# EMS FastStart Manual


## Abstract

This manual provides information about the EMS FastStart program including installation, event message design, instructions for running the program, and customizing EMS FastStart to fit the specific needs of your application. In addition, this manual covers event message design to help you make the best use of the EMS FastStart program. The code examples in this manual are written in COBOL85.


## Product Version

Guardian 90 C20

## Supported Releases

This manual supports function added to C20.00, and supports all subsequent releases until otherwise indicated in a new edition.


| Part Number | Published | Release ID |
|---|---|---|
| 133701 | August 1997 | D44.00 |

## Document History

| Part Number | Product Version | Published |
|---|---|---|
| 031022 | Guardian 90 C20 | March 1990 |
| 058659 | Guardian 90 C20 | February 1991 |
| 133701 | Guardian 90 C20 | August 1997 |

New editions incorporate any updates issued since the previous edition.

A plus sign (+) after a release ID indicates that this manual describes function added to the base release, either by an interim product modification (IPM) or by a new product version on a .99 site update tape (SUT).

## Ordering Information

For manual ordering information: domestic U.S. customers, call 1-800-243-6886; international customers, contact your local sales representative.

## Document Disclaimer

Information contained in a manual is subject to change without notice. Please check with your authorized Tandem representative to make sure you have the most recent information.

## Export Statement

Export of the information contained in this manual may require authorization from the U.S. Department of Commerce.

## Examples

Examples and sample programs are for illustration only and may not be suited for your particular purpose. Tandem does not warrant, guarantee, or make any representations regarding the use or the results of the use of any examples or sample programs in any documentation. You should verify the applicability of any example or sample program before placing the software into productive use.

## U.S. Government Customers

FOR U.S. GOVERNMENT CUSTOMERS REGARDING THIS DOCUMENTATION AND THE ASSOCIATED SOFTWARE:

These notices shall be marked on any reproduction of this data, in whole or in part.

**NOTICE:** Notwithstanding any other lease or license that may pertain to, or accompany the delivery of, this computer software, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Section 52.227-19 of the FARS Computer Software—Restricted Rights clause.

**RESTRICTED RIGHTS NOTICE:** Use, duplication, or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013.

**RESTRICTED RIGHTS LEGEND:** Use, duplication or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the rights in Technical Data and Computer Software clause in DAR 7-104.9(a). This computer software is submitted with "restricted rights." Use, duplication or disclosure is subject to the restrictions as set forth in NASA FAR SUP 18-52 227-79 (April 1985) "Commercial Computer Software—Restricted Rights (April 1985)." If the contract contains the Clause at 18-52 227-74 "Rights in Data General" then the "Alternate III" clause applies.

U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

# New and Changed Information

This is the third edition of the *EMS FastStart Manual*. This is only a revision of the format. No content has been changed. The following is the New and Changed Information from the second edition, the last edition with changes in content:

## New Features

EMS FastStart has the following new features:

- The token ZEMS-TKN-SUPPRESS-DISPLAY is now supported. Viewpoint and the EMS distributors use this token to select whether a specific event is displayed. A new field in the egen-record (suppress-display) was added to support this token. See Section 6, Building Your Application for Event Generation.
- The token ZSPI-TKN-MANAGER is now supported. This token specifies the process name of a particular subsystem process. A new field in the egen-record (subsystem-manager) was added to support this token. See Section 6, Building Your Application for Event Generation.
- EMS FastStart now supports template facilities. See Section 4, Preparing the Application Configuration File.
- The following data types were added: ZSPI-DDL-TIMESTAMP, ZSPI-DDL-TRANSID, ZSPI-DDL-SSID.
- EMS FastStart now supports the C language. EMS FastStart includes a version of the ATM example written in the C language. See Section 3, Installing and Configuring EMS FastStart.
- EMS FastStart has a new ACF format. See Section 4, Preparing the Application Configuration File.)
    - SAVE-DDL--DICTIONARY on page 4-4 key word was added. This is useful in saving the DDL dictionary. The DDL dictionary is needed for creating the templates for your application.

      The parameter specifies if the DDL dictionary created in the subvolume specified by the USER-SUBVOL parameter will be saved. Valid parameter values are YES (save the dictionary) and NO (do not save the dictionary; purge it). This dictionary will contain all the definitions for ZSPIDDL and ZEMSDDL plus your own subsystem definitions (tokens and events). This dictionary is required if you want to use the EMS Template Services and create templates for your application.
    - USER-DDL--FILE on page 4-4 key word in ACF has been added so that you can include your own DDL definitions during the compilation of EMS FastStart DDL.

      The parameter specifies the name of a user defined DDL source file. Valid parameter values are file-name (a valid DDL source file) and NOT-USED (do not source a file). The content of this file will be sourced at the end of the MAIN DDL file and added to the dictionary. You may use this file to source your event

definitions, which will then be used later by the EMS Template Services for generating your application template file.

# Changed Information

EMS FastStart has the following changed information:

- The file names of the ATM example have changed. See Section 3, Installing and Configuring EMS FastStart.
- In Table 6-1, Procedures Required for Each Mode, the procedures required for EGEN modes 1 to 4 has been corrected in Section 6, Building Your Application for Event Generation.

# Contents

## Examples



## Figures

## Tables

# About This Manual

This manual provides information about the EMS FastStart program including installation, event message design, instructions for running the program and customizing EMS FastStart to fit the specific needs of your application. In addition, this manual covers event message design to help you make the best use of the EMS FastStart program. The code examples in this manual are written in COBOL85.

## Who Should Read This Manual?

This manual will be of interest to anyone involved with designing event messages or designing and coding applications that will use event message management. This includes system managers and programmers who use the Event Management Service and are installing EMS FastStart.

## What You Should Know

System managers and programmers should be familiar with the GUARDIAN 90 operating system and the Event Management Service software. Programmers should be familiar with C, COBOL85, or TAL.

## What This Manual Covers

This manual consists of seven sections and five appendixes as follows:

- Section 1, Introduction, presents a broad view of the EMS FastStart program including its applications and uses; required hardware and software; how to choose between EMS and EMS FastStart; and information about running EMS FastStart.
- Section 2, Brief Review of Event Design, is a review of event message design and a set of instructions with examples to help you optimize event message design in order to make the best use of EMS FastStart.
- Section 3, Installing and Configuring EMS FastStart, describes the program installation and configuration, how to modify the Compiler Configuration File (CCF), and security considerations.
- Section 4, Preparing the Application Configuration File, describes the Application Configuration File (ACF), including a listing of the default file and instructions for modifying it to fit your application's needs.
- Section 5, Running EMS FastStart, tells you how to run (and stop) the program and includes an example of the program output you see on your terminal. This section also lists the user subvolume files and briefly describes the resulting program components.
- Section 6, Building Your Application for Event Generation, is the most important section for programmers. It tells you the tasks to perform to build your own application and generate your event messages with EMS FastStart.

- Section 7, Testing Program and Filter, contains information on the EMS FastStart test program and the use of the filter program with a printing distributor.
- Appendixes A through E contain the EMSFS and EGEN error, warning, and advisory messages, a COBOL85 program example, copy libraries, a sample template source file, and the filter program.

# Related Documents

This manual assumes you are familiar with the GUARDIAN 90 operating system, the Event Management Service (EMS) program, and TACL. In using EMS FastStart, you may find it helpful to refer to the following documents:

- *Distributed Systems Management (DSM) Programming Manual*
- *VIEWPOINT Manual*
- *Introduction to Distributed Systems Management (DSM)*
- *Event Management Service (EMS) Manual*
- *DSM Template Services Manual*

# Your Comments Invited

After using this manual, please take a moment to send us your comments. You can do this by returning a Reader Comment Card or by sending an Internet mail message.

A Reader Comment Card is located at the back of printed manuals and as a separate file on the Tandem CD Read disc. You can either FAX or mail the card to us. The FAX number and mailing address are provided on the card.

Also provided on the Reader Comment Card is an Internet mail address. When you send an Internet mail message to us, we immediately acknowledge receipt of your message. A detailed response to your message is sent as soon as possible. Be sure to include your name, company name, address, and phone number in your message. If your comments are specific to a particular manual, also include the part number and title of the manual.

Many of the improvements you see in Tandem manuals are a result of suggestions from our customers. Please take this opportunity to help us improve future manuals.

# Notation Conventions

## General Syntax Notation

The following list summarizes the notation conventions for syntax presentation in this manual.

**UPPERCASE LETTERS.** Uppercase letters indicate keywords and reserved words; enter these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

**lowercase italic letters.** Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

```
file-name
```

**[ ] Brackets.** Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name
```

```
INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list may be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
LIGHTS [ ON              ]
       [ OFF             ]
       [ SMOOTH [ num ] ]
```

```
K [ X | D ] address-1
```

**{ } Braces.** A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list may be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }
```

```
ALLOWSU { ON | OFF }
```

**| Vertical Line.** A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

**… Ellipsis.** An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address-1 [ , new-value ]...
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char..."
```

**Punctuation.** Parentheses, commas, semicolons, and other symbols not previously described must be entered as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;

LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must enter as shown. For example:

```
"[" repetition-constant-list "]"
```

**Item Spacing.** Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In the following example, there are no spaces permitted between the period and any other items:

```
$process-name.#su-name
```

**Line Spacing.** If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] CONTROLLER

   [ , attribute-spec ]...
```

# Notation for Messages

The following list summarizes the notation conventions for the presentation of displayed messages in this manual.

**Nonitalic text.** Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

**lowercase italic letters.** Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

```
p-register

process-name
```

**[ ] Brackets.** Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list might be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
LDEV ldev [ CU %ccu | CU %... ] UP [ (cpu,chan,%ctlr,%unit) ]
```

**{ } Braces.** A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list might be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LBU { X | Y } POWER FAIL
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

**| Vertical Line.** A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

**% Percent Sign.** A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400
```

```
P=%p-register E=%e-register
```

# Notation for Management Programming Interfaces

**UPPERCASE LETTERS.** Uppercase letters indicate names from definition files; enter these names exactly as shown. For example:

```
ZCOM-TKN-SUBJ-SERV
```

**lowercase letters.** Words in lowercase letters are words that are part of the notation, including Data Definition Language (DDL) keywords. For example:

```
token-type
```

# Change Bar Notation

Change bars are used to indicate substantive differences between this edition of the manual and the preceding edition. Change bars are vertical rules placed in the right margin of changed portions of text, figures, tables, examples, and so on. Change bars highlight new or revised information. For example:

The message types specified in the REPORT clause are different in the COBOL85 environment and the Common Run-Time Environment (CRE).

The CRE has many new message types and some new message type codes for old message types. In the CRE, the message type SYSTEM includes all messages except LOGICAL-CLOSE and LOGICAL-OPEN.

# 1 Introduction

EMS FastStart enhances development of applications under Distributed Systems Management (DSM). EMS FastStart provides a simple, cost-effective way for programmers to develop and test EMS event messages. Event Management Service (EMS) is a component of DSM that is used for collecting and distributing events.

EMS FastStart generates and compiles a number of source files which are used to simplify event generation and testing. EMS FastStart uses a configuration file to generate a module that is bound into the user application. This module, called EGEN, serves as an interface between the user application and EMS. This enables programmers to generate events in the same syntax as the user application.

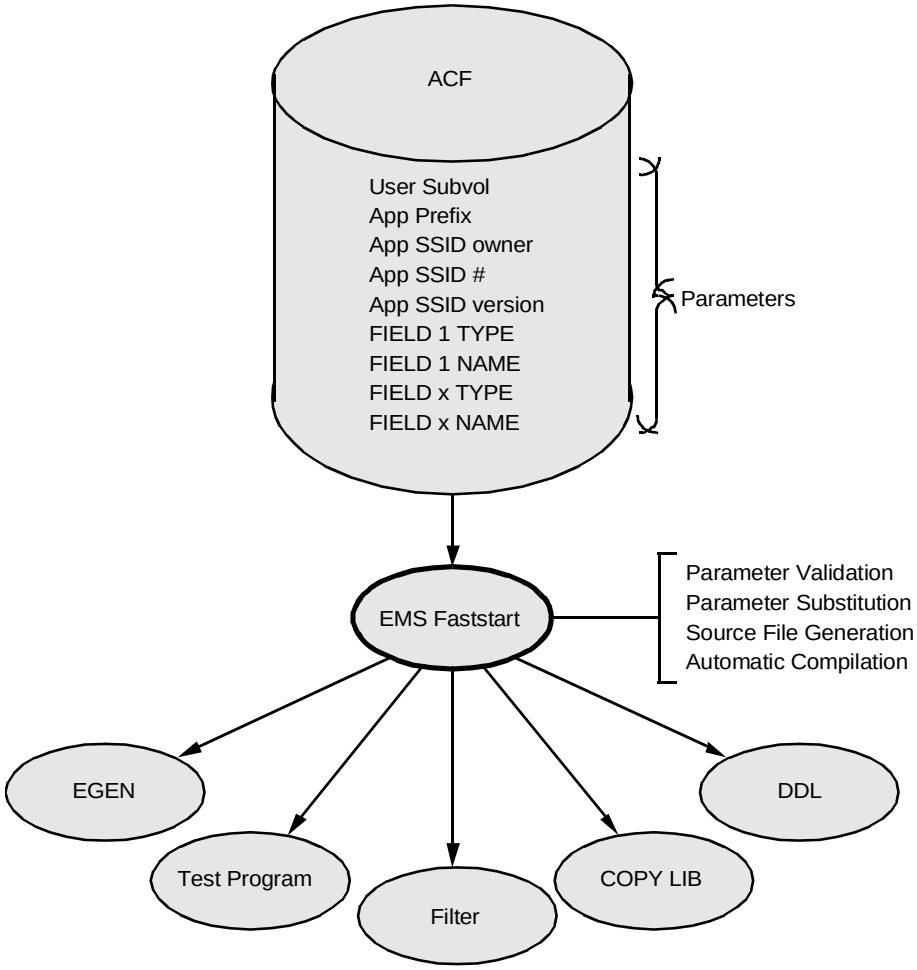
## EMS FastStart Features

EMS FastStart does the following:

- Automates many EMS steps to increase accuracy and save time
- Generates events in the same language as the user application
- Simplifies event generation and saves programming time and effort
- Provides an interactive testing facility to validate newly-defined fields, filters and templates
- Provides complete examples in C, COBOL85 and TAL
- Promotes standardization of error and event processing within applications
- Provides automatic file generation and compilation
- Facilitates testing of the DSM interfaces and functions separately from application developments
- Facilitates using DSM template services
- Supports TAL, C, and COBOL85

Figure 1-1 shows the EMS FastStart process and the resulting components.

**Figure 1-1. The EMS FastStart Process**




*Legend*

The ACF (Application Configuration File) is an EDIT file you use to set up your event parameters.

001

# How EMS FastStart Works

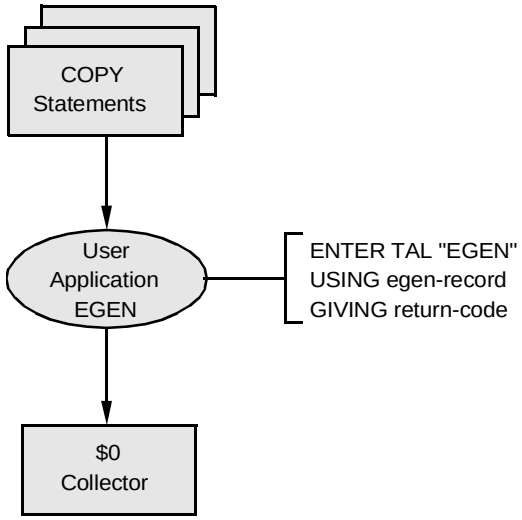
EMS FastStart is a TACL-based code generator which generates and compiles a number of source files which are used to simplify event generation and testing. EMS FastStart creates these routines, programs, and copy libraries:

- EGEN
- Test Program and Filter
- DDL Source Files for C, COBOL85, and TAL

## EGEN

EGEN is a TAL routine that provides a high-level interface between your application and the Event Management Service (EMS). The application interfaces with EGEN by moving values into the EGEN record structure (defined by the Application Configuration File). EGEN generates tokenized events on behalf of the application by performing a WRITE-READ of the event buffer to the collector $0 or an alternate location you specify using run-time DEFINEs. Figure 1-2 illustrates how EGEN interfaces with your application. EGEN supports four modes of operation which allow it to be used in COBOL85 applications as well as multi-threaded TAL applications.

**Figure 1-2. EGEN Interface**




## Test Program and Filter

The EMS FastStart test program lets you enter values for events based on the ACF. It is an interactive COBOL85 program that calls EGEN to generate events and is compiled as part of EMSFS. This program lets you test the interface between your application and EGEN. It lets you test EGEN with your ACF, DSM applications, filters and templates before an application has been completed, saving time and effort in filter and event

validation. Also, you can isolate testing data from production data by writing events to a location other than the event log.

When used with EGEN, the testing program and filter create an interactive testing facility which allows immediate testing of newly defined events through an easy-to-use conversational interface.

EMS FastStart creates and compiles a test filter that can be run with a printing distributor or Viewpoint to test your application events.

## DDL Source Files for C, COBOL85, and TAL

EMS FastStart automatically generates copy statements for COBOL85 and source definitions for TAL and C that can be sourced in your application based on the ACF.

- The COBOL85 copy libraries are located in the file *xxxx*COB (for instance, the example in this manual is ATM1COB because the file prefix is ATM1).
- The TAL source definitions are located in the file *xxxx*TAL.
- The C source definitions are located in the file xxxxC.

These files contain the event record layout (EGEN-RECORD), the EGEN interface definitions and the EGEN return codes.

# Choosing Between EMS and EMS FastStart

EMS FastStart is designed to streamline the event generation process. It offers programmers a simplified interface with the Event Management Service (EMS) for building event buffers. Although EMS FastStart provides a comprehensive subset of EMS functions, the demands of your application might require you to use EMS rather than EMS FastStart. To determine whether to use EMS or EMS FastStart, you must evaluate the needs of your application.

---

**Note.** The EMS term “token” is used here to explain the differences in using EMS and EMS FastStart. In the rest of this manual, the term “variable data field” (specific to EMS FastStart) is substituted for “token” (specific to EMS). For an explanation of token, refer to any of the Distributed System Management manuals such as the *Event Management Service (EMS) Manual*.

---

The following are examples of situations where you may want to use EMS rather than EMS FastStart for generating events:

- The application has events with more than one subject (EMS FastStart has a limit of one subject per event).
- The application specifies values for header tokens that are not supported by EMS FastStart. EMS allows the application to override the identification of the event generator and timestamp in an event. This is useful if one process generates events on behalf of another.
- The application includes multiple occurrences of a specific token in an event (EMS FastStart allows only one occurrence per event).

- The application has events that include error lists or foreign tokens (tokens owned by a different subsystem than the subsystem generating the event).
- The user of the application needs to check that the correct set of tokens is included in an event at the time the event is generated. EMS FastStart cannot perform explicit checking of the event-generating subprogram.
- The application needs these specialized tokens for the event messages which EMS FastStart cannot generate:
  - Extensible structured tokens
  - Error lists
  - Structured tokens containing numeric values
- The application cannot run within the default values of EMS FastStart for numeric fields. (See Modifying the ACF in Section 4, Preparing the Application Configuration File, for a list of the default values.)

# System and Program Requirements

EMS FastStart runs on the GUARDIAN 90 operating system version C20 or higher on Tandem Nonstop systems.

EMS FastStart uses the following software:

- DDL—The DDL compiler is used to generate the language-specific definition files and the user COPYLIB files.
- TAL—The TAL compiler generates the EMS FastStart customized EGEN procedure.
- EMS—EMS (Event Management Services) is a component of DSM used for collecting and distributing events. The EMS FastStart EGEN routine (a TAL routine bound into the user's application) interfaces with EMS to write events to the collector $0.
- EMF—EMF (Event Management Filter compiler) is used by EMS FastStart to produce a default filter which can be used with VIEWPOINT to display specific events coming from the application.
- COBOL85 (optional)—The COBOL85 compiler compiles the EMS FastStart /EGEN testing application (event generator) written in COBOL85.
- C (optional)—C is required only if you use an application programmed in C.

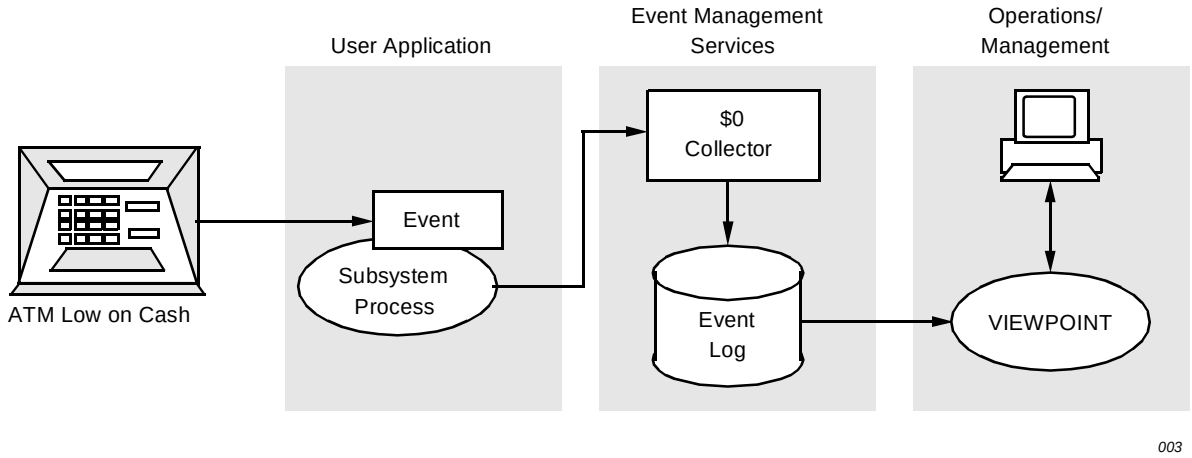
# Using Template Services with EMS FastStart

The template compiler allows you to create templates for formatting and displaying events. For more information, see the *DSM Template Services Manual*.

# 2 Brief Review of Event Design

This section provides guidelines and standards for programmers to follow when designing events. EMS FastStart is designed to simplify the process of generating EMS event messages. It is necessary to design event standards for programmers to follow, whether EMS is used by itself or with EMS FastStart. These standards can increase the overall efficiency of development efforts as well as enable more efficient operation and management of the subsystem. Figure 2-1 shows the EMS event generating and reporting process.

**Figure 2-1. Event Generation and Reporting**




An event is any significant hardware or software incident that you want reported, either to a person or to a program. For example, a message to a person goes to Viewpoint to be read and a message to a program can be used by an application to cause an action or to accumulate statistics. The number of messages created and their design depends on what you need to convey to the people or programs that use the messages.

Some of the questions that need to be considered when developing and using the standards for event message design for applications are:

- What type of information is included in an event?
- What events are critical?
- What event number is assigned to an event?
- Should the event be displayed?
- Who or what will be reacting to the event (people or programs)?

Good event message planning and design before using EMS FastStart helps you benefit from the streamlined event message generation and testing the EMS FastStart program provides. Refer to the *Event Management Service (EMS) Manual* for more information on events. Event message design is described in this section in a series of steps with 12 sample event messages.

# How to Design Events

To help give an overview of event design, the process has been divided into ten steps. Each step uses the same set of 12 sample event messages as an example. These are the same 12 event messages used in the COBOL85 program in Appendix C, COBOL85 Program Example



---

**Note.** Steps 1 through 9 apply to both EMS and EMS FastStart. Step 10 applies to EMS FastStart only.

---

## Step 1: Identify Event Owner (SSID)

Before defining events, you must identify the owner of the events. The DSM architecture and the Event Management Services requires that each event generated by an application be identified by a specific subsystem ID (SSID). The SSID is made up of three parts: a subsystem owner, a subsystem number and a version field. In the COBOL85 ATM example, the SSID is CUSTOMER.1.J00 where CUSTOMER is the subsystem owner, 1 is the subsystem number, and the version is J00.

The SSID identifies the owner of each event in the EMS log files. It can be used to build a filter which selects events generated by a specific subsystem, for example, print only the messages generated by the ATM application. Each application or subsystem should have a different SSID; this will make event filtering and selection much easier.

For your application, you must assign an SSID using the standards defined by EMS. Please refer to the *EMS Manual* for a detailed description of the valid SSID values.

## Step 2: List Messages

Make a list of all event messages necessary for your application. For existing applications, list all of the current messages (for example, do a code search). For new applications, create a list of the messages you will need to generate. The example below shows a typical list of event messages for a banking application.

## Example:

1. ATM SFMAIN01 is up at 245 A St. San Francisco.
2. ATM SFMAIN02 is up at 245 A St. San Francisco.
3. Insufficient funds in account 34503933. Access denied on ATM OAKWEST1 at 245 Oak St. Oakland.
4. Insufficient funds in account 23409344. Access denied on ATM SACTO99 at 341 Main St. Sacramento.
5. ATM SFMAIN02 at 245 A St. San Francisco is low on funds.
6. ATM SFMAIN01 at 245 A St. San Francisco is low on funds.
7. ATM LACENT99 is down at 9320 Main St. Los Angeles.
8. ATM LACENT91 is down at 9320 Main St. Los Angeles.
9. Security breach on account 23457320. Number of accesses attempted 3; ATM SACTO02 is down at 230 State St. Sacramento.
10. Security breach on account 34094443. Number of accesses attempted 2; ATM SACTO01 is down at 230 State St. Sacramento.
11. Hardware failure on ATM LACENT99 at 125 8th Avenue. Component failed is cash dispenser; subcomponent 321561ac; serial number 231234093; sense status 0101110101.
12. Hardware failure on ATM LACENT98 at 125 8th Avenue. Component failed is cash dispenser; subcomponent 321576xx; serial number 343223480; sense status 1010111011.

# Step 3: Identify Data in the Messages

Identify each part of a message as either fixed text or variable data. The variable data and the fixed text are the two elements that define each event and comprise its template. The variable data become the fields into which you move values when you generate your event messages.

The sample messages below show the variables underlined. Fixed text is not underlined.

## Example:

1. ATM SFMAIN01 is up at 245 A St. San Francisco.
2. ATM SFMAIN02 is up at 245 A St. San Francisco.
3. Insufficient funds in account 23457320. Access denied on ATM OAKWEST1 at 245 Oak St. Oakland.
4. Insufficient funds in account 23409344. Access denied on ATM SACTO99 at 341 Main St. Sacramento.
5. ATM SFMAIN02 at 245 A St. San Francisco is low on funds.

