Program Customization

Version 2.0 February 9, 2005

Some definitions:

A program segment is a universal code fragment, used in many programs
A code segment is a version of a program segment for a particular program
GS means "green screen" or a character based screen usable on a dumb terminal
GUI means a "GUI" application using the AccuTerm GUI environment

Mostly you customize a program my creating code segments. You may make "global" changes by modifying program segments but there could be problems with future releases.

These segments can be divided into two groups:

"Infrastructure" segments like the start and close of the program. There is only one of any particular type in any given program.

"Field" segments – repeated for each field in the program.

In addition, some of both kinds can have code segments defined at the file level. That is, custom code written for a "postread" routine will be inserted in any program where the file is read (unless overridden by a segment for a particular program)

When generating the program there is a particular hierarchical sequence for the various segments.

For "infrastructure" segments:

Code Segment	"program*name" e.g. UPD.PARTS*START
Code Segment	"file*name" (if applicable) e.g. PARTS*POSTREAD
Program Segment	"type.name" (where type is ATG for GUI, UPD for GS) e.g. ATG.FORM, UPD.SCREEN
Program Segment	"ALL.name" (used for both GUI & GS) e.g. ALL.START

For "field" segments:

Code Segment	"program*attr*event" e.g. UPD.PARTS*PART-NO*VALIDATE
Code Segment	"file*attr*event" e.g. PARTS*PART-NO*VALIDATE
Program Segment	"type.event.class" e.g. UPD.VALIDATE.FILE (for file validations)
Program Segment	"ALL.event.class" eg. ALL.VALIDATE.FILE

In previous versions, the "Input" program segments (UPD.SIMPLE, UPD.FILE, etc) have been split a bit, in order to make code re-usable between GS at ATG programs. There are now four pieces to one input…

ACTIVATE	(before any input. Saves orig value, etc.)
INPUT	(the call to TPH.INPUT and the LOOP UNTIL CORRECT logic. Does not exist for ATG – that is done by the GUI driver)
VALIDATE	(Checks if on file, etc.)
DEACTIVATE	(Does any calculations, displays any "display" fields, etc.)

Sequence of operation:

Define the program (as before)
Create the GS layout (with new painter)
Generate, compile and test GS program
Build GUI layout from GS. This creates an item that is compatible with the AccuTerm GED editor, which can be used to change look of the GUI screen (move things around, change fonts, colors etc.)
Generate, compile and test ATG program

Note: when I say "program" I mean the base program, e.g. "UPD.CUSTOMER" -- there is a code segment variable that you can use to include code only for GS or ATG.

The idea is to be able to re-use code customizations between the different platforms. Obviously this can't be 100% but it should be pretty close.

Segment Variables:

Each program or code segment can have "replacement variables" that will be resolved to actual values when the program is generated. In this way one fragment can be used a template for many programs.

Like the segments, they can be roughly divided into "infrastructure" and "field" variables. Below are some samples, but this list is by no means complete (there are about 90 defined at this point)

Infrastructure Variables

%PGM **Program name**. This variable holds the name of the program being generated.

%TYPE **Program Type.** This variable will be (for example) GS or ATG

%TITLE **Program Title.** This variable holds the title of the program as entered in the **Update Program Definition** screen.

%FN **Data File Name**. This is the name of the file as it was designated at the time of creation. For example, if the file was called **CUSTOMER**, the **%FN** variable would contain the value **CUSTOMER**.

%FV **File variable name**. This is the internal BASIC file variable name used in the **OPEN** statement. For example, if the file name is **CUSTOMER**,

the **%FV** variable would contain the value **CUSTOMER.FILE**. All file names used by **The Programmer's Helper** are appended with "**.FILE**" for internal use. (**NOTE:** if the file was originally named with the ".FILE" extension, the code generator ignores it and leaves it as is.)

%RECORD **Record Name**. This is the name of the array that will be used to **MATREAD** and **MATWRITE** the item. For example, if the file name used in the program is **CUSTOMER.FILE**, the name of the data array would be **CUSTOMER.REC**. All arrays used to contain file data use the file name with an suffix of "**.REC**".

Field Variables

The following replacement variables are called "**field**" or "**step**" variables. Their values will continually change from step to step and they are only defined within a data entry step.

%STEP **Step Number**. This variable contains the step number of the attribute being entered.

%TEXT **Prompt Text**. This is the prompt text that is displayed for each step. For example, let's look at step 7.

%COL **Input Column**. This is the data entry input column number for this step.

%LINE **Input Line**. This is the data entry input line number for this step.

%LEN **Input Length**. This is the maximum allowable input length for this step.

%REQ **Required Flag**. This variable contains the value "**R**" if the step is required and cannot be bypassed.

%TPOS **Text Position**. This is the column number where the prompt text will be printed for this step.

%TLINE **Text Line.** This is the line number where the prompt text will be displayed.

%MV **Multivalue Flag**. This variable contains the value one (1) if the step is part of a multivalued window.

Conditional Statements in a Segment

There can be conditional statements in a program or code segment. In this manner code that is applicable for one type of program or one type of field can be included (or excluded)

The general format is

```
%IF %VAR op value
<statements to be included if expression is true>
%ELSE
<statements to be included if the expression is false>
%END
```

“op” can be

EQ	Equals
NE	Not Equal
GT	Greater Than
LT	Less Than
GE	Greater than or equal
LE	Less than or equal
$	“contains” (similar to the basic INDEX() function)

A special case of

```
%IF %VAR
```

Test for a non-zero, non-null value

The value is not entered in quotes, for example:

```
%IF %TYPE EQ GS
*% Green screen (this is a sgmt comment)
        <green screen statements>
%ELSE
*% Gui
        <GUI statements>
%END
```