



Using Messages

Overview

Starting with FACTS 7, messages may be entered into a database (SMMSGs) of messages. Instead of setting the traditional Z\$ to the actual text of the message, it may be set to the code associated with the desired message. The message program, SMC020, then reads the message file to get the actual text of the message.

A custom messages file may also be used, SMCMSG. Messages put into SMCMSG take precedence over messages with the same code in SMMSGs. This allows for customization of the messages without changing the standard message file distributed with the software.

Standard messages may be entered or edited with the file maintenance SMF996, and custom messages may be entered or edited with the file maintenance SMF995.

One purpose of creating and using a message database is to allow for future language independence. Software Solutions, Inc., recommends that all messages be placed in the message database.

► Chapter Outline

Overview

- Messages in Driver Programs
- Messages in Other Programs
- Assigning Message Codes
- Using Variables in Messages
- Telling the Drives to Issue No Message

Messages in Driver Programs

When working with the driver programs, to give a message to the user, the variable `MESSAGE$` should be set to the message code. In most cases, this variable should be set in conjunction with `GO_BACK` or `FAILED` being set to 1.

Messages in Other Programs

Most existing programs set the variable `Z$` and do a `GOSUB 8810`. To use the message file, simply set `Z$` to the message code instead of the message text.

Assigning Message Codes

Messages that occur frequently should be given descriptive code, such as “`BAD_DATE`”, “`CURR_GL_PER`”, etc., so they are easier to remember and recognize. Messages that will likely be used only once should be given numbers as the code.

Custom messages should be given codes that are unlikely to be used for standard messages. Typically that would involve using some type of consistent prefix that would not make sense for a descriptive code.

Using Variables in Messages

Frequently a message needs to have values filled in at runtime, like inserting the bad Customer Number the user entered. To do this, you must first insert the place holder where the value is to be inserted. Place holders are numbered and are preceded by a percent sign.

In the following example there are two values that would be filled in at runtime:

Code: `CREDIT_LIMIT`

Text: Customer ‘%1’ has exceed the credit limit of \$%2.

The program that issues the message should set `Z$` or `MESSAGE$` (depending on context) to:

`“CREDIT_LIMIT”+$0a$+customer_number$+$0a$+credit_limit$+$0a$`

The value following the first `$0a$` replaces %1 and the value following the second `$0a$` replaces %2. If `customer_number$=“ C100”` and `credit_limit$=“ 1000”`, the message would be displayed as follows:

Customer ‘ C100’ has exceeded the credit limit of \$ 1000.

To prevent the blank spaces, CVS() the variables with ,3.

Telling the Drivers to Issue No Message

When you set GO_BACK=1 in a validation procedure, the driver will issue the generic message “Invalid data in field” if you do not set MESSAGE\$. To set GO_BACK=1 and tell the driver that it should not issue any message at all, set MESSAGE\$=”<none>”.