6. ATM SFMAIN01 at 245 A St. San Francisco is low on funds.
7. ATM LACENT99 is down at 320 Main St. Los Angeles.
8. ATM LACENT91 is down at 320 Main St. Los Angeles.
9. Security breach on account 23457320. Number of accesses attempted 3; ATM SACTO02 is down at 230 State St. Sacramento.
10. Security breach on account 34094443. Number of access attempted 2; ATM SACTO01 is down at 230 State St. Sacramento.
11. Hardware failure on ATM LACENT99 at 125 8th Ave. Los Angeles. Component failed is cash dispenser; subcomponent 321561ac; serial number 231234093; sense status 0101110101.
12. Hardware failure on ATM LACENT98 at 125 8th Ave. Los Angeles. Component failed is cash dispenser; subcomponent 321561ac; serial number 343223480; sense status 1010111011.

## Step 4: Identify Groups of Variables

In this step you identify groups of variables, or variable types, in your messages (for example, account numbers). It is important to understand that a single variable type can occur in any number of event messages (for example, account number).

In the sample event messages there are eight types or groups of variables:

1. Name
2. Location
3. Account number
4. Hardware component
5. Hardware subcomponent
6. Serial number
7. Sense status
8. Retry limit

The variables from the messages are shown below in the appropriate group.

1. Name

| | |
|---|---|
| SFMAIN01 | SACTO99 |
| SFMAIN02 | LACENT91 |
| OAKWEST1 | LACENT98 |
| SACTO01 | LACENT99 |
| SACTO02 | |

2. Location

245 A St. San Francisco
245 Oak St. Oakland
341 Main St. Sacramento
320 Main St. Los Angeles
230 State St. Sacramento
125 8th Ave. Los Angeles

3. Account Number

   23409344
   23457320
   34094443

4. Hardware Component

   cash dispenser

5. Hardware Subcomponent

   321561ac

6. Serial Number

   231234093
   343223480

7. Sense Status

   0101110101
   1010111011

8. Retry Limit

   3
   2

There is additional standard event information automatically included by EMS as part of each event:

- Timestamp showing the time the event was logged to the collector $0
- System number of the system that generated the event
- PIN of the reporting subsystem process
- CPU of the reporting subsystem process

For a complete list of standard event information, see the *Event Management Service (EMS) Manual*.

# Step 5: Assign Field Name, Type, and Number to Variables

This step is the beginning of defining the variable data fields for your application. Each variable data field has three components:

- FIELD-*x*-NAME
- FIELD-*x*-TYPE
- FIELD-*x*-NUMBER

Note that the number between the hyphens (for example, *x* in the example above) is the *group field number*. This number is associated with all three components that make up a field in the ACF. Each group of three components shares a group field number. This group field number is distinct from the field NUMBER and the event number. (See Step 6: Assign Event Numbers for information on event numbers and Step 9: Build Template File for information on group field numbers.)

## Assign a Field Name for Each Variable

Give each field a unique name that identifies the type of information passed in the event buffer. The field name must be COBOL85 compatible and is limited to 20 alphanumeric characters and hyphens. Below is an example of a field name (**bold type**):

| | |
|---|---|
| FIELD-7-NAME | **ATM-SENSE-STATUS** |
| FIELD-7-TYPE | ZSPI-DDL-INT2 |
| FIELD-7-NUMBER | 700 |

## Assign a Field Type for Each Variable

Each variable data field has a DDL type defined as ZSPI-DDL-*xxxx*, where *xxxx* is the data type you define to suit the needs of your application. Below is an example of a field type (**bold type**):

| | |
|---|---|
| FIELD-7-NAME | ATM-SENSE-STATUS |
| FIELD-7-TYPE | **ZSPI-DDL-INT2** |
| FIELD-7-NUMBER | 700 |

FastStart uses the data types (see the *DSM Programming Manual* for more information) listed in Table 2-1 below.

**Table 2-1. Data Types**

| | |
|---|---|
| ZSPI-DDL-INT | ZSPI-DDL-CHAR24 |
| ZSPI-DDL-INT2 | ZSPI-DDL-CHAR50 |
| ZSPI-DDL-INT4 | ZSPI-DDL-CHAR64 |
| ZSPI-DDL-UINT | ZSPI-DDL-CHAR128 |
| ZSPI-DDL-ENUM | ZSPI-DDL-CHAR254 * |
| ZSPI-DDL-BOOLEAN | ZSPI-DDL-USERID |
| ZSPI-DDL-CHAR | ZSPI-DDL-USERNAME |
| ZSPI-DDL-CHAR-PAIR | ZSPI-DDL-FNAME |
| ZSPI-DDL-CHAR4 | ZSPI-DDL-FNAME32 |
| ZSPI-DDL-CHAR6 | ZSPI-DDL-SUBVOL |
| ZSPI-DDL-CHAR8 | ZSPI-DDL-CRTPID |
| ZSPI-DDL-CHAR16 | ZSPI-DDL-DEVICE |
| ZSPI-DDL-TRANSID | ZSPI-DDL-TIMESTAMP |
| ZSPI-DDL-SSID | |

*ZSPI-DDL--CHAR254 is defined in EXTRADDL. Refer to Section 4, Preparing the Application Configuration File, for detailed information.

## Assign a Field Number for Each Variable

A field number identifies each separate variable data type. The field number can be any number from 1 to 9990. You can assign an arbitrary field number, although you may wish to assign field numbers with some logic for your application. Below is an example of a field number (**bold type):**

| | |
|---|---|
| FIELD-7-NAME | ATM-SENSE-STATUS |
| FIELD-7-TYPE | ZSPI-DDL-INT2 |
| FIELD-7-NUMBER | **700** |

## Examples of Field Names, Types, and Numbers Assigned to Variables

Here are examples of field names, types, and numbers assigned to variables:

| | |
|---|---|
| Field name: | ATM-NAME |
| Field type (DDL): | ZSPI-DDL-CHAR8 |
| Field number: | 100 |
| Variables: | SFMAIN01<br>SFMAIN02<br>OAKWEST1<br>SACTO01<br>SACTO02<br>SACTO99<br>LACENT98<br>LACENT99 |

| | |
|---|---|
| Field name: | ATM-LOCATION |
| Field type (DDL): | ZSPI-DDL-CHAR24 |
| Field number: | 200 |
| Variables: | 245 A St. San Francisco<br>245 Oak St. Oakland<br>341 Main St. Sacramento<br>230 State St. Sacramento<br>125 8th Ave. Los Angeles<br>320 Main St. Los Angeles |

| | |
|---|---|
| Field name: | ATM-ACCOUNT-NUMBER |
| Field type (DDL): | ZSPI-DDL-INT2 |
| Field number: | 300 |
| Variables: | 23409344<br>23457320<br>34094443 |

| | |
|---|---|
| Field name: | ATM-RETRY-LIMIT |
| Field type (DDL): | ZSPI-DDL-INT |
| Field number: | 400 |

| | |
|---|---|
| Variables: | 3<br>2 |

| | |
|---|---|
| Field name: | ATM-HW-COMPONENT |
| Field type (DDL): | ZSPI-DDL-CHAR24 |
| Field number: | 500 |
| Variables: | cash dispenser |

| | |
|---|---|
| Field name: | ATM-HW-SUBCOMPONENT |
| Field type (DDL): | ZSPI-DDL-CHAR24 |
| Field number: | 600 |
| Variables: | 321561AC |

| | |
|---|---|
| Field name: | ATM-SERIAL-NUMBER |
| Field type (DDL): | ZSPI-DDL-INT2 |
| Field number: | 700 |
| Variables: | 231234093<br>343223480 |

| | |
|---|---|
| Field name: | ATM-SENSE-STATUS |
| Field type (DDL): | ZSPI-DDL-INT2 |
| Field number: | 800 |
| Variables: | 0101110101<br>1010111011 |

## Step 6: Assign Event Numbers

In this step, you group all event messages that have *exactly* the same fixed text and the same variable data fields and assign one event number to each group. You should also assign a literal that describe each of the event numbers. These literals will enhance the documentation of your application and will be needed to create the template source file in Step 9: Build Template File. Then create one event message template for each group (numbered event).

An event message template is the design of the fixed text and variable data for one event. All of the event messages produced by the numbered event type will differ only in the contents of the variable data fields.

There are several reasons to assign unique event numbers:

- An event can have a probable cause and a recommended action database within Viewpoint. This database is keyed into by a subsystem ID and an event number. A unique event number should be assigned when the causes and corrective actions are not the same. In our example, the cause of Event 202 is hardware failure and the cause of Event 201 is a security breach. Each of these events has a different recommended action to fix the problem. Therefore, each event has a unique event number.
- The DSM template facility, which formats the event messages for display, uses a key comprised of the subsystem ID and event number. When it is desirable to format the fixed and variable portions of the event buffer differently, unique subsystem IDs are required.
- Unique event numbers enable you to write programs to watch for certain events and automatically react to them. You can use the same subsystem ID and event number for events that differ only in their variable data values.

Six of the twelve messages in our example have identical fixed text and variable data fields (underlined text). Shown below are six unique event numbers assigned to those six messages:

EVENT 1 ATM-EVT-UP

ATM SFMAIN01 is up at 245 A St. San Francisco.

ATM SFMAIN02 is up at 245 A St. San Francisco.

EVENT 2 ATM-EVT-ACCT-INSUF-FUNDS

Insufficient funds in account 23457320. Access denied on ATM OAKWEST1 at 245 Oak St. Oakland.

Insufficient funds in account 23409344. Access denied on ATM SACTO99 at 341 Main St. Sacramento.

EVENT 100 ATM-EVT-LOW-ON-CASH

ATM SFMAIN02 at 245 A St. San Francisco is low on funds.

ATM SFMAIN01 at 245 A St. San Francisco is low on funds.

EVENT 200 ATM-EVT-DOWN

ATM LACENT99 is down at 320 Main St. Los Angeles.

ATM LACENT91 is down at 320 Main St. Los Angeles.

EVENT 201 ATM-EVT-SECURITY-BREACH

Security breach on account 23457320. Number of accesses attempted 3; ATM SACTO02 is down at 230 State St. Sacramento.

Security breach on account 34094443. Number of accesses attempted 3; ATM SACTO01 is down at 230 State St. Sacramento.

EVENT 202 ATM-EVT-HW-FAILURE

Hardware failure on ATM LACENT99 at 125 8th Ave. Los Angeles. Component failed is cash dispenser; subcomponent 321561ac; serial number 231234093; sense status 0101110101.

Hardware failure on ATM LACENT98 at 125 8th Ave. Los Angeles. Component failed is cash dispenser; subcomponent 321561ac; serial number 343223480; sense status 1010111011.

## Step 7: Determine Event Subject

The subject of an event identifies the system component that is most directly involved in the event. The subject of an event can be a hardware component (for example, ATM or processor) or a software component (for example, a process, protocol layer, or file). Any component of the system can be the subject.

Viewpoint provides one specific screen designed to display event messages based on event subjects. Viewpoint can display the last "n" events database sorted by subject. This enables you to sort events by subject prior to taking corrective actions. For more information on Viewpoint, consult the *Viewpoint* manual.

The subject of all of the events in our example is the ATM name. The subject field names for our example are listed below.

| Event Number | Event Literal | Field Name of the Event Subject |
|---|---|---|
| Event 1 | ATM-EVT-UP | ATM-NAME |
| Event 2 | ATM-EVT-ACCT-INSUF-FUNDS | ATM-NAME |
| Event 100 | ATM-EVT-LOW-ON-CASH | ATM-NAME |
| Event 200 | ATM-EVT-DOWN | ATM-NAME |
| Event 201 | ATM-EVT-SECURITY-BREACH | ATM-NAME |
| Event 202 | ATM-EVT-HW-FAILURE | ATM-NAME |

In all of our events the ATM-NAME seemed to be the most important piece of information so it was chosen as the subject. However, you may choose any field as the subject. EMS FastStart has a limit of one subject per event, but any field can be defined as the subject. (For more information about the subjects of events see the *Event Management Service (EMS) Manual*.)

## Step 8: Define Event Type – Informative, Action-Attention, Action-Completion, or Critical

Each event must be one of the four types: Informative, Action-Attention, Action-Completion, or Critical. The four event types are described below. Note that the Action-Attention and Action-Completion event types are mutually dependent, so that neither

can be used without the other. An event is classified informative by default unless otherwise specified.

Section 6, Building Your Application for Event Generation, has more information on event types and lists sample code for each. Each programmer decides which type is appropriate for each message needed for an application.

### Informative

An informative event message informs you about any system or program status message you want, for example, the program started normally. This is the default event message type for FastStart.

### Action-Attention

An action-attention event message means operator intervention is needed, for example, a message to change a tape. This is the first half of a pair with an Action-Completion type message being the second half.

### Action-Completion

An action-completion event message informs you that the necessary action of an action-attention event message was completed and the job is continuing.

### Critical

A critical event message informs you that a critical event has occurred, for example, an error has occurred which requires immediate attention.

You determine the event type by deciding the level of importance of each event, for example:

| Event Number | Event Literal | Topic | Event Type |
|---|---|---|---|
| Event 1 | ATM-EVT-UP | ATM is up | Informative |
| Event 2 | ATM-EVT-ACCT-INSUF-FUNDS | Insufficient funds in account | Informative |
| Event 100 | ATM-EVT-LOW-ON-CASH | ATM is low on cash | Action |
| Event 200 | ATM-EVT-DOWN | ATM is down | Critical |
| Event 201 | ATM-EVT-SECURITY-BREACH | ATM security breach | Critical |
| Event 202 | ATM-EVT-HW-FAILURE | ATM hardware failure | Critical |

## Step 9: Build Template File

Based on the previous steps, you can now build your event templates. During this step, you will use the syntax defined in the *DSM Template Services Manual* to create a template source file for you application. In this file, you need to describe your subsystem, each event with their fixed text and the variable data fields (tokens) which will contain the variables data for each event.

For more information on how to build this file, see the *DSM Template Services Manual*.

In order to create and then compile your application template source file, you need to know the name of the corresponding token that matches each of the application fields you define in the ACF. These tokens are created by EMS FastStart during the generation process and are defined in a DDL source file (for example, ATM1DDLS). The information about these fields is also be stored in the DDL dictionary.

The name of each of these tokens is built according to these rules:

- Each field name you define in the ACF will have two prefixes added to them. The TAL/TACL syntax is used to describe the token names here since this is the same syntax used by the template compiler language.
- The first prefix added is "-TKN-".
- The second prefix added is the USER-VARIABLES-PREFIX parameter (for example, ATM). See Section 4, Preparing the Application Configuration File, for a description of how to define the USER-VARIABLE-PREFIX.

The following examples show the token name created by the ATM example used throughout this manual. Note that the token field name syntax is shown as TACL compatible since this syntax will be used by the filter compiler. In this example, the USER-VARIABLES-PREFIX is ATM.

| ACF field name | Token field name |
|---|---|
| ATM-NAME | ATM-TKN-ATM-NAME |
| ATM-LOCATION | ATM-TKN-ATM-NAME-LOCATION |
| ATM-ACCOUNT-NUMBER | ATM-TKN-ATM-NAME-ACCOUNT-NUMBER |
| ATM-RETRY-LIMIT | ATM-TKN-ATM-RETRY-LIMIT |
| ATM-HW-COMPONENT | ATM-TKN-ATM-HW-COMPONENT |
| ATM-HW-SUBCOMPONENT | ATM-TKN-ATM-HW-SUBCOMPONENT |
| ATM-SERIAL-NUMBER | ATM-TKN-ATM-SERIAL-NUMBER |
| ATM-SENSE-STATUS | ATM-TKN-ATM-SENSE-STATUS |

Please refer to Appendix D, ATM Example, EVENT DEFINITION SOURCE FILE: SATMDDL on page D-1. This is a template source file corresponding to this example in this section.

---

**Note.** All of the steps you have completed so far apply to both EMS and EMS FastStart. Beginning with Step 10, Assign a Group Field Number to Each Field, the information applies only to EMS FastStart.

---

## Step 10: Assign a Group Field Number to Each Field

Assign a group field number to each field (for example, "1" in FIELD-1-NUMBER). You can assign any number from 1 through 999. When you assign group field numbers, you must start with "1" and continue assigning consecutive numbers (for example, 2, 3, 4...). If you try assigning group field numbers in any other way, EMS FastStart does

not work. (The group field number is an internal value used by the program to keep track of the number of fields.)

Note that the group field number is distinct from the field number (for example, 1 in FIELD-1-NUMBER) and the event number.

When you follow these steps in designing your event messages, you create the subsystem portion of the ACF. The ACF is used as input for EMS FastStart (see Section 4, Preparing the Application Configuration File and Section 6, Building Your Application for Event Generation for detailed information about the ACF).

Below are examples of group field numbers (**bold type)**.

| **Fields** | **Field Definitions** |
|---|---|
| FIELD-**1**-NAME | ATM-NAME |
| FIELD-**1**-TYPE | ZSPI-DDL-CHAR8 |
| FIELD-**1**-NUMBER | **100** |
| FIELD-**2**-NAME | ATM-LOCATION |
| FIELD-**2**-TYPE | ZSPI-DDL-CHAR24 |
| FIELD-**2**-NUMBER | **200** |
| FIELD-**3**-NAME | ATM-ACCOUNT-NUMBER |
| FIELD-**3**-TYPE | ZSPI-DDL-INT2 |
| FIELD-**3**-NUMBER | **300** |
| FIELD-**4**-NAME | ATM-RETRY-LIMIT |
| FIELD-**4**-TYPE | ZSPI-DDL-INT |
| FIELD-**4**-NUMBER | **400** |
| FIELD-**5**-NAME | ATM-HW-COMPONENT |
| FIELD-**5**-TYPE | ZSPI-DDL-CHAR24 |
| FIELD-**5**-NUMBER | **500** |
| FIELD-**6**-NAME | ATM-HW-SUBCOMPONENT |
| FIELD-**6**-TYPE | ZSPI-DDL-CHAR24 |
| FIELD-**6**-NUMBER | **600** |
| FIELD-**7**-NAME | ATM-SERIAL-NUMBER |
| FIELD-**7**-TYPE | ZSPI-DDL-INT2 |
| FIELD-**7**-NUMBER | **700** |
| FIELD-**8**-NAME | ATM-SENSE-STATUS |
| FIELD-**8**-TYPE | ZSPI-DDL-INT2 |
| FIELD-**8**-NUMBER | **800** |

# 3

# Installing and Configuring EMS FastStart

This section has three parts. The first part describes the EMS FastStart Installation procedure, including Installing EMS FastStart on the Installation Subvolume and Installing EMS FastStart on a Working Subvolume, and a list of Post-installation Files that should be on your system after successful installation of EMS FastStart.

Next, the EMS FastStart Configuration is described, including code for The Default Compiler Configuration File (CCF) and a description of the key words and their parameters used in Modifying the CCF for a specific environment.

In the third part of this section, Security Considerations are addressed with specific information about Compiler Access and READ Access to Files.

## EMS FastStart Installation

EMS FastStart is installed at your site using the INSTALL program documented in the *System Generation Manual*. This is normally done by the system manager.

It is not necessary to perform a SYSGEN to install EMS FastStart since EMS FastStart is a stand-alone product that does not bind any files in the operating system.

### Installing EMS FastStart on the Installation Subvolume

Install EMS FastStart software files from the Site Update Tape (SUT). The INSTALL program restores the EMS FastStart files from the SUT onto the distribution subvolume (DSV) and then copies them to the Installation Subvolume (ISV) during the installation process. The subvolume name on the installation subvolume (ISV) is ZEMSFS.

### Installing EMS FastStart on a Working Subvolume

Copy all the EMS FastStart files from the ISV (ZEMSFS) to the working subvolume you choose. You can use the subvolume name ZEMSFS again or another subvolume name if you prefer. Do not run EMS FastStart from the original ISV subvolume; make a working copy.

△ **Caution.** We strongly recommend that you do not run EMS FastStart from the original ISV subvolume; make a working copy. When you install software from SUT tapes, the INSTALL program overwrites all files on the ISV subvolume.

## Post-installation Files

After the installation is complete, the following files will be in the EMS FastStart installation subvolume (ISV) and in your working subvolume:

| | |
|---|---|
| CCF | Compiler Configuration File, used to modify the run time environment of EMS FastStart. |
| CEGNDECS | C Library Header for EGEN Procedure (Needed by C programs). |
| EGENDECS | EGEN TAL Procedures Declaration File. |
| EMSFS | TACL macro file used to attach the segment file. |
| EMSFSC20 | EMS FastStart Segment File for release C20. |
| EXTRADDL | DDL Schema Source File used for user-defined variable data field types. |
| SATMACF | ATM Example, ACF source code. |
| SATMC | ATM Example, C Source code. |
| SATMCOB | ATM Example, COBOL85 source code. |
| SATMDDL | ATM Example, DDL event definitions |
| SATMDOC | ATM Example, Programming and Compilation Notes. |
| SATMTAL | ATM Example, TAL source code. |
| SATMTMPL | ATM Example, EMF source template. |
| SOFTDOC | Programmer notes supplied in the release tape. |

# EMS FastStart Configuration

EMS FastStart lets you adapt the code generation process to your environment by using a configuration file called CCF (Compiler Configuration File). The CCF should be modified by your system manager if your installation has different configurations than the default values specified in the CCF (for example, your COBOL85 compiler is not on $SYSTEM.SYSTEM). The default CCF is shown in Example 3-1.

---

**Note.** Be aware that EMS FastStart may not work properly if you use a batch product to invoke the TAL, COBOL85, and DDL compilers at your installation. EMS FastStart uses the completion codes returned by these compilers to determine if the compilation was successful. Since some batch products do not return completion codes, EMS FastStart can wait indefinitely. To correct this problem, specify the exact location of the Tandem compilers in the CCF.

---

# The Default Compiler Configuration File (CCF)

The Compiler Configuration File (CCF) is used to modify the EMS FastStart generation compiler attributes to reflect a specific installation. The CCF contains key words which have parameters you can modify for each installation.

Example 3-1 shows the contents of the default CCF for EMS FastStart.

**Example 3-1. Default CCF in EMS FastStart**

```
------------------------------------------------------------------
--
-- EMS FastStart - T9263C20 - (17MAR91)
--
-- File Type:         CCF
--
-- File Version:      A00
--
-- Creation Date:     September 17, 1989 11:01:43
--
-- Source File Name:  CCF
--
-- Description:This is the CCF file distributed with EMS FastStart
--
-- Modifications Summary:                    Date of Modification
--
-- This is the first release of this file      16 September,  1989
--
------------------------------------------------------------------


CCF-VERSION               A00
COBOL85-LOCATION          $SYSTEM.SYSTEM.COBOL85
COBOL85-CPU               0
COBOL85-PRIORITY          100
COBOL85-WORK-VOLUME       $SYSTEM
DDL-LOCATION              $SYSTEM.SYSTEM.DDL
DDL-CPU                   1
DDL-PRIORITY              100
DDL-WORK-VOLUME           $SYSTEM
EMF-CPU                   0
EMF-PRIORITY              100
EMF-WORK-VOLUME           $SYSTEM
TAL-LOCATION              $SYSTEM.SYSTEM.TAL
TAL-CPU                   1
TAL-PRIORITY              100
TAL-WORK-VOLUME           $SYSTEM
SPOOLER-COLLECTOR         $S
```

# Modifying the CCF

The CCF contains key words which are reserved by EMS FastStart and cannot be modified. However, each key word has a parameter which you can modify to customize the CCF for a specific installation. The list below and on the following page shows the EMS FastStart CCF key words and their parameters.

| Key Word | Parameter |
|---|---|
| CCF-VERSION | *ccf-version*<br>The parameter specifies the version of the CCF; the only version currently supported is A00.<br>Note: A00 is the internal version number of the CCF and is not related to the GUARDIAN 90 version number. |
| COBOL85-LOCATION | *cobol85-location*<br>The parameter specifies the location of the COBOL85 compiler (for example, $SYSTEM.SYSTEM.COBOL85). If your COBOL85 compiler is located somewhere other than $SYSTEM.SYSTEM, make the appropriate changes. You must specify the exact location of the Tandem compilers.<br>Note: If you do not have the COBOL85 compiler you must specify NOT-USED as a parameter to the COBOL85-LOCATION key word. Even if you do not use the COBOL85 compiler you must enter valid parameter values for the COBOL85 key words (COBOL85-CPU, COBOL85-PRIORITY and COBOL85-WORK-VOLUME). |
| COBOL85-CPU | *cobol85-cpu*<br>The parameter specifies the CPU in which the COBOL85 compiler will execute (for example, 0). Valid values for this parameter are 0 through 15. |
| COBOL85-PRIORITY | *cobol85-priority*<br>The parameter specifies the priority at which the COBOL85 compiler will execute (for example, 100). Valid values for this parameter are 1 through 199. |
| COBOL85-WORK-VOLUME | *cobol85-work-volume*<br>The parameter specifies the volume which will be used by the COBOL85 compiler to create its temporary files during the compilation process (for example, $SWAP). |
| DDL-LOCATION | *ddl-location*<br>The parameter specifies the location of the DDL compiler (for example, $SYSTEM.SYSTEM.DDL). |
| DDL-CPU | *ddl-cpu*<br>The parameter specifies the CPU in which the DDL compiler will execute (for example, 0). Valid values for this parameter are 0 through 15. |

| Key Word | Parameter |
|---|---|
| DDL-PRIORITY | *ddl-priority*<br>The parameter specifies the priority at which the DDL compiler will execute (for example, 100). Valid values for this parameter are 1 through 199. |
| DDL-WORK-VOLUME | *ddl-work-volume*<br>The parameter specifies the volume which will be used by the DDL compiler to create its temporary files during the compilation process (for example, $SWAP). |
| EMF-CPU | *emf-cpu*<br>The parameter specifies the CPU in which the EMF compiler will execute (for example, 0). Valid values for this parameter are 0 through 15. |
| EMF-PRIORITY | *emf-priority*<br>The parameter specifies the priority at which the EMF compiler will execute (for example, 100). Valid values for this parameter are 1 through 199. |
| EMF-WORK-VOLUME | *emf-work-volume*<br>The parameter specifies the volume which will be used by the EMF compiler to create its temporary files during the compilation process (for example, $SWAP). |
| TAL-LOCATION | *tal-location*<br>The parameter specifies the location of the TAL compiler (for example, $SYSTEM.SYSTEM.TAL). |
| TAL-CPU | *tal-cpu*<br>The parameter specifies in which CPU the TAL compiler will execute (for example, 0). Valid values for this parameter are 0 through 15. |
| TAL-PRIORITY | *tal-priority*<br>The parameter specifies in which CPU the TAL compiler will execute (for example, 0). Valid values for this parameter are 1 through 199. |
| TAL-WORK-VOLUME | *tal-work-volume*<br>The parameter specifies the volume which will be used by the TAL compiler to create its temporary files during the compilation process (for example, $SWAP). |
| SPOOLER-COLLECTOR | *spooler-collector*<br>The parameter specifies the name of the spooler collector on your system. Please note that EMS FastStart only verifies if the process specified exists and does not verify if this is really a collector process (for example, $S). |

After the CCF is modified to reflect your specific environment, the system manager can run EMS FastStart with the default ACF (ATMACF) supplied to test the installation and configuration (also see Section 5, Running EMS FastStart, and Section 7, Testing Program and Filter, for more information).

