****

IM/East
External Reference Specification

****

1.0 INTRODUCTION

1.0 INTRODUCTION

1.1 WHAT IS IM/FAST

IM/Fast is part of the IM/VE family of products available on NOS/VE. It is a system which lets users develop, maintain, and use large interactive applications. A large application typically has many users and transactions. IM/Fast runs on the NOS/VE Operating System.

1.2 WHY USE IM/FAST

You may not want to use IM/Fast for every one of your applications. It will fit some applications better than others. There are products in NOS/VE and in the IM/VE product family which may be better for you. If you have a database application, evaluate IM/Quick, IM/SMART, and the DMFQM module of IM/DM first. By using these products, you may save a considerable amount of programming effort.

However, use IM/Fast if your application has all of the requirements listed below.

- Special needs of the application are beyond the scope of standard products and you must write either COBOL or FORTRAN programs to address those needs. Note that IM/Fast lets you integrate your programs with system programs. You will still be able to mix programs you write with programs provided by the system to develop the application.

- The application is mainly interactive and used by many users simultaneously. Furthermore, you want someone to be responsible for the application to ensure proper operation and use.

- When performance and response (to users) are important factors. Using IM/Fast can improve performance because it lets you use sub-routines for on-line transaction processing.

IM/Fast uses standard NOS/VE interfaces and facilities to build the environment for a user's application. This is an important consideration in evaluating IM/Fast. Anyone willing to spend enough time and effort can develop their own

IM/Fast-like application. IM/Fast gives you a set of existing tools which can assist you during the lifetime of your application, including development, usage, and maintenance.

1.3 IM/FAST FEATURES

This section describes the main features of IM/Fast.

1.3.1 ON-LINE APPLICATION MAINTENANCE

IM/Fast has a "Workbench" which is a collection of menus for the application administrator to use. The workbench lets an administrator of an IM/Fast application create and change the application definition and control what other users can do. Application maintenance functions can be performed on-line, while other users are also active.

1.3.2 MENU DRIVEN WORK ENVIRONMENT

Each IM/Fast application has a set of menus, called a menu system. All menus in a menu system are interconnected to allow moving between them. Menus also contain entries that let you execute transactions.

The way IM/Fast processes transaction requests is by showing each user the options from a menu in the menu system. The user can pick any one of the options. The menu option selected either executes a transaction or causes the selection of another menu.

1.3.3 TRANSACTION PROCESSING OPTIONS

In IM/Fast, a transaction request is processed by a NOS/VE program. NOS/VE gives you the option of executing a program as a command-processor or a task. A command-processor is a program that is called to process a NOS/VE System Command Language (SCL) command. IM/Fast in addition lets you execute sub-programs to process transaction requests. The use of a sub-program is very efficient. IM/Fast lets you combine the different ways of calling transaction processors in the same application or menu so that you may use programs or tasks when

performance is not important and use sub-programs when performance is desired.

1.3.4 SCHEDULING AND TUNING

IM/Fast provides a scheduling scheme for your applications that is based on NOS/VE Service Class concept.

Different applications running on a single host can be given different "Priority Levels". A "Priority Level" is equivalent to the NOS/VE Scheduler Service Class. Applications in different Scheduler Service Classes receive different levels of service from the system. By changing attributes of the Service Class you can "tune" the application according to your throughput and response time requirements.

The NOS/VE System Scheduler also provides the tools that can be used for tuning IM/Fast applications.

1.3.5 ACCOUNTING

NOS/VE Application Accounting is used to provide transaction accounting under IM/Fast. NOS/VE Application Accounting provides two types of accounting: Resource Measured and Unit Measured. Both accounting methods can be used by IM/Fast applications. However, the Unit Measured accounting method is preferred and IM/Fast will provide an interface to it in a future release. It lets you keep track of how many times a transaction is used, and the pricing may be based on this count. Resource measured accounting lets you collect the actual system resources (such as CPU time and memory) used.

The IM/Fast interface to Unit Measured Application accounting will not be available in the first release.

1.3.6 SECURITY

There are four ways in which access to IM/Fast applications can be controlled. They are User-authorization, User-privilege, Menu-access, and Access-level.

