGE will define a new field (see the section "Using fields in the form") using the field name from the Unidata file dictionary. Both PRINT.PAGE fields will refer to the same Unidata field, but any clauses describing the first PRINT.PAGE field will not apply to this second field because PRINT.PAGE views them as two different fields.

iii. Using fields in the form

One of the most useful features of PRINT.PAGE is that fields do not need to be defined in the definition section to be used in the form section. To place a field from the Unidata file dictionary in the form section of the form, simply use the field name from the dictionary as if it were already defined, and PRINT.PAGE will automatically define the field to be used based on the dictionary record for that field. If no file name is specified, PRINT.PAGE will use the dictionary for the PRIMARY.FILE to search for the field.

Any PRINT.PAGE field defined in the definition section may be used anywhere and as many times as needed in the form section simply by enclosing it with the character "%". For example, the string "%NAME%" placed in the form section would print the value for the PRINT.PAGE field "NAME".

To specify a particular file for a field, separate the file name and the field name with the character "/" and enclose the whole string with the character "%". Eg. The string "%STUDENTS/NAME%" would use the PRINT.PAGE field "NAME" defined for the file "STUDENTS".

iv. Using options with fields

If any of the PRINT.PAGE options are to be used for a particular field, that field must be defined in the definition section with the FIELD clause. All optional clauses for a particular field must directly follow the FIELD clause for that field. Options are described in chapter 3 of this documentation.

d. Printing options

i. Form options

There are options to control the length of the form being used, the number of lines to actually print on a form, the method used to skip to the top of the next form, and how many alignment copies of the form to print initially for setting up the printer.

The following example sends 3 copies of the output to printer FINANCE:

```
GET.LIST MY.LIST
PRINT.PAGE GENLEDGER GL.REP EOM.REPORT \
    -FORM FINANCE \
    -COPIES 3
```

ii. Controlling paging on the form

Page control in PRINT.PAGE is done using a combination of three different items: the form options in the definition section, the three different subsections (&HEADING, &BODY, and &FOOTING) of the form section, and the print directives embedded in the form section. PRINT.PAGE keeps track of both the physical page number being printed and the current record count. Both of these counts may be put into the form section by using predefined fields in the form.

iii. Generating e-mail

PRINT.PAGE can generate e-mail messages in addition to generating printed output. Optionally, each e-mail message may also have an attachment. Consider the following example:

```
GET.LIST GOOD.STUDENTS
PRINT.PAGE STUDENTS REG.WP CONGRATS \
    -MAIL STU.EMAIL \
    -SENDER "baker@stateu.edu" \
    -ATTACH NEWSFORM
```

The output will be generated as e-mail messages. The recipient's e-mail address will be determined from the i-descriptor STU.EMAIL on the STUDENTS file. The

body of the message will be determined by processing the form definition fiile CONGRATS from directory REG.WP.

The sender's e-mail address will be set to "baker@stateu.edu".

Each e-mail message will have as an attachment the result of PRINT.PAGE processing form definition NEWSFORM from the REG.WP file.

Chapter 3. Setting Up PRINT.PAGE

a. Introduction

There are four major sections to a PRINT.PAGE form description file, the initialization section, termination section, definition section and the form section. Each section begins with the section declaration on a line by itself and ends when another section declaration is found. The form section must be the last section. If a form description file begins with no section declaration, then it is assumed to be the definition section.

Within the initialization, termination and definition sections, null lines and any lines that begin with the comment character "*" will be ignored by PRINT.PAGE. Within the form section no lines will be ignored.

b. Initialization section

The initialization section begins with "&INITIALIZATION" and may contain Unidata commands that should be executed before the main processing takes place. The most common reason to place something in this section is to initialize a labeled common storage area that will be used by an UniBasic subroutine in the definition section of the form,

c. Termination section

The termination section begins with "&TERMINATION" and may contain Unidata commands that should be executed after the main processing takes place.

d. Definition section

The definition section begins with "&DEFINITION".

i. File clauses

The file clauses are optional clauses that are only required when multiple Unidata files are used within the same form description file. The PRIMARY.FILE clause may

be used for clarity even when there is only one Unidata file being used.

(1) PRIMARY.FILE

Definition: The PRIMARY.FILE clause identifies the Unidata file that should be specified on the command line when PRINT.PAGE is run. If no other files are defined through the use of the SECONDARY.FILE clause, PRINT.PAGE will use the dictionary from this file to define and verify every dictionary field used in the form.

When used: PRIMARY.FILE is an optional clause which can be used for the sake of clarity within the form description file, and also as a verification against the file specified on the command line. If this clause is included, and the file name on the command line does not match the file name given in this clause, then PRINT.PAGE will report an error.

Syntax: The syntax is:

```
PRIMARY.FILE Unidata_file_name
```

where "unidata_file_name" is the Unidata file name specified on the command line. This must be the name of a file defined in the VOC.

Options: None.

Defaults: The PRIMARY.FILE defaults to the Unidata file name entered on the command line when PRINT.PAGE was run.

Examples: Suppose that PRINT.PAGE is run from the command line like this:

```
PRINT.PAGE EMPLOYEES WORDPROC EMPLOYEE.LTR \
    -FORM EMPLTR
```

The file EMPLOYEE.LTR in directory WORDPROC could contain:

```
Place at beginning   1:  &DEFINITION
of file --------->   2:  PRIMARY.FILE EMPLOYEES
                     3:  etc...
```

(2) SECONDARY.FILE

Definition: The SECONDARY.FILE clause identifies the other Unidata files (up

to a maximum of 8) that are used in the PRINT.PAGE form. The key to this file can be defined as a PRINT.PAGE dictionary field from the PRIMARY.FILE, a PRINT.PAGE prompt field, or an evaluate field based on some combination of these.

When used: This clause allows the use of related files in the same PRINT.PAGE form without the necessity of creating I-descriptors in the dictionary of the PRIMARY.FILE. Any data field or I-descriptor from the SECONDARY.FILE may now be used anywhere in the form.

Syntax: The syntax is:

```
SECONDARY.FILE secondary_unidata_file
key_to_file
```

where "secondary_unidata_file" is the name of the Unidata file to be used, and "key_to_file" is the PRINT.PAGE field name that is the record key to "secondary_unidata_file".

The key field "key_to_file" can be a PRINT.PAGE dictionary field from the PRIMARY.FILE, a PRINT.PAGE prompt field, or an evaluate field based on some combination of these.

If "key_to_file" is a multi-valued field, then multiple records will be returned for each PRIMARY.FILE record. Within each field in the multiple records, subvalue marks will be changed to text marks, value marks will be changed to subvalue marks, and the field from each record will be treated as a single value. This is the same processing done with I-descriptors using the TRANS() function for translating multi-valued fields.

Options: None.

Defaults: None.

Examples: Given a system with an EMPLOYEES file keyed by employee id, and an EMPLOYEES.TAX.INFO file keyed by social security number, both files could be used by declaring the EMPLOYEES file to be the PRIMARY.FILE and the EMPLOYEES.TAX.INFO file to be a secondary file.

If SS.NUM is a field in the EMPLOYEES file dictionary, then the set up for such a form would be:

```
1:  &DEFINITION
```

```
                        2:  PRIMARY.FILE EMPLOYEES
  Placed after          3:    fields from EMPLOYEES.
  PRIMARY.FILE          4:    etc...
  clauses -------->     5:
                        6:  SECONDARY.FILE EMPLOYEES.TAX.INFO SSN
                        7:    fields from EMPLOYEES.TAX.INFO
                        8:    etc.
                        9:
                       10:  &BODY
                       11:
```

ii. Field clauses

The FIELD clause introduces a field to be defined to PRINT.PAGE. All optional clauses must follow directly after the FIELD clause for the field that they modify.

(1) Overview of FIELD clauses

Definition: The FIELD clause defines a PRINT.PAGE field that may be used anywhere in the form section.

When used: The FIELD clause must be used to introduce prompt fields and evaluate fields, but PRINT.PAGE will recognize dictionary fields and predefined fields without a FIELD clause.

The FIELD clause does become necessary for dictionary fields and predefined fields if any of the field options need to be used. For example, here are some reasons why it may be necessary to define the field with the FIELD clause:

1. The defaults for using a field in the form section are to be changed (eg, declaring a field to be INSERTING as opposed to the default of overwriting).

2. The default format for a field is to be changed (eg, the value is to be printed with a FMT code of "5L" instead of the default FMT code from the dictionary).

3. The dictionary field name is too long to be used in the form section (eg, the Unidata field "LAST.PAYMENT.DATE" needs to be defined as "LPAYDT").

4. Not all the values of a multi-valued field are to be printed on the form, so the NUM.VALUES option must be used to limit the number of values to print.

5. A constant number of values from a multi-valued field are to print on a fixed form, whether or not there is actually that many values in the multi-valued field, so the EXPAND.LINES OFF option must be used.

(2) FIELD clause for dictionary fields

Syntax: The syntax can be one of the following:

```
FIELD pp_field_name unidata_field_name
FIELD pp_field_name
```

where "pp_field_name" is the name that PRINT.PAGE will use to identify this field throughout the form description file, and "unidata_field_name" is the field name from the Unidata file dictionary.

PRINT.PAGE will use the current file to define and verify dictionary fields. If there is only one Unidata file being used in the form description file, then that is always the currently defined file. If there is more than one Unidata file being used (by using the PRIMARY.FILE and SECONDARY.FILE clauses), then PRINT.PAGE uses the last file clause to determine the current file.

So, if multiple files are being used, each time PRINT.PAGE encounters the PRIMARY.FILE or SECONDARY.FILE clauses, all field definitions following that clause will use that file as the Unidata file to verify the dictionary fields against. The current file is reset to the PRIMARY.FILE when the form section begins, so that the PRIMARY.FILE is the default file for all fields in the form section that were not defined in the definition section.

If the Unidata field name is not included, and this field is not defined as a prompt or evaluate field, PRINT.PAGE assumes that the field is a dictionary field and that the Unidata field name is the same as the PRINT.PAGE field name.

Options: The Unidata field name is optional.

Defaults: The Unidata field name defaults to the PRINT.PAGE field name.

Examples: For a normal PRINT.PAGE form using a single file:

```
Defining              1:  &DEFINITION
Dictionary fields     2:  PRIMARY.FILE CUSTOMERS
    ------------->    3:    FIELD CNAME FULL.NAME
    ------------->    4:    FIELD FIRST FIRST.NAME
                      5:      INSERTING
    ------------->    6:    FIELD ACCT.BALANCE
                      7:      FMT 10R
                      8:  &BODY
```

Three dictionary fields have been defined in the definition section. Any other dictionary fields that may be used later in the form section will assume default values and usage based on the dictionary record since they are not defined here.

The PRINT.PAGE field "CNAME" is defined as the Unidata field "FULL.NAME" from the file "CUSTOMERS". There are no other changes from how it is defined in the dictionary.

The PRINT.PAGE field "FIRST" is defined as the Unidata field "FIRST.NAME" from the file "CUSTOMERS". It is also being modified so that it will use the INSERTING option in the form section.

The PRINT.PAGE field "ACCT.BALANCE" is defined as the Unidata field with the same name from the file "CUSTOMERS" since an Unidata field name was not included. It is being modified so that it will be formatted using the FMT code "10R" in the form section rather than the FMT code that is in the dictionary record.

For a PRINT.PAGE form using multiple files:

```
                       1:  &DEFINITION
                       2:  PRIMARY.FILE CUSTOMERS
                       3:    FIELD CNAME FULL.NAME
                       4:    FIELD FIRST FIRST.NAME
                       5:      INSERTING
Dictionary fields      6:    FIELD ACCT.BALANCE
from the secondary     7:      FMT 10R
file                   8:  SECONDARY.FILE ORDER.FILE ORDER.NUMS
    ------------->     9:    FIELD O.DATE ORDER.DATE
    ------------->    10:    FIELD O.ITEM ITEM.DESCRIPTION
                      11:  &BODY
                      12:  etc...
```

Five dictionary fields have been defined in the definition section.

The PRINT.PAGE fields "CNAME", "FIRST", "ACCT.BALANCE" are defined in the same way as in the single-file example.

The field "ORDER.NUMS"" on the SECONDARY.FILE clause is a multi-valued field in the Unidata file "CUSTOMERS" that contains the record keys to all of the customer's current orders in the Unidata file "ORDER.FILE". Since there will be multiple "ORDER.FILE" records per customer (because the field "ORDER.NUMS" from the "CUSTOMERS" file is multi-valued), all of the fields from the file "ORDER.FILE" will be defined as multi-valued automatically.

The PRINT.PAGE fields "O.DATE" and "O.ITEM" are defined as the Unidata fields "ORDER.DATE" and "ITEM.DESCRIPTION" from the file "ORDER.FILE". These fields are automatically assumed to be multi-valued (see previous paragraph) and EXPAND.LINES is automatically ON.

(3) FIELD clause for evaluate fields

Syntax: The syntax is:

```
FIELD pp_field_name
    EVALUATE evaluate_phrase
```

where "pp_field_name" is the name that PRINT.PAGE will use to identify this field throughout the form description file, and "evaluate_phrase" is the phrase to evaluate. The EVALUATE option identifies this as an evaluate field.

Options: None.

Defaults: None.

Examples: The key to the transaction file "SHIPPED" is to be the concatenation of the field "INV.NO" from the primary file "INVENTORY" and a prompt field that requests the warehouse code.

```
1:  &DEFINITION
2:  PRIMARY.FILE INVENTORY
3:    FIELD WAREHOUSE.CODE
```

```
Evaluate field       4:      PROMPT "Warehouse code: " "##"
   -------------->   5:    FIELD SHIPPED.KEY
                     6:      EVALUATE INV.NO:'*':WAREHOUSE.CODE
                     7:  SECONDARY.FILE SHIPPED SHIPPED.KEY
                     8:  etc...
```

In this example, any data field or I-descriptor from either the "INVENTORY" file or the "SHIPPED" file may be used. Only a single record will be returned for the "SHIPPED" file based on the evaluate field "SHIPPED.KEY", since "SHIPPED.KEY" is single-valued.

See the section on the EVALUATE option itself for more information and examples.

(4) FIELD clause for predefined fields

Syntax: The syntax is:

```
FIELD pp_field_name #predefined_field_name
```

where "pp_field_name" is the name that PRINT.PAGE will use to identify this field throughout the form description file, the character "#" identifies this as a predefined field name, and "predefined_field_name" is one of PRINT.PAGE predefined fields included in the list below.

PRINT.PAGE Predefined Fields			
Field	Default CONV	Default FMT	Description
@DATE	D2/	8R	Date PRINT.PAGE started
@DAY		2R	Day of month PRINT.PAGE started
@LOGNAME		15L	User login name
@MONTH		2R	Month PRINT.PAGE started
@PATH		25L	Unix path where PRINT.PAGE is being run
@TIME	MTH	8R	time PRINT.PAGE started

@USERNO		3R	User number running PRINT.PAGE
@WHO		15L	Unidata account name
@YEAR		2R	Year PRINT.PAGE started
DATE()	D2/	8R	Current date
FONT			Change fonts at execution time
PAGE.COUNT		5R	Current page count
RECORD.COUNT		5R	Current record count
TIME()	MTH	8R	Current time
TIMEDATE()		20L	Current time and date

Options: FONT is the only predefined field which requires options. For each FONT, you should specify a font name and size as shown in the examples below.

Defaults: See the table listed above.

Examples: To include the page number and current date in the heading of a PRINT.PAGE form, use the predefined fields DATE() and PAGE.COUNT in the form section.

```
                      1:  &DEFINITION
Define a font —>      2:      FIELD XCHG #FONT
                      3:          ARGS ARIAL 10
                      4:  &HEADING
Current date ---->    5:  Date: %#DATE()%
Current page ---->    6:  Page: %#PAGE.COUNT%
                      7:  &BODY
                      8:  Dear %FIRST%:
Change font —>        9:  %XCHG%
                     10:  This letter will serve to inform you
                          that you have
```

The predefined fields "DATE()" and "PAGE.COUNT" have been used in the form section without a FIELD clause in the definition section. Each of these will then use the default FMT and CONV codes.

The predefined field "DATE()" has a default CONV code of "D2/" and a default FMT code of "8R" so line 3 in the example will be printed as:

```
Date: 05/23/98
```

The predefined field "PAGE.COUNT" has a default CONV code of "" and a default FMT code of "5R" so line 6 in the example will be printed as:

```
Page:    23
```

The first few lines of the report will be printed in the default font (which varies with the printer being used). The body of the letter will be printed in 10-point Arial font. (See Appendix E for information on defining fonts to PRINT.PAGE.)

Any of the predefined fields may be further defined with a FIELD clause in order to use an abbreviated PRINT.PAGE field name, change the default FMT or CONV code, or to use another option such as INSERTING:

```
                     1:  &DEFINITION
Current date --->    2:  FIELD TODAY #DATE()
                     3:     CONV D4/
Change CONV —>       4:     INSERTING
                     5:  FIELD DATE.DUE
Mark inserting —>    6:     INSERTING
                     7:  &BODY
                     8:  This is your account status as of
                     9:  %TODAY%.  Please pay by %DATE.DUE% or
                    10:  we will need to
```

The predefined field "DATE()" is being modified to print as an inserting field for a word-processing application.

(5) FIELD clause for prompt fields

Syntax: The syntax is:

```
FIELD pp_field_name
    PROMPT prompt_options
```

where "pp_field_name" is the name that PRINT.PAGE will use to identify this field throughout the form description file, and "prompt_options" is the information needed by the PROMPT option. The PROMPT option identifies this as a prompt field.

Options: None.

Defaults: None.

Examples: To prompt for a due date when printing invoices:

```
                      1:  &DEFINITION
Prompting ------->    2:   FIELD DUE.DATE
                      3:    PROMPT "Invoice due date: " ##/##/##
                      4:
```

The due date will be requested at run time and will be printed exactly as entered.

See the section the PROMPT option itself for more information and examples.

(6) FIELD clause for subroutine fields

Syntax: The syntax is:

```
FIELD pp_field_name
    SUBROUTINE subroutine_name
```

where "pp_field_name" is the name that PRINT.PAGE will use to identify this field throughout the form description file, and "subroutine_name" is the catalogued UniBasic subroutine. The SUBROUTINE option identifies this as a subroutine field.

Options: None.

Defaults: None.

Examples: The subroutine GET.NAME.ADDRESS calls the UniBasic function ITYPE to determine which is the correct address to use, and therefore can not be called from an I-descriptor in a file.

The following UniBasic subroutine PP.GET.NA is written to provide an interface between GET.NAME.ADDRESS which requires three arguments and PRINT.PAGE which expects a single argument for the subroutine call.

```
SUBROUTINE PP.GET.NA(RETURN.VALUE)
```

```
ADDRESS.TYPE = 'HOME'
DATE.TO.SEND = @DATE
CALL GET.NAME.ADDRESS(CALCULATED.ADDRESS,
        ADDRESS.TYPE,DATE.TO.SEND)
RETURN.VALUE = CALCULATED.ADDRESS
RETURN
END
```

Use the following lines in the definition section of the form description file to call the subroutine PP.GET.NA which returns a multi-value address.

```
                    1:  &DEFINITION
                    2:  ...
Subroutine field    3:
   -------------->  4:  FIELD NAME.ADDRESS
                    5:    SUBROUTINE PP.GET.NA
                    6:    MULTI.VALUE
                    7:
                    8:  etc...
```

iii. Field options

All field options are indeed optional, but when they are used they must follow directly after the FIELD clause for the field that they modify.

(1) BY.EXP

Definition: The BY.EXP option is used in conjunction with an active exploded select list. It identifies a multi-valued field that should print out the single value that corresponds to the value number in the exploded select list.

When used: Use the BY.EXP option when using an exploded select list designed to print out single values from multi-valued fields.

This is how PRINT.PAGE handles associations and the special selects that may be used with them such as:

```
SELECT FILENAME BY.EXP MV.FLD WHEN \
    MV.FLD = 'Value'
```

All fields in a PRINT.PAGE form defined with this option will be treated as associated, and will only print out the values from the positions in the lists that the exploded select list indicates.

Syntax: The syntax is:

```
BY.EXP
```

Options: None.

Defaults: None.

Examples: For example, if in a library system the Unidata file "PATRONS" contains an association of overdue books for each patron with the fields "TITLE" and "DUE.DATE", and a separate form is to be printed for each overdue book requesting information from the patron, then use a SELECT BY.EXP and the BY.EXP option in the PRINT.PAGE form.

```
Identifies these     1:  &DEFINITION
fields as BY.EXP     2:    FIELD TITLE
  ------------->     3:      BY.EXP
                     4:      INSERTING
                     5:    FIELD DUE.DATE
  ------------->     6:      BY.EXP
                     7:      INSERTING
                     8:    FIELD NAME PATRON.NAME
                     9:      INSERTING
                    10:  &BODY
```

```
11:  Dear %NAME%:
12:
13:  According to our records, you have the
14:  book "%TITLE%" which was due back on
15:  %DUE.DATE%.  Please let us know if you
16:  still have it, and when you will
17:  return
     it.
```

The definition file would be used with a select like this:

```
:SELECT PATRONS BY.EXP DUE.DATE WHEN DUE.DATE # '' \
    AND DUE.DATE >= "3/1/98"
:PRINT.PAGE PATRONS PP.FORMS OVERDUE.LETTERS \
    -FORM OVRDUE
```

For each record id and value number in the active exploded select list, a separate form will be printed. If a record in "PATRONS" has more than one overdue book, then that patron will receive a letter for each overdue book.

(2) CONV

Definition: The CONV option specifies the CONV code to use in printing a field.

When used: Use the CONV option when a dictionary field is to be printed with a different conversion, or when the default conversion for other field types (pre-defined, prompt, and evaluate) needs to be changed.

Syntax: The syntax is:

```
CONV conversion.code
```

where "conversion.code" is any legal conversion code.

Options: None.

Defaults: The default CONV is obtained from the dictionary for dictionary fields. Other defaults are described in the sections on pre-defined fields, prompt fields, and evaluate fields.

Examples:

```
1:  &DEFINITION
2:    FIELD PAY.DATE
```

```
Conversion ------>   3:      CONV D4/
```

The conversion for the field "PAY.DATE" (which may have been "D2/" in the Unidata file dictionary) is being changed to "D4/" so that all four digits of the year will print.

(3) DEFAULT.VALUE

Definition: The DEFAULT.VALUE option specifies a value to be used any time the field is null. (Note that the value "0" is not a null value so the default value will not be used when a field contains a zero.)

When used: Use the DEFAULT.VALUE to provide a default value for any type of PRINT.PAGE field.

Syntax: The syntax is:

```
DEFAULT.VALUE default_value
```

where "default_value" is the value to use when the field is null.

Options: None.

Defaults: None.

Examples: For example, to make sure that a number always prints for a field that may contain either a number or a null value, set up the form like this:

```
       1:  &DEFINITION
       2:    FIELD AMOUNT.PAID
DEFAULT.VALUE --->   3:      DEFAULT.VALUE 0
```

When the form is printed, if the field "AMOUNT.PAID" is null, then the value "0" will be used instead. All of the normal formatting and conversion will then be done using the value "0" rather than "".

(4) EVALUATE

Definition: The EVALUATE option will do calculations based on the fields from

multiple Unidata files, prompt fields, predefined fields, and other evaluate fields. The calculations that may be performed are a subset of the calculations available with I-descriptors.

When used: Use EVALUATE when an I-descriptor style field is needed for the PRINT.PAGE form and it is not desirable or not possible to create the I-descriptor. Creation of I-descriptors is not desirable when dictionaries are becoming cluttered. Creation of I-descriptors may be impossible due to policies established by the system administrator, or because the I-descriptor would need to reference PRINT.PAGE fields (prompt fields, evaluate fields, pre-defined fields).

Syntax: The syntax is:

```
EVALUATE evaluate.phrase
```

where "evaluate.phrase" is a phrase or expression similar to what would be used in an I-descriptor.

Options: The expression syntax is a subset of the logic available for I-descriptors. Operands may be any PRINT.PAGE field or literal values. "IF" statements and functions are not supported. Field, value, and subvalue marks are treated as normal characters, so no special handling is done for multi-valued fields. Allowable operators are summarized below.

Allowable “Evaluate” Operators	
(	left parenthesis
)	right parenthesis
** *or* ^	exponentiation
*	multiplication
/	division
+	addition
-	subtraction
: *or* CAT	concatenation
LT *or* <	less than

GT *or* >	greater than
EQ *or* =	equal to
NE *or* <> *or* >< *or* #	not equal to
LE *or* <= *or* =< *or* #>	less than or equal to
GE *or* >= *or* => *or* #<	greater than or equal to
MATH *or* MATCHES *or* LIKE	matches
NOT	logical NOT
AND *or* &	logical AND
OR *or* !	logical OR

Defaults: None.

Examples: This example produces a report that contains information from a student demographic file "STUDENTS", and from a student schedule file "STUD.SCHEDS" that is related to the student file by the student id and a particular term. The term is prompted for, and the student id comes from the "STUDENTS" file.

```
                     1:  &DEFINITION
Create a field       2:  PRIMARY.FILE STUDENTS
to be used for a     3:    FIELD TERM
record key into      4:      PROMPT 'Term: ' VERIFY.FILE TERMS
a secondary file     5:    FIELD SS.KEY
  --------------->   6:      EVALUATE TERM:'*':@ID
  --------------->   7:  SECONDARY.FILE STUD.SCHEDS SS.KEY
                      8:    FIELD SS.COURSE COURSE
                      9:    FIELD SS.TITLE COURSE.TITLE
                     10:  &BODY
                     11:
                     12:  Student name.....: %STUDENTS/NAME%
                     13:  Registration term: %TERM%
                     14:
                     15:    Course Id              Course Title
                     16:  %SS.COURSE%            %SS.TITLE%
                     17:
                     18:  etc...
```

This allows the courses and any other information from the schedule file to be printed on the same form with the student demographics without creating I-descriptors between them.

(5) EXPAND.LINES

Definition: The EXPAND.LINES option controls whether PRINT.PAGE inserts extra lines into the form as needed to print multi-valued fields. By default, all multi-valued fields used in the form are defined with EXPAND.LINES ON.

When used: Use the EXPAND.LINES option to turn off this feature, or to explicitly declare it to be on.

Syntax: The syntax can be one of the following:

```
EXPAND.LINES
EXPAND.LINES ON
EXPAND.LINES OFF
```

Options: EXPAND.LINES may be turned off for a field with the "OFF" option. The other options are present for documentation purposes, to reflect the fact that EXPAND.LINES is on.

Defaults: The default for all multi-valued fields is EXPAND.LINES ON, and if EXPAND.LINES is included for a field without specifying "ON" or "OFF", then "ON" is assumed.

Examples: For example, if a multi-valued field such as "ITEM.DESC" has a variable number of description lines, but a fixed layout is being used (like a preprinted form), use the EXPAND.LINES OFF option to disable the insertion of print lines.

```
                     1:  &DEFINITION
                     2:    FIELD ITEM.DESC
                     3:      NUM.VALUES 10
EXPAND.LINES ---->   4:      EXPAND.LINES OFF
All of these lines   5:  &BODY
are set aside to     6:
print descriptions   7:
         -------->   8:  Description: %ITEM.DESC%
         -------->   9:
         -------->  10:
         -------->  11:
         -------->  12:
         -------->  13:
         -------->  14:
         -------->  15:
         -------->  16:
```

```
--------->  17:  More text may appear here without
            18:  being
            19:  overwritten.
            20:  etc...
```

(6) FMT

Definition: The FMT option specifies the FMT code to use in printing a field.

When used: Use the FMT option when a dictionary field is to be printed with a different format, or when the default format for other field types (pre-defined, prompt, and evaluate) needs to be changed.

Syntax: The syntax is:

```
FMT fmt.code
```

where "format.code" is any legal format code.

Options: None.

Defaults: The default FMT is obtained from the dictionary for dictionary fields. Other defaults are described in the sections on pre-defined fields, prompt fields, and evaluate fields.

Examples:

```
                    1:  &DEFINITION
                    2:    FIELD JOB.DEPT
Format ---------->  3:      FMT 20R
```

The format for the field "JOB.DEPT" (which may have been "20L" in the Unidata file dictionary) is being changed to "20R" so that the field is right-justified on the printout.

(7) INSERTING

Definition: The INSERTING option specifies that this field is to be inserted into the text line, rather than the default action of overlaying the line.

When used: Use the INSERTING option whenever a value is to use only as much

space as it needs, no more and no less. It is especially appropriate for use in the middle of paragraphs or letters.

Syntax: The syntax is:

```
INSERTING
```

Options: None.

Defaults: None.

Examples: In this example, the first name is to be used as the salutation and an appointment date is to be inserted.

```
Marks these fields   1:  &DEFINITION
to insert into       2:    FIELD FIRST.NAME
the text -------->   3:      INSERTING
                     4:    FIELD APPOINTMENT.DATE
         -------->   5:      INSERTING
                     6:  &BODY
                     7:  %FULL.NAME%
                     8:  %ADDRESS%
                     9:
                    10:  Dear %FIRST.NAME%:
                    11:
                    12:  Your appointment is scheduled for
                    13:  %APPOINTMENT.DATE%. Please notify
                    14:  our office...etc...
                    15:
```

The PRINT.PAGE fields "FIRST.NAME" and "APPOINTMENT.DATE" are being modified to be inserted into the text of the letter, rather than overlaying it.

A record with the field "FIRST.NAME" having the value "Joyce" and an "APPOINTMENT.DATE" OF 05/18/92 would print out as:

```
Joyce Humphrey
318 West Washington Avenue
Chicago, IL 60601

Dear Joyce:

Your apppointment is scheduled for
05/18/97.  Please notify
our office...etc...
```

(8) MAX.LENGTH

Definition: The MAX.LENGTH option specifies the maximum length to print for a field.

When used: Use the MAX.LENGTH option whenever a long field needs to be truncated.

Syntax: The syntax is:

```
MAX.LENGTH length
```

where "length" is the maximum number of characters to print.

Options: None.

Defaults: Without the MAX.LENGTH clause, all of the characters in the field are always printed. The default width is controlled by the FMT clause (if present) or the FMT entry in the file dictionary for this field, but PRINT.PAGE will use a longer output width if the actual text does not fit within the FMT. (The default behavior functions just like I-descriptors printing fields wider than the FMT clause.)

Examples:

```
                    1:  &DEFINITION
                    2:    FIELD NAME
Maximum length -->  3:      MAX.LENGTH 25
```

In this example, only the first 25 characters of the field "NAME" will be printed.

(9) MULTI.VALUE

Definition: The MULTI.VALUE option specifies that a field is multi-valued.

When used: Use the MULTI.VALUE option to override the single/multi-valued flag in the dictionary record of a dictionary field. When used in conjunction with a prompt field, PRINT.PAGE continues to prompt for values until a null value is entered.

Syntax: The syntax is:

```
MULTI.VALUE
```

Options: None.

Defaults: None.

Examples: For example, to prompt for multiple comment lines for an invoice, set up the form like:

```
                    1:  &DEFINITION
                    2:    FIELD COMMENTS
                    3:      PROMPT "Comment lines: "
                    4:      MAX.LENGTH 60
MULTI.VALUE ----->  5:      MULTI.VALUE
                    6:      EXPAND.LINES
```

When PRINT.PAGE is run, it will prompt for comment lines until a null value is entered and, because of the EXPAND.LINES option, all of the comments will be printed.

(10) NUM.SUBVALUES

Definition: The NUM.SUBVALUES option specifies the maximum number of subvalues to be printed (per value) for a multi-valued field.

For fields defined as multi-valued, the EXPAND.LINES option is on by default, so extra lines will be inserted (up to the product of NUM.SUBVALUES and NUM.VALUES) unless EXPAND.LINES is explicitly turned off.

For fields defined as single-valued, only the first value and first subvalue will be printed.

When used: Use the NUM.SUBVALUES option when a maximum number of subvalues are to print for a particular multi-valued field.

Syntax: The syntax is:

```
NUM.SUBVALUES number.of.subvalues
```

where "number.of.subvalues" is the number of subvalues to be printed.

Options: None.

Defaults: The default "number.of.subvalues" is all subvalues unless the EXPAND.LINES OFF option has been used, then the default is 1.

Examples:

```
                        1:  &DEFINITION
                        2:    FIELD CONTACT.NAMES
                        3:      NUM.VALUES 3
                        4:    FIELD CONTACT.DATES
                        5:      NUM.VALUES 3
NUM.SUBVALUES --->      6:      NUM.SUBVALUES 2
                        7:  &BODY
                        8:
                        9:  %CONTACT.NAMES%       %CONTACT.DATES%
                       10:
                       11:  etc...
```

In this example, if the fields "CONTACT.NAMES" and "CONTACT.DATES" are associated, with the field "CONTACT.DATES" having multiple subvalues for each corresponding name, then up to three names will be printed on the report, and up to two dates will be printed for each name that prints.

(11) NUM.VALUES

Definition: The NUM.VALUES option specifies the maximum number of values to be printed for a multi-valued field.

For fields defined as multi-valued, the EXPAND.LINES option is on by default, so extra lines will be inserted (up to NUM.VALUES) unless EXPAND.LINES is explicitly turned off.

For fields defined as single-valued, only the first value and first subvalue will be printed.

When used: Use the NUM.VALUES option when a maximum number of values are to print for a particular multi-valued field.

Syntax: The syntax can be one of the following:

```
NUM.VALUES number.of.values
```

```
NUM.VALUES number.of.values LAST.VALUES
```

where "number.of.values" is the number of values to be printed.

Options: The LAST.VALUES option specifies that the last "number.of.values" values are to print, rather than the default first "number.of.values".

Defaults: The default "number.of.values" is all values unless the EXPAND.LINES OFF option has been used, then the default is 1.

Examples:

```
                     1:  &DEFINITION
                     2:    FIELD CONTACT.NAMES
NUM.VALUES ------>   3:      NUM.VALUES 5
                     4:    FIELD SALES.DATES
NUM.VALUES ------>   5:      NUM.VALUES 3 LAST.VALUES
                     6:  &BODY
                     7:
                     8:  %CONTACT.NAMES%
                     9:
                    10:  %SALES.DATES%
                    11:  etc...
```

Up to five values in the field "CONTACT.NAMES" will be printed on the report. If there are less than five values, only as many lines as are needed to print the number of values will be printed.

The last three values in the field "SALES.DATES" will be printed on the report. If the field "SALES.DATES" has less than three values, only as many lines as are needed to print the number of values will be printed.

(12) PROMPT

Definition: The PROMPT option allows PRINT.PAGE to prompt the user for the values of fields at the time PRINT.PAGE is run.

When used: The PROMPT option should be used any time there will be some variation in the text of the printed for each batch that is run. This allows the variable text to be input at run time rather than edit the text of the form description file itself for each batch.

Syntax: The syntax can be one of the following:

```
PROMPT promp.text input.mask VERIFY.FILE    ver
                                            ify
                                            nam
                                            e
PROMPT prompt.text input.mask
PROMPT prompt.text VERIFY.FILE verify.name
PROMPT VERIFY.FILE verify.name
PROMPT
```

where "prompt.text" is the text to be displayed when the user is prompted for a value, "input.mask" is a set of characters to be displayed as a pattern for user input, and "verify.name" is the name of a file to be used to validate responses to the prompt.

Options: "Prompt.text", "input.mask", and "VERIFY.FILE verify.name" are all optional.

Defaults: "Prompt.text" defaults to the "pp_field_name" listed in the FIELD clause.

"input.mask" defaults to a single "#".

There is no default for "VERIFY.FILE verify.name".

Examples: To prompt for a due date when printing invoices:

```
                    1:   FIELD DUE.DATE
Prompting ------->  2:    PROMPT "Invoice due date: " ##/##/##
                    3:
```

The due date will be requested at run time and will be printed exactly as entered.

To prompt for a payroll type when printing time cards:

```
                    1:  FIELD PAYROLL.TYPE
Prompting ------->  2:   PROMPT "Type: " VERIFY.FILE PAY.TYPES
                    3:
```

The payroll type will be requested at run time and will be validated against file PAY.TYPES. Any user response must be a record key on file PAY.TYPES.

(13) SKIP.NULL.LINES

Definition: The SKIP.NULL.LINES option identifies fields that should not have blank values printed.

When used: Use the SKIP.NULL.LINES to mark lines that should not be printed when the entire line is blank. It is usually used in conjunction with a multi-valued field that may have no values at all or null values, but it may be used on any field where a blank line should not be printed. PRINT.PAGE will do a TRIM() function on any line with this option, and will not print it if the trimmed final print line is null.

Syntax: The syntax is:

```
SKIP.NULL.LINES
```

Options: None.

Defaults: Lines that are blank after inserting values or fields will be printed.

Examples: For example, to print a field containing comments in the middle of a form and exclude any blank comment lines, set up the form like this:

```
                    1:  &DEFINITION
                    2:    FIELD COMMENTS
SKIP.NULL.LINES ->  3:      SKIP.NULL.LINES
```

When the form is printed, no comment lines that are all blank or null will be printed.

(14) SPECIAL.CONV

Definition: The SPECIAL.CONV option specifies an enhanced conversion code to use in printing a date field.

When used: Use the SPECIAL.CONV option whenever the enhanced appearance is needed.

Syntax: The syntax is:

```
SPECIAL.CONV enhanced.conv.code
```

where "enhanced.conv.code" is any code from the following table.

Special Conversion Codes	
DAY.OF.WEEK	Returns the day number (where Sunday = 1) of the date field being used.
DAY.OF.WEEK.TEXT	Returns the day in text format (e.g., "Sunday", "Friday") of the date field being used.
MONTH.TEXT	Returns the month in text format (e.g., "April", "June") of the date field being used.
DATE.TEXT	Returns the entire date in text format (Eg. "July 4, 1776") of the date field being used.

Options: None.

Defaults: None.

Examples:

```
Special                1:  &DEFINITION
Conversions            2:    FIELD TEXT.DAY BIRTHDATE
  --------------->     3:      SPECIAL.CONV DAY.OF.WEEK.TEXT
                       4:      INSERTING
                       5:    FIELD TEXT.BD BIRTHDATE
  --------------->     6:      SPECIAL.CONV DATE.TEXT
                       7:      INSERTING
                       8:  &BODY
                       9:
                      10:  Your birthday is
                      11:  %TEXT.DAY%, %TEXT.BD%.
```

The PRINT.PAGE field "TEXT.DAY" is defined as the Unidata field "BIRTHDATE", but is being displayed as the name of the day. The PRINT.PAGE field "TEXT.BD" is defined as the same Unidata field "BIRTHDATE", but it is being display as the text version of the full date.

The SPECIAL.CONV option overrides the conversion code in the Unidata file dictionary for dictionary fields. Also, the SPECIAL.CONV and CONV options are mutually exclusive.

A record with the field "BIRTHDATE" having the value "12/04/60" would print out as:

```
Your birthday is
Sunday, December 4, 1960.
```

(15) SUBROUTINE

Definition: The SUBROUTINE option allows PRINT.PAGE to call a catalogued UniBasic subroutine that has one argument and returns a value in that argument. The returned value may be either single-valued or multi-valued.

When used: Use the SUBROUTINE option whenever the normal I-descriptor call to a subroutine (...SUBR("subname",etc.)...) is not a workable choice (eg. when the subroutine needs to call the UniBasic function ITYPE.) Use the normal I-descriptor whenever possible as it provides more functionality.

Syntax: The syntax is:

```
SUBROUTINE subroutine.name
```

where "subroutine.name" is a catalogued UniBasic subroutine that has a single argument and returns some sort of value in that argument.

Options: None.

Defaults: None.

Examples: There is a subroutine GET.EXPECTED.BALANCE that uses various I-descriptors to determine billing information and returns the expected balance in its single argument. It is based on the value of the current record and record key (@RECORD and @ID, which are what PRINT.PAGE uses to evaluate other fields.)

```
                      1:   FIELD EXPECTED.BALANCE
Subroutine field->    2:     SUBROUTINE GET.EXPECTED.BALANCE
                      3:     CONV MD2,
                      4:     FMT 10R
```

iv. Form clauses

All form clauses are optional, as all of them have default values. The form clauses can be placed anywhere in the definition section except between a FIELD clause and its options. It is best though, to put the form clauses all together at the beginning or end of the definition section for clarity.

(1) FORM.LENGTH

Definition: The FORM.LENGTH clause defines the physical length of the form being used. It is used in conjunction with the PRINT.LINES.ON.FORM clause.

When used: Use the FORM.LENGTH clause any time a form with a non-standard length is begin used.

Syntax: The syntax is:

```
FORM.LENGTH number_lines
```

where "number_lines" is the physical number of lines on the form being used.

Options: None.

Defaults: The default number of physical lines per page is 66.

Examples: For example, if a form such as a sales receipt that is three inches long is being used, there would be 18 physical lines on the form (assuming that the printer is set for 6 lines per inch). To tell PRINT.PAGE that there are 18 lines, set up the form like this:

```
                    1:  &DEFINITION
FORM.LENGTH ----->  2:    FORM.LENGTH 18
```

(2) LINE.UP

Definition: The LINE.UP clause tells PRINT.PAGE how many alignment forms to print at the beginning of each run. PRINT.PAGE will use the text in the form section, including PRINT.PAGE field names, for the alignment forms. Each time the PRINT.PAGE form description file is modified, the alignment form that prints out will automatically match the new form.

When used: Use the LINE.UP clause anytime a preprinted form is being used so the printer operator can make sure the printer and the forms are aligned properly. Note that if the PRINT.PAGE form is more than one physical page (by using the $NEW.PAGE print directive, or having a form section that will not fit on a page), then each alignment copy will be more than one page. This allows the use of multi-page preprinted forms.

Syntax: The syntax is:

```
LINE.UP number_alignment
```

where "number_alignment" is the number of alignment copies to print.

Options: None.

Defaults: None.

Examples: For example, to have PRINT.PAGE print two alignment copies at the beginning of each run, set up the form like this:

```
                1:  &DEFINITION
LINE.UP --->    2:    LINE.UP 2
```

PRINT.PAGE will now print two alignment copies before any actual forms are processed.

(3) PRINT.LINES.ON.FORM

Definition: The PRINT.LINES.ON.FORM clause defines the number of lines to actually print on the form being used. It is normally used in conjunction with the FORM.LENGTH clause. If the number of print lines is over this limit, then PRINT.PAGE will proceed to the next form using the method defined with the TOP.OF.FORM clause, and will continue printing. If there is a &HEADING or &FOOTING section, PRINT.PAGE will finish the page with the footing, and continue the next page with the heading.

When used: Use the PRINT.LINES.ON.FORM clause to change the default value, or when the FORM.LENGTH clause is used to change the physical form size.

Syntax: The syntax is:

```
PRINT.LINES.ON.FORM number_lines
```

where "number_lines" is the number of lines to be printed on the form being used.

Options: None.

Defaults: The number of lines to print per form defaults to six less than the physical lines on the form (the FORM.LENGTH clause) unless the form length is less than six, then the default is equal to the form length.

If no FORM.LENGTH or PRINT.LINES.ON.FORM clause is used, the default form length is 66, and the default for PRINT.LINES.ON.FORM is 60.

Examples: For example, if a sales receipt is three inches long, (18 physical lines), to print on all 18 lines, set up the form like this:

```
                    1:  &DEFINITION
                    2:    FORM.LENGTH 18
------------------->   3:    PRINT.LINES.ON.FORM 18
```

As a second example, consider a gummed-label which is two inches high (12 physical lines). To make maximum use of the label, but ensure that a blank line is left between labels, use a setup like this:

```
                    1:  &DEFINITION
                    2:    FORM.LENGTH 12
------------------->   3:    PRINT.LINES.ON.FORM 11
```

(4) TOP.OF.FORM

Definition: The TOP.OF.FORM clause determines which method is used to advance to the beginning of the next form.

When used: Use the TOP.OF.FORM clause to change the default method when a non-standard form length is used, the printer being used will not handle form feeds properly, or no action should be taken between

forms.

Syntax: The syntax is:

```
TOP.OF.FORM method
```

where "method" is one of the keywords FORM.FEED, BLANKS, or NONE. The FORM.FEED method uses ASCII character 12 (Undata octal 214) to cause the printer to advance to the next form (this is the standard of most printers). The BLANKS method prints blank lines until the number of lines printed is equal to the length set with the FORM.LENGTH clause. The NONE method does no extra printing between forms.

Options: None.

Defaults: If no TOP.OF.FORM clause is given, the default depends on what the FORM.LENGTH is set to. If the FORM.LENGTH clause is not used, or it is set to the default 66 lines, then the default TOP.OF.FORM method is FORM.FEED, otherwise the default is BLANKS.

Examples: For example, to disable any advancing between forms, set up the form like this:

```
                    1:  &DEFINITION
----------------->  2:    TOP.OF.FORM NONE
```

e. Form section

The form section is where the actual text is placed and the layout of fields is specified. Any literal text, PRINT.PAGE field defined in the definition section, dictionary field from either the PRIMARY.FILE or a defined SECONDARY.FILE, or print directive may be placed here. The form section allows for multiple pages per record to be printed, along with page handling features that include headings and footings.

i. Subsections

There are three subsections to the form section: the heading, body, and footing sections. The subsection begins when the character "&" followed by either HEADING, BODY, or FOOTING is found on a line by itself (the obsolete method of

%END% is equivalent to &BODY). The definition section ends and the form section begins when any one of these three subsections is introduced. These three subsections may be in any order, but may only appear once in a form description file. All text following the subsection keyword, and up to the next subsection or the end of form description file, is a part of that subsection. Any item is allowed in any subsection, the only difference is when and where the items are printed on the page.

(1) &HEADING

The text and fields in the &HEADING subsection will be used for a heading for each page that is printed for a particular record. This section of text will be printed at the beginning of the form for each record processed. If there is not enough room for the entire form to be printed on the same page (as determined by the length of the heading and footing sections, and the PRINT.LINES.ON.FORM option) then the heading section will be printed again after advancing to the next form.

(2) &BODY

The text and fields in the &BODY subsection will be printed only once per record. If this requires multiple pages to be printed, then PRINT.PAGE handles the paging using the form options and the heading and footing subsections.

(3) &FOOTING

The text and fields in the &FOOTING subsection will be printed at the bottom of each page required for this record. PRINT.PAGE handles all of the line spacing, paging, heading, and footings automatically.

ii. Print directives

The text in the form section is printed just as it is entered, with the exception of any print directives included in the text. Each print directive must be on a line itself, and must start with the character "$".

(1) $NEW.PAGE

The print directive $NEW.PAGE forces a new page at that line in the text. PRINT.PAGE automatically handles the paging, headings, and footings as set up

with the form options and subsections. This print directive should only be used in the &BODY subsection.

Chapter 4. Hints for Using PRINT.PAGE

a. Using multi-valued fields

There are a variety of ways to deal with multi-valued fields within PRINT.PAGE: using the EXPAND.LINES, NUM.VALUES, and NUM.SUBVALUES options, as well as how the fields are actually placed in the text in the form section.

i. Using a multi-valued field with no options

When a multi-valued field is used, whether it is defined in the definition section or just used in the form section, the default usage is to print all values and subvalues on expanding lines. That is, all of the information in the field will be printed with as many print lines as needed being inserted.

The defaults for these options are:

```
EXPAND.LINES        ON
NUM.VALUES          all values
NUM.SUBVALUES       all subvalues
```

The following example where "DED.AMTS" and "DEDUCTIONS" are multi-valued fields and each has five values:

```
                    1:  &DEFINITION
                    2:  PRIMARY.FILE EMPLOYEES
                    3:  &BODY
                    4:  Employee..: %NAME%
                    5:  Salary....: %SALARY%
Fields ---------->  6:  Deductions: %DED.AMTS%  %DEDUCTIONS%
                    7:  Date hired: %HIRE.DATE%
```

would have four lines inserted between lines 6 and 7 and would print out like this:

```
Employee..: Whittenburg, Samuel
Salary....:  21,250.00
Deductions:      50.00  Life Insurance
                 25.00  Pension Plan
                 10.00  Union Dues
                 25.00  Medical Plan
                 20.00  United Way
Date hired: 12/01/87
```

ii. Using NUM.VALUES or NUM.SUBVALUES with EXPAND.LINES ON

Using either NUM.VALUES or NUM.SUBVALUES for a multi-valued field simply sets a maximum number of values or subvalues to print. Print lines will still be inserted as needed up to the maximum set by these options.

The following example where "MEETING.DATE" and "MEETING.DESC" are multi-valued fields and each has eight values:

```
Use the NUM.VALUES   1:  &DEFINITION
to restrict how      2:  PRIMARY.FILE CUSTOMERS
many values print    3:  FIELD MTG.DT MEETING.DATE
       ---------->   4:    NUM.VALUES 3 LAST.VALUES
                     5:  FIELD MTG.DESC MEETING.DESC
       ---------->   6:    NUM.VALUES 3 LAST.VALUES
                     7:  &BODY
                     8:
                     9:  Customer.......: %CUSTOMER.NAME%
                    10:  Meetings
                    11:     %MTG.DT% %MTG.DESC%
                    12:  Last order date: %LAST.ORDER.DATE%
```

would have only two lines inserted between lines 11 and 12, would print only the last three values from each field, and would print out like this:

```
Customer.......: Scarborough, Jim
Meetings
   12/15/98 Met to discuss new products
   12/19/98 Follow up discussion
   01/25/99 Finalized sale
Last order date: 01/25/99
```

iii. Using NUM.VALUES or NUM.SUBVALUES with EXPAND.LINES OFF

Using the EXPAND.LINES OFF option causes values and subvalues after the first value to be printed overlaying the next print lines defined in the form section. Enough space on the subsequent print lines need to be left to allow for the values and subvalues that will be printed there. With EXPAND.LINES OFF, no values will be printed past the last physical line in the form section, even if the logical form length is longer than the physical form length.

The following example where "MAIL.ADDRESS" is a multi-valued field and has four values:

```
                      1:  &DEFINITION
Use EXPAND.LINES      2:  PRIMARY.FILE PROSPECTS
OFF to print on       3:  FIELD MAIL.ADDRESS
fixed lines in        4:    NUM.VALUES 5
the form  ----->      5:    EXPAND.LINES OFF
                      6:  FIELD TODAY #DATE()
                      7:    SPECIAL.CONV DATE.TEXT
                      8:  FIELD FIRST
                      9:    INSERTING
                     10:  &BODY
                     11:
Reserve five         12:  %TODAY%
lines for the        13:
address  ------>     14:  %MAIL.ADDRESS%
         ------>     15:
         ------>     16:
         ------>     17:
         ------>     18:
                     19:
                     20:  Dear %FIRST%,
                     21:
                     22:  etc...
```

would not insert any lines but rather the values would print on lines 14-18 and would print out like this:

```
        March 17, 1997

        Mr. Timothy Brown
        Department 35G
        11321 West Main
        Highland, IN  46322


        Dear Timothy,

        etc...
```

iv. Using EXPAND.LINES OFF without NUM.VALUES or NUM.SUBVALUES

Using the EXPAND.LINES OFF by itself changes the default number of values and

subvalues to print for a field to 1.

v. Using two or more multi-valued fields with EXPAND.LINES ON

Placing two or more multi-valued fields with EXPAND.LINES ON (the default) on the same line in the form section will cause the expanded lines to include the values for these fields on corresponding lines. Values and subvalues within multi-valued fields on the same line to be expanded will be aligned so that corresponding values will print on the same print line (the same effect as using LIST to list multi-valued fields with subvalues.)

The following example where "ITEM.NUMBER", "ITEM.COST", and "ITEM.DESC" are multi-valued fields each having two values, and "ITEM.DESC" has three subvalues for the first item and four subvalues for the second item:

```
                     1:  &DEFINITION
                     2:  PRIMARY.FILE PURCHASE.ORDERS
                     3:    FIELD I.NUM ITEM.NUMBER
                     4:    FIELD I.DESC ITEM.DESC
                     5:    FIELD I.COST ITEM.COST
                     6:
                     7:  &BODY
                     8:  Purchase Order: %PO.NUMBER%
                     9:
                    10:    Item      Description         Cost
Fields --------->   11:    %I.NUM%   %I.DESC%        %I.COST%
                    12:                              ----------
                    13:                           %TOTAL.COST%
                    14:
```

would need seven lines to print and so would insert six lines between the lines 11 and 12, and would print out like this:

```
      Purchase Order: P0039285

        Item      Description         Cost
        A352-30   Midget            125.35
                  brown
                  widgets
        X8-AB39   Automobile tire    25.50
                  and motorcycle
                  tire instant
```

```
                    repair kit
                                    ----------
                                       150.85
```

If the same example included the NUM.SUBVALUES option to limit the printing of subvalues:

```
                      1:  &DEFINITION
                      2:  PRIMARY.FILE PURCHASE.ORDERS
NUM.SUBVALUES on      3:    FIELD I.NUM ITEM.NUMBER
each field ------>    4:      NUM.SUBVALUES 2
                      5:    FIELD I.DESC ITEM.DESC
                -->   6:      NUM.SUBVALUES 2
                      7:    FIELD I.COST ITEM.COST
                -->   8:      NUM.SUBVALUES 2
                      9:
                     10:  &BODY
                     11:  Purchase Order: %PO.NUMBER%
                     12:
                     13:    Item      Description         Cost
                     14:    %I.NUM%   %I.DESC%        %I.COST%
                     15:                              ----------
                     16:                           %TOTAL.COST%
```

the output would look like this:

```
      Purchase Order: P0039285

        Item      Description         Cost
        A352-30   Midget            125.35
                  brown
        X8-AB39   Automobile tire    25.50
                  and motorcycle
                                    ----------
                                       150.85
```

b. Using subroutines that call ITYPE

There are several ways that the subroutine field type may be used effectively.

If the subroutine to be used has a single argument and returns a value in that argument, then there is no extra setup or additional routine necessary, simply use the subroutine field type as described above.

If the subroutine does not fit the above description, then another subroutine (an interface routine) must be written which calls the subroutine desired, and returns the appropriate value in its single argument. This interface routine will most likely be short and simple and able to handle any necessary setup.

If the interface routine can not handle the setup for the subroutine to be called, or if it would be too inefficient to do so, another routine (the setup routine) may be written to handle the setup of labelled common and any other initialization needed. This setup routine would then be listed in the &INITIALIZATION section in the form description file. The setup routine is called simply as a Unidata sentence or paragraph and may include command line arguments.

i. Writing an interface routine.

The interface routine needs to set up the appropriate fields necessary to call the subroutine and then pass the appropriate information back through its single argument.

For example, the subroutine to be called (GET.CHARGES) calculates charges to be billed on an account based on the ID number and date, and it returns two multi-valued associated lists, descriptions and amounts. If the only valued needed in the PRINT.PAGE document is the sum of the amounts, then the following subroutine and form description file would be what is needed:

Interface routine, PP.CALL.GET.CHARGES:

```
SUBROUTINE PP.CALL.GET.CHARGES(RETURN.AMOUNT)
DESC.LIST = ''
AMT.LIST = ''
CALL GET.CHARGES(@ID, @DATE, DESC.LIST, AMT.LIST)
RETURN.AMOUNT = SUM(AMT.LIST)
RETURN
END
```

Form description file:

```
List the interface   1:  &DEFINITION
  routine as a       2:  PRIMARY.FILE ACCTS.RECEIVABLE
  subroutine --->    3:  FIELD TO.BE.BILLED
  field -------->    4:    SUBROUTINE PP.CALL.GET.CHARGES
                     5:    FMT "10R"
                     6:    CONV "MD2,"
                     7:  &BODY
                     8:  Account ID...: %@ID%
```

```
 9:  Account name.: %NAME%
10:  Date.........: %#@DATE%
11:  Charges......: %TO.BE.BILLED%
12:  etc.
```

The form would print out like this:

```
Account ID...: 103992
Account name.: Jones, JoAnne P.
Date.........: 03/15/98
Charges......:   1,255.23
etc.
```

ii. Writing a setup routine.

If in the above example, the subroutine GET.CHARGES used a labelled common which it assumes the calling program had set up in its own initialization section, a setup routine for the PRINT.PAGE document would need to be written for it. It would most likely be short and simple.

For example, if the labelled common contained a table of codes, descriptions, and amounts, the setup routine would need to load those tables just as an program calling GET.CHARGES would need to do. Assuming the tables are loaded from the file AR.CODES with the fields listed, the following setup routine may be used and then listed in the initialization section of the form description file:

Setup routine, PP.SET.UP.GET.CHARGES:

```
EQU MAX.CODES TO 99
COMMON /AR.GCHG/ NUM.CODES,
    CODE.LIST,
    DESC.LIST(MAX.CODES),
    AMT.LIST(MAX.CODES)
OPEN '', 'AR.CODES' TO F.AR.CODES ELSE
   STOP 'FATAL ERROR: CAN NOT OPEN "AR.CODES"'
END

CODE.LIST = ''
NUM.CODES = 0
SELECT F.AR.CODES TO 10
DONE.LIST = 0
LOOP
  READNEXT AR.CODE FROM 10 ELSE
    DONE.LIST = 1
```

```
    END
  UNTIL DONE.LIST
    READ AR.CODES.REC FROM F.AR.CODES, AR.CODE THEN
      NUM.CODES += 1
      CODE.LIST<1,NUM.CODES> = AR.CODE
      DESC.LIST(NUM.CODES) = AR.CODES.REC<1>
      AMT.LIST(NUM.CODES) = AR.CODES.REC<2>
    END
  REPEAT
  END
```

Form description file:

```
List the setup -->   1:  &INITIALIZATION
  routine in the     2:    PP.SET.UP.GET.CHARGES
  initialization     3:  &DEFINITION
  section            4:  PRIMARY.FILE ACCTS.RECEIVABLE
                     5:  FIELD TO.BE.BILLED
                     6:    SUBROUTINE PP.CALL.GET.CHARGES
                     7:    FMT "10R"
                     8:    CONV "MD2,"
                     9:  &BODY
                    10:  Account ID...: %@ID%
                    11:  Account name.: %NAME%
                    12:  Date.........: %#@DATE%
                    13:  Charges......: %TO.BE.BILLED%
                    14:  etc.
```

An entry in the VOC file named PP.SET.UP.GET.CHARGES would need to be created in order for PRINT.PAGE to be able to execute the program during initialization.

c. Using multiple files

There may be up to eight secondary files in a single form description file, and the key field from the primary file for each of the files may be either single or multi-valued. If the key field for a particular secondary file is single-valued, then for each primary record, one secondary record will be returned. If the key field for a secondary file is multi-valued, then for each primary record, multiple secondary records will be returned, one for each value from the key field.

i. Using one secondary file with a single-valued key field

If a system of two files has the file "INVENTORY" as the primary file and the file "SUPPLIERS" as a secondary file, and there is a field in the "INVENTORY" file named "MAIN.SUPPLIER" that is a key to a record in the "SUPPLIERS" file, then both files may be used in the form description file by setting it up as:

```
Key to secondary     1:  &DEFINITION
file is the field    2:  PRIMARY.FILE INVENTORY
"MAIN.SUPPLIER" ->   3:  SECONDARY.FILE SUPPLIERS MAIN.SUPPLIER
from the primary     4:  &BODY
file.                5:
                     6:              Inventory Worksheet
Fields from the      7:
primary file --->    8:  Item number..: %ITEM.NUMBER%
       --------->    9:  Description..: %DESCRIPTION%
                    10:
       --------->   11:  Stock on Hand: %QTY.ON.HAND%
       --------->   12:  Backordered..: %QTY.BACKORDERED%
                    13:
Fields from the     14:
secondary file      15:
       --------->   16:  Main Supplier: %SUPPLIERS/NAME%
       --------->   17:                 %SUPPLIERS/ADDRESS%
       --------->   18:  Payment terms: %SUPPLIERS/PAY.TERMS%
```

The form would print out like this:

```
                Inventory Worksheet

       Item number..: P35-X253.3
       Description..: Single shot, bolt
                      action, mahogany
                      stock, .22 gauge
                      rifle.

       Stock on Hand:     13
       Backordered..:      0



       Main Supplier: LongHorn Rifles
                      Purchasing Department
                      352 Oliver Avenue
                      Springfield, IL 60885
       Payment terms: 2%/10, net in 30
```

ii. Using two secondary files with single-valued key fields

If a system of three files has the file "PERSONNEL" as the primary file, and the files "DEPTS" and "PAYROLL" as secondary files, and the field "JOB.DEPT" from the primary file is the key to "DEPTS", and the field "SOC.SEC.NO" is the key to "PAYROLL", then all three files may be used in the form description file by setting it up as:

```
                     1:  &DEFINITION
                     2:  PRIMARY.FILE PERSONNEL
Fields may be        3:    FIELD TODAY #DATE()
defined for any      4:      SPECIAL.CONV DATE.TEXT
of the secondary     5:    FIELD P.FIRST FIRST
files in the         6:      INSERTING
definition section   7:  SECONDARY.FILE DEPTS JOB.DEPT
                -->  8:    FIELD SUP.NAME
                     9:      EVALUATE FIRST:' ':LAST
                    10:      INSERTING
                    11:  SECONDARY.FILE PAYROLL SOC.SEC.NO
                -->  12:   FIELD BONUS
                    13:      EVALUATE YTD.OT.HRS * (RATE * .05)
                    14:      CONV MD2,
                    15:      INSERTING
                    16:  &BODY
                    17:
or they may simply  18:
be used in the      19:  %TODAY%
text of the form    20:
section as needed   21:  %PERSONNEL/NAME%
                -->  22:  %PERSONNEL/ADDRESS%
                    23:  %DEPTS/DEPT.NAME%
                    24:
                    25:  Dear %P.FIRST%:
                    26:
                    27:  Your supervisor, %SUP.NAME%, has
                    28:  recommended you for a special
                    29:  citation because of your willingness
                    30:  to work overtime many different times
                    31:  throughout the year.  As a special
                    32:  reward, the company would like to
                    33:  present you with a bonus of $%BONUS%
                    34:  for all of your hard work.
                    35:
                    36:  Thanks again,
```

```
37:  The Widget Company management
38:
```

The form would print out like this:

```
January 15, 1998

Mr. Joseph Kohlmeyer
985 West 43rd Street
Columbus, OH  47823
Quality Control Department

Dear Joseph:

Your supervisor, Sam Jones, has
recommended you for a special
citation because of your willingness
to work overtime many different times
throughout the year.  As a special
reward, the company would like to
present you with a bonus of $1,025.25
for all of your hard work.

Thanks again,

The Widget Company management
```

iii. Using one secondary file with a multi-valued key field

If a system of two files has the file "SALES.PERSONNEL" as the primary file and the file "CLIENTS" as a secondary file, and there is a multi-valued field in the "SALES.PERSONNEL" file named "CLIENT.LIST" that contains keys to records in the "CLIENTS" file, then both files may be used in the form description file by setting it up as:

```
1:  &DEFINITION
2:  PRIMARY.FILE SALES.PERSONNEL
3:    FIELD SP.NAME NAME
4:    FIELD SP.TER TERRITORY
5:  SECONDARY.FILE CLIENTS CLIENT.LIST
6:    FIELD CL.NAME.ADD NAME.ADDRESS
7:    FIELD CL.DT LAST.SALES.DATE
8:    FIELD CL.ITEMS LAST.SALES.ITEMS
```

```
                     9:  &BODY
                    10:
                    11:  Current Sales Summary
                    12:
Each Client's       13:  Salesperson: %SP.NAME%
name/address, last  14:  Territory..: %SP.TER%
sales date, and     15:
the items sold      16:  Customers:
will be grouped     17:     Name        ---- Last sale ----
together --------->  18:     Address     Date         Items
                    19:  %CL.NAME.ADD% %CL.DT%   %CL.ITEMS%
                    20:
```

The form would print out like this:

```
Current Sales Summary

Salesperson: Timmons, Susan
Territory..: Los Angeles, California

Customers:
   Name        ---- Last sale ----
   Address     Date         Items
MTC Engines   08/30/96  Calipers
Box 3153                Brake Drums
13 W. Third             Mufflers
Pamona, CA              Spark Plugs
                        Filters

Joe's Garage  03/15/97  Hubcaps
1536 Main               Wrenches
Holly, NV

Fixit Shop    01/03/97  Filters
Box 105
RR #13
Vale, CO
```

The example assumes that the field "NAME.ADDRESS" in the "CLIENTS" file is a multi-valued I-descriptor which appends a null value after the last line of the address. Also, field "LAST.SALES.ITEMS" does not contain subvalues in the data record.

If two multi-valued fields (with EXPAND.LINES ON) are printed on the same line, the fields are treated as associated for printing purposes.

Chapter 5. Calling PRINT.PAGE from UniBasic Programs

a. Overview

While the command-line interface for PRINT.PAGE is quite powerful, there may be occasions where the use of subroutine calls for PRINT.PAGE would provide some additional flexibility in generating output. Using the subroutine calls, a programmer could produce the "easy" output using PRINT.PAGE and then write his or her own code for producing the more difficult output.

The subroutine calls which are available perform validation of the form description files, generate the print lines, and return counter information to the calling program. All of the command-line flexibility is available by calling the appropriate subroutines.

Debugging the calling program can be simplified by testing the individual form description files with the command-line interface to PRINT.PAGE.

For most situations, the calling program should handle paging, so the form description files should contain "TOP.OF.FORM NONE".

b. Calling sequence

There are three steps to follow when using the subroutine interface for PRINT.PAGE:

1. Initialize the PRINT.PAGE tables by calling PRINT.PAGE.INIT. This is done only once per program.

2. Each form description file to be used (maximum of four description files per program) must be loaded using PRINT.PAGE.LOAD.

3. For each time the calling program needs to generate a PRINT.PAGE section, a call is made to PRINT.PAGE.PROCESS. Typically, this means one call per data record being processed.

c. Subroutine descriptions

PRINT.PAGE.INIT

Calling sequence:

```
CALL PRINT.PAGE.INIT
```

Arguments:

None.

Comments:

PRINT.PAGE.INIT sets up the COMMON areas used by PRINT.PAGE. This subroutine should be called prior to calling any other PRINT.PAGE subroutine.

i. PRINT.PAGE.LOAD

Calling sequence:

```
CALL PRINT.PAGE.LOAD(wp.dir, wp.file, unidata.file,
sequence.num, errmsg)
```

Arguments:

wp.dir	the type-1 file where the PRINT.PAGE form description file is stored.
wp.file	the name of the form description file (record) in wp.dir.
unidata.file	the primary Unidata data file from which the PRINT.PAGE form is being generated. (This form description file may contain references to secondary data files.)
sequence.num	the sequence number (between 1 and 4) to be used in PRINT.PAGE.PROCESS to uniquely identify this form description.
errmsg	multi-valued list of error messages returned by PRINT.PAGE.LOAD. A successful load will return a null error-message list.

Comments:

PRINT.PAGE.LOAD parses the form description file to verify that correct syntax has been used and that the referenced dictionary fields actually exist in the appropriate files.

PRINT.PAGE.LOAD prompts for any prompt fields in the forms definition file. Any evaluate fields present are tested with a null @RECORD and a null @ID.

ii. PRINT.PAGE.PROCESS

Calling sequence:

```
CALL PRINT.PAGE.PROCESS (sequence.num, lines.printed,
warnmsg)
```

Arguments:

sequence.num	the sequence number (between 1 and 4) used in calling PRINT.PAGE.LOAD for this particular section of output.
lines.printed	the number of physical lines generated by PRINT.PAGE.PROCESS.
warnmsg	multi-valued list of warning messages returned by PRINT.PAGE.PROCESS. A successful execution will return a null warning-message list.

Comments:

PRINT.PAGE.PROCESS performs the actual work of generating the output. The calling program is responsible for setting @RECORD and @ID prior to calling PRINT.PAGE.PROCESS. (@ID my contain value marks, since PRINT.PAGE supports exploded select lists.)

d. Sample program

The following program skeleton illustrates how a calling program could use PRINT.PAGE.

```
0001:  * program to produce customer invoices
0002:  * illustrating calls to PRINT.PAGE
0003:  *
0004:      GOSUB PRINT.PAGE.SET.UP
0005:      GOSUB OTHER.SET.UP
0006:      PRINTER ON
0007:      DONE = 0
0008:      LOOP
0009:          READNEXT @ID ELSE
0010:              DONE = 1
0011:          END
0012:      UNTIL DONE
0013:          READ @RECORD FROM F.Unidata.FILE, @ID THEN
```

```
0014:               GOSUB PRINT.HEADER
0015:               GOSUB PRINT.BODY
0016:               GOSUB PRINT.FOOTER
0017:           END ELSE
0018:               CRT @ID:' NOT FOUND ON ':Unidata.FILE
0019:           END
0020:       REPEAT
0021:       PRINTER OFF
0022:       STOP
0023:
0024:
0025:   PRINT.PAGE.SET.UP:
0026:       CALL PRINT.PAGE.INIT
0027:       Unidata.FILE = 'CUSTOMERS'
0028:       WP.DIR = 'FORM.FILES'
0029:       WP.RECORD = 'INVOICE.HEADER'
0030:       SEQUENCE = 1
0031:       ERRMSG = ''
0032:       CALL PRINT.PAGE.LOAD (WP.DIR, WP.RECORD, Unidata.FILE,
                SEQUENCE, ERRMSG)
0033:       IF ERRMSG # '' THEN
0034:           GOSUB DISPLAY.ERRORS
0035:           STOP
0036:       END
0037:       WP.RECORD = 'INVOICE.FOOTER'
0038:       SEQUENCE = 2
0039:       CALL PRINT.PAGE.LOAD (WP.DIR, WP.RECORD, Unidata.FILE,
                SEQUENCE, ERRMSG)
0040:       IF ERRMSG # '' THEN
0041:           GOSUB DISPLAY.ERRORS
0042:           STOP
0043:       END
0044:       RETURN
0045:
0046:   OTHER.SET.UP:
0047:       PAGE.SIZE = 60 ;* for use by this program, not
0048:       PRINT.PAGE
0049:       ...custom code goes here (including file opens)...
0050:       RETURN
0051:
0052:   PRINT.HEADER:
0053:       SEQUENCE = 1
0054:       NUM.LINES = 0
0055:       CALL PRINT.PAGE.PROCESS (SEQUENCE, NUM.LINES, ERRMSG)
0056:       IF ERRMSG # '' THEN
0057:           GOSUB DISPLAY.ERRORS
0058:       END
0059:       LINES.PRINTED.THIS.PAGE = NUM.LINES
0060:       RETURN
```

```
0061:  PRINT.BODY:
0062:      ...custom code goes here, and may contain logic like:
0063:          LINE.TO.PRINT = ...
0064:          PRINT LINE.TO.PRINT
0065:          LINES.PRINTED.THIS.PAGE += 1
0066:          IF LINES.PRINTED.THIS.PAGE > PAGE.SIZE THEN
0067:              GOSUB PRINT.HEADER
0068:          END
0069:      ...end of custom code...
0070:      RETURN
0071:
0072:  PRINT.FOOTER
0073:      SEQUENCE = 2
0074:      CALL PRINT.PAGE.PROCESS (SEQUENCE, NUM.LINES, ERRMSG)
0075:      IF ERRMSG # '' THEN
0076:          GOSUB DISPLAY.ERRORS
0077:      END
0078:      LINES.PRINTED.THIS.PAGE += NUM.LINES
0079:      RETURN
0080:
0081:  DISPLAY.ERRORS:
0082:      COUNT.ERRORS = COUNT(ERRMSG,@VM) + 1
0083:      FOR WHICH.ERROR = 1 TO COUNT.ERRORS
0084:          CRT ERRMSG<1,WHICH.ERROR>
0085:      NEXT WHICH.ERROR
0086:      RETURN
0087:  END
```

Chapter 6. Appendices

Appendix A -- Special Characters

*	Comment character (definition section)
&	Section declaration character
#	Predefined field character
$	Print directives character (form section)
%	Field name delimiter (form section)
/	File name-field name delimiter

Appendix B -- PRINT.PAGE command line syntax

The syntax for running PRINT.PAGE from the command line is:

```
PRINT.PAGE unidata_file wp_dir wp_file
        [-FORM form]
        [-AT printer]
        [-TTY]
        [-NODISPLAY]
        [-COPIES n]
        [-HELP]
        [-NOERRORS]
        [-MAIL addressee.field -SENDER sender.name
            -SUBJECT email.subject [-ATTACH form.to.attach]]
```

Where:

"unidata_file"	is the Unidata file that the records have been selected from, and is also the primary file for the PRINT.PAGE form.
"wp_dir"	is the type-1 file that contains the PRINT.PAGE form description file.
"wp_file"	is the PRINT.PAGE form description file.
"-FORM form"	is an optional clause to set the form name (or spool attribute) to "form".
"-AT printer"	is an optional clause to set the printer name to "printer".
"-TTY"	is an optional clause that causes the form to be displayed on the terminal screen instead of being written to the spool queue.
"-NODISPLAY"	is an optional clause that suppresses the normal status and record count display that PRINT.PAGE updates as it is running. This is useful for running PRINT.PAGE from an UniBasic EXECUTE statement.
“-COPIES n”	is an optional clause that specifices the number of copies to produce.
“-HELP”	displays a short help file.

"-NOERRORS" suppresses error messages during form printing (but not during reading of the form definition file)

"-MAIL" sends the output via e-mail rather than printing it. The "addressee.field" specifies an i-descriptor that returns the recipient's e-mail address. The "sender.name" field is a literal which specifies the sender's e-mail address. The "subject" is a literal which will be used as the subject line on the e-mail message. The "-ATTACH" option specifies a PRINT.PAGE form-definition file to be used to generate an attachment to the e-mail message.

PRINT.PAGE runs from the default active select list (select list 0), so either a SELECT command or a GET.LIST command must be executed prior to running PRINT.PAGE in order for any records to be processed.

Appendix C -- PRINT.PAGE syntax

```
*============================================================
*  Initialization section
*
&INITIALIZATION
  initialization routine (paragraph or sentence)
*
*
*
*============================================================
*  Termination section
*
&TERMINATION
  termination routine (paragraph or sentence)
*
*
*
*============================================================
*  Definition section
*
*
&DEFINITION
*
*
*   Form Options
*
LINE.UP num_line_up_forms
TOP.OF.FORM [FORM.FEED\BLANKS\NONE]
FORM.LENGTH number_of_lines_on_page
PRINT.LINES.ON.FORM number_of_lines_to_use
*
*
*   File Options
*
PRIMARY.FILE primary_file_name
SECONDARY.FILE secondary_file_name key_from_primary
*
*
*   Field options
*
*     Dictionary fields
*
  FIELD pp_field [unidata_field]
    FMT fmt_code
    CONV conv_code
    SPECIAL.CONV special_conv_code
    MAX.LENGTH maximum_print_length
```

```
    DEFAULT.VALUE literal_value
    BY.EXP
    NUM.VALUES num_to_use [LAST.VALUES]
    NUM.SUBVALUES num_to_use
    EXPAND.LINES [ON\OFF]
    INSERTING
    MULTI.VALUE
*
*
*     Predefined fields
*
  FIELD pp_field #predefined_field
    IGNORE.ERROR [fmt|conv]
    FMT fmt_code
    CONV conv_code
    SPECIAL.CONV special_conv_code
    MAX.LENGTH maximum_print_length
    INSERTING
*
*
*     Prompt fields
*
  FIELD pp_field
    PROMPT [text [input_mask]] [VERIFY.FILE file_name]
    FMT fmt_code
    MAX.LENGTH maximum_print_length
    DEFAULT.VALUE literal_value
    NUM.VALUES num_to_use [LAST.VALUES]
    EXPAND.LINES [ON\OFF]
    INSERTING
    MULTI.VALUE
*
*
*     Evaluate fields
*
  FIELD pp_field
    EVALUATE phrase_to_evaluate
    FMT fmt_code
    CONV conv_code
    SPECIAL.CONV special_conv_code
    MAX.LENGTH maximum_print_length
    DEFAULT.VALUE literal_value
    BY.EXP
    NUM.VALUES num_to_use [LAST.VALUES]
    NUM.SUBVALUES num_to_use
    EXPAND.LINES [ON\OFF]
    INSERTING
    MULTI.VALUE
*
```

```
*
*     Subroutine fields
*
  FIELD pp_field
    SUBROUTINE subroutine_name
    FMT fmt_code
    CONV conv_code
    SPECIAL.CONV special_conv_code
    MAX.LENGTH maximum_print_length
    DEFAULT.VALUE literal_value
    BY.EXP
    NUM.VALUES num_to_use [LAST.VALUES]
    NUM.SUBVALUES num_to_use
    EXPAND.LINES [ON\OFF]
    INSERTING
    MULTI.VALUE
*
*
*
*============================================================
*  Form section
*    (HEADING, BODY, and FOOTING subsections)
*
&HEADING
&BODY
&FOOTING

   %pp_field%      literal text
   %unidata_field%
   %unidata_file/unidata_field%
   %#predefined_field%

$NEW.PAGE
literal text
```

Appendix D -- PRINT.PAGE defaults

D.1 Form options

FORM.LENGTH	66
LINE.UP	0
PRINT.LINES.ON.FORM	If FORM.LENGTH is greater than 6 then FORM.LENGTH - 6 else FORM.LENGTH
TOP.OF.FORM	If FORM.LENGTH is 66 then FORM.FEED else BLANKS

D.2 File options

PRIMARY.FILE	The Unidata file name entered on the command line.
SECONDARY.FILE	(not applicable)

D.3 Dictionary fields

FIELD	The PRINT.PAGE field name defaults to the dictionary field name.
CONV	CONV code from the dictionary
FMT	FMT code from the dictionary
SPECIAL.CONV	(not applicable)
MAX.LENGTH	none
DEFAULT.VALUE	(not applicable)
BY.EXP	(not applicable)

NUM.VALUES	If EXPAND.LINES is ON, then all values else 1 value
LAST.VALUES	off (use first values in list)
NUM.SUBVALUES	If EXPAND.LINES is ON, then all subvalues else 1 subvalue
EXPAND.LINES	On (shift text lines down to fit)
INSERTING	Off (overlay text to the right)
MULTI.VALUE	SM flag from the dictionary

D.4 Predefined fields

FIELD	The PRINT.PAGE field name defaults to the predefined field name.
CONV	CONV code from predefined table
FMT	FMT code from predefined table
SPECIAL.CONV	(not applicable)
MAX.LENGTH	none
DEFAULT.VALUE	(not applicable)
BY.EXP	(not applicable)
NUM.VALUES	(not applicable)
LAST.VALUES	(not applicable)
NUM.SUBVALUES	(not applicable)
EXPAND.LINES	(not applicable)
INSERTING	Off (overlay text to the right)
MULTI.VALUE	Off (single value)

D.5 Prompt fields

FIELD	none
PROMPT	The prompt text defaults to the PRINT.PAGE field name. The input mask defaults to a single pound sign. By default, there is no VERIFY.FILE.
CONV	none
FMT	none
SPECIAL.CONV	(not applicable)
MAX.LENGTH	none
DEFAULT.VALUE	(not applicable)
BY.EXP	(not applicable)
NUM.VALUES	If EXPAND.LINES is ON, then all values else 1 value
LAST.VALUES	off (use first values in list)
NUM.SUBVALUES	(not applicable)
EXPAND.LINES	On (shift text lines down to fit)
INSERTING	Off (overlay text to the right)
MULTI.VALUE	Off (single value)

D.6 Evaluate fields

FIELD	none
CONV	none
FMT	none
SPECIAL.CONV	(not applicable)
MAX.LENGTH	none
DEFAULT.VALUE	(not applicable)

BY.EXP	(not applicable)
NUM.VALUES	If EXPAND.LINES is ON, then all values else 1 value
LAST.VALUES	off (use first values in list)
NUM.SUBVALUES	If EXPAND.LINES is ON, then all subvalues else 1 subvalue
EXPAND.LINES	On (shift text lines down to fit)
INSERTING	Off (overlay text to the right)
MULTI.VALUE	Off (single value)

D.7 Subroutine fields

FIELD	none
CONV	none
FMT	none
SPECIAL.CONV	(not applicable)
MAX.LENGTH	none
DEFAULT.VALUE	(not applicable)
BY.EXP	(not applicable)
NUM.VALUES	If EXPAND.LINES is ON, then all values else 1 value
LAST.VALUES	off (use first values in list)
NUM.SUBVALUES	If EXPAND.LINES is ON, then all subvalues else 1 subvalue
EXPAND.LINES	On (shift text lines down to fit)
INSERTING	Off (overlay text to the right)

MULTI.VALUE	Off (single value)

Appendix E -- Defining Fonts to PRINT.PAGE

When PRINT.PAGE output is spooled, the font used is the default font for the printer. The font can be changed for the entire PRINT.PAGE output, or for sections of that output. Font definitions are stored in a file I_PP.FONTS.COMMON in the PRINT.PAGE source directory. A system administrator can change the fonts and/or add new fonts by editing this file and then recompiling PRINT.PAGE.LOAD.

The standard I_PP.FONTS.COMMON file assumes that the output is being sent to a Hewlett-Packard LaserJet printer using PCL coding. The following fonts are defined in the standard file:

Courier	abcdefghijk 1234567890
CGTimes	abcdefghijk 1234567890
Arial	abcdefghijk 1234567890
Courier Bold	**abcdefghijk 1234567890**
Courier Italic	*abcdefghijk 1234567890*
Helvetica	abcdefghijk 1234567890