# Security Considerations

There are two security considerations: compiler access and READ access to files. The system manager may be involved in securing these files for EMS FastStart users.

## Compiler Access

Users of EMSFS need the correct security to access and execute the following compilers: DDL, TAL, EMF, C, and COBOL85 (if COBOL85 is on your system).

## READ Access to Files

To run EMS FastStart properly, you need READ access to the following files in the subvolume: EMSFS, EMSFSC20, CCF, EXTRADDL and the ACF you specify. The locations of these files are specified in the ACF with the EMSFS-SUBVOL key word.

You also need READ access to the following ZSPIDEF files: ZEMSDDL, ZSPIDDL, ZSPITACL, ZSPITAL, ZEMSTACL and ZEMSTAL. The locations of these files are specified in the ACF with the ZSPIDEF-SUBVOL. (See Section 4, Preparing the Application Configuration File, for more information.)

# 4

# Preparing the Application Configuration File

Before you use EMS FastStart, you must modify the Application Configuration File (ACF) parameters to generate the customized EGEN procedure for your application. This section outlines the procedure for Modifying the ACF. In Application Configuration File the features and restrictions of the ACF are described. Next, a listing of the Default Application Configuration File is shown followed by the key words and parameters used to modify the ACF. A subsection about Field Definitions is included to help explain the field key words and parameters. The end of this section covers adding data types to the ACF with Adding Data Types with EXTRADDL and EGEN Default Values.

This section assumes that you have already prepared your application by designating the owner of the events, number and version (SSID), and the event message details (for example, name, number, type, subject, etc.).

## Application Configuration File

The Application Configuration File (ACF) is the source file for the EMS FastStart generation process which produces the customized EGEN procedure. The ACF contains parameters which you can modify to use with your application.

The EMS FastStart generation process starts by validating each ACF file parameter entered. If an error is detected, an informative message is displayed and the generation process stops. The message informs you of the probable cause and recommended action. Therefore, it is important to specify the ACF parameters carefully. (See Section 5, Running EMS FastStart, and Appendix A, EMS FastStart Messages, for specific information.)

The ACF has the following features:

- The ACF supports a comment character "--" (two hyphens) which may be placed in any column.
- A version field is provided to support future extensions.
- USER-SUBVOL-FILES-PREFIX is a key word used to prefix all the files in the user subvolume. It enables you to have multiple versions of the EGEN module in the same subvolume.
- Any field defined in the ACF can become the subject (see Section 6, Building Your Application for Event Generation for information on Multiple Subject Data Type Support).

The ACF has the following restrictions:

- The variables used by EGEN must be unique.

- The FIELD-X-TYPE parameter must be a Subsystem Programmatic Interface (SPI) definition supported by EMS FastStart (for example, ZSPI-DDL-CHAR8).

If you are creating an application, you can design your application's events to take advantage of EMS FastStart's features. If you are converting an existing application, you can search your application's code for event messages and organize them for EMS FastStart. See Section 2, Brief Review of Event Design, for information and examples of event message design.

# Default Application Configuration File

The ACF contains key words with parameters which can be modified for each installation. You can modify the default ACF for your application or replace it with your own ACF. There are 27 possible data types; 8 are shown in the default ACF below. A complete list of the data types supported can be found under Section 4, Preparing the Application Configuration File. Example 4-1 shows the contents of the default ACF for EMS FastStart (ATMACF) which you see after the program is installed.

**Example 4-1. Default Application Configuration File** (page 1 of 2)

```
--
-- EMS FastStart - T9263C20 - (17MAR91)
--
-- File Type:         ACF
--
-- File Version:      B00
--
-- Creation Date:     September 17, 1989 10:54:31
--
-- Source File Name:  SATMACF
--
-- Description:       This ACF implements the ATM example.
--
-- Modifications Summary:                   Date of Modification
--
-- This is the first release of this file     16 September,  1989
--
-- Modified for the T9263C20 release          21 November,   1990
--
-- 1- Added the following key words: SAVE-DDL-DICTIONARY
--                                   USER-DDL-FILE
--
-- 2- Changed the version from A00 to B00.
--
-----------------------------------------------------------------
--
--       ACF-VERSION                B00
--
       EMSFS-SUBVOL               $data.zemsfs
       ZSPIDEF-SUBVOL             $dsv.zspidef
       USER-SUBVOL                $data.atm
       USER-SUBVOL-FILES-PREFIX   ATM1
       SAVE-DDL-DICTIONARY        YES
       USER-DDL-FILE              SATMDDL
       USER-VARIABLES-PREFIX      ATM
--
       APPLICATION-SSID-OWNER     CUSTOMER
       APPLICATION-SSID-NUMBER    1
       APPLICATION-SSID-VERSION   J00
--
       EVENT-TEXT-TYPE            ZSPI-DDL-CHAR254
```

**Example 4-1. Default Application Configuration File** (page 2 of 2)

```
--
-- Start of field definitions for application ATM !
--
       FIELD-1-NAME               atm-name
       FIELD-1-TYPE               ZSPI-DDL-CHAR8
       FIELD-1-NUMBER             100
       FIELD-2-NAME               atm-location
       FIELD-2-TYPE               ZSPI-DDL-CHAR24
       FIELD-2-NUMBER             200
       FIELD-3-NAME               atm-account-num
       FIELD-3-TYPE               ZSPI-DDL-INT2
       FIELD-3-NUMBER             300
       FIELD-4-NAME               atm-retry-limit
       FIELD-4-TYPE               ZSPI-DDL-INT
       FIELD-4-NUMBER             400
       FIELD-5-NAME               atm-hw-component
       FIELD-5-TYPE               ZSPI-DDL-CHAR24
       FIELD-5-NUMBER             500
       FIELD-6-NAME               atm-hw-subcomponent
       FIELD-6-TYPE               ZSPI-DDL-CHAR24
       FIELD-6-NUMBER             600
       FIELD-7-NAME               atm-serial-number
       FIELD-7-TYPE               ZSPI-DDL-INT2
       FIELD-7-NUMBER             700
       FIELD-8-NAME               atm-sense-status
       FIELD-8-TYPE               ZSPI-DDL-INT2
       FIELD-8-NUMBER             800
```

# Modifying the ACF

The ACF contains key words reserved by EMS FastStart which you cannot modify. However, each key word has a parameter which you can modify in order to customize the ACF for a specific installation. Following is a list of the ACF key words and their parameters.

| Key Word | Parameter |
|---|---|
| ACF-VERSION | *acf-version*<br>The parameter specifies the version of the ACF; the only version currently supported is B00. This is the internal version number of the ACF and should not be altered. |
| EMSFS-SUBVOL | *emsfs-subvol*<br>The parameter specifies the location of the working EMSFS subvolume. EMS FastStart expects to find the following files in this subvolume: CCF and EXTRADDL. Note that EMS FastStart users must have READ access for these two files. |
| ZSPIDEF-SUBVOL | *zspidef-subvol*<br>The parameter specifies the subvolume which contains the ZSPIDDL, ZEMSDDL, ZSPITAL, ZEMSTAL, ZSPITACL and ZEMSTACL definition files used by the different compilers during the generation phases. Note that EMS FastStart users must have READ access to all these files. |

| Key Word | Parameter |
|---|---|
| USER-SUBVOL | *user-subvol*<br><br>The parameter specifies the location where EMS FastStart will create all the source and object files. EMS FastStart creates files on behalf of the user. Therefore, the user must have CREATE access on the designated USER-SUBVOL. |
| USER-SUBVOL-FILES-PREFIX | *user-subvol-file-prefix*<br><br>The parameter specifies the prefix that EMS FastStart will use to build the filenames of each file in the subvolume specified by USER-SUBVOL. For example, APP1 will be used to create the following files: APP1DDLS, APP1EGES, APP1INDX, etc. |
| SAVE-DDL--DICTIONARY | *user-subvol-file-prefix*<br><br>The parameter specifies if the DDL dictionary created in the subvolume specified by the USER-SUBVOL parameter will be saved. Valid parameter values are YES (save the dictionary) and NO (do not save the dictionary; purge it). This dictionary will contain all the definitions for ZSPIDDL and ZEMSDDL plus your own subsystem definitions (tokens and events). This dictionary is required if you want to use the DSM Template Services and create templates for your application. |
| USER-DDL--FILE | *user-subvol-file-prefix*<br><br>The parameter specifies the name of a user defined DDL source file. Valid parameter values are *file-name* (a valid DDL source file) and NOT-USED (do not source a file). The content of this file will be sourced at the end of the MAIN DDL file and added to the dictionary. You may use this file to source your event definitions, which will then be used later by the DSM Template Services for generating your application template file. |
| USER-VARIABLES-PREFIX | *user-variables-prefix*<br><br>The parameter is used to make EGEN variable names unique. The name cannot be longer than four characters, must start with a letter and must contain only alphanumeric characters. For example: A123, Z111, ZZZZ, ABCD, etc. The token names created by EMS FastStart will also use this prefix. For example: PROGRAM-NAME will have an associated token called ATM-TKN-PROGRAM-NAME (If the USER-VARIABLES-PREFIX is ATM). |

| Key Word | Parameter |
|---|---|
| APPLICATION-SSID-OWNER | *application-ssid-owner*<br><br>The parameter defines the owner of the event buffer and must be an eight-character string or less to identify your company or organization.<br><br>The owner name must start with an alpha character and contain only alphanumeric characters and hyphens. Blanks are allowed only at the end for padding. The alpha characters are case sensitive (for example, “COMPANY” and “Company” are recognized as different names). To avoid confusion, it is recommended that you define your owner ID with all alpha characters in uppercase, as is done for the Tandem owner ID “TANDEM_ _” (where the underscores represent blanks). |
| APPLICATION-SSID-NUMBER | *application-ssid-number*<br><br>This field defines the SSID subsystem number and is identified by a 16-bit signed integer value that identifies the subsystem within the set of subsystems provided by the subsystem owner. |
| APPLICATION-SSID-VERSION | *application-ssid-version*<br><br>The parameter represents the software release version of the subsystem and is a 16-bit unsigned integer value. In the format of the field the left byte contains the letter part of the version as an ASCII uppercase alpha character, and the right byte contains the numeric part of the version as an unsigned integer value. |
| EVENT-TEXT-TYPE | *event-text-type*<br><br>The parameter controls the length of the displayable text describing the event. The values supported by EMS FastStart are ZSPI-DDL-CHAR128 and ZSPI-DDL-CHAR254. |
| FIELD-X-NAME | *field-x-name*<br><br>The parameter specifies the name of each field in the EGEN-RECORD and has a maximum 20 character string. The name must start with an alpha character and contain only alpha and numeric characters and hyphens. Blanks are allowed only at the end for padding. Replace the “X” for each different field with an integer value (a field includes NAME, TYPE, and NUMBER definitions). The value for “X” must be sequential and start with 1. (See Field Definitions below for more information.) |

| Key Word | Parameter |
|---|---|
| FIELD-X-TYPE | *field-x-type* |

The parameter specifies which variable data types are supported by EGEN. The following 27 variable data types are supported:

| | |
|---|---|
| ZSPI-DDL-INT | ZSPI-DDL-CHAR24 |
| ZSPI-DDL-INT2 | ZSPI-DDL-CHAR50 |
| ZSPI-DDL-INT4 | ZSPI-DDL-CHAR64 |
| ZSPI-DDL-UINT | ZSPI-DDL-CHAR128 |
| ZSPI-DDL-ENUM | ZSPI-DDL-CHAR254 * |
| ZSPI-DDL-BOOLEAN | ZSPI-DDL-USERID |
| ZSPI-DDL-CHAR | ZSPI-DDL-USERNAME |
| ZSPI-DDL-CHAR-PAIR | ZSPI-DDL-FNAME |
| ZSPI-DDL-CHAR4 | ZSPI-DDL-FNAME32 |
| ZSPI-DDL-CHAR6 | ZSPI-DDL-SUBVOL |
| ZSPI-DDL-CHAR8 | ZSPI-DDL-CRTPID |
| ZSPI-DDL-CHAR16 | ZSPI-DDL-DEVICE |
| ZSPI-DDL-TRANSID | ZSPI-DDL-TIMESTAMP |
| ZSPI-DDL-SSID | |

*ZSPI-DDL-CHAR254 is defined in EXTRADDL. (See Adding Data Types with EXTRADDL below.)

Replace the “X” for each different field with an integer value (a field includes NAME, TYPE, and NUMBER definitions). The value for “X” must be sequential and start with 1. (See Field Definitions below.)

| | |
|---|---|
| FIELD-X-NUMBER | *field-x-number* |

The parameter specifies the number of the variable data type that will be added to the event buffer by EGEN. A field number identifies each separate variable data field (token). The field number can be any number from 1 to 9990. You can assign any number from 1 through 9990 arbitrarily to any variable data field, although you may wish to assign them with some logic for your application.

Replace the “X” for each different field with an integer value (a field includes NAME, TYPE, and NUMBER definitions). The value for “X” must be sequential and start with 1. (See Field Definitions below.)

# Field Definitions

For each field defined in the ACF, you need to specify three things: the field name, the field type, and the field number. Each must share the same value for X, the group field number. For example:

| | |
|---|---|
| FIELD-x-NAME | *field-x-name* |
| FIELD-x-TYPE | *field-x-type* |
| FIELD-x-NUMBER | *field-x-number* |

The field name, type, and number are always defined as a group. The “x” represents the group field number.

The group field number “X” is replaced by an integer value for each different field. Valid values for “X” are 1 through 9990. Numbers from 9991 through 9998 are reserved by EMS FastStart. The first “X” value must be 1 and the values must continue sequentially (2, 3, 4, 5, etc.).

Fields 1 and 2 are defined as follows in the sample ACF:

| | |
|---|---|
| FIELD-1-NAME | *atm-name* |
| FIELD-1-TYPE | ZSPI-DDL-CHAR8 |
| FIELD-1-NUMBER | 100 |
| FIELD-2-NAME | *atm-location* |
| FIELD-2-TYPE | ZSPI-DDL-CHAR24 |
| FIELD-2-NUMBER | 200 |

For more information, see the *Distributed Systems Management (DSM) Programming Manual*.

# Adding Data Types with EXTRADDL

If you need a data type not found in the standard types supported by EMSFS, you can define another in the EXTRADDL file.

EMS FastStart provides a DDL source file to allow you to expand on the basic data types allowed. New data types are defined in the DDL source file, EXTRADDL, located in the EMS FastStart distribution subvolume. Only character fields can be defined in the EXTRADDL. EMS FastStart automatically sources in the EXTRADDL file during the main DDL compilation step. The name of the new data type you define should always be of the form ZSPI-DDL-CHAR*, where the asterisk (*) represents the number of characters you define. For example, ZSPI-DDL-CHAR20.

The EXTRADDL file contains a definition for the ZSPI-DDL-CHAR254 data type. You can add a new data type to the EXTRADDL file by adding a section that defines the new data type. The sample code in Example 4-2 shows the EXTRADDL file with added code for a new data type, CHAR20. To add your own data types, use this code as a guide.

## Example 4-2. Adding a New Data Type to the EXTRADDL File

```
*-------------------------------------------------------------------
*
*  EMS Fast Start - T9263020 - (17MAR91)
*
*  File Type:                     DDL Source Schema
*
*  Source File Name:              Extraddl
*
*  Generation Time:               July 7,  1988
*
*  Language Compiler Required:    Data Definition Language (DDL)
*
*  Compiler Version Required:     C20
*
*  Source Library File Produced:  None, see below.
*
*
*  File Description:  This DDL source schema file is an example of DDL
*  definitions which may be added to the base ZSPIDDL definitions
*  provided by Tandem.  These definitions can then be used by
*  EMS Fast Start and EGEN to create tokens of specific types.
*
*  Modifications Summary:    Date of Modification
*
*  1 - Added the Zspi-ddl-char254 token.  Used by     21 October 1988
*      EGEN to generate an event message with a
*      ZEMS-TKN-TEXT of up to 254 bytes.
*      N.B. 254 is the maximum bytes length for a
*      fixed token code.
*
*-------------------------------------------------------------------
?SECTION Zspi-ddl-char254
!
!  254 ASCII characters - addressable as a STRUCT, bytes, or INT's
!
?TALBOUND 0
DEFINITION  Zspi-ddl-char254.
   02 z-c                PIC X(254)          SPI-NULL " ".
   02 z-s  REDEFINES z-c.
      03   z-i           TYPE BINARY 16      OCCURS 127 TIMES.
   02 z-b  REDEFINES z-c PIC X               OCCURS 254 TIMES.
END
TOKEN-TYPE  Zspi-typ-char254      VALUE IS Zspi-tdt-char
                                  DEF   IS Zspi-ddl-char254.

!
!This section shows how to add a field type of 20 characters.
!
?SECTION Zspi-ddl-char20
!
! 20 ASCII characters - addressable as a STRUCT, bytes, or INT's
!
?TALBOUND 0
DEFINITION  Zspi-ddl-char20.
   02 z-c                PIC X(20)           SPI-NULL " ".
   02 z-s  REDEFINES z-c.
      03   z-i           TYPE BINARY 16      OCCURS 10 TIMES.
   02 z-b  REDEFINES z-c PIC X               OCCURS 20 TIMES.
END
TOKEN-TYPE  Zspi-typ-char20       VALUE IS Zspi-tdt-char
                                  DEF   IS Zspi-ddl-char20.
```

# EGEN Default Values

EGEN default values are used by the EGEN procedure to determine if information should be added to the event buffer. EGEN compares the value of each field defined in your ACF against a set of default values. Each field type supported by EGEN has a specific default value.

Before generating an event, an application always initializes all fields of the egen-record (the egen-record contains all the fields declared in the ACF) with default values. This is done with the initialize^egen^record procedure. The following values are entered into the field types:

| Field type | EGEN Default Values |
|---|---|
| ZSPI-DDL-INT | 32767 |
| ZSPI-DDL-INT2 | 2147483647 |
| ZSPI-DDL-INT4 | 9223372036854775807 |
| ZSPI-DDL-UINT | 65535 |
| ZSPI-DDL-ENUM | 32767 |
| ZSPI-DDL-TRANSID | 9223372036854775807 |
| ZSPI-DDL-TIMESTAMP | 9223372036854775807 |

These default values were assigned because they represent values which are unlikely to occur during the course of running an application. If the actual value passed to EGEN differs from the default value, that new value is passed to the event message buffer. If the actual value matches the default value, EGEN assumes nothing has changed, and no value is added to the event buffer. The objective is to supply information which does not match the default values. If you feel that one of the EGEN default values may in fact match a legitimate value passed by an event, change the default value for that field type. If you do not do this, variable data will be missing from your event message buffer.

## Example

Table 4-1 shows the contents of the following fields after a call to the initialize^egen^record procedure.

**Table 4-1. Field Content After Call to Initialize^egen^record Procedure** (page 1 of 2)

| FIELD-NAME | FIELD-TYPE | FIELD-VALUE |
|---|---|---|
| ATM-NAME | ZSPI-DDL-CHAR8 | 8 spaces |
| ATM-LOCATION | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-ACCOUNT-NUM | ZSPI-DDL-INT2 | 2147483647 |
| ATM-RETRY-LIMIT | ZSPI-DDL-INT | 32767 |
| ATM-HW-COMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |

**Table 4-1. Field Content After Call to Initialize^egen^record Procedure** (page 2 of 2)

| FIELD-NAME | FIELD-TYPE | FIELD-VALUE |
|---|---|---|
| ATM-HW-SUBCOMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-SERIAL-NUMBER | ZSPI-DDL-INT2 | 2147483647 |
| ATM-SENSE-STATUS | ZSPI-DDL-INT2 | 2147483647 |

**Note.** For character fields such as ZSPI-DDL-CHAR8 and ZSPI-DDL-CHAR24, the default value is 8 and 24 spaces respectively.

Once the fields are initialized, information needed for each specific event must be moved into the fields, replacing the default values. Table 4-2 shows the ATM-ACCT-INSUF-FUNDS event generated by the ATM COBOL85 example. This shows all the fields values after event-specific information was moved into three fields: ATM-NAME, ATM-LOCATION, and ATM-ACCOUNT-NUM. These fields are shown in boldface type below.

**Table 4-2. ATM COBOL85 Example**

| FIELD-NAME | FIELD-TYPE | FIELD-VALUE |
|---|---|---|
| **ATM-NAME** | **ZSPI-DDL-CHAR8** | **OAKWEST1** |
| **ATM-LOCATION** | **ZSPI-DDL-CHAR24** | **245 Oak St., Oakland** |
| **ATM-ACCOUNT-NUM** | **ZSPI-DDL-INT2** | **34503933** |
| ATM-RETRY-LIMIT | ZSPI-DDL-INT | 32767 |
| ATM-HW-COMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-HW-SUBCOMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-SERIAL-NUMBER | ZSPI-DDL-INT2 | 2147483647 |
| ATM-SENSE-STATUS | ZSPI-DDL-INT2 | 2147483647 |

The application now calls EGEN. EGEN looks at the contents of the field-value and compares it with the EGEN default values. Only those fields whose contents differ from the default values are added to the event buffer. In the example above, only those fields which are shown in **bold type** will be added to the event buffer. Note that if the field value is not modified, and the default value is used, nothing is added to the event buffer. Should this happen, you may find some variable data missing from the events you generate.

**Note.** The term “default” is used to mean a standard value to which other values are compared. The default value is one that is not expected so that when the user-supplied value does not equal the default, the user-supplied value is added to the event buffer.

To change the default field values, generate your application and modify the *xxxx*TAL (for example, ATM1TAL for our sample) DDL file (Example 4-3). Enter the new default values. Then re-compile the EGEN module with the new TAL DDL and re-bind this new

TAL module with your application. Remember, it is important to choose default values which are unlikely to occur in your application. Do not use a common or likely value as a default. In the example above, a “real-life” ATM-RETRY-LIMIT value is unlikely to match the default value of 32767; however, it is likely that a value of 4 might be passed to the field-value. Therefore, 4 would not be a wise choice as a default value for that field type.

**Example 4-3. EGEN Default Values**

```
!---------------------------------------------------------------------
!
! Constants used by Egen to define the default values of a field
!
LITERAL EMSFS^DEFAULT^INT = 32767;
LITERAL EMSFS^DEFAULT^INT2 = 2147483647D;
LITERAL EMSFS^DEFAULT^INT4 = 922337203685477580 7F;
LITERAL EMSFS^DEFAULT^UINT = %177777;
LITERAL EMSFS^DEFAULT^ENUM = 32767;
LITERAL EMSFS^DEFAULT^TRANSID = 9223372036854775807F;
LITERAL EMSFS^DEFAULT^TMESTAMP = 9223372036854775807F;
```

# 5 Running EMS FastStart

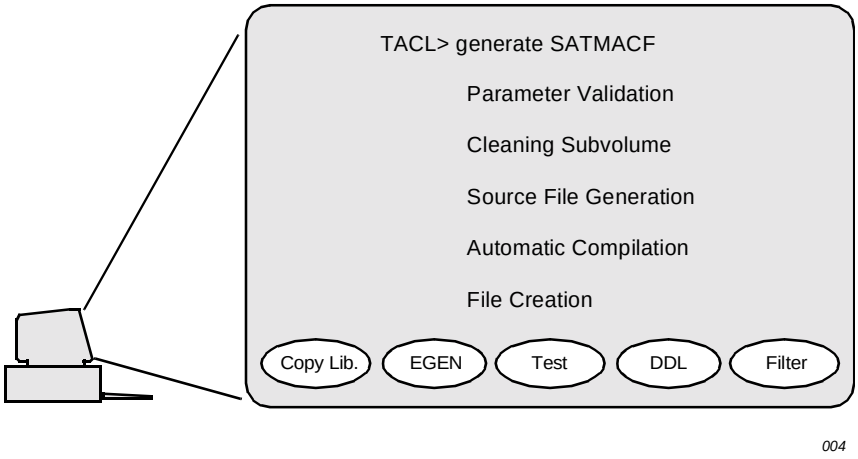
This section describes how to run EMS FastStart. It outlines the procedure for first Setting Up the EMS FastStart Environment and then Running EMSFS with the GENERATE command.

The message you receive after a successful completion of the setup as well as two possible error messages if the setup cannot be completed are shown and explained. Then, Stopping EMSFS and Detaching the Segment File after the run is completed is described. Next, a sample run is shown including the commands you must enter and the text you see on your terminal.

At the end of this section, the User Subvolume Files and EMSFS Components are listed and described. There is also information about the warning, advisory, and error messages you may receive when you run EMS FastStart, (listed and described in detail in Appendix A, EMS FastStart Messages).

Figure 5-1 illustrates the generation process: parameter validation, cleaning the subvolume, source file generation, automatic compilation, and file creation. The figure also shows the resulting EMS FastStart components: copy libraries, the EGEN routine, the test program, the DDL source code, and the filter.

**Figure 5-1. EMS FastStart Generation Process**



# Setting Up the EMS FastStart Environment

Before using EMS FastStart, you must attach the EMSFSC20 segment file to your TACL.

A TACL macro is provided to simplify this operation. Enter the following commands exactly as specified to be sure that your working environment is properly initialized.

At the TACL prompt enter:

```
1> VOLUME EMS-FastStart-working-subvolume
```

This command sets your default subvolume to the EMS FastStart development subvolume (for example, VOLUME $DEV.ZEMSFS).

At the TACL prompt enter:

```
2> RUN EMSFS
```

This command invokes the EMSFS TACL macro which attaches the EMSFSC20 segment file to your TACL. This macro also checks that the version of EMSFS matches the version of the segment file.

The following message appears on your terminal if the EMS FastStart setup was successfully completed.

```
EMS FastStart - T9263C20 - (17MAR91)
Copyright Tandem Computers Incorporated 1989,1990,1991

** Comment 1 ** EMS FastStart set up completed.

Enter the GENERATE <acf-filename> command to start an EMS FastStart
session.
```

One of the following two error messages will appear on your terminal if the EMS FastStart set up was unsuccessful.

## Error Message 1

```
EMS FastStart - T9263C20 - (17MAR91)
Copyright Tandem Computers Incorporated 1989,1990,1991

** Error 1 ** EMS FastStart set up failed.
Probable Cause:  The segment file version does not match the startup file
version.

     Segment file version = <segment file version>
     Startup file version = <startup file version>

Recommended Action:  Reinstall EMS FastStart to ensure that the segment
file and the startup file have the same version.
```

### Error Message 2

```
EMS FastStart - T9263C20 - (17MAR91)
Copyright Tandem Computers Incorporated 1989,1990,1991

** Error 2 ** Unable to locate the EMSFSC20 segment files.

Probable Cause:  Your current default subvolume (<user-defaults-subvol>)
does not contain EMSFSC20 segment files.

Recommended Action:  Please change your default subvolume to the location
of the EMS FastStart subvolume and reissue the RUN command by entering the
following commands:

VOLUME $<emsfs-volume>.<emsfs-subvol>
RUN EMSFS
```

# Running EMSFS

To start EMS FastStart, enter the GENERATE command at the TACL prompt.

```
TACL> GENERATE acf-filename
```

This command accepts one parameter: the filename of the ACF customized for your application (for example, GENERATE SATMACF). The parameter acf-filename specifies an edit file which contains the information specific to your application. If the file does not exist, EMS FastStart creates a default ACF that you must edit to describe your particular environment. (See Section 4, Preparing the Application Configuration File,, for information about the ACF.)

The GENERATE command starts the five automatic processes of EMS FastStart as given in Running EMS FastStart–ATM Example: parameter validation, cleaning the subvolume, source file generation, automatic compilation, and file creation. When complete, EMS FastStart will have created a group of files in the user subvolume and the TACL prompt will be returned. To stop the GENERATE process and EMS FastStart at any time, press the BREAK key.

## Stopping EMSFS and Detaching the Segment File

After running EMS FastStart, use the EMSFS:STOP command to free memory segment space within your TACL by detaching the EMSFSC20 segment file.

At the TACL prompt, enter the EMSFS:STOP command:

```
TACL> EMSFS:STOP
```

This message is displayed:

```
EMS FastStart - T9263C20 - (17MAR91)
Copyright Tandem Computers Incorporated 1989,1990,1991

** Comment 2 ** Environment stopped.
```

# Running EMS FastStart–ATM Example

This section shows you how to generate a sample run of EMS FastStart using the default ACF called SATMACF.

Following the command entries, a listing of what you will see on your screen is shown. In order to clarify the process, the listing is divided into steps which are grouped as follows:

Parameter Validation (Steps 1-2)

Cleaning the Subvolume (Step 3)

Source File Generation (Steps 4 - 8)

Automatic Compilation (Steps 9 - 13)

File Creation: ATM1TEST and ATM1INDX (Steps 14 - 16)

Note that the EMS FastStart run will stop if an error occurs. You will receive an error message advising you of the probable cause and the recommended action (listed and described in detail in Appendix A, EMS FastStart Messages. Correct the error and run the program again.

Here is a sample EMS FastStart run:

```
TACL> volume $data.zemsfs
TACL> run emsfs
EMS FastStart - T9263C20 - (17MAR91)
Copyright Tandem Computers Incorporated 1989,1990,1991

** Comment 1 ** EMS FastStart set up completed.
```

Enter the GENERATE acf-filename command to start an EMS FastStart session.

```
TACL> generate satmacf
EMS FastStart - T9263C20 - (17MAR91), Operating System C20, System name \MTL
October 15, 1989 22:24:28
```

The EMS FastStart response following the RUN and GENERATE commands is shown below as it will appear on your screen.

# Parameter Validation (Steps 1-2)

Steps 1 and 2 show the validation of the ACF and CCF parameters.

```
STEP #1: Reading and validating the ACF: $DATA.ZEMSFS.SATMACF.

------------------------------------------------------------------------
--
-- EMS FastStart - T9263C20 - (17MAR91)
--
-- File Type:         ACF
--
-- File Version:      B00
--
-- Creation Date:     September 17, 1989 10:54:31
```

```
--
-- Source File Name:  SATMACF
--
-- Description:       This ACF implements the ATM example.
--
-- Modifications Summary:                          Date of Modification
--
-- This is the first release of this file          16 September,  1989
--
-- Modified for the T9263C20 release               21 November,   1990
--
-- 1- Added the following key words: SAVE-DDL-DICTIONARY
--                                   USER-DDL-FILE
--
-- 2- Changed the version from A00 to B00.
--
-------------------------------------------------------------------------
--
--
ACF-VERSION                 B00
--
EMSFS-SUBVOL                $data.zemsfs
ZSPIDEF-SUBVOL              $dsv.zspidef
USER-SUBVOL                 $data.atm
USER-SUBVOL-FILES-PREFIX    ATM1
SAVE-DDL-DICTIONARY         YES
USER-DDL-FILE               $DATA.ATM.SATMDDL

USER-VARIABLES-PREFIX       ATM
--
APPLICATION-SSID-OWNER      CUSTOMER
APPLICATION-SSID-NUMBER     1
APPLICATION-SSID-VERSION    J00
--
EVENT-TEXT-TYPE             ZSPI-DDL-CHAR254
--
-- Start of Field definitions for application ATM
--
FIELD-1-NAME                atm-name
FIELD-1-TYPE                ZSPI-DDL-CHAR8
FIELD-1-NUMBER              100
FIELD-2-NAME                atm-location
FIELD-2-TYPE                ZSPI-DDL-CHAR24
FIELD-2-NUMBER              200
FIELD-3-NAME                atm-account-num
FIELD-3-TYPE                ZSPI-DDL-INT2
FIELD-3-NUMBER              300
FIELD-4-NAME                atm-retry-limit
FIELD-4-TYPE                ZSPI-DDL-INT
FIELD-4-NUMBER              400
FIELD-5-NAME                atm-hw-component
FIELD-5-TYPE                ZSPI-DDL-CHAR24
FIELD-5-NUMBER              500
FIELD-6-NAME                atm-hw-subcomponent
FIELD-6-TYPE                ZSPI-DDL-CHAR24
FIELD-6-NUMBER              600
FIELD-7-NAME                atm-serial-number
FIELD-7-TYPE                ZSPI-DDL-INT2
FIELD-7-NUMBER              700
FIELD-8-NAME                atm-sense-status
FIELD-8-TYPE                ZSPI-DDL-INT2
FIELD-8-NUMBER              800
October 15, 1989 22:24:52
```

```
STEP #2: Reading and validating the CCF: $DATA.ZEMSFS.CCF.
------------------------------------------------------------------------
--
-- EMS FastStart - T9263C20 - (17MAR91)
--
-- File Type:         CCF
--
-- File Version:      A00
--
-- Creation Date:     September 17, 1989 11:01:43
--
-- Source File Name:  CCF
--
-- Description:       This is the CCF file distributed with EMS FastStart
--
-- Modifications Summary:                          Date of Modification
--
-- This is the first release of this file           16 September,  1989
--
------------------------------------------------------------------------
CCF-VERSION               A00
COBOL85-LOCATION          $SYSTEM.SYSTEM.COBOL85
COBOL85-CPU               0
COBOL85-PRIORITY          100
COBOL85-WORK-VOLUME       $SYSTEM
DDL-LOCATION              $SYSTEM.SYSTEM.DDL
DDL-CPU                   1
DDL-PRIORITY              100
DDL-WORK-VOLUME           $SYSTEM
EMF-CPU                   0
EMF-PRIORITY              100
EMF-WORK-VOLUME           $SYSTEM
TAL-LOCATION              $SYSTEM.SYSTEM.TAL
TAL-CPU                   1
TAL-PRIORITY              100
TAL-WORK-VOLUME           $SYSTEM
SPOOLER-COLLECTOR         $S

October 15, 1989 22:25:00
```

## Cleaning the Subvolume (Step 3)

This step is included to ensure that old versions of EMS FastStart files do not exist on the subvolume. If files are found, they are purged; therefore it is important that your subvolume be empty of any files.

```
STEP #3: Cleaning the subvolume: $DATA.ATM.

EMS FastStart continuing, no files found in this subvolume

October 15, 1989 22:25:01
```

## Source File Generation (Steps 4 - 8)

All the source files required by the compilers will be created during the following steps.

```
STEP #4: Creating the MAIN DDL source file: $DATA.ATM.ATM1DDLS.

October 15, 1989 22:25:06

STEP #5: Creating the USER DDL source library file: $DATA.ATM.ATM1UDDL.
```

```
October 15, 1989 22:25:08

STEP #6: Creating the FILTER source file: $DATA.ATM.ATM1EMFS.

October 15, 1989 22:25:10

STEP #7: Creating the COBOL85 test program source file: $DATA.ATM.ATM1PROS.

October 15, 1989 22:25:17

STEP #8: Creating the TAL EGEN module source file: $DATA.ATM.ATM1EGES.

October 15, 1989 22:25:38
```

## Automatic Compilation (Steps 9 - 13)

Next, EMS FastStart automatically compiles the various files just created. Compilation of the main DDL source file is the longest step within the EMS FastStart generation process. Depending on your system type and the ACF configuration, the automatic compilation can take from 5 to 20 minutes or longer.

---

**Note.** DDL compilation warnings may be returned to you. These warnings are caused by DDL when it creates the COBOL85 copylib files. They are a result of an inconsistency between COBOL85 and SPI substructures. Some variable data fields like ZSPI-DDL-FNAME and ZSPI-DDL-FNAME32 as defined by SPI have substructures with variable data fields not supported by COBOL85. You can ignore these types of warnings since they will not affect your application or the EGEN module, but relate to the DDL compilation phase.

---

```
STEP #9: Starting the compilation of the MAIN DDL schema source file.

DDL compiler location:               $SYSTEM.SYSTEM.DDL.
DDL compiler execution cpu:          1.
DDL compiler execution priority:     100.
DDL compiler work volume:            $SYSTEM.

Source file:  $DATA.ATM.ATM1DDLS.
Listing file: $S.#EMSFS.ATM1DDLS.

October 15, 1989 22:32:46

STEP #10: Starting the compilation of the USER DDL schema source file.

DDL compiler location:               $SYSTEM.SYSTEM.DDL.
DDL compiler execution cpu:          1.
DDL compiler execution priority:     100.
DDL compiler work volume:            $SYSTEM.

Source file:  $DATA.ATM.ATM1UDDL.
Listing file: $S.#EMSFS.ATM1UDDL.

October 15, 1989 22:32:54

STEP #11: Starting the compilation of the EGEN module.

TAL compiler location:               $SYSTEM.SYSTEM.TAL.
TAL compiler execution cpu:          1.
TAL compiler execution priority:     100.
TAL compiler work volume:            $SYSTEM.

Source file:  $DATA.ATM.ATM1EGES.
```

```
Object file:  $DATA.ATM.ATM1EGEN.
Listing file: $S.#EMSFS.ATM1EGEN.

October 15, 1989 22:33:49

STEP #12: Starting the compilation of the COBOL85 test program.

COBOL85 compiler location:           $SYSTEM.SYSTEM.COBOL85.
COBOL85 compiler execution cpu:      0.
COBOL85 compiler execution priority: 100.
COBOL85 compiler work volume:        $SYSTEM.

Source file:  $DATA.ATM.ATM1PROS.
Object file:  $DATA.ATM.ATM1PROG.
Listing file: $S.#EMSFS.ATM1PROG.

October 15, 1989 22:34:26

STEP #13: Starting the compilation of the EMF Filter example.

EMF compiler execution cpu:          0.
EMF compiler execution priority:     100.
EMF compiler work volume:            $SYSTEM.

Source file:  $DATA.ATM.ATM1EMFS.
Object file:  $DATA.ATM.ATM1EMFO.
Listing file: $S.#EMSFS.ATM1EMFO.

October 15, 1989 22:35:06
```

# File Creation: ATM1TEST and ATM1INDX (Steps 14 - 16)

After these three steps, the EMS FastStart generation process is complete and the statistics for the run are shown.

```
STEP #14: Creating the COBOL85 program test file: $DATA.ATM.ATM1TEST

October 15, 1989 22:35:07

STEP #15: Creating the INDEX file: $DATA.ATM.ATM1INDX.

October 15, 1989 22:35:09

STEP #16: Duplicating the ACF file: $DATA.ZEMSFS.SATMACF.

File Utility Program - T9074C20 - (15FEB89)    System  \MTL
Copyright Tandem Computers Incorporated 1981, 1983, 1985, 1986, 1987, 1988
ALLOW 1 ERRORS
DUP $DATA.ZEMSFS.SATMACF, $DATA.ATM.ATM1ACF, PURGE, SAVEALL
FILES DUPLICATED:  1

EMS FastStart - T9263C20 - (17MAR91)

Number of generation errors = 0
Number of generation warnings = 0
Number of files created = 8
Number of files compiled = 5
Number of fields validated = 8
Generation cpu time = 00:00:59
Total elapsed time = 0:10:47

TACL>
```

The EMS FastStart generation process is now complete. Use the EMSFS:STOP command to free memory segment space within your TACL by detaching the EMSFSC10 or EMSFSC20 segment file. At the TACL prompt enter the STOP command:

```
TACL>  EMSFS:STOP
```

This message will be displayed:

```
EMS FastStart - T9263C20 - (17MAR91)
Copyright Tandem Computers Incorporated 1989,1990,1991

** Comment 2 ** Environment stopped.
```

The remainder of this section has information about EMSFS messages, the files that will now be on your user subvolume, and the EMSFS components.

# EMSFS Messages

Each step of the EMS FastStart generation process is validated and if an error or warning occurs, a message appears on the terminal. If it is an error message, the generation process stops to let you correct the problem. Each error and warning has a number associated with it and a probable cause and recommended action to help you identify the problem and correct it. The error messages and warnings are listed and described in Appendix A, EMS FastStart Messages.

# User Subvolume Files

After the successful completion of the generation process, the following files are located on the user subvolume (in this example, $DATA.ATM). The files are also listed in an index file (in this example, ATM1INDX) located on the user subvolume.

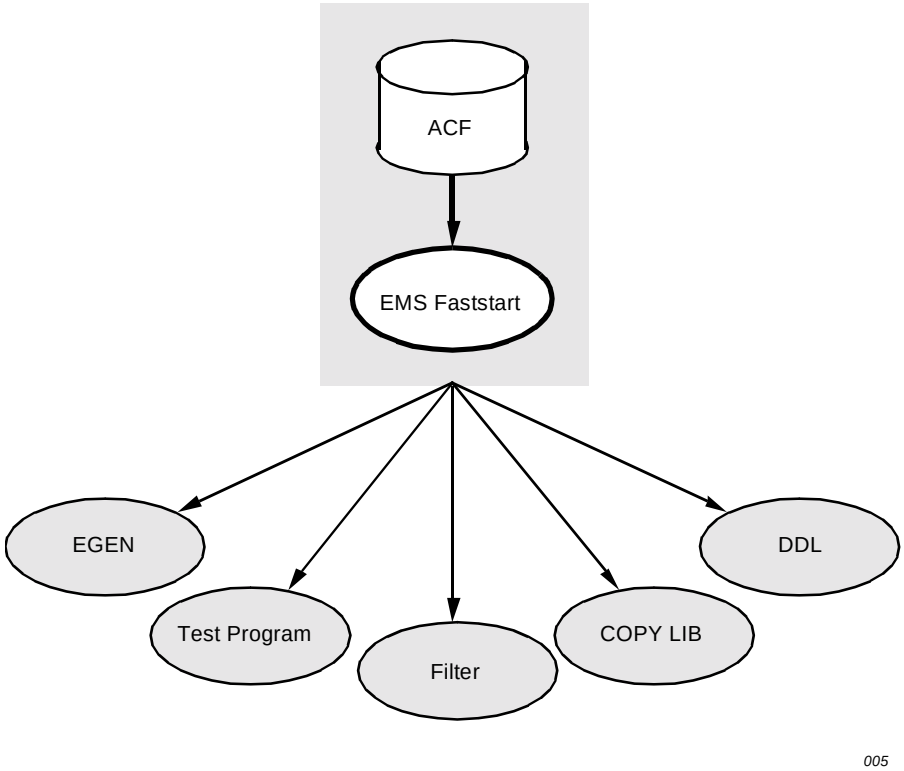
| | |
|---|---|
| ATM1ACF | ACF used to build this subvolume. |
| ATM1COB | COBOL85 definitions of the EGEN structure. |
| ATM1C | C definitions of the EGEN structure. |
| ATM1DDLS | Application DDL schema source file. |
| ATM1EGEN | TAL object which will be bound within a program. |
| ATM1EGES | TAL source code for EGEN. |
| ATM1EMFO | EMF filter object file. |
| ATM1EMFS | EMF filter source file example (compatible with Viewpoint). |
| ATM1INDX | Index file of the subvolume. |
| ATM1PROG | COBOL85 program to test the EGEN module. |
| ATM1PROS | COBOL85 source file code of ATM1PROG. |
| ATM1TACL | TACL definitions used by the EMF compiler. |

| | |
|---|---|
| ATM1TAL | TAL definitions used by EGEN. |
| ATM1TEST | TACL macro file to set up the defines for EGEN and start the ATM1PROG. |
| ATM1UCOB | User-defined event numbers in a COBOL85 copylib format. |
| ATM1UDDL | User-defined event numbers in a DDL schema source file. |

# EMSFS Components

The EMS FastStart generation and compilation process produces a TAL routine called EGEN, a test program, a filter, copy libraries, and DDL source files. Figure 5-2 illustrates the process and the resulting components.

**Figure 5-2. EMS FastStart Components**




A description of the components that result from the EMS FastStart generation and compilation process is shown below as well as the files related to each component.

## DDL

After the generation process, the user subvolume contains DDL source code that produces the copy libraries for COBOL85 and TAL, the two programming languages supported by EMS FastStart.

Files: ATM1DDLS

## Copy Libraries

Copy libraries for your application are produced by the compilation of the DDL source file.

Files: ATM1COB

ATM1TAL

ATM1TACL

ATM1UCOB

ATM1UDDL

ATM1C

## EGEN

EMS FastStart uses an Application Configuration File (ACF) to generate a TAL routine called EGEN (Event GENerator). EGEN is bound into the user application and generates events by performing a WRITE-READ of the event buffer to the collector $0, or to an alternate location you specify.

Files: ATM1EGEN

ATM1EGES

## Test Program

An interactive COBOL85 test program (ATM1PROG) lets you enter values for events based on your ACF. It calls EGEN to generate events and is compiled as part of EMSFS. This program lets you test the interface between your application and EGEN. When testing events, you can isolate testing data from production data by writing events to a location other than the event log.

Files: ATM1PROG

ATM1PROS

ATM1TEST

## Filter

A filter is created and compiled by EMS FastStart which can be run with a printing distributor or Viewpoint to test your application events. The interactive testing facility allows immediate testing of newly-defined events and filters through an easy-to-use conversational interface.

Files: ATM1EMFS

ATM1EMFO

## Index File

In addition to the components and their files listed above, an index file is created that is located in the user subvolume and contains a list of all of the files created when you generated the EMS FastStart.

File: ATM1INDX

# 6

# Building Your Application for Event Generation

To use EMSFS, you must modify your application to use the EGEN module generated by EMS FastStart. These modifications will enable you to generate the event messages needed for your application. This section includes information on defining, modifying, and compiling an application (EGEN Operating Modes). It details the specific steps a programmer must take to use EGEN with a specific application and shows how to use EGEN with a COBOL85 program (Mode 1). For a description of TAL and C examples using EGEN, see the ATM example in the EMS FastStart Distribution subvolume.

## How EGEN Works

EGEN is the TAL module which links the application process and the EMS system. Your application interfaces with EGEN by moving values to the event record structure (called egen-record in this section) defined by the ACF. The ENTER TAL constructs pass these parameters to EGEN. EGEN then generates the tokenized event by performing a WRITE-READ of the event buffer to the collector $0 or to an alternate location.

## EGEN Operating Modes

EGEN is designed to be used by a number of different application types such as COBOL85 batch programs, COBOL85, C or TAL servers, or multi-threaded TAL programs. Each type of application has specific requirements (for example, a simple interface or full control of all I/O operations) which can be addressed by choosing one of the four operating modes supported by EGEN. Before any modifications can be made, you must select the correct EGEN operating mode. Each mode has a pre-defined set of procedures and parameters provided within the EGEN module. Table 6-1 and Table 6-2 summarize the procedures and parameters required for each mode of operation.

### Mode 1

This is the simplest operating mode and is used mainly by COBOL85 applications. In mode 1, EGEN sends approximately three messages to the collector for each event generated. Each of the other operating modes sends only one message per event. Because of the heavy use of CPU resources, it is recommended that mode 1 be used only by applications which generate few events.

For each event generated, EGEN opens the defined collector, writes the event, completes the write operation and then closes the defined collector.

## Mode 2

Mode 2 can be used by applications which generate many events. Mode 2 is like mode 1 but opens the defined collector only once, which reduces the use of system resources to generate events. The only disadvantage of this mode is that EGEN completes the write operation before returning to the calling program. In some cases, this may cause unacceptable delays for the application.

## Mode 3

Mode 3 allows EGEN to be used within a multi-threaded program because it completes the write operation within the calling program.

## Mode 4

Mode 4 uses EGEN only to format an event buffer based on the contents of the egen-record passed to EGEN. When you use this mode, you must write the event within the user application.

**Table 6-1. Procedures Required for Each Mode**

| | Procedure called for mode # | | | |
|---|---|---|---|---|
| **Procedure Name** | **1** | **2** | **3** | **4** |
| Initialize^egen^record | YES | YES | YES | YES |
| Open^egen^collector | NO | YES | YES | * |
| EGEN | YES | YES | YES | YES |
| Complete^egen^operation | NO | NO | YES | * |
| Close^egen^collector | NO | YES | YES | * |

*You can use these EGEN procedures or use your own customized procedures.

## Initialize^egen^record

This procedure initializes the egen^record with default values. A default value is specified in the DDL source file for each field within the record. See EGEN Default Values later in this section for detailed information on EGEN default values.

For all modes, this procedure must be called before calling the EGEN procedure. You must call this procedure each time you want to generate an event message, otherwise EGEN will return an error code to your application. Example 6-1 shows the syntax for this procedure.

**Example 6-1. Syntax for the Initialize^egen^record Procedure**

```
  { status := } INITIALIZE^EGEN^RECORD ( egen^record ) ! i/o
  { CALL    }

  status         returned value

       INT

        on return, is one of the following numbers:

        0   No Error

    <>  0   An error occurred when initializing the
            egen^record.  Please refer to Appendix B
            for the list of warnings or errors that can be returned by this
procedure.

egen^record     input, output

  INT     .EXT:ref:*

  is the structure to be initialized with default values.
```

# Open^egen^collector

This procedure opens the collector on behalf of the user application. It uses a run time parameter, called a DEFINE, which allows you to select the name of the collector to which events will be written. This DEFINE, =_EMS_COLLECTOR is discussed in Define Run-time Parameters on page 6-20. If this DEFINE is not used, the default collector $0 is opened.

The defined collector (or the default collector) is opened with a default sync depth of 1 (if not otherwise specified by the caller) and for nowait I/O. Example 6-2 shows the syntax for this procedure.

**Example 6-2. Syntax of the Open^egen^collector Procedure**

```
  { status := } Open^egen^collector ( collector^file^number, ! o
  { CALL      }                       [sync^depth],          ! i
                                      error^detail )         ! o

  status         returned value

       INT

        on return, is one of the following numbers:

                   0   No Error

             <>    0   An error occurred when opening the collector.
                       Please refer to Appendix B for the list of
                       warnings or errors that can be returned by this
                       procedure.

collector^file^number     output

  INT:ref:1

  is used to pass back the filenumber which was opened by this
  procedure.

sync^depth     input

INT:value

is used to specify a sync depth value used when opening the collector.
If not specified, a value of one is assumed.

error^detail   output

  INT:ref:1

  is used to pass back the file system error if an error occurs
  within this procedure.
```

# EGEN

The EGEN procedure accepts a record structure from the user application and formats a tokenized event buffer based on the information received. The EGEN mode is a function of the combination of parameters passed to EGEN by the application. Sample syntax for EGEN procedure is shown in Example 6-3.

## Example 6-3. Syntax for the EGEN Procedure

```
  { status := } EGEN (  egen^record,            ! i
  { CALL    }          [collector^file^number], ! i
                       [event^buffer^ptr],      ! i
                       [tag],                   ! i
                       [event^buffer^used] )    ! o

  status         returned value

       INT

       on return, is one of the following numbers:

        0   No Error

    <>  0   An error occurred within Egen.  Please
            refer to Appendix B for the list of
            warnings or errors that can be returned by
            this procedure.

egen^record     input

  INT     .EXT:ref:*

  is the structure which contains the event message information.

collector^file^number     input

  INT:ref:1

  is used to pass the file number on which to WRITEX or WRITEREADX
  the event message formatted by Egen.

event^buffer^ptr     input

  INT     .EXT:ref:1

  is the address of the buffer that will contain the tokenized
  event message formatted by Egen.

tag        input

  INT(32):value

  is used to pass a tag that will be associated with the WRITEX or
  WRITEREADX operation to be done by Egen.

event^buffer^used     output

  INT     .EXT:ref:1

  is used to pass back the length of the tokenized event message formatted by
Egen.
```

# Complete^egen^operation

This procedure is used to complete the WRITEX or WRITE-READX nowaited I/O on the collector file on behalf of the user application. Example 6-4 shows the syntax for this procedure.

**Example 6-4. Syntax for the Complete^egen^operation Procedure**

```
  { status := } COMPLETE^EGEN^OPERATION (  collector^file^number,  ! i
  { CALL      }                           tag,                    ! o
                                          [time^limit],           ! i
                                          error^detail )          ! o

  status         returned value

       INT

        on return, is one of the following numbers:

        0   No Error

    <>  0   An error occurred when completing the write of
            the event buffer.  Please refer to Appendix B
            for the list of warnings or errors that can
            be returned by this procedure.

collector^file^number     input

  INT:ref:1

  is used to pass the file number on which to complete the
  write/writeread operation.

tag     output

  INT(32)  .EXT:ref:1

  is used to pass back the tag associated with the operation just
  completed.

time^limit     input

  INT(32):value

  is used to pass a time limit value to the AWAITIOX procedure call.
  If no value is passed, the value of -1D will be used.

error^detail   output

  INT:ref:1

  is used to pass back the file system error if an error occurred within
  this procedure.
```

## Close^egen^collector

The Close^egen^collector procedure is used to close the current collector file on behalf of the user application. Example 6-5 shows the syntax for this procedure.

**Example 6-5. Syntax for the Close^egen^collector Procedure**

```
  { status := } CLOSE^EGEN^COLLECTOR ( collector^file^number ) ! i
  { CALL      }

  status         returned value

       INT

        on return, is one of the following numbers:

        0   No Error

    <>  0   An error occurred when closing the collector.
            Please refer to Appendix B for the list of
            warnings or errors that can be returned by
            this procedure.

collector^file^number

  INT:value

  is used to pass the filenumber to close.
```

# EGEN Parameters

EGEN internally detects the mode of operation by analyzing the combination of parameters passed to it. There are four combinations of parameters which are valid for EGEN. If you call EGEN with any other combination, an error message will be returned to you. Table 6-2 shows the parameters that must be passed to EGEN for each operating mode.

**Table 6-2. Parameters Required for Modes**

| | Mode # | | | |
|---|---|---|---|---|
| **Name Passed** | **1** | **2** | **3** | **4** |
| egen-record | YES | YES | YES | YES |
| user-file-number | NO | YES | YES | NO |
| event-buffer-ptr | NO | NO | YES | YES |
| user-tag | NO | NO | YES | NO |
| event-buffer-length | NO | NO | NO | YES |

# Egen-record Fields Definition

The Egen-record structure is the principal interface between an application and EGEN. Based on the values of these fields, EGEN will generate an event. Table 6-3 shows each of the fields and their descriptions. Note that all the user defined fields in the ACF will also be added to the egen-record. These fields are not described here because they are specific to each application.

**Table 6-3. Egen-record Fields and Descriptions** (page 1 of 2)

| Field Name | Field Data Type and Description |
|---|---|
| acf-version | ZSPI-DDL-INT.<br>This field contains the version of the ACF; the only version currently supported is B00. This is the internal version number of the ACF and should not be altered. This field is initialized each time you call the Initialize^egen^record procedure and is used internally by the EGEN procedure. You should not alter the value of this field. |
| ssid-owner | ZSPI-DDL-CHAR8<br>This field contains the owner of the event buffer and must be an eight-character string or less to identify your company or organization. |
| ssid-subsystem-number | ZSPI-DDL-INT<br>This field contains the SSID subsystem number and is identified by a 16-bit signed integer value that identifies the subsystem within the set of subsystems provided by the subsystem owner. |
| ssid-version | ZSPI-DDL-UINT<br>This field contains the software release version of the subsystem and is a 16-bit unsigned integer value. In the format of the field the left byte contains the letter part of the version as an ASCII uppercase alpha character, and the right byte contains the numeric part of the version as an unsigned integer value. |
| egen-error | ZSPI-DDL-INT<br>This field contains more detailed information about errors detected by an EGEN procedure. |
| event-type | ZSPI-DDL-ENUM<br>This field contains the type of event to be generated by EGEN. One of four types can be generated: Informative-event, Critical-event, Action-attention-event and Action-completion-event. |
| event-number | ZSPI-DDL-INT<br>This field contains the event number of the event to be generated by EGEN. |
| action-id | ZSPI-DDL-UINT<br>This field contains the action identifier of an action-attention-event or an action-completion-event. |

**Table 6-3. Egen-record Fields and Descriptions** (page 2 of 2)

| Field Name | Field Data Type and Description |
|---|---|
| suppress-display | ZSPI-DDL-BOOLEAN |
| | If TRUE, tells the Viewpoint application not to display the event message; Viewpoint will display it if the token is either FALSE or missing. By default, the value is FALSE, which mean that the event will be displayed on Viewpoint. See the EMS Manual, under the ZEMS-TKN-SUPPRESS-DISPLAY heading, for more information. |
| subsystem-manager | ZSPI-DDL-FNAME32 |
| | This field contains the process name of a particular subsystem process. See the DSM programing manual, under the ZSPI-TKN-MANGER heading for more information. |
| event-text | ZSPI-DDL-CHAR254 |
| | This field contains the displayable text describing the event. If you use the DSM templates services, then you do not need to put text into this field. See the EMS manual, under the ZEMS-TKN-TEXT heading for more information. |
| subject-field-name | ZSPI-DDL-CHAR24 |
| | This field contains the name of the other field in the egen-record which will contains the subject of this event. (Example, Initializing subject-field-name with ATM-NAME will cause the subject of this event to be the value contained by ATM-NAME (for example, “SFMAIN01”). |

## An Example Using Mode 2

After you have selected the operating mode for your application, modify the application code accordingly. Example 6-6 shows sample code for using EGEN in mode 2 within a COBOL85 program. (For a detailed example refer to Appendix C, COBOL85 Program Example.) As shown in Table 6-1, mode 2 requires the following procedures:

- Initialize^egen^record
- Open^egen^collector
- Egen
- Close^egen^collector

Calling these procedures opens the collector at the beginning of the program and then closes it before the end of the program.

Next, look at Table 6-2 for the parameters to use with the mode 2 EGEN procedure:

- egen-record
- user-file-number

Example 6-6 opens the collector at the beginning of the program and stores the file-number in a variable (section 205-open-collector in the example). In Example 6-6 the variable is called file-number. In section 210-atm-up, the egen-record is initialized in section 300-initialize-egen-record and the informative event message “atm is up” is moved into the event buffer along with the appropriate variable data (ATM name, location, and event type). EGEN is called next to generate event messages with two parameters (egen-record and file-number).

The collector is closed in the section 365-close-collector.

**Example 6-6. Sample Code for EGEN in Mode 2**

```
 205-open-collector.

  MOVE ZERO TO file-number.
    ENTER TAL "Open^egen^collector" USING file-number, omitted, error-detail
                                    GIVING return-code.
    IF return-code IS NOT EQUAL TO ZERO
       PERFORM 400-validate-return-code.

*--------------------------------------------------------------------------
* EVENT # 1: ATM IS UP:
*
* Fields within the egen-record which are used for event # 1:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*--------------------------------------------------------------------------

 210-atm-up.

    PERFORM 300-initialize-egen-record.

    MOVE INFORMATIVE-EVENT          TO event-type   OF egen-record.
    MOVE ATM-EVT-UP                 TO event-number OF egen-record.
    MOVE "SFMAIN01"                 TO atm-name     OF egen-record.
    MOVE "atm-name"                 TO subject-field-name OF egen-record.
    MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

    ENTER TAL "Egen" USING egen-record, file-number
                     GIVING return-code.
    IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.

 300-initialize-egen-record.

    ENTER TAL "Initialize^egen^record" USING egen-record
                                       GIVING return-code.
    IF return-code IS NOT EQUAL TO ZERO
       PERFORM 400-validate-return-code.

    MOVE ZERO            TO return-code.
    MOVE ATM-VAL-OWNER   TO ssid-owner            OF egen-record.
    MOVE ATM-SSN-NUMBER  TO ssid-subsystem-number OF egen-record.
    MOVE ATM-VAL-VERSION TO ssid-version          OF egen-record.

 365-close-collector.

    ENTER TAL "Close^egen^collector"  USING file-number
                                      GIVING return-code.
    IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.
```

# EGEN Default Values

EGEN default values are used by the EGEN procedure to determine if information should be added to the event buffer. EGEN compares the value of each field defined in your ACF against a set of default values. Each field type supported by EGEN has a specific default value.

Before generating an event, an application always initializes all fields of the egen-record (the egen-record contains all the fields declared in the ACF) with default values. This is done with the initialize^egen^record procedure. The following values are entered into the field types:

| Field type | EGEN Default Values |
|---|---|
| ZSPI-DDL-INT | 32767 |
| ZSPI-DDL-INT2 | 2147483647 |
| ZSPI-DDL-INT4 | 9223372036854775807 |
| ZSPI-DDL-UINT | 65535 |
| ZSPI-DDL-ENUM | 32767 |
| ZSPI-DDL-TRANSID | 9223372036854775807 |
| ZSPI-DDL-TIMESTAMP | 9223372036854775807 |

These default values were assigned because they represent values which are unlikely to occur during the course of running an application. If the actual value passed to EGEN differs from the default value, that new value is passed to the event message buffer. If the actual value matches the default value, EGEN assumes nothing has changed, and no value is added to the event buffer. The objective is to supply information which does not match the default values. If you feel that one of the EGEN default values may in fact match a legitimate value passed by an event, change the default value for that field type. If you do not do this, variable data will be missing from your event message buffer.

## Example

Table 6-4 shows the contents of the following fields after a call to the initialize^egen^record procedure.

**Table 6-4. Field Content After Call to Initialize^egen^record Procedure** (page 1 of 2)

| FIELD-NAME | FIELD-TYPE | FIELD-VALUE |
|---|---|---|
| ATM-NAME | ZSPI-DDL-CHAR8 | 8 spaces |
| ATM-LOCATION | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-ACCOUNT-NUM | ZSPI-DDL-INT2 | 2147483647 |
| ATM-RETRY-LIMIT | ZSPI-DDL-INT | 32767 |
| ATM-HW-COMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |

**Table 6-4. Field Content After Call to Initialize^egen^record Procedure** (page 2 of 2)

| FIELD-NAME | FIELD-TYPE | FIELD-VALUE |
|---|---|---|
| ATM-HW-SUBCOMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-SERIAL-NUMBER | ZSPI-DDL-INT2 | 2147483647 |
| ATM-SENSE-STATUS | ZSPI-DDL-INT2 | 2147483647 |

**Note.** For character fields such as ZSPI-DDL-CHAR8 and ZSPI-DDL-CHAR24, the default value is 8 and 24 blanks respectively.

Once the fields are initialized, information needed for each specific event must be moved into the fields, replacing the default values. Table 6-5 shows the ATM-EVT-ACCT-INSUF-FUNDS event generated by the ATM COBOL85 example. This shows all the fields values after event-specific information was moved into three fields: ATM-NAME, ATM-LOCATION, and ATM-ACCOUNT-NUM. These fields are shown below (**bold**).

**Table 6-5. ATM COBOL85 Example**

| FIELD-NAME | FIELD-TYPE | FIELD-VALUE |
|---|---|---|
| **ATM-NAME** | ZSPI-DDL-CHAR8 | OAKWEST1 |
| **ATM-LOCATION** | ZSPI-DDL-CHAR24 | 245 Oak St., Oakland |
| **ATM-ACCOUNT-NUM** | ZSPI-DDL-INT2 | 34503933 |
| ATM-RETRY-LIMIT | ZSPI-DDL-INT | 32767 |
| ATM-HW-COMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-HW-SUBCOMPONENT | ZSPI-DDL-CHAR24 | 24 spaces |
| ATM-SERIAL-NUMBER | ZSPI-DDL-INT2 | 2147483647 |
| ATM-SENSE-STATUS | ZSPI-DDL-INT2 | 2147483647 |

The application now calls EGEN. EGEN looks at the contents of the field-value and compares it with the EGEN default values. Only those fields whose contents differ from the default values are added to the event buffer. In the example above, only those fields which are shown in bold type will be added to the event buffer. Note that if the field value is not modified, and the default value is used, nothing is added to the event buffer. Should this happen, you may find some variable data missing from the events you generate.

To change the default field values, generate your application and modify the *xxxx*TAL (for example, ATM1TAL for our sample) DDL file (Example 6-7). Enter the new default values. Then re-compile the EGEN module with the new TAL DDL and re-bind this new TAL module with your application.

Remember, it is important to choose default values which are unlikely to occur in your application. Do not use a common or likely value as a default. In the example above, a “real-life” ATM-RETRY-LIMIT value is unlikely to match the default value of 32767;

however, it is likely that a value of 4 might be passed to the field-value. Therefore, 4 would not be a wise choice as a default value for that field type.

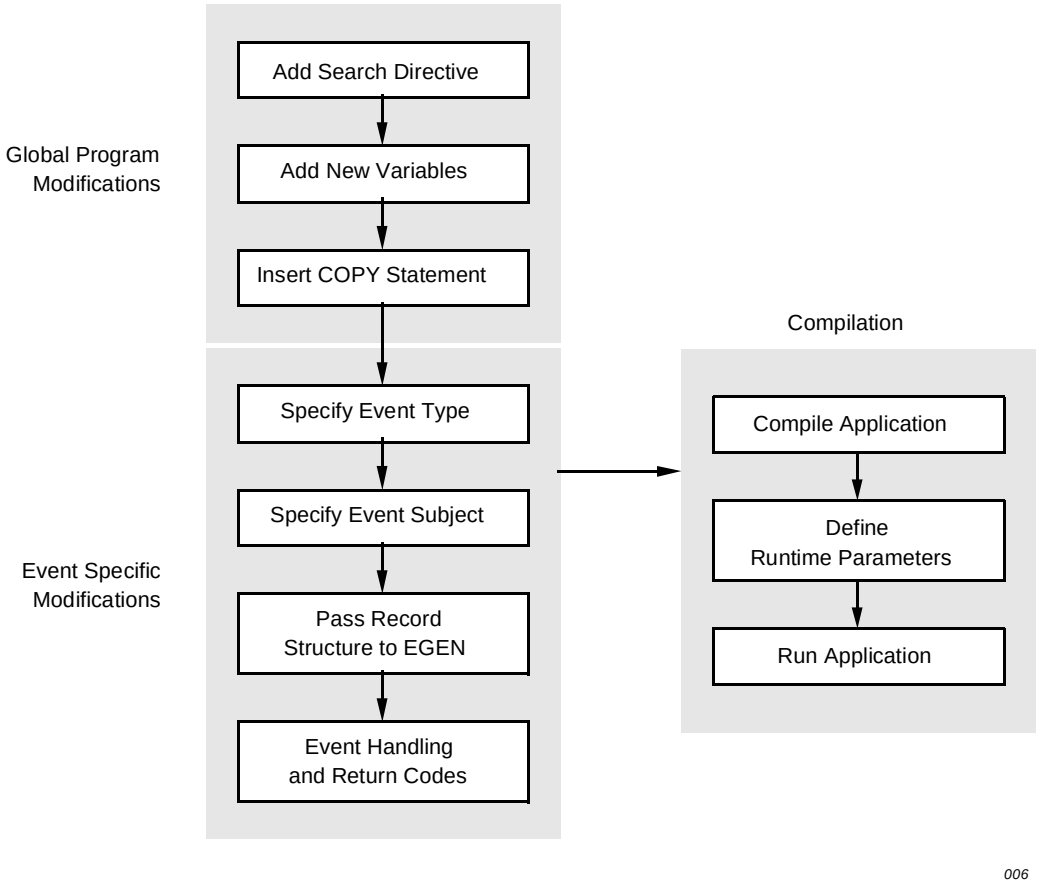
**Example 6-7. EGEN Default Values**

```
!----------------------------------------------------------------------
!
! Constants used by Egen to define the default values of a field
!
LITERAL EMSFS^DEFAULT^INT = 32767;
LITERAL EMSFS^DEFAULT^INT2 = 2147483647D;
LITERAL EMSFS^DEFAULT^INT4 = 9223372036854775807F;
LITERAL EMSFS^DEFAULT^UINT = %177777;
LITERAL EMSFS^DEFAULT^ENUM = 32767;
LITERAL EMSFS^DEFAULT^TRANSID = 9223372036854775807F;
LITERAL EMSFS^DEFAULT^TMESTAMP = 9223372036854775807F;
```

# Application Modifications

In order for EGEN to work with your application, certain application modifications must be made. These modifications are grouped into three parts: global program modifications, event-specific modifications, and compilation. Figure 6-1 shows these phases.

**Figure 6-1. Application Modification Phases**



# Global Program Modifications

## Add the SEARCH Directive

Add the ?SEARCH directive to the user application, specifying the location of EGEN. The compiler must know where EGEN resides in order to bind it with your application.

Example:

```
?SEARCH $data.atm.atm1egen
```

## Add New Variables

Add variables for the specific values needed by EGEN. Example of variables added to the COBOL85 ATM program to support the EGEN interface:

```
01 cobol-val-true          PIC xx value high-values*
01 cobol-val-false         PIC xx value low-values*
01 return-code             PIC S9(4)
01 file-number             PIC 9(4) COMP
01 error-detail            PIC 9(4) COMP.
```

*These need to be moved into the suppress-display field of the egen record to suppress the display.

## Insert Copy Statements

The five copy statements described below must be inserted in the record-definition of the Data Division in your program. You can also add copy statements to include specific definitions for your program. Example 6-8 below shows a copy statement.

| | |
|---|---|
| **egen-record** | The definition of the egen-record as specified in your ACF. It contains other fields needed by EGEN (for example, egen-error) |
| **atm-ssid** | The definition for your SSID. |
| **egen-interfaces-definitions** | Defines constants used to tell EGEN which type of event to generate and constants used to define all the return codes that EGEN can return to your application. |
| **user-event-numbers** | Defines constants used to specify each event generated by your application (for example, ATM-DOWN). |
| **user-action-id** | Defines constants used to generate action attention and action completion events. |

**Example 6-8. Copy Statement Example from the COBOL85 ATM Sample Program**

```
COPY egen-record                    OF "$data.atm.atm1cob".
COPY atm-ssid                       OF "$data.atm.atm1cob".
COPY egen-interface-definitions     OF "$data.atm.atm1cob".

COPY user-event-numbers             OF "$data.atm.atm1cob".
COPY user-action-id                 OF "$data.atm.atm1cob".
```

# Event-Specific Modifications

## Specify Event Types

Events are classified as one of four types: Informative (the default type), Action attention, Action completion, or Critical. This section describes the event types and includes sample COBOL85 code to show how to generate each type. The sample code is part of the COBOL85 ATM example listed in Appendix C, COBOL85 Program Example, and is included on the ISV. For a more detailed description on the event types, refer the *Event Management Service (EMS) Manual*.

## Informative

An informative event message informs you about any system or program status (for example, the program started normally). This is the default event message type for EMS FastStart unless you specify otherwise. See Example 6-9 below.

**Example 6-9. Sample Code for Generating Informative Events**

```
GENERATE-INFORMATIVE-EVENT.

MOVE INFORMATIVE-EVENT     TO event-type   OF egen-record.
MOVE ATM-EVT-UP            TO event-number OF egen-record.
MOVE "SFMAIN01"            TO atm-name     OF egen-record.
MOVE "atm-name"            TO subject-field-name OF egen-record.
MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

ENTER TAL "Egen" USING egen-record, file-number
                              GIVING return-code.
IF return-code NOT = ZERO
  PERFORM   400-validate-return-code.
```

## Action Attention

An action-attention event message means operator intervention is required (for example, a message to change a tape or add paper to a printer). This is the first half of a pair with an action-completion event message being the second half. These two events must be linked. To tie each action-attention event with a specific action-completion event, move the same user-action-id value to action-id for both the action-attention events and action-completion events and use the same subject value for each. See Example 6-10 below.

**Example 6-10. Sample Code for Generating Action-Attention Events**

```
GENERATE-ACTION-ATTENTION.

PERFORM 300-initialize-egen-record.

MOVE ACTION-ATTENTION-EVENT     TO event-type   OF egen-record.
MOVE USER-ACTION-ID-1           TO action-id    OF egen-record.
MOVE ATM-EVT-LOW-ON-CASH        TO event-number OF egen-record.
MOVE "SFMAIN02"                 TO atm-name     OF egen-record.
MOVE "atm-name"                 TO subject-field-name OF egen-record.
MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

ENTER TAL "Egen" USING egen-record, file-number
                               GIVING return-code.
IF return-code NOT = ZERO
     PERFORM 400-validate-return-code.
```

## Action Completion

An action-completion event message informs you that the action required in an action-attention event message was completed and the job is continuing. Note in this Example 6-11, that the user-action-id (user-action-id-1) from the action-attention example above was moved into the field action-id. Also, the same subject value was used.

**Example 6-11. Sample Code for Generating Action-Completion Events**

```
GENERATE-ACTION-COMPLETION.

PERFORM 300-initialize-egen-record.

MOVE ACTION-COMPLETION-EVENT    TO event-type     OF egen-record.
MOVE ATM-ACTION-ID-1            TO action-id      OF egen-record.
MOVE ATM-EVT-BACK-ONLINE        TO event-number   OF egen-record.
MOVE "SFMAIN02"                 TO atm-name       OF egen-record.
MOVE "atm-name"                 TO subject-field-name OF egen-record.
MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

ENTER TAL "Egen" USING egen-record, file-number
                               GIVING return-code.
IF return-code NOT = ZERO
     PERFORM 400-validate-return-code.
```

Example 6-12 shows sample code for generating action completion events with the display suppressed.

**Example 6-12. Sample Code for Generating Action Completion Events with Suppress Display**

```
GENERATE-ACTION-COMPLETION.

PERFORM 300-initialize-egen-record.

MOVE ACTION-COMPLETION-EVENT    TO event-type    OF egen-record.
MOVE COBOL-VAL-TRUE             TO SUPPRESS-DISPLAY OF egen-record.
MOVE ATM-ACTION-ID-1            TO action-id    OF egen-record.
MOVE ATM-EVT-BACK-ONLINE        TO event-number OF egen-record.
MOVE "SFMAIN02"                 TO atm-name     OF egen-record.
MOVE "atm-name"                 TO subject-field-name OF egen-record.
MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

ENTER TAL "Egen" USING egen-record, file-number
                               GIVING return-code.
IF return-code NOT = ZERO
     PERFORM 400-validate-return-code.
```

## Critical

A critical event message informs you that an event such as a system error, component failure, or security breach has occurred and requires immediate attention. See Example 6-13 below.

**Example 6-13. Sample Code for Generating Critical Events**

```
GENERATE-CRITICAL-EVENT.

MOVE CRITICAL-EVENT             TO event-type   OF egen-record.
MOVE ATM-EVT-SECURITY-BREACH    TO event-number OF egen-record.
MOVE "SACTO02"                  TO atm-name     OF egen-record.
MOVE "atm-name"                 TO subject-field-name OF egen-record.
MOVE "230 State St. Sacramento" TO atm-location OF egen-record.
MOVE 23457320                   TO atm-account-num OF egen-record.
MOVE 3                          TO atm-retry-limit OF egen-record.

ENTER TAL "Egen" USING egen-record, file-number
                                  GIVING return-code.

IF return-code NOT = ZERO
     PERFORM 400-validate-return-code.
```

# Specify Event Subject

An event message is generated by a subsystem whenever a significant occurrence is detected. An event message includes many kinds of information besides “what happened.” Every event message must include a subject: the name or the number of the hardware or software component most directly involved in the event.

Since each event can have a different subject, EMS FastStart has implemented a facility which provides the capability to select a unique subject for each event. In fact, any field name that you declare in the ACF can become the subject of an event. However, you are restricted to one subject per event and the field which will become the subject cannot be added as another field in the event record.

The application needs to tell EGEN which field in the event buffer is the subject of the event. This is accomplished in the user's application by moving the name of the selected field to another field called subject-field-name. At this point, you can move the specific values into the field that will become the subject.

The following example illustrates the point. An informative event will be generated which will alert the operations staff that the ATM named SFMAIN01 is now up and running. Since this event is about this specific ATM, we want the subject of this event to be SFMAIN01. To accomplish this, we use the field named “atm-name” to contain the name of the ATM (SFMAIN01) and then we initialize the field called subject-field-name with the name of the field containing the subject, in this case atm-name. Example 6-14 shows the subject fields from the COBOL85 ATM program.

**Example 6-14. Sample Code Showing Subject Data**

```
MOVE INFORMATIVE-EVENT      TO event-type    OF egen-record.
MOVE ATM-EVT-UP             TO event-number  OF egen-record.
MOVE "SFMAIN01"             TO atm-name   OF egen-record.
MOVE "atm-name"             TO subject-field-name OF egen-record.
MOVE "245 A St., San Francisco" TO atm-location OF egen-record.
```

## Move Values to the Event Record

Move the values associated with the specific events into the fields that were defined in the ACF. In Example 6-15 below, the fields associated with the event #200 (ATM DOWN) are:

- Event type
- Event number
- ATM name
- ATM location
- Event text

The programmer is responsible for determining the values for each field and moving them into the event record.Example 6-15 shows the ATM COBOL85 example.

**Example 6-15. ATM COBOL85 Example**

```
?HEADING "240-ATM-DOWN SECTION"
/
 240-atm-down.
*---------------------------------------------------------------------
* EVENT # 200: ATM IS DOWN:
*
* the fields within the EGEN record which are used for event # 200:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*      event-text         PIC X(254).
*
*---------------------------------------------------------------------

  PERFORM 300-initialize-egen-record.

  MOVE CRITICAL-EVENT               TO event-type   OF egen-record.
  MOVE ATM-EVT-DOWN                 TO event-number OF egen-record.
  MOVE "LACENT99"                   TO atm-name     OF egen-record.
  MOVE "atm-name"                   TO subject-field-name OF egen-record.
  MOVE "9320 Main St., Los Angeles" TO atm-location OF egen-record.

  ENTER TAL "Egen" USING egen-record, file-number
                                  GIVING return-code.

  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.
```

## Pass the Record Structure to EGEN

Pass the record structure to EGEN by issuing the following statement from within the user application:

```
ENTER TAL "Egen" USING egen-record GIVING return-code.
```

## Error Handling and Return Codes

Each time you call one of the procedures implemented in the EGEN module you should check to see if any errors occurred when generating an event by verifying the value of the return-code. Appendix B, EGEN Messages, describes the return codes defined between EGEN and a user application (implemented by EGEN as a set of TAL procedures).

A return-code for each procedure implemented in the EGEN module informs the calling application of the status of the operation requested. If the return-code is equal to zero (0), the operation was successful. If the return-code is not equal to zero (0), it indicates a warning or an error. Also, a special field called egen-error (Integer) in the EGEN record structure is used to give you more detailed information about the error. The contents of this field depend on the return-code of the EGEN procedures.

The following is a list of procedures for which errors, warnings, and advisory return codes are returned to the calling application. (The return codes for each procedure are listed and described in Appendix B, EGEN Messages.)

- Initialize^egen^record
- Open^egen^collector
- Close^egen^collector
- Complete^egen^operation
- Get^egen^event^text^define
- Initialize^event^buffer
- Write^event^buffer
- EGEN

# Compile Application

After you have completed the global and the event specific modifications, compile the application program. If the application compiles successfully, you will receive no system error messages. If you receive a system error message, correct the problem and re-compile the program. Consult your Tandem compiler manual for information about specific system errors.

## Define Run-time Parameters

EGEN supports two run time parameters which add flexibility to applications that use EMS FastStart. These parameters allow you to specify an alternate EMS collector (the default is $0.). These parameters are implemented with two DEFINEs which can be changed each time you run your application. (For more information about DEFINEs see the *GUARDIAN 90 Operating System User's Guide*.)

The =_EMS_COLLECTOR DEFINE allows you to specify the location where you want EGEN to pass the buffer via a write-read command. The default location is the primary collector $0. You can also specify a disk file for testing purposes before you install your application in production. If you do not specify another collector in this DEFINE, EGEN will use $0 as the default collector.

---

**Note.** If you specify a disk file as a collector, it must have the same structure and attributes as the EMS event log file if you want to use a distributor to print the contents of this file.

---

The second DEFINE, =_EGEN_ADD_EVENT_TEXT tells EGEN whether to add the token ZEMS^TKN^TEXT (plus its value, which in this case is the event text) to the event buffer. The valid values are $YES (add the event text) or $NO (do not add the event text). $NO is the default value.

## Using TACL to Set up and Change DEFINEs

To add the DEFINEs to your application, use the following TACL macro. Example 6-16 shows how to add DEFINE values for =_EMS_COLLECTOR and =_EGEN_ADD_EVENT_TEXT.

### Example 6-16. TACL Macro File

```
==
==
?TACL MACRO
==
#OUTPUT Adding the =_EMS_COLLECTOR define.
==
==
== The define =_EMS_COLLECTOR is used to specify the location of the
== collector to EGEN. (The default collector is $0). If a filename
== is entered which has a file code of 843 (EMS event log files,
== type) EGEN will write formatted event messages to that file.
== As an aid in testing the functionality of an application using
== EGEN, those messages can be printed from this file by using a
== standard EMS distributor (EMSDIST).
==
SET DEFMODE ON
SET DEFINE CLASS MAP
SET DEFINE FILE $0
ADD DEFINE =_EMS_COLLECTOR

#OUTPUT

INFO DEFINE (=_EMS_COLLECTOR)

#OUTPUT Adding the =_EGEN_ADD_EVENT_TEXT define.
==
==
== The define =_EGEN_ADD_EVENT_TEXT is used to specify that EGEN
== should add the EVENT-TEXT field to the event buffer.  The
== following values are valid: $YES and $NO.
==
SET DEFINE CLASS MAP
SET DEFINE FILE $YES
ADD DEFINE =_EGEN_ADD_EVENT_TEXT
#OUTPUT
INFO DEFINE (=_EGEN_ADD_EVENT_TEXT)
#OUTPUT
#OUTPUT Starting $DATA.ATM.ATM1PROG.
#OUTPUT
RUN $DATA.ATM.ATM1PROG
```

# 7 Testing Program and Filter

EMS FastStart generates a Testing Program, *xxxx*PROG (where *xxxx* is the name of your USER-SUBVOL-FILE-PREFIX), and an EMF filter (Using the Filter Program with a Printing Distributor), *xxxx*EMFO, to help you test your customized EGEN module before you bind EGEN with your application.

You can use the testing program and the filter to generate event messages similar to those you want your application to generate. You can display the event messages using Viewpoint or a printing distributor. The default filter can be used by Viewpoint or a printing distributor to select only those events generated by the testing program. The testing program can also be used to test other filters and templates as well as DSM programmed applications prior to completing your application using EMS FastStart.

This section describes the five event types the testing program can generate, shows you a sample run of the testing program generated from the ATM example, and describes the use of the filter program with a printing distributor.

## Testing Program

The EMS FastStart testing program, *xxxx*PROG, is written in COBOL85 and supports a simple conversational interface which allows you to generate four types of "hard-coded" events or a customized event type.

---

**Note.** The testing program (*xxxx*PROG) uses EGEN mode 2 as a default.

---

The four hard-coded event types are:

- INFORMATIVE-EVENT
- ACTION-ATTENTION-EVENT
- ACTION-COMPLETION-EVENT
- CRITICAL-EVENT

Each of these event types is supported by EGEN and has specific parameters which you cannot change. (See Section 6, Building Your Application for Event Generation, for more information on event types.)

The fifth event type supported by EGEN is DATA-ENTRY-EVENT. This event type allows you to enter the parameters of your customized EGEN procedure.

The DATA-ENTRY event type lets you select:

- Owner of the event (SSID)
- Type of the event
- Value of the suppress display flag
- Number of the event
- Name of the subsystem manager

- Text of the event
- Value for each field defined in the ACF
- Field which will be the subject of this event.

# Testing Program Sample Session

A TACL macro file is created for you on the subvolume which contains the EMSFS testing program. This macro file adds the =_EMS_COLLECTOR and the =_EGEN_ADD_EVENT_TEXT DEFINES and then initializes them with default values ($NO is the =_EGEN_ADD_EVENT_TEXT default value). (For more information about these defines please refer to Section 6, Building Your Application for Event Generation.)

---

**Note.** In the following sample session based on the ATMACF example, the USER-SUBVOL-FILE-PREFIX parameter was ATM1 so the TACL macro file is called ATM1TEST and the EMS FastStart testing program is called ATM1PROG.

---

Example 7-1 shows a sample session using the EMS FastStart testing program generated for the ATM example. The user input is shown in **bold** type.

---

**Example 7-1. Testing Program Sample Session** (page 1 of 2)

```
1> RUN ATM1TEST
Adding the =_EMS_COLLECTOR define.

    Define Name         =_EMS_COLLECTOR
    CLASS               MAP
    FILE                $0

Adding the =_EGEN_ADD_EVENT_TEXT define.
    Define Name         =_EGEN_ADD_EVENT_TEXT
    CLASS               MAP
    FILE                $YES

Starting $DATA.ATM.ATM1PROG

EMS Fast Start - T9263C20 - (17MAR91) - ATM1PROG

This program will prompt you for the type of Event Message
to generate

The following event message types are currently supported
 I-  Informative Event
 AA- Action Attention Event
 AC- Action Completion Event
 C-  Critical Event
 D-  Data entry event.  Enter necessary fields for testing.
 H-  Help
 E-  Exit Program
Enter the event message type to generate: I, AA, AC, C, D or H
>I
EGEN generated the event message successfully

>AA
EGEN generated the event message successfully

>AC
EGEN generated the event message successfully
```

---

**Example 7-1. Testing Program Sample Session** (page 2 of 2)

```
>D
Please enter the required information:

ssid owner (ZSPI-TYP-CHAR8): CUSTOMER
ssid subsystem number (ZSPI-TYP-INT): 1
ssid version (ZSPI-TYP-UINT): 12345
event type (I,AA,AC,C): AA
action identifier (ZSPI-TYP-INT): 2
suppress-display (T,F): F
event number (ZSPI-TYP-INT): 100
subsystem manager (ZSPI-DDL-FNAME32):
event text (ZSPI-TYP-CHAR254): This is a test event generated by ATM1PROG
atm-name (ZSPI-TYP-CHAR8): MTL0122
atm-location (ZSPI-TYP-CHAR24): 221 Rene Levesque, MTL.
atm-account-num (ZSPI-TYP-INT2): 122345
atm-retry-limit (ZSPI-TYP-INT): 3
atm-hw-component (ZSPI-TYP-CHAR24):
atm-hw-subcomponent (ZSPI-TYP-CHAR24:
atm-serial-number (ZSPI-TYP-INT2): 334356
atm-sense-status (ZSPI-TYP-INT2): 03303

Please enter the name of the field which value will be the subject
of this event (e.g. Program-name, Atm-name, ...)

Event subject field name (ZSPI-TYP-CHAR24): atm-name


This is the current setting of EGEN-RECORD

Ssid owner: CUSTOMER
Ssid subsystem number: 00001
Ssid version: 12345
event type: Action-attention event
action-identifier: 2
suppress display: False
event number: 00100
subsystem manager name:
Event subject field name: atm-name
Event text: This is a test event generated by ATM1PROG
atm-name: MTL0122
atm-location: 221 Rene Levesque, MTL
atm-account-num: 0000122345
atm-retry-limit: 00003
atm-hw-component:
atm-hw-subcomponent:
atm-serial-number: 0000334356
atm-sense-status: 0000003303

EGEN generated the event message successfully

>H
This program will prompt you for the type of Event Message
to generate

The following event messages type are currently supported
 I-  Informative Event
 AA- Action Attention Event
 AC- Action Completion Event
 C-  Critical Event
 D-  Data entry event.. you enter the event fields
 H-  Help
 E-  Exit Program

>E
```

# Using the Filter Program with a Printing Distributor

This section shows how to display the event messages generated by the EMS FastStart testing program. Please refer to the *Event Management Service Manual* for a detailed explanation of how to start and configure an EMS distributor (EMSDIST).

The following command syntax starts a printing distributor that reads events from the collector $0 and prints them to the terminal named $term31.

```
TACL> EMSDIST /NAME $mydist/TYPE p, COLLECTOR $0, TEXTOUT
$term31, FILTER $data.atm.atm1emfo , TIME 11:00
```

The filter object file $data.atm.atm1emfo is used by the printing distributor to select the events generated by the ATM1PROG (EMS FastStart testing program) after 11:00 (the specified time). Example 7-2 shows the screen output of the EMSDIST program.

**Example 7-2. Screen Output of the EMSDIST Program**

```
89-11-20 14:21:14 \MTL.02,171       CUSTOMER.1.J00        000004 Program:
                                    MYPROG of application: PAYROLL, BATCH RUN
                                    started
89-11-20 14:21:44 \MTL.02,171       CUSTOMER.1.J00        000100 Please mount
                                    tape (BA5277) on $TAPE1 for the PAYROLL
                                    batch run
89-11-20 14:22:07 \MTL.02,171       CUSTOMER.1.J00        000100 Tape (BA5277)
                                    accepted on $TAPE1, PAYROLL batch
                                    continuing ...
89-11-20 14:38:10 \MTL.02,171       CUSTOMER.1.12345      000100 This is a
                                    test event generated by ATM1PROG
```

## Filtering on Specific Tokens

Each of the applications fields you define in the ACF will have a token identifier associated with them. These tokens are created by EMS FastStart during the generation process and are defined in the DDL file. You can then write a filter that will be used by an EMS Distributor to print specific events. This filter could be used to verify the existence of a specific token in your event messages or compare the value of a specific field. The name of each of these tokens is built according to these rules:

- Each field name you define in the ACF will have two prefixes added to them. The TAL/TACL syntax is used to describe the token names here since this is the same syntax used by the filter compiler language.
- The first prefix added is “^TKN^”.
- The second prefix added is the USER-VARIABLES-PREFIX parameter (for example, ATM).

The following examples show the token name created by the ATM example used throughout this manual. Please note that the token field name syntax is shown as TACL compatible since this syntax will be used by the filter compiler. In this example, the USER-VARIABLES-PREFIX is ATM.

| **ACF field name** | **Token field name** |
|---|---|
| ATM-NAME | ATM^TKN^ATM^NAME |
| ATM-LOCATION | ATM^TKN^ATM^NAME^LOCATION |
| ATM-ACCOUNT-NUMBER | ATM^TKN^ATM^NAME^ACCOUNT^NUMBER |
| ATM-RETRY-LIMIT | ATM^TKN^ATM^RETRY^LIMIT |
| ATM-HW-COMPONENT | ATM^TKN^ATM^HW^COMPONENT |
| ATM-HW-SUBCOMPONENT | ATM^TKN^ATM^HW^SUBCOMPONENT |
| ATM-SERIAL-NUMBER | ATM^TKN^ATM^SERIAL^NUMBER |
| ATM-SENSE-STATUS | ATM^TKN^ATM^SENSE^STATUS |

# A EMS FastStart Messages

These code samples show the run-time errors, warnings, and advisory messages you may receive when you run EMS FastStart.

You see the following lines of code on your terminal after a successful EMS FastStart run:

```
EMS FastStart - T9263C20 - (17MAR91)

Number of generation errors = <integer value>
Number of generation warnings = <integer value>
Number of files created = <integer value>
Number of files compiled = <integer value>
Number of fields validated = <integer value>
Generation cpu time = <hh:mm:ss>
Total elapsed time = <hh:mm:ss>
```

If you press the BREAK key to terminate an EMS FastStart run, the following warning appears.

```
EMS FastStart - T9263C20 - (17MAR91)

** Warning 2 ** EMS FastStart stopping, BREAK key entered.

Number of generation errors = <integer value>
Number of generation warnings = <integer value>
Number of files created = <integer value>
Number of files compiled = <integer value>
Number of fields validated = <integer value>
Generation cpu time = <hh:mm:ss>
Total elapsed time = <hh:mm:ss>
```

If EMS FastStart terminates because of an undefined error, the following message appears.

```
EMS FastStart - T9263C20 - (17MAR91)

** Error 3 ** EMS FastStart terminating, an undefined error occurred.

Number of generation errors = <integer value>
Number of generation warnings = <integer value>
Number of files created = <integer value>
Number of files compiled = <integer value>
Number of fields validated = <integer value>
Generation cpu time = <hh:mm:ss>
Total elapsed time = <hh:mm:ss>
```

The following errors and warnings and their probable causes and recommended actions may appear on your terminal during an EMS FastStart run.

```
** Error 4 ** Unable to purge a file in the <user-subvol> subvolume.

Probable cause:

Purge error: <file system error> on file <filename>.

Recommended action:

EMS FastStart cannot continue because of this error and is terminating.
Please check the file security on your user subvolume.
```

```
** Warning 50 ** The ACF does not exist.


Probable cause:

The file: <acf-filename> does not exist and a default ACF was created.

Recommended action:

Please edit the prototype ACF to include your specific application
parameters.
```

```
** Error 51 ** An error occurred when trying to process the ACF.

Probable cause:

Unable to open the file: <acf-filename>, file error: <file system error>.

Recommended action:

Please correct the error by specifying a valid ACF.
```

```
** Error 52 ** Invalid key word in the ACF.

Probable cause:

EMS FastStart was expecting the <expected-keyword> key word and the
current key word is <current-keyword>.

Recommended action:

Please correct the key word in the ACF file.
```

```
** Error 53 ** Missing Parameter or Invalid key word.

Probable cause:

EMS FastStart detected that no parameters were supplied for this keyword
or the key word expected was: <expected-keyword>.

Recommended action:

Please correct the ACF by specifying a valid key word or a valid
parameter.
```

```
** Error 54 ** Invalid ACF file or missing key word.

Probable cause:

EMS FastStart was expecting the <expected-keyword>.

Recommended action:

Please supply a valid ACF name or add the key word in the ACF.
```

```
** Error 56 ** Invalid ACF-VERSION parameter.

Probable cause:

The ACF-VERSION is invalid; the only version currently supported is B00.

Recommended action:

Please correct the ACF-VERSION parameter in the ACF file.
```

```
** Error 57 ** Invalid filename format.

Probable cause:

EMS FastStart tried to validate the subvolume name or the whole filename
and detected an error.

Recommended action:

Please specify a valid subvolume name or a valid file name.
```

```
** Error 58 ** The location specified for the EMSFS-SUBVOL parameter is
invalid.

Probable cause:

The file: <filename> does not exist.

Recommended action:

Please correct the EMSFS-SUBVOL parameter in the ACF file.
```

```
** Error 59 ** The ZSPIDEF-SUBVOL parameter is invalid.

Probable cause:

One of the following files: ZEMSTACL, ZEMSTAL, ZEMSDDL, ZSPIDDL,
ZSPITACL, or ZSPITAL was not validated by EMS FastStart because of the
following  reason:  The file <filename> does not exist.

Recommended action:

Please correct the ZSPIDEF-SUBVOL parameter to point to the right
location.
```

```
** Error 60 ** The USER-SUBVOL parameter is invalid.

Probable cause:

The USER-SUBVOL parameter is invalid or specifies a non-existent volume or
is equal to the EMSFS-SUBVOL.

Recommended action:

Please correct the USER-SUBVOL parameter in the ACF.
```

```
** Error 61 ** The USER-SUBVOL-FILES-PREFIX parameter is invalid.

Probable cause:

The USER-SUBVOL-FILES-PREFIX parameter is invalid. It should be exactly 4
characters long and start with an alphabetic character (e.g. APP1).

Recommended action:

Please correct the USER-SUBVOL-FILES-PREFIX parameter in the ACF.
```

```
** Error 62 ** The USER-VARIABLES-PREFIX parameter is invalid.

Probable cause:

It should be exactly 4 characters long and start with an alphabetic
character(e.g. APPL).

Recommended action:

Please correct the USER-VARIABLES-PREFIX parameter in the ACF.
```

```
** Error 63 ** The APPLICATION-SSID-OWNER parameter is invalid.

Probable cause:

It should be a valid subsystem identifier as specified by DSM.

Recommended action:

Please correct the APPLICATION-SSID-OWNER parameter in the ACF.
```

```
** Error 64 ** The APPLICATION-SSID-NUMBER parameter is invalid.

Probable cause:

It must be between 0 and 32767.

Recommended action:

Please correct the APPLICATION-SSID-NUMBER parameter in the ACF.
```

```
** Error 65 ** The APPLICATION-SSID-VERSION parameter is invalid.

Probable cause:

It must be exactly 3 characters long ( 1 alphabetic character + 2 numeric
characters (e.g.  A00, C10).

Recommended action:

Please correct the APPLICATION-SSID-VERSION parameter in the ACF.
```

```
** Error 67 ** The EVENT-TEXT-TYPE parameter is invalid.

Probable cause:

The following values are valid: ZSPI-DDL-CHAR128, ZSPI-DDL-CHAR254.

Recommended action:

Please correct the EVENT-TEXT-TYPE parameter in the ACF.
```

```
** Error 68 ** Duplicate FIELD-<field-number>-NAME parameter.

Probable cause:

This parameter value has the same name as a previous FIELD-NAME or has the
same name as one of the EGEN-RECORD reserved names:  ACF-VERSION, SSID-
OWNER, SSID-SUBSYSTEM-NUMBER, SSID-VERSION, EGEN-ERROR, EVENT-TYPE, EVENT-
NUMBER, ACTION-ID, SUPPRESS-DISPLAY, SUBSYSTEM-MANAGER, EVENT-TEXT, or
SUBJECT-FIELD-NAME.

Recommended action:

Please correct the FIELD-<field-number>-NAME parameter in the ACF to make
the FIELD-<field-number>-NAME unique.
```

```
** Error 69 ** The FIELD-<field-number>-NAME parameter is invalid.

Probable cause:
It must be from 1 to 20 characters long and be COBOL compatible (e.g.
Progra m-name, File-error, Text-detail-1, etc .....).

Recommended action:

Please correct the FIELD-<field-number>-NAME parameter in the ACF.
```

```
** Warning 70 ** The FIELD-<field-number>-TYPE parameter was not
validated.

Probable cause:

EMS FastStart was not able to validate this field data type since it is
not one of the default data type supported.  The supported data type are
described in the EMS FastStart manual.

Recommended action:

If this data type was added in the EXTRADDL file, then you can ignore this
warning.  You should assure that this type is declared as a character
field,since EMS FastStart does not support structures.
```

```
** Error 71 ** The FIELD-<field-number>-NUMBER parameter is invalid.

Probable cause:

It must be between 0 and 9990. The field numbers 9991 through 9998  are
reserved for EGEN.

Recommended action:

Please correct the FIELD-<field-number>-NUMBER in the ACF.
```

```
** Error 72 ** Duplicated FIELD-<field-number>-NUMBER parameter.

Probable cause:

The FIELD-<field-number>-NUMBER parameter was already used by another
field.

Recommended action:

Please correct the FIELD-<field-number>-NUMBER in the ACF.
```

```
** Error 73 ** EMS FastStart internal error.

Probable cause:

This is an internal error that occurred when trying to validate the ACF
parameters.  It should never occur unless there is an integrity problem
within the routine Utils:parse^acf^file.

Recommended action:

Please submit the problem to Tandem with the following files: EMSFS,
EMSFSC20,CCF, EXTRADDL, and your ACF.
```

```
** Error 74 ** Invalid field type.

Probable cause:

The FIELD-<field-number>-TYPE keyword parameter does not start with a
ZSPI-  DDL-CHAR form.

Recommended action:

Please specify a valid FIELD-<field-number>-TYPE.
```

```
** Error 75 ** The SAVE-DDL-DICTIONARY parameter is invalid.

Probable cause:

The following values are valid: YES (to save the dictionary) and NO (to
purge the dictionary after the compilation).

Recommended action:

Please correct the SAVE-DDL-DICTIONARY parameter in the ACF.
```

```
** Error 76 ** The USER-DDL-FILE parameter is invalid.

Probable cause:

The following values are valid: NOT-USED (do not source a DDL file
definition) or a valid filename.
The file <filename> does not exist.

Recommended action:

Please correct the USER-DDL-FILE parameter in the ACF.
```

```
** Error 100 ** An error occurred when trying to process the CCF.

Probable cause:

<error description>

Recommended action:

Please correct the error regarding the CCF.
```

```
** Error 101 ** Invalid key word in the CCF.

Probable cause:

EMS FastStart was expecting the <expected keyword> and the current key
word is <current keyword>.

Recommended action:

Please correct the key word in the CCF file.
```

```
** Error 102 ** Missing Parameter or Invalid key word.

Probable cause:

EMS FastStart detected that no parameters were supplied for this keyword
or the key word expected was <expected keyword>.

Recommended action:

Please correct the CCF by specifying a valid key word or a valid
parameter.
```

```
** Error 103 ** Invalid CCF or missing key word.

Probable cause:

EMS FastStart was expecting the <expected keyword>.

Recommended action:

Please correct the key word in the CCF file.
```

```
** Warning 104 ** The COBOL85 compiler will not be used.

Probable cause:

EMS FastStart detected that the COBOL85-LOCATION parameter was NOT-USED.

Recommended action:

There is no action required if you do not want to use the COBOL85
compiler.
```

```
** Error 105 ** Invalid CCF-VERSION parameter.

Probable cause:

The CCF-VERSION is invalid; the only version currently supported is A00.

Recommended action:

Please correct the CCF-VERSION parameter in the ACF file.
```

```
** Error 106 ** Invalid <compiler-name-LOCATION> parameter.

Probable cause:

The <compiler-name-LOCATION> is invalid because of the following reason:
The file: <compiler-filename> does not exist.

Recommended action:

Please correct the <compiler-name-LOCATION> parameter in the CCF file.
```

```
** Error 107 ** Invalid <compiler-name-CPU> parameter.

Probable cause:

The <compiler-name-CPU> parameter should be between 0 and 15 or the CPU is
not currently available.

Recommended action:

Please correct the <compiler-name-CPU> parameter in the CCF file.
```

```
** Error 108 ** Invalid <compiler-name-PRIORITY> parameter.

Probable cause:

The <compiler-name-PRIORITY> parameter should be between 1 and 199.

Recommended action:

Please correct the <compiler-name-PRIORITY> parameter in the CCF file.
```

```
** Error 109 ** Invalid <compiler-name-WORK-VOLUME> parameter.

Probable cause:

The <compiler-name-WORK-VOLUME> parameter should point to a valid volume.

Recommended action:

Please correct the <compiler-name-WORK-VOLUME> parameter in the CCF file.
```

```
** Error 110 ** Invalid SPOOLER-COLLECTOR parameter.

Probable cause:

The SPOOLER-COLLECTOR should be a valid process name that exists.

Recommended action:

Please correct the SPOOLER-COLLECTOR parameter in the CCF file.
```

```
** Warning 150 ** Compiler terminated with warnings diagnostic.

Probable cause:

The compilation terminated but warnings were detected by <compiler-name>.

Recommended action:

Please verify the listing file for detailed information.  You may have
compiler warnings during the DDL compilation due to the conversion of data
types from SPI definitions (example, ZSPI-DDL-FNAME) to COBOL85.  You can
ignore these warnings.
```

```
** Error 151 ** Compiler abended.

Probable cause:

The compilation was not successful and terminated abnormally.

Recommended action:

EMS FastStart cannot continue operation because of this error, please
verify the listing file for detailed information.
```

```
** Error 152 ** Compiler failed because of CPU failure.

Probable cause:

The compilation was not successful and terminated abnormally.

Recommended action:

EMS FastStart cannot continue operation because of this error, please
verify the listing file for detailed information.
```

```
** Error 153 ** Compiler failed because of NETWORK problem.

Probable cause:

The compilation was not successful and terminated abnormally.

Recommended action:

EMS FastStart cannot continue operation because of this error, please
verify the listing file for detailed information.
```

```
** Error 154 ** Compiler failed; unknown error.

Probable cause:

The compilation was not successful and terminated abnormally.

Recommended action:

EMS FastStart cannot continue operation because of this error, please
verify the listing file for detailed information.
```

# B EGEN Messages

Each time you call one of the procedures implemented in the EGEN module you should verify the value of the return code to determine whether or not an error occurred during event generation. This appendix describes the return codes defined between EGEN and a user application (implemented by EGEN as a set of TAL procedures).

A return code for each procedure implemented in the EGEN module informs the calling application of the status of the operation requested. If the return code is equal to zero (0), the operation was successful. If the return code is not equal to zero (0), it indicates a warning or an error. Also, a special field called egen-error (integer) in the EGEN record structure is used to give you more detailed information about the error. The contents of this field depends on the return code of the EGEN procedures.

The following is a list of procedures for which error, warning, and advisory return codes are returned to the calling application:

- Initialize^egen^record (Initialize^egen^record Return Codes)
- Open^egen^collector (Open^egen^collector Return Codes)
- Close^egen^collector (Close^egen^collector Return Codes)
- Complete^egen^operation (Complete^egen^operation Return Codes)
- Get^egen^event^text^define (Get^egen^event^text^define Return Codes)*
- Initialize^event^buffer (Initialize^event^buffer Return Codes)*
- Write^event^buffer (Write^event^buffer Return Codes)*
- EGEN (EGEN Return Codes)

*These procedures are not called directly by the user application but are called from within EGEN.

## Initialize^egen^record Return Codes

0: Egen-initialize-record-ok

Returned when the egen-record initializes properly. The value of egen-error has no meaning for this message.

1: Egen-initialize-missing-param

Returned if this procedure is called without the egen-record parameters. The value of egen-error has no meaning for this message.

## Open^egen^collector Return Codes

0: Egen-open-collector-ok

Returned when the collector is opened. The value of egen-error has no meaning for this message.

10: Egen-open-missing-param

Returned if this procedure is called without the collector^file^number and the error^detail parameters. The value of egen-error has no meaning for this message.

11: Egen-open-invalid-sync-depth

Returned if this procedure is called with an invalid sync^depth parameter. The value of egen-error has no meaning for this message. The value must be 0 through 15.

12: Egen-open-collector-error

Returned if this procedure gets an error when trying to open the collector. The value of egen-error will contain the GUARDIAN 90 file system error returned by the FILEINFO procedure.

13: Egen-open-collector-warning

Returned if this procedure gets a warning when trying to open the collector. The value of egen-error will contain the GUARDIAN 90 file system error returned by the FILEINFO procedure.

## Close^egen^collector Return Codes

0: Egen-collector-closed-ok

Returned when the collector is closed properly. The value of egen-error has no meaning for this message.

20: Egen-collector-missing-param

Returned if this procedure is called without the collector^file^number parameter. The value of egen-error has no meaning for this message.

21: Egen-collector-already-closed

Returned if the collector is already closed. The value of egen-error has no meaning for this message.

## Complete^egen^operation Return Codes

0: Egen-complete-operation-ok

Returned when the last outstanding operation is terminated properly. The value of egen-error has no meaning for this message.

30: Egen-complete-missing-param

Returned when one of the collector^file^number or tag or error^detail parameters is missing. The value of egen-error has no meaning for this message.

31: Egen-complete-operation-error

Returned when the last outstanding operation is terminated with an error or warning. The value of egen-error will contain the GUARDIAN 90 file system error returned by the FILEINFO procedure.

## Get^egen^event^text^define Return Codes

0: Egen-get-text-define-ok

Returned when the procedure processes the =_EGEN_ADD_EVENT_TEXT Define properly. The value of egen-error has no meaning for this message.

40: Egen-get-text-definemode-error

Returned when the procedure detects an error with the DEFINEMODE procedure. The value of egen-error will contain the define error returned by DEFINEMODE.

41: Egen-get-text-defineinfo-error

Returned when the procedure detects an error with the DEFINEINFO procedure. The value of egen-error will contain the define error returned by DEFINEINFO.

## Initialize^event^buffer Return Codes

0: Egen-initialize-event-ok

Returned when the event-buffer is initialized properly. The value of egen-error has no meaning for this message.

50: Egen-initialize-type-error

Returned when the event-type field of egen-record does not contain a valid event type (Informative-event, Action-attention-event, Action-completion-event or Critical-event). The value of egen-error has no meaning for this message.

51: Egen-initialize-event-number

Returned when the event-number field of egen-record does not contain a valid event-number. The value of egen-error has no meaning for this message.

52: Egen-initialize-emsinit-error

Returned when the EMSINIT procedure detects an error when trying to initialize the event-buffer. The value of egen-error will contain the SPI error returned from EMSINIT.

53: Egen-initialize-subject-error

Returned when the subject-name field of egen-record does not contain a valid field-name. The value of egen-error has no meaning for this message.

54: Egen-initialize-flags-error

Returned when the EMSADDTOKENS procedure detects an error when trying to add variable data fields (tokens) ZEMS^TKN^EMPHASIS or ZEMS^TKN^SUPPRESS^DISPLAY to the event-buffer. The value of egen-error will contain the SPI error returned from EMSADDTOKENS.

55: Egen-initialize-action-id

Returned when the event-type is an action-attention-event or an action-completion-event and the action ADD ID is not initialized properly. The value of egen-error has no meaning for this message.

56: Egen-initialize-action-error

Returned when the EMSADDTOKENS procedure detects an error when trying to add action variable data fields (tokens) ZEMS^TKN^ACTION^NEEDED, ZEMS^TKN^ACTION^ID to the event-buffer. The value of egen-error will contain the SPI error returned from EMSADDTOKENS.

57: Egen-initialize-text-error

Returned when the EMSADDTOKENS procedure detects an error when trying to add the text variable data field (token) ZEMS^TKN^TEXT to the event-buffer. The value of egen-error will contain the SPI error returned from EMSADDTOKENS.

58: Egen-initialize-tokens-error

Returned when the EMSADDTOKENS procedure detects an error when trying to add the user defined variable data fields (tokens) to the event-buffer. The value of egen-error will contain the SPI error returned from EMSADDTOKENS.

59: Egen-initialize-ssgettkn-error

Returned when the SSGETTKN procedure detects an error when trying to get the length of the event-buffer. The value of egen-error will contain the SPI error returned from EMSADDTOKENS.

## Write^event^buffer Return Codes

0: Egen-write-event-ok

Returned when the event is written properly. The value of egen-error has no meaning for this message.

60: Egen-write-event-warning

Returned when a warning is detected after the write of the event-buffer. The value of egen-error will contain the GUARDIAN 90 file system error returned by the FILEINFO procedure.

61: Egen-write-event-error

Returned when an error is detected after the write of the event-buffer. The value of egen-error will contain the GUARDIAN 90 file system error returned by the FILEINFO procedure.

## EGEN Return Codes

0: Egen-generate-event-ok

Returned when the event is generated properly. The value of egen-error has no meaning for this message.

70: Egen-missing-parameter-error

Returned if this procedure is called without a valid combination of parameters. The value of egen-error has no meaning for this message.

71: Egen-record-not-initialized

Returned if this procedure is called but the egen-record is not initialized properly. This can happen if you call EGEN without first calling the Initialize^egen^record procedure. The value of egen-error has no meaning for this message.

# C COBOL85 Program Example

This appendix contains the COBOL85 source code file SATMCOB that is located on the EMS FastStart subvolume after installation:

```
?HEADING "SATMCOB : COBOL PROGRAM EXAMPLE USING EGEN FOR AN ATM APPLICATION"
?SYMBOLS, INSPECT, SAVEABEND, NOMAP, NOICODE, ERRORS 0
*-------------------------------------------------------------------
*
*  File Type:                  COBOL85 Source File for the ATM example
*  Source File Name:           SATMCOB
*  Program File Produced:      OATMCOB
*  Generation Time:            October 31, 1989 09:42
*  Language Compiler Required: COBOL85
*  Compiler Version Required:  C20
*
*  File Description:  This program shows an example of how to use the
*  EGEN module within a COBOL85 program.  This program will use EGEN in
*  mode 2, which is the mode that we recommend for the COBOL85 program.
*-------------------------------------------------------------------
*  How to use this program example: To use this program example you will
*  have to generate a set of EMS FastStart files by using the SATMACF
*  file provided with the EMS FastStart product.  After a successful
*  generation, please modify the following statement to point to the
*  volume and subvolume where your EMS FastStart generated files are.
*
*  1-?SEARCH <user-subvolume>.<application-prefix>EGEN
*
*  2-COPY egen-record OF "<user-subvolume>.<application-prefix>COB".
*    COPY atm-ssid    OF "<user-subvolume>.<application-prefix>COB".
*    COPY egen-interface-definitions OF
*                         "<user-subvolume>.<application-prefix>COB".
*
*  3-COPY atm-event-numbers OF "<user-subvolume>.<application-prefix>COB".
*    COPY atm-action-id     OF "<user-subvolume>.<application-prefix>COB".
*    COPY atm-constant-values OF "<user-subvolume>.<application-prefix>COB".
*-------------------------------------------------------------------
* The ?SEARCH directive tells the compiler to look in the proper object
* file to find the EGEN module and resolve the external references.
*-------------------------------------------------------------------

?SEARCH atm1egen

?HEADING "SATMCOB VARIABLES DECLARATIONS"
/
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SATMCOB.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01 return-code             PIC S9(4).
 01 file-number             PIC 9(4) COMP.
 01 user-tag                PIC 9(8) COMP.
 01 event-buffer-used       PIC 9(4) COMP.
 01 sync-depth              PIC 9(4) COMP.
 01 read-count              PIC 9(4) COMP.
 01 error-detail            PIC 9(4) COMP.


?HEADING "COPY LIBRARIES STATEMENT"
/
*-------------------------------------------------------------------
*
```

```
*      egen-record & atm-ssid:
*           COPY libraries specific to each ACF.
*
*      egen-interface-definitions:
*           COPY library that is used to interface between user
*           application and Egen.
*
*      atm-event-numbers, atm-action-id & atm-constant-values:
*           COPY libraries that are defined by the application
*           programmer and sourced by the compilation of the
*           main DDL.
*-----------------------------------------------------------------------

 COPY egen-record                     OF "atm1cob".
 COPY atm-ssid                        OF "atm1cob".
 COPY egen-interface-definitions      OF "atm1cob".

 COPY atm-event-numbers               OF "atm1cob".
 COPY atm-action-id                   OF "atm1cob".
 COPY atm-constant-values             OF "atm1cob".

?HEADING "START-OF-PROGRAM SECTION"
/
 PROCEDURE DIVISION.

 100-main.

  PERFORM 205-open-collector.
  PERFORM 210-atm-up.
  PERFORM 215-atm-back-online.
  PERFORM 220-atm-acct-insuf-funds.
  PERFORM 230-atm-low-on-cash.
  PERFORM 240-atm-down.
  PERFORM 250-atm-security.
  PERFORM 260-atm-hardware.
  PERFORM 365-close-collector.

  STOP RUN.


?HEADING "205-OPEN-COLLECTOR SECTION"
/
 205-open-collector.

  MOVE ZERO TO file-number.
  ENTER TAL "Open^egen^collector" USING file-number, omitted, error-detail
                                  GIVING return-code.
  IF return-code IS NOT EQUAL TO ZERO
        PERFORM 400-validate-return-code.


?HEADING "210-ATM-UP SECTION"
/
 210-atm-up.
*-----------------------------------------------------------------------
* EVENT # 1: ATM IS UP:
*
* The fields within the egen-record which are used for event # 1:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*-----------------------------------------------------------------------

 PERFORM 300-initialize-egen-record.
```

```
  MOVE INFORMATIVE-EVENT          TO event-type   OF egen-record.
  MOVE ATM-EVT-UP                 TO event-number OF egen-record.
  MOVE "SFMAIN01"                 TO atm-name     OF egen-record.
  MOVE "atm-name"                 TO subject-field-name OF egen-record.
  MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

  ENTER TAL "Egen" USING egen-record, file-number
                                  GIVING return-code.
  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.


?HEADING "215-ATM-BACK-ONLINE SECTION"
/
 215-atm-back-online.
*---------------------------------------------------------------------------
* EVENT # 3: ATM IS BACK ONLINE.
*
* The fields within the egen-record which are used for event # 3:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*---------------------------------------------------------------------------

  PERFORM 300-initialize-egen-record.

  MOVE ACTION-COMPLETION-EVENT    TO event-type       OF egen-record.
  MOVE ATM-ACTION-ID-1            TO action-id        OF egen-record.
  MOVE ATM-EVT-BACK-ONLINE        TO event-number     OF egen-record.
  MOVE ATM-VAL-TRUE               TO suppress-display OF egen-record.
  MOVE "SFMAIN02"                 TO atm-name         OF egen-record.
  MOVE "atm-name"                 TO subject-field-name OF egen-record.
  MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

  ENTER TAL "Egen" USING egen-record, file-number
                                  GIVING return-code.
  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.


?HEADING "220-ATM-ACCT-INSUF-FUNDS SECTION"
/
 220-atm-acct-insuf-funds.
*---------------------------------------------------------------------------
* EVENT # 2: ATM INSUFFICIENT FUNDS IN USER ACCOUNT.
*
* The fields within the egen-record which are used for event # 2:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*      atm-account-num    NATIVE-4.
*---------------------------------------------------------------------------

  PERFORM 300-initialize-egen-record.

  MOVE INFORMATIVE-EVENT          TO event-type   OF egen-record.
  MOVE ATM-EVT-ACCT-INSUF-FUNDS   TO event-number OF egen-record.
  MOVE "OAKWEST1"                 TO atm-name     OF egen-record.
  MOVE "atm-name"                 TO subject-field-name OF egen-record.
  MOVE "245 Oak St., Oakland"     TO atm-location OF egen-record.
  MOVE 34503933                   TO atm-account-num OF egen-record.
```

```
  ENTER TAL "Egen" USING egen-record, file-number
                                 GIVING return-code.
  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.


?HEADING "230-ATM-LOW-ON-CASH SECTION"
/
 230-atm-low-on-cash.
*---------------------------------------------------------------------------
* EVENT # 100: ATM IS LOW ON CASH.
*
* The fields within the egen-record which are used for event # 100:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*---------------------------------------------------------------------------

  PERFORM 300-initialize-egen-record.

  MOVE ACTION-ATTENTION-EVENT     TO event-type   OF egen-record.
  MOVE ATM-ACTION-ID-1            TO action-id    OF egen-record.
  MOVE ATM-EVT-LOW-ON-CASH        TO event-number OF egen-record.
  MOVE "SFMAIN02"                 TO atm-name     OF egen-record.
  MOVE "atm-name"                 TO subject-field-name OF egen-record.
  MOVE "245 A St., San Francisco" TO atm-location OF egen-record.

  ENTER TAL "Egen" USING egen-record, file-number
                                 GIVING return-code.
  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.


?HEADING "240-ATM-DOWN SECTION"
/
 240-atm-down.
*---------------------------------------------------------------------------
* EVENT # 200: ATM IS DOWN.
*
* the fields within the egen-record which are used for event # 200:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*---------------------------------------------------------------------------
  PERFORM 300-initialize-egen-record.
  MOVE CRITICAL-EVENT               TO event-type   OF egen-record.
  MOVE ATM-EVT-DOWN                 TO event-number OF egen-record.
  MOVE "LACENT99"                   TO atm-name     OF egen-record.
  MOVE "atm-name"                   TO subject-field-name OF egen-record.
  MOVE "320 Main St. Los Angeles"   TO atm-location OF egen-record.


  ENTER TAL "Egen" USING egen-record, file-number
                                 GIVING return-code.
  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.

?HEADING "250-ATM-SECURITY SECTION"
/
 250-atm-security.
*---------------------------------------------------------------------------
```

```
* EVENT # 201 ATM SECURITY BREACH.
*
* The fields within the egen-record which are used for event # 201:
*
*      event-type         PIC S9(4).
*      event-number       NATIVE-2.
*      atm-name           PIC X(8).
*      atm-location       PIC X(24).
*      atm-account-num    NATIVE-4.
*      atm-retry-limit    NATIVE-2.
*-------------------------------------------------------------------------

  PERFORM 300-initialize-egen-record.

  MOVE CRITICAL-EVENT             TO event-type   OF egen-record.
  MOVE ATM-EVT-SECURITY-BREACH    TO event-number OF egen-record.
  MOVE "SACTO02"                  TO atm-name     OF egen-record.
  MOVE "atm-name"                 TO subject-field-name OF egen-record.
  MOVE "230 State St. Sacramento" TO atm-location OF egen-record.
  MOVE 23457320                   TO atm-account-num OF egen-record.
  MOVE 3                          TO atm-retry-limit OF egen-record.

  ENTER TAL "Egen" USING egen-record, file-number
                                  GIVING return-code.

 IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.


?HEADING "260-ATM-HARDWARE SECTION"
/
 260-atm-hardware.
*-------------------------------------------------------------------------
* EVENT # 202: ATM-HARDWARE FAILURE EVENT.
*
* The fields within the egen-record which are used for event # 202:
*
*      event-type                 PIC S9(4).
*      event-number               NATIVE-2.
*      atm-name                   PIC X(8).
*      atm-location               PIC X(24).
*      atm-hardware-component     PIC X(24).
*      atm-hardware-subcomponent  PIC X(24).
*      atm-serial-number          NATIVE-4.
*      atm-sense-status           NATIVE-4.
*-------------------------------------------------------------------------

  PERFORM 300-initialize-egen-record.

  MOVE CRITICAL-EVENT             TO event-type   OF egen-record.
  MOVE ATM-EVT-HW-FAILURE         TO event-number OF egen-record.
  MOVE "LACENT99"                 TO atm-name     OF egen-record.
  MOVE "atm-name"                 TO subject-field-name OF egen-record.
  MOVE "125 8th Ave. Los Angeles" TO atm-location OF egen-record.
  MOVE "Cash Dispenser"           TO atm-hw-component OF egen-record.
  MOVE "321561ac"                 TO atm-hw-subcomponent OF egen-record.
  MOVE  231234093                 TO atm-serial-number OF egen-record.
  MOVE  0101110101                TO atm-sense-status OF egen-record.

  ENTER TAL "Egen" USING egen-record, file-number
                                  GIVING return-code.

  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.
```

```
?HEADING "300-INITIALIZE-EGEN-RECORD SECTION"
/
 300-initialize-egen-record.
*---------------------------------------------------------------------
* This section will initialize the egen-record by calling the
* Initialize^egen^record procedure.  This procedure is included in the
* Egen object file and sourced with the ?SEARCH compiler directive.
*
* Initialize^egen^record will move spaces to character fields and
* high values to all fields in egen-record.  The following is a list
* of high values used to initialize non-character fields.
*
*           LITERAL EMSFS^DEFAULT^INT  = 32767;
*           LITERAL EMSFS^DEFAULT^INT2 = 2147483647D;
*           LITERAL EMSFS^DEFAULT^INT4 = 92233720368545775807F;
*           LITERAL EMSFS^DEFAULT^TRANSID = 9223372036854775807;
*           LITERAL EMSFS^DEFAULT^TIMESTAMP = 9223372036854775807;
*           LITERAL EMSFS^DEFAULT^UINT = %177777;
*           LITERAL EMSFS^DEFAULT^ENUM = 32767;
*
* After the egen-record is initialized, we will move the application
* ssid to the specific fields in this record.
*---------------------------------------------------------------------

  ENTER TAL "Initialize^egen^record" USING egen-record GIVING return-code.

  IF return-code IS NOT EQUAL TO ZERO
     PERFORM 400-validate-return-code.

  MOVE ZERO            TO return-code.
  MOVE ATM-VAL-OWNER   TO ssid-owner            OF egen-record.
  MOVE ATM-SSN-NUMBER  TO ssid-subsystem-number OF egen-record.
  MOVE ATM-VAL-VERSION TO ssid-version          OF egen-record.


?HEADING "365-CLOSE-COLLECTOR SECTION"
/
 365-close-collector.

  ENTER TAL "Close^egen^collector"  USING file-number
                                     GIVING return-code.

  IF return-code NOT = ZERO
       PERFORM 400-validate-return-code.


?HEADING "400-VALIDATE-RETURN-CODE SECTION"
/
 400-validate-return-code.

  IF return-code = EGEN-GENERATE-EVENT-OK
     DISPLAY "Egen generated the event message successfully"
  ELSE
  IF return-code = EGEN-INITIALIZE-MISSING-PARAM
     DISPLAY "Egen Error: ** Egen-initialize-missing-param, error #1 **"
     DISPLAY "Error Detail: " WITH NO ADVANCING
     DISPLAY egen-error OF egen-record
  ELSE
  IF return-code = EGEN-OPEN-MISSING-PARAM
     DISPLAY "Egen Error: ** Egen-open-missing-param, error #10 **"
     DISPLAY "Error Detail: " WITH NO ADVANCING
     DISPLAY egen-error OF egen-record
  ELSE
  IF return-code = EGEN-OPEN-INVALID-SYNC-DEPTH
     DISPLAY "Egen Error: ** Egen-open-invalid-sync-depth, error #11 **"
```

```
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-OPEN-COLLECTOR-ERROR
           DISPLAY "Egen Error: ** Egen-open-collector-error, error #12 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-OPEN-COLLECTOR-WARNING
           DISPLAY "Egen Error: ** Egen-open-collector-warning, error #13 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-COLLECTOR-MISSING-PARAM
           DISPLAY "Egen Error: ** Egen-collector-missing-param, error #20 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-COLLECTOR-ALREADY-CLOSED
           DISPLAY "Egen Error: ** Egen-collector-already-closed, error #21 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-COMPLETE-MISSING-PARAM
           DISPLAY "Egen Error: ** Egen-complete-missing-param, error #30 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-COMPLETE-OPERATION-ERROR
           DISPLAY "Egen Error: ** Egen-complete-operation-error, error #31 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-GET-TEXT-DEFINEMODE-ERROR
           DISPLAY "Egen Error: ** Egen-get-text-definemode-error, error #40 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-GET-TEXT-DEFINEINFO-ERROR
           DISPLAY "Egen Error: ** Egen-get-text-defineinfo-error, error #41 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-INITIALIZE-TYPE-ERROR
           DISPLAY "Egen Error: ** Egen-initialize-type-error, error #50 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-INITIALIZE-EVENT-NUMBER
           DISPLAY "Egen Error: ** Egen-initialize-event-number, error #51 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-INITIALIZE-EMSINIT-ERROR
           DISPLAY "Egen Error: ** Egen-initialize-emsinit-error, error #52 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-INITIALIZE-SUBJECT-ERROR
           DISPLAY "Egen Error: ** Egen-initialize-subject-error, error #53 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record
        ELSE
        IF return-code = EGEN-INITIALIZE-FLAGS-ERROR
           DISPLAY "Egen Error: ** Egen-initialize-flags-error, error #54 **"
           DISPLAY "Error Detail: " WITH NO ADVANCING
```

```
           DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-INITIALIZE-ACTION-ID
            DISPLAY "Egen Error: ** Egen-initialize-action-id, error #55 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-INITIALIZE-ACTION-ERROR
            DISPLAY "Egen Error: ** Egen-initialize-action-error, error #56 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-INITIALIZE-TEXT-ERROR
            DISPLAY "Egen Error: ** Egen-initialize-text-error, error #57 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-INITIALIZE-TOKENS-ERROR
            DISPLAY "Egen Error: ** Egen-initialize-tokens-error, error #58 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-INITIALIZE-SSGETTKN-ERROR
            DISPLAY "Egen Error: ** Egen-initialize-ssgettkn-error, error #59 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-WRITE-EVENT-WARNING
            DISPLAY "Egen Error: ** Egen-write-event-warning, error #60 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-WRITE-EVENT-ERROR
            DISPLAY "Egen Error: ** Egen-write-event-error, error #61 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-MISSING-PARAMETER-ERROR
            DISPLAY "Egen Error: ** Egen-missing-parameter-error, error #70 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF return-code = EGEN-RECORD-NOT-INITIALIZED
            DISPLAY "Egen Error: ** Egen-record-not-intialized, error #71 **"
            DISPLAY "Error Detail: " WITH NO ADVANCING
            DISPLAY egen-error OF egen-record
         ELSE
         IF ( return-code < EGEN-GENERATE-EVENT-OK )      OR
            ( return-code > EGEN-RECORD-NOT-INITIALIZED )
           DISPLAY "Egen Error: ** Invalid return code, error # " WITH NO ADVANCING
           DISPLAY return-code
           DISPLAY "Error Detail: " WITH NO ADVANCING
           DISPLAY egen-error OF egen-record.

       END-PROGRAM.
```

# D

# DDL, Copy Libraries and Templates Example

This appendix contains the ATM DDL source file (ATM Example, EVENT DEFINITION SOURCE FILE: SATMDDL), the ATM COBOL85 copylib (ATM Example, COBOL85 COPYLIB: ATM1COB), and the ATM Template Source file (ATM Example, DSM Templates Services Source File: SATMTMPL).

The first source file for ATM DDL event is SATMDDL and is provided on the distribution subvolume. You can modify the ATM DDL event source as necessary to meet the needs of your application. This file contains the description of all the events needed by the ATM example. It will be sourced during the compilation of the main DDL. You specify the location of this file with the USER-DDL-LOCATION key word parameter in the ACF.

The second source file for ATM COBOL85 copylib is ATM1COB, which will be located on your USER-SUBVOL subvolume after the generation process of EMS FastStart. This file contains all the definitions about EGEN and at the end the definitions for events from the file SATMDDL. The definitions in this file will be needed to compile a template file.

The last source file is the template source file for the ATM example, SATMTMPL which can be used with the DSM Templates Services to create a template file for the ATM application. Please note that to compile this file and generate the object template file, you need to keep the dictionary on the USER-SUBVOL by specifying the parameter of the SAVE-DDL-DICTIONARY to YES.

# ATM Example, EVENT DEFINITION SOURCE FILE: SATMDDL

This is a new file.

```
?PAGE "EMS FastStart - T9263C20 - (17MAR91) - Atm DDL schema source file"
*------------------------------------------------------------------------
*
*  EMS FastStart - T9263C20 - (17MAR91)
*
*  File Type:                     DDL Event Definition  Source Schema
*
*  Source File Name:              SATMDDL
*
*  Source Library File Produced:  Sourced during the compilation of
*                                 the main DDL.
*
*  Generation Time:               December 17, 1990
*
*  Language Compiler Required:    Data Definition Language (DDL)
*
*  Compiler Version Required:     C20
*
*
```

```
*  File Description: This DDL source schema file is an example of DDL
*  definitions which are used by the ATM example provided with EMS
*  FastStart.  These definitions will ease the documentation and the
*  maintenance of an EGEN based application.
*
*-----------------------------------------------------------------------

?SETSECTION Atm-event-numbers
*-----------------------------------------------------------------------
*
* Each event message generated by an application will be identified
* by a unique number called the event message number. This section
* presents an example of a typical implementation.
*
* Event Numbers                 Usage Description
*
*    0 -  99                    Informative Events
*  100 - 199                    Action Events
*  200 - 499                    Critical Events
*  500 - 999                    Reserved for future enhancement
*
*-----------------------------------------------------------------------

*-----------------------------------------------------------------------
*
* INFORMATIVE EVENTS DESCRIPTION:
*
* These event message numbers describe the current set of INFORMATIVE
* EVENTS supported by this application.  For each event generated by a
* Atm application, the programmer should select one of the following
* event message numbers and move it into the EVENT-NUMBER field of
* EGEN-RECORD.
*
*-----------------------------------------------------------------------

CONSTANT Atm-evt-up                         VALUE IS 1.
CONSTANT Atm-evt-acct-insuf-funds           VALUE IS 2.
CONSTANT Atm-evt-back-online                VALUE IS 3.

*-----------------------------------------------------------------------
*
* ACTION EVENTS DESCRIPTION:
*
* These event message numbers describe the current set of ACTION
* EVENTS supported by this application.  For each event generated by a
* Atm application, the programmer should select one of the following
* event message numbers and move it into the EVENT-NUMBER field of
* EGEN-RECORD.
*
*-----------------------------------------------------------------------

CONSTANT Atm-evt-low-on-cash                VALUE IS 100.

*-----------------------------------------------------------------------
*
* CRITICAL EVENTS DESCRIPTION:
*
* These event message numbers describe the current set of CRITICAL
* EVENTS supported by this application.  For each event generated by a
* Atm application, the programmer should select one of the following
* event message numbers and move it into the EVENT-NUMBER field of
* EGEN-RECORD.
*
*-----------------------------------------------------------------------

CONSTANT Atm-evt-down                       VALUE IS 200.
```

```
CONSTANT Atm-evt-security-breach            VALUE IS 201.
CONSTANT Atm-evt-hw-failure                 VALUE IS 202.
?SETSECTION Atm-action-id
*---------------------------------------------------------------------
*
* ACTION EVENT IDENTIFIER DESCRIPTION:
*
* There is an action identifier associated with each action event
* message.  This Action ID is used by VIEWPOINT to match an
* action-attention event message with the corresponding
* action-completion event message.
*
*---------------------------------------------------------------------

CONSTANT Atm-action-id-1                    VALUE IS 1.
CONSTANT Atm-action-id-2                    VALUE IS 2.
CONSTANT Atm-action-id-3                    VALUE IS 3.
CONSTANT Atm-action-id-4                    VALUE IS 4.
CONSTANT Atm-action-id-5                    VALUE IS 5.

?SETSECTION Atm-constant-values
*---------------------------------------------------------------------
*
* CONSTANT VALUE IDENTIFIER DESCRIPTION:
*
* These are constants that can be used globally in the ATM subsystem.
*
*---------------------------------------------------------------------
CONSTANT Atm-val-false                      VALUE IS 0.
CONSTANT Atm-val-true                       VALUE IS 1.
```

# ATM Example, COBOL85 COPYLIB: ATM1COB

```
*---------------------------------------------------------------------
*
*  EMS FastStart - T9263C20 - (17MAR91)
*
*  DDL Source Library File, Language Dependant
*
*  Produced by the DDL compilation of ATM1DDLS
*
*  Generation Time:   January 18, 1991 13:58:07
*
*---------------------------------------------------------------------
?SECTION ZSPI-DDL-CHAR254,TANDEM
*---------------------------------------------------------------------
* Source EXTRADDL source schema file for user defined SPI data types.
*---------------------------------------------------------------------
*
*  EMS FastStart - T9263C20 - (17MAR91)
*
*  File Type:                    DDL Source Schema
*
*  Source File Name:             Extraddl
*
*  Generation Time:              July 7, 1988
*
*  Language Compiler Required:   Data Definition Language (DDL)
*
*  Compiler Version Required:    C20
*
*  Source Library File Produced:   None, see below.
*
*
```

```
*  File Description: This DDL source schema file is an example of DDL
*  definitions which may me added to the base ZSPIDDL definitions
*  provided by Tandem.  These definitions can then be used by
*  EMS FastStart and EGEN to create tokens of specific types.
*
*  Modifications Summary:                          Date of Modification
*
*  1- Added the Zspi-ddl-char254 token.  Used by       21 October, 1988
*     EGEN to generate an event message with a
*     ZEMS-TKN-TEXT of up to 254 bytes.
*     N.B. 254 is the maximum bytes length for a
*          fixed token code.
*
*-----------------------------------------------------------------------
  01 ZSPI-DDL-CHAR254.
    02 Z-C                          PIC X(254).
    02 Z-S REDEFINES Z-C.
      03 Z-I                        NATIVE-2
                                    OCCURS 127 TIMES.
    02 Z-B REDEFINES Z-C            PIC X
                                    OCCURS 254 TIMES.
?SECTION ZSPI-TYP-CHAR254,TANDEM
    01 ZSPI-TYP-CHAR254 NATIVE-2 VALUE IS 510.
?Section ATM-SSID,Tandem
*-----------------------------------------------------------------------
*
* ATM SSID is defined here and will be passed to the EGEN
* procedure to identify the owner of the event.  The SSID definition
* will also be used by EMF to compile the FILTER example.
*
*   Description          Value
*   -----------          -----
*
*  ATM-VAL-OWNER:         CUSTOMER
*
*  ATM-SSN-NUMBER:        1
*
*  ATM-VAL-VERSION:       J00
*
*-----------------------------------------------------------------------
*
*
*
    01 ATM-VAL-OWNER PIC X(8), VALUE IS "CUSTOMER".
    01 ATM-SSN-NUMBER NATIVE-2 VALUE IS 1.
    01 ATM-VAL-VERSION NATIVE-2 VALUE IS 18944.
*
*
*
  01 ATM-VAL-SSID.
    02 Z-FILLER                     PIC X(8)
                                    VALUE "CUSTOMER".
    02 Z-OWNER REDEFINES Z-FILLER.
      03 Z-C                        PIC X(8).
      03 Z-S REDEFINES Z-C.
        04 Z-I                      NATIVE-2
                                    OCCURS 4 TIMES.
      03 Z-B REDEFINES Z-C          PIC X
                                    OCCURS 8 TIMES.
    02 Z-NUMBER                     NATIVE-2
                                    VALUE 1.
    02 Z-VERSION                    NATIVE-2
                                    VALUE 18944.
?Section EGEN-RECORD,Tandem
  01 EGEN-RECORD.
    02 ACF-VERSION                  NATIVE-2.
```

```
02 SSID-OWNER.
  03 Z-C                          PIC X(8).
  03 Z-S REDEFINES Z-C.
    04 Z-I                        NATIVE-2
                                   OCCURS 4 TIMES.
  03 Z-B REDEFINES Z-C            PIC X
                                   OCCURS 8 TIMES.
02 SSID-SUBSYSTEM-NUMBER          NATIVE-2.
02 SSID-VERSION                   NATIVE-2.
02 EGEN-ERROR                     NATIVE-2.
02 EVENT-TYPE                     PIC S9(4)      COMP.
02 EVENT-NUMBER                   NATIVE-2.
02 ACTION-ID                      NATIVE-2.
02 SUPPRESS-DISPLAY               PIC X(2).
02 SUBSYSTEM-MANAGER.
  03 Z-SYSNAME.
    04 Z-C                        PIC X(8).
    04 Z-S REDEFINES Z-C.
      05 Z-I                      NATIVE-2
                                   OCCURS 4 TIMES.
    04 Z-B REDEFINES Z-C          PIC X
                                   OCCURS 8 TIMES.
  03 Z-LOCALNAME.
    04 Z-DISC.
      05 Z-VOLUME.
        06 Z-C                    PIC X(8).
        06 Z-S REDEFINES Z-C.
          07 Z-I                  NATIVE-2
                                   OCCURS 4 TIMES.
        06 Z-B REDEFINES Z-C      PIC X
                                   OCCURS 8 TIMES.
      05 Z-SUBVOLUME.
        06 Z-C                    PIC X(8).
        06 Z-S REDEFINES Z-C.
          07 Z-I                  NATIVE-2
                                   OCCURS 4 TIMES.
        06 Z-B REDEFINES Z-C      PIC X
                                   OCCURS 8 TIMES.
      05 Z-FILENAME.
        06 Z-C                    PIC X(8).
        06 Z-S REDEFINES Z-C.
          07 Z-I                  NATIVE-2
                                   OCCURS 4 TIMES.
        06 Z-B REDEFINES Z-C      PIC X
                                   OCCURS 8 TIMES.
    04 Z-PROCESS REDEFINES Z-DISC.
      05 Z-CRTPID.
        06 Z-PROCNAME.
          07 Z-C                  PIC X(6).
          07 Z-S REDEFINES Z-C.
            08 Z-I                NATIVE-2
                                   OCCURS 3 TIMES.
          07 Z-B REDEFINES Z-C    PIC X
                                   OCCURS 6 TIMES.
        06 Z-CRT REDEFINES Z-PROCNAME NATIVE-2
                                   OCCURS 3 TIMES.
        06 Z-PID.
          07 Z-CPU                PIC X(1).
          07 Z-PIN                PIC X(1).
        06 Z-CPUPIN REDEFINES Z-PID NATIVE-2.
      05 Z-QUAL1.
        06 Z-C                    PIC X(8).
        06 Z-S REDEFINES Z-C.
          07 Z-I                  NATIVE-2
                                   OCCURS 4 TIMES.
        06 Z-B REDEFINES Z-C      PIC X
```

```
                                OCCURS 8 TIMES.
      05 Z-QUAL2.
        06 Z-C                  PIC X(8).
        06 Z-S REDEFINES Z-C.
          07 Z-I                NATIVE-2
                                OCCURS 4 TIMES.
        06 Z-B REDEFINES Z-C    PIC X
                                OCCURS 8 TIMES.
    04 Z-DEVICE REDEFINES Z-DISC.
      05 Z-DEVNAME.
        06 Z-C                  PIC X(8).
        06 Z-S REDEFINES Z-C.
          07 Z-I                NATIVE-2
                                OCCURS 4 TIMES.
        06 Z-B REDEFINES Z-C    PIC X
                                OCCURS 8 TIMES.
      05 Z-SUBDEVNAME.
        06 Z-C                  PIC X(8).
        06 Z-S REDEFINES Z-C.
          07 Z-I                NATIVE-2
                                OCCURS 4 TIMES.
        06 Z-B REDEFINES Z-C    PIC X
                                OCCURS 8 TIMES.
      05 Z-FILLER               PIC X(8).
02 EVENT-TEXT.
  03 Z-C                        PIC X(254).
  03 Z-S REDEFINES Z-C.
    04 Z-I                      NATIVE-2
                                OCCURS 127 TIMES.
  03 Z-B REDEFINES Z-C          PIC X
                                OCCURS 254 TIMES.
02 SUBJECT-FIELD-NAME.
  03 Z-C                        PIC X(24).
  03 Z-S REDEFINES Z-C.
    04 Z-I                      NATIVE-2
                                OCCURS 12 TIMES.
  03 Z-B REDEFINES Z-C          PIC X
                                OCCURS 24 TIMES.
02 ATM-NAME.
  03 Z-C                        PIC X(8).
  03 Z-S REDEFINES Z-C.
    04 Z-I                      NATIVE-2
                                OCCURS 4 TIMES.
  03 Z-B REDEFINES Z-C          PIC X
                                OCCURS 8 TIMES.
02 ATM-LOCATION.
  03 Z-C                        PIC X(24).
  03 Z-S REDEFINES Z-C.
    04 Z-I                      NATIVE-2
                                OCCURS 12 TIMES.
  03 Z-B REDEFINES Z-C          PIC X
                                OCCURS 24 TIMES.
02 ATM-ACCOUNT-NUM              NATIVE-4.
02 ATM-RETRY-LIMIT              NATIVE-2.
02 ATM-HW-COMPONENT.
  03 Z-C                        PIC X(24).
  03 Z-S REDEFINES Z-C.
    04 Z-I                      NATIVE-2
                                OCCURS 12 TIMES.
  03 Z-B REDEFINES Z-C          PIC X
                                OCCURS 24 TIMES.
02 ATM-HW-SUBCOMPONENT.
  03 Z-C                        PIC X(24).
  03 Z-S REDEFINES Z-C.
    04 Z-I                      NATIVE-2
                                OCCURS 12 TIMES.
```

```
      03 Z-B REDEFINES Z-C           PIC X
                                     OCCURS 24 TIMES.
    02 ATM-SERIAL-NUMBER             NATIVE-4.
    02 ATM-SENSE-STATUS              NATIVE-4.
?Section EGEN-INTERFACE-DEFINITIONS,Tandem
*---------------------------------------------------------------------
* EGEN module interface variables definitions.
*---------------------------------------------------------------------
*
* Used to define the EVENT-TYPE field of EGEN-RECORD.
*
    01 INFORMATIVE-EVENT NATIVE-2 VALUE IS 1.
    01 ACTION-ATTENTION-EVENT NATIVE-2 VALUE IS 2.
    01 ACTION-COMPLETION-EVENT NATIVE-2 VALUE IS 3.
    01 CRITICAL-EVENT NATIVE-2 VALUE IS 4.
    01 ACF-VERSION NATIVE-2 VALUE IS 16896.
*
*
*---------------------------------------------------------------------
* When the EGEN module is invoked, it will return a status variable,
* called Return-code, to the calling program.  Please note that this
* return code does not correspond to a file system error, the field
* egen-error of egen-record will contains more information. This
* table lists the possible values returned by EGEN.
*
* Return-Code Description
*
*      0      EGEN successfully generated the event message
*
*      1      The Initialize^egen^record procedure detected that
*             there was no parameter passed to this procedure.
*
*     10-13   The Open^egen^collector procedure detected an error
*             when opening the collector.  The Egen-error field
*             of Egen-record contains detailed information.
*
*     20-21   The Close^egen^collector procedure detected an error
*             when closing the collector.  The Egen-error field
*             of Egen-record contains detailed information.
*
*     30-31   The Complete^egen^operation procedure detected an error
*             when completing the write operation. The Egen-error field
*             of Egen-record contains detailed information.
*
*     40-41   The Get^egen^event^text^define procedure detected an error
*             when processing the =_EGEN_ADD_EVENT_TEXT define.  The
*             Egen-error field of Egen-record contains detailed
*             information.
*
*     50-59   The Initialize^event^buffer procedure detected an error
*             when initializing the event buffer.  The Egen-error field
*             of Egen-record contains detailed information.
*
*     60-61   The Write^event^buffer procedure detected an error or a
*             warning when writing the event buffer.  The Egen-error field
*             of Egen-record contains detailed information.
*
*     70      The Egen procedure detected that a required parameter
*             was not passed or that the combination of parameters was
*             not valid.
*
*     71      The Egen procedure detected that the Egen-record was not
*             initialized by the Initialize^egen^record procedure
*             before calling Egen.
*
*---------------------------------------------------------------------
```

```
*
* Constants and returns code used by the Initialize^egen^record procedure
*
    01 EGEN-INITIALIZE-RECORD-OK NATIVE-2 VALUE IS 0.
    01 EGEN-INITIALIZE-MISSING-PARAM NATIVE-2 VALUE IS 1.
*
* Constants and returns code used by the Open^egen^collector procedure
*
    01 EGEN-OPEN-COLLECTOR-OK NATIVE-2 VALUE IS 0.
    01 EGEN-OPEN-MISSING-PARAM NATIVE-2 VALUE IS 10.
    01 EGEN-OPEN-INVALID-SYNC-DEPTH NATIVE-2 VALUE IS 11.
    01 EGEN-OPEN-COLLECTOR-ERROR NATIVE-2 VALUE IS 12.
    01 EGEN-OPEN-COLLECTOR-WARNING NATIVE-2 VALUE IS 13.
*
* Constants and returns code used by the Close^egen^collector procedure
*
    01 EGEN-COLLECTOR-CLOSED-OK NATIVE-2 VALUE IS 0.
    01 EGEN-COLLECTOR-MISSING-PARAM NATIVE-2 VALUE IS 20.
    01 EGEN-COLLECTOR-ALREADY-CLOSED NATIVE-2 VALUE IS 21.
*
* Constants and returns code used by the Complete^egen^operation procedure
*
    01 EGEN-COMPLETE-OPERATION-OK NATIVE-2 VALUE IS 0.
    01 EGEN-COMPLETE-MISSING-PARAM NATIVE-2 VALUE IS 30.
    01 EGEN-COMPLETE-OPERATION-ERROR NATIVE-2 VALUE IS 31.
*
* Constants and returns code used by the Get^egen^event^text^define procedure
*
    01 EGEN-GET-TEXT-DEFINE-OK NATIVE-2 VALUE IS 0.
    01 EGEN-GET-TEXT-DEFINEMODE-ERROR NATIVE-2 VALUE IS 40.
    01 EGEN-GET-TEXT-DEFINEINFO-ERROR NATIVE-2 VALUE IS 41.
*
* Constants and returns code used by the Initialize^event^buffer procedure
*
    01 EGEN-INITIALIZE-EVENT-OK NATIVE-2 VALUE IS 0.
    01 EGEN-INITIALIZE-TYPE-ERROR NATIVE-2 VALUE IS 50.
    01 EGEN-INITIALIZE-EVENT-NUMBER NATIVE-2 VALUE IS 51.
    01 EGEN-INITIALIZE-EMSINIT-ERROR NATIVE-2 VALUE IS 52.
    01 EGEN-INITIALIZE-SUBJECT-ERROR NATIVE-2 VALUE IS 53.
    01 EGEN-INITIALIZE-FLAGS-ERROR NATIVE-2 VALUE IS 54.
    01 EGEN-INITIALIZE-ACTION-ID NATIVE-2 VALUE IS 55.
    01 EGEN-INITIALIZE-ACTION-ERROR NATIVE-2 VALUE IS 56.
    01 EGEN-INITIALIZE-TEXT-ERROR NATIVE-2 VALUE IS 57.
    01 EGEN-INITIALIZE-TOKENS-ERROR NATIVE-2 VALUE IS 58.
    01 EGEN-INITIALIZE-SSGETTKN-ERROR NATIVE-2 VALUE IS 59.
*
* Constants and returns code used by the Write^event^buffer procedure
*
    01 EGEN-WRITE-EVENT-OK NATIVE-2 VALUE IS 0.
    01 EGEN-WRITE-EVENT-WARNING NATIVE-2 VALUE IS 60.
    01 EGEN-WRITE-EVENT-ERROR NATIVE-2 VALUE IS 61.
*
* Constants and returns code used by the Egen procedure
*
    01 EGEN-GENERATE-EVENT-OK NATIVE-2 VALUE IS 0.
    01 EGEN-MISSING-PARAMETER-ERROR NATIVE-2 VALUE IS 70.
    01 EGEN-RECORD-NOT-INITIALIZED NATIVE-2 VALUE IS 71.
?Section ATM-EVENT-NUMBERS,Tandem
*---------------------------------------------------------------------
* If a file was specified in the USER-DDL-FILE of the ACF, we will
* source it here.  We also add these definitions in the COBOL
* definition file to be used by COBOL programs.
*---------------------------------------------------------------------
*---------------------------------------------------------------------
*
*  EMS FastStart - T9263C20 - (17MAR91)
```

```
*
*  File Type:                     DDL Source Schema
*
*  Source File Name:              SATMDDL
*
*  Source Library File Produced:  Sourced during the compilation of
*                                 the main DDL.
*
*  Generation Time:               December 17, 1990
*
*  Language Compiler Required:    Data Definition Language (DDL)
*
*  Compiler Version Required:     C20
*
*
*
*  File Description: This DDL source schema file is an example of DDL
*  definitions which is used by the ATM example provided with EMS
*  FastStart.  These definitions will ease the documentation and the
*  maintenance of an EGEN based application.
*
*---------------------------------------------------------------------
*---------------------------------------------------------------------
*
* Each event message generated by an application will be identified
* by a unique number called the event message number. This section
* presents an example of a typical implementation.
*
* Event Numbers                  Usage Description
*
*    0 -  99                     Informative Events
*  100 - 199                     Action Events
*  200 - 499                     Critical Events
*  500 - 999                     Reserved for future enhancement
*
*---------------------------------------------------------------------
*---------------------------------------------------------------------
*
* INFORMATIVE EVENTS DESCRIPTION:
*
* These event message numbers describe the current set of INFORMATIVE
* EVENTS supported by this application.  For each event generated by a
* Atm application, the programmer should select one of the following
* event message numbers and move it into the EVENT-NUMBER field of
* EGEN-RECORD.
*
*---------------------------------------------------------------------
    01 ATM-EVT-UP NATIVE-2 VALUE IS 1.
    01 ATM-EVT-ACCT-INSUF-FUNDS NATIVE-2 VALUE IS 2.
    01 ATM-EVT-BACK-ONLINE NATIVE-2 VALUE IS 3.
*---------------------------------------------------------------------
*
* ACTION EVENTS DESCRIPTION:
*
* These event message numbers describe the current set of ACTION
* EVENTS supported by this application.  For each event generated by a
* Atm application, the programmer should select one of the following
* event message numbers and move it into the EVENT-NUMBER field of
* EGEN-RECORD.
*
*---------------------------------------------------------------------
    01 ATM-EVT-LOW-ON-CASH NATIVE-2 VALUE IS 100.
*---------------------------------------------------------------------
*
* CRITICAL EVENTS DESCRIPTION:
*
```

```
* These event message numbers describe the current set of CRITICAL
* EVENTS supported by this application.  For each event generated by a
* Atm application, the programmer should select one of the following
* event message numbers and move it into the EVENT-NUMBER field of
* EGEN-RECORD.
*
*--------------------------------------------------------------------
    01 ATM-EVT-DOWN NATIVE-2 VALUE IS 200.
    01 ATM-EVT-SECURITY-BREACH NATIVE-2 VALUE IS 201.
    01 ATM-EVT-HW-FAILURE NATIVE-2 VALUE IS 202.
?Section ATM-ACTION-ID,Tandem
*--------------------------------------------------------------------
*
* ACTION EVENT IDENTIFIER DESCRIPTION:
*
* There is an action identifier associated with each action event
* message.  This Action ID is used by VIEWPOINT to match an
* action-attention event message with the corresponding
* action-completion event message.
*
*--------------------------------------------------------------------
    01 ATM-ACTION-ID-1 NATIVE-2 VALUE IS 1.
    01 ATM-ACTION-ID-2 NATIVE-2 VALUE IS 2.
    01 ATM-ACTION-ID-3 NATIVE-2 VALUE IS 3.
    01 ATM-ACTION-ID-4 NATIVE-2 VALUE IS 4.
    01 ATM-ACTION-ID-5 NATIVE-2 VALUE IS 5.
?Section ATM-CONSTANT-VALUES,Tandem
*--------------------------------------------------------------------
*
* CONSTANT VALUE IDENTIFIER DESCRIPTION:
*
* These are constants that can be used globally in the ATM subsystem.
*
*--------------------------------------------------------------------
    01 ATM-VAL-FALSE NATIVE-2 VALUE IS 0.
    01 ATM-VAL-TRUE NATIVE-2 VALUE IS 1.
```

# ATM Example, DSM Templates Services Source File: SATMTMPL

```
============================================================
== Program Name      : EMS FastStart - T9263C20 - (17MAR91)
==
== File State        : Production
== File Version      : 1
== Generation Time   : December 15, 1990 14:50:35
== Source File Name  : SATMTMPL
== Object File Name  : ZATMTMPL
== Compiler Version  : TACL and TEMPL C20
==
== File Description  : This file is the input file to the TEMPL
==                     compiler.  It contains the ATM example
==                     templates.
==
== Program Modifications:              Date of Modification
== First release of this file             December 15, 1990
============================================================

VERSION: "T9263C20 - (17MAR91)"
SSID: ATM-VAL-SSID
SSNAME: "CUSTOMER"
```

```
==
==                GENERAL FORMATTING TEMPLATES
==

MSG: ZEMS-TKN-EVENTNUMBER, ATM-EVT-UP

     "ATM <1> is up at <2>."

     1: ATM-TKN-ATM-NAME
     2: ATM-TKN-ATM-LOCATION


MSG: ZEMS-TKN-EVENTNUMBER, ATM-EVT-ACCT-INSUF-FUNDS

     "Insufficient funds in account <1>."
     " Access denied on ATM <2>, at <3>."

     1: ATM-TKN-ATM-ACCOUNT-NUM, ZI2
     2: ATM-TKN-ATM-NAME
     3: ATM-TKN-ATM-LOCATION


MSG: ZEMS-TKN-EVENTNUMBER, ATM-EVT-LOW-ON-CASH

      "ATM <1> at <2> is low on funds."

      1: ATM-TKN-ATM-NAME
      2: ATM-TKN-ATM-LOCATION


MSG: ZEMS-TKN-EVENTNUMBER, ATM-EVT-DOWN

      "ATM <1> is down at <2>."

      1: ATM-TKN-ATM-NAME
      2: ATM-TKN-ATM-LOCATION


MSG: ZEMS-TKN-EVENTNUMBER, ATM-EVT-SECURITY-BREACH

     "Security breach on account <1>. Number of accesses "
     "attempted <2>; ATM <3> is down at <4>."

      1: ATM-TKN-ATM-ACCOUNT-NUM, ZI2
      2: ATM-TKN-ATM-RETRY-LIMIT, I
      3: ATM-TKN-ATM-NAME
      4: ATM-TKN-ATM-LOCATION


MSG: ZEMS-TKN-EVENTNUMBER, ATM-EVT-HW-FAILURE

     "Hardware failure on ATM <1> at <2>. Component failed is <3>; "
     "subcomponent is <4>; serial number <5>; sense status <6>."

      1: ATM-TKN-ATM-NAME
      2: ATM-TKN-ATM-LOCATION
      3: ATM-TKN-ATM-HW-COMPONENT
      4: ATM-TKN-ATM-HW-SUBCOMPONENT
      5: ATM-TKN-ATM-SERIAL-NUMBER, ZI2
      6: ATM-TKN-ATM-SENSE-STATUS, ZI2
```

# E Filter

This is the program code for the EMS FastStart default filter program ATM1EMFS.

```
------------------------------------------------------------------------
--
-- EMS Fast Start - T9263C20 - (17MAR91)
--
-- File Type:                      EMF Source file
--
-- Source File Name:               $DATA.ATM.ATM1EMFS
--
-- Object File Produced:           $DATA.ATM.ATM1EMFO
--
-- Generation Time:                November 6, 1989 21:41:40
--
-- Language Compiler Required:     Event Management Filter (EMF)
--
-- Compiler Version Required:      C20
--
-- File Description:  This file contains a filter which passes only
-- event messages generated by a specific subsystem.  This filter is
-- compatible with Viewpoint and can be used to select event messages
-- generated by the ATM1PROG program.
------------------------------------------------------------------------
--
-- Filter description: This filter will pass only event messages which
-- are generated by the specific subsystem: ATM.  It is compatible
-- with VIEWPOINT and can be used to display events generated by the
-- $DATA.ATM.ATM1PROG program.
--
-- Author :            Please take ownership.
--
-- Date Created :      November 6, 1989 21:41:40
-- Date Last Changed : November 6, 1989 21:41:40
--
-- Source Filter Name: $DATA.ATM.ATM1EMFS
-- Object Filter Name: $DATA.ATM.ATM1EMFO
--
-- Function Keywords:
--        Parameterized: No
--        Distributor Type:
--            Forwarding: .....................  No
--            Printing: .......................  Yes
--            Consumer: .......................  Yes
--        Security: ...........................  NONO
--        Private: ............................  Yes
--        Shareable: ..........................  No
--        User Group:
--            Help Desk: ......................  No
--            Support Center: .................  No
--            Other: ..........................  Development
--        Mode of Operation:
--            Monitor: ........................  No
--            Troubleshooting: ................  No
--            Other: ..........................  Program Testing
--        Tokens:
--            Emphasis:
--                Action: .....................  Pass 1
--                Critical: ...................  Pass 2
--                Normal: .....................  Pass
--            Scope:
--             1. Node Name: ..................  Pass
--                                Node Name =     *
```

```
--
--             2. Subsystem ID: ..............  Pass
--                              Subsystem ID =  ATM^VAL^SSID
--
--             3. Event Number: ..............  Pass
--                              Event Number =  *
--
--             4. Token Present: .............  Pass
--                              Token Name =    *
--
--             5. Specific Token Value: .......  Pass
--                              Token Value =   *
--             6. Text: ......................  Pass
--                              Text =          *
--             7. Other ......................  Pass
--                              Other =         *
--
-------------------------------------------------------------------------

[#SET ZEMS^VAL^SSID [ZSPI^VAL^TANDEM].[ZSPI^SSN^ZEMS].0]

[#SET ATM^VAL^SSID [ATM^VAL^OWNER].[ATM^SSN^NUMBER].0]


FILTER ATM^DEFAULT^FILTER;

BEGIN SSID ( ZEMS^VAL^SSID )

 IF ZSPI^TKN^SSID = SSID ( ATM^VAL^SSID ) THEN
     BEGIN
        --
        -- fails on suppress^display events which are not
        -- action-completion.
        --
        IF ZEMS^TKN^SUPPRESS^DISPLAY = [ZSPI^VAL^TRUE] THEN
        BEGIN
          IF TOKENPRESENT ( ZEMS^TKN^ACTION^NEEDED ) AND
             ZEMS^TKN^ACTION^NEEDED = [ZSPI^VAL^FALSE]
          THEN PASS 3
          ELSE FAIL;
        END;
        --
        -- passes action-attention and action-completion events
        --
        IF TOKENPRESENT ( ZEMS^TKN^ACTION^NEEDED ) THEN PASS 1;
        --
        --
        -- testing for <> false for critical events
        --
        IF ZEMS^TKN^EMPHASIS <> [ZSPI^VAL^FALSE] THEN PASS 2;
        --
        --
        -- all other events from ATM^VAL^SSID are passed
        --
        PASS;
     END
  ELSE FAIL;
END;
```

# Index

# E



# F

# U



# V



# W