A user must be given user-authorization to access each IM/Fast application. Any NOS/VE user can be authorized to

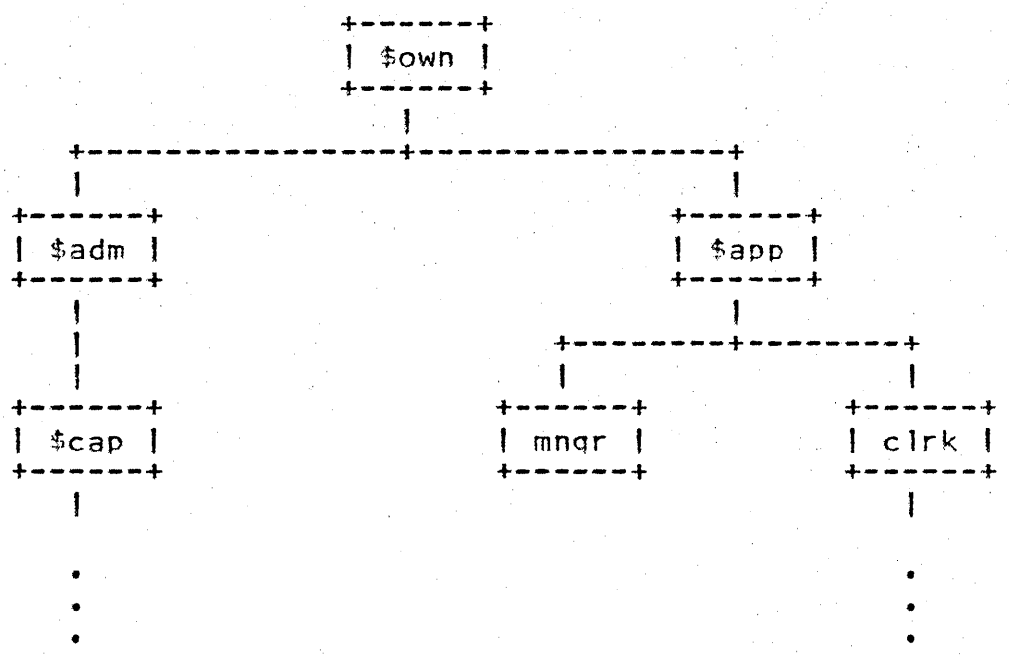
access an IM/Fast application. The user must first be a valid NOS/VE user and then be authorized to use one or more IM/Fast applications. The Family administrator authorizes NOS/VE users while an IM/Fast application administrator (defined below) authorizes users of the application.

User-privilege determines the type of work a user can do in the application. This is done by letting a person use only a subset of menus in the menu system. This partitioning of the menu system is done for the following privileges supported by IM/Fast:

Application_owner
Application_administrator
Database_administrator
Application_programmer
Application_user

Each user may have one or more of these privileges. Authorized users with the Application-administrator privilege are called Application administrators.

Menus in the user's privileged subset can be further restricted by a menu-access parameter, called the "root" menu. If you think of the menu system as an inverted tree of menus, as shown in the picture below, the "root" menu is the root of the tree of menus the user can access. Menus that are not under the root menu either directly or indirectly cannot be accessed by the user. In the picture below, if the user's root menu is $app, the user cannot access $cap.



Access levels are used by IM/Fast when it constructs a menu for display purposes. A menu may have more options than are shown to a given user. A transaction appears in a user's menu only if the user's access level is greater than or equal to the access level of the transaction. Two users, with different access levels, looking at the same menu may see different options. If a transaction does not appear on any menu for a user, it cannot be used.

1.3.7 TRANSACTION LOGGING

IM/Fast logs transaction input requests to provide a historical record of work performed. Application logging may be enabled or disabled by application and user. This logging is in addition to the account-log, system-log, network-log, and database-logs maintained by other parts of NOS/VE system. In addition, application transactions may use the system statistics and logging facilities directly if that is needed to meet some special needs.

The Transaction Logging will not be provided in the first release of IM/Fast.

1.3.8 SHARED DATA AREAS

Transaction processors have a need to store data that can be used by other transaction processors or the same transaction processor when executed at a later time. In most cases this data is stored in the application database. However, there are times when the data, due to its transient nature or for rapid access or both, cannot be stored in the application database.

For this class of data storage and retrieval, IM/Fast transaction processors are given access to a special data area. IM/Fast allocates the data area on a segment access NOS/VE file. The transaction processors access this area like it is stored in memory; no special calls to read/write are needed.

IM/Fast provides two such data areas to each transaction processor. One data area is allocated on a file that is shared by all users and transactions of the application. The other is on a file which is not shared. The data stored in this file is available to transactions submitted by one user. Both of these data areas are managed by the application, not IM/Fast. The only limit on the size of these areas is the NOS/VE file size limit. Certain restrictions apply when using the global data area for update.

1.3.9 ON-LINE APPLICATION LIBRARY MAINTENANCE

IM/Fast provides a way for you to change programs on your Application Object Library without requiring on-line users to logout first to make use of them. This capability is in addition to the Object Code Utility (OCU) of NOS/VE which is required for Application Library maintenance.

1.3.10 INTEGRATION

"Integration" is a way to customize your application for all users, even when they may have different needs. IM/Fast lets you use NOS/VE system commands and other programs, along with any of your own programs in the application menus, to tailor the menus for a specific use. For example, you may let a programmer use the Professional Programming Environment, a database administrator use the DMDBA module of IM/DM, and a

data entry clerk use a program specially developed for data entry work, all in the same application at the same time.

1.4 IM/FAST APPLICATION CONCEPTS

Each IM/Fast application is a collection of Users, User-groups, Transactions, Menus, and Application Libraries predefined to IM/Fast by an Administrator. The Application Definition is stored and used by IM/Fast to provide authorized users access to application menus and transactions.

Several independent IM/Fast Applications are possible on one CYBER system. Each application can be administered separately and provide different services to different users.

This section, 1.4, describes IM/Fast terms and concepts.

1.4.1 APPLICATION DEFINITION FILES (ADFS)

Each IM/Fast application definition is stored on a set of NOS/VE permanent files, called the Application Definition Files or ADFs for short. There is only one set of ADFs per application. The ADFs are created when the application definition is created. The application definition is accessed for every user of the application. Consequently, ADFs are shared by all users of the application. For some users the ADFs are attached so that the definition can be changed, for others only read access is required. The ADFs are accessed by IM/Fast and not the application. Normally, users are not aware that the application definition is stored on files and being referenced for them.

ADFs are legible sequential files.

1.4.2 APPLICATION OBJECT LIBRARY

The Application Object Library is a NOS/VE permanent file on which user-developed application programs, procedures, forms, and messages are stored. Application administrators are responsible for creating and maintaining this file. IM/Fast uses this file for all users of an application.

1.4.3 MENU

A menu is a collection of choices. A menu choice can be a transaction or another menu. Each menu is known by a code. A menu code must be unique among other menu codes and transaction codes within the application.

1.4.4 MENU SYSTEM

All menus, except the root menu, must belong to another menu, setting up a parent-child relationship among menus. A menu may belong to only one other menu either directly or indirectly. A menu may not belong to itself. The collection of all application menus is the Menu System. Each application menu system includes some menus supplied by IM/Fast and one or more application menus.

1.4.4.1 Root_menu

IM/Fast menu system is a hierarchy of menus and transactions. Each non-terminal node of the hierarchy is a menu and terminal nodes are transactions. The menu at the top of the hierarchy is the root menu. All other menus and transactions belong to the root menu either directly or indirectly. The menus and transactions at the top two levels of the system are supplied by IM/Fast. User developed menus and transactions must be added to these menus directly or to other menus that belong to them.

There is only one root menu per application. This is the $own menu which is provided by IM/Fast.

1.4.4.2 User_root_menu

Each privilege supported by IM/Fast (logically) divides the menu system into sub-sets of menus (and transactions). These were called the privilege sub-sets in section 1.3.6 (SECURITY). Each privilege subset is in turn a hierarchy belonging to the larger hierarchy of the entire menu system, and therefore has a root menu, which may not be the same as the root of the entire menu hierarchy.

Note that the privilege subset for application-owner privilege is the entire menu hierarchy, and has the $own menu as its root menu. The privilege subset for a user with only application-user privilege is the hierarchy of which $app menu is the root.

Each user is authorized to access the privilege subset of the menu hierarchy or a further subset of it. This is controlled by the menu-access attributes of the user. The menu-access attributes establish a root menu, an initial menu, and an initial transaction. The user's root menu is the root of the menu hierarchy that is actually available to the user. The user's root menu must be within the privilege subset and may be different for different users. A user's root menu should not be confused with the root menu for the entire menu hierarchy or the root menu of the privilege sub-sets, even though they may be the same because they are different conceptually.

1.4.4.3 Initial_menu

The initial menu is the first menu available to the user at the start of a new session. If an initial transaction is specified, the initial menu is the menu of the initial transaction. If there is no initial transaction, the initial menu can be the user's root menu or any menu that belongs to the hierarchy established by the root menu.

1.4.5 MENU PROLOG/EPILOG

IM/Fast menus and any user-defined menus may have prolog and epilog processors defined. Menu prolog and epilog processors are optional. Menu prolog and epilog processors are sub-programs provided by the application developers.

Menu prologs and epilogs are executed in hierarchical fashion. A menu prolog, if one is specified, is executed when the menu is selected for display when you are moving down in the menu system (away from the root menu). The menu epilog of the current menu is executed, if one is specified, when the current menu is left to display a different menu and you are moving up in the menu system (toward the root menu).

IM/Fast allows you to jump from one menu to another. IM/Fast automatically executes all required menu epilogs as it

moves up in the menu system to a common node, then it automatically executes all required menu prologs as it moves down in the menu system to the new menu.

On initialization, IM/Fast automatically executes all required menu prologs from $own through the initial menu. On termination, IM/Fast automatically executes all required menu epilogs from the currently displayed menu through the $own menu.

If a menu prolog returns an abnormal status, the abnormal status is displayed, the process of moving from one menu to another is stopped, and the menu which is the parent of the menu whose prolog returned the abnormal status is displayed. If a menu epilog returns an abnormal status, the abnormal status is displayed. After the user acknowledges the abnormal status, the process of moving from one menu to another is continued.

1.4.6 PRIVILEGE

Every user of an IM/Fast application has one or more privileges given to him. The user's privilege determines what part of the Menu System the user can see and use. Names of IM/Fast privileges are:

application_owner
application_administrator
database_administrator
application_programmer
application_user

The term "application user" or "user", for short, refers to an authorized IM/Fast application user without regard to the user's privilege. The term "owner" refers to a user with the application_owner privilege, the term "administrator" refers to a user with the application_administrator privilege, the phrase "database administrator" refers to a user with the database_administrator privilege, and the term "developer" or "programmer" refers to a user with the application_programmer privilege.

1.4.7 SESSION

A session refers to the use of one IM/Fast application by one user. It manifests itself in the system as a NOS/VE task in the user's job (between login and logout). Between a login and logout, a NOS/VE user can have several sessions with different or the same IM/Fast application. A user can only have one IM/Fast session at a time, that is, two users cannot be running IM/Fast on two different terminals using the same NOS/VE user identification at the same time. The application is specified at the beginning of each session; this application stays active during the session. The definition of the entire active application or parts of it (that apply to the current user) is available to IM/Fast throughout the session. User requests are processed by IM/Fast during a session and are interpreted using the definition of the active application.

1.4.8 APPLICATION PROLOG AND EPILOG

IM/Fast provides the ability to execute a file of SCL commands at the beginning and at the end of a user's session. The application prolog, if any, is executed at the beginning of a user's session; the application epilog, if any, is executed at the end of a user's session. The application prolog and epilog are specified for the application, but a user may specify a different file of SCL commands as parameters on the Fast control statement.

If the application prolog returns an abnormal status, IM/Fast displays the abnormal status and terminates. If the application epilog returns an abnormal status, IM/Fast displays the abnormal status and continues the termination process.

1.4.9 SESSION PROCESSORS (INITIAL/FINAL)

IM/Fast provides user processing at the beginning and at the end of a user's session. This is done by session processors. Session processors can be specified for every user or for the application. Session processors are sub-programs provided by application developers. The use of these processors is optional. Processing done by the session processors is determined by application developers.

1.0 INTRODUCTION
1.4.9 SESSION PROCESSORS (INITIAL/FINAL)

The initial session processor, if any, is called at the beginning of each session after the application prolog, if any. The final session processor, if any, is called at the end of the session before the application epilog, if any.

If the initial session processor returns an abnormal status, IM/Fast displays the abnormal status and terminates. If the final session processor returns an abnormal status, IM/Fast displays the abnormal status and continues the termination process.

1.4.10 TRANSACTION

IM/Fast lets you define transactions. A transaction is the definition of processing required by the computer to satisfy a user request. Transactions are arranged into menus, and the menus are arranged into a system of menus. A menu option can either be a transaction or another menu.

A transaction is identified by a transaction code. The same transaction code can be added to more than one menu. Transaction codes are entered by users of IM/Fast applications to initiate the processing defined for it, without knowing the name or type of the actual transaction processor. All transaction codes which are valid for an application are predefined by the application administrator. Each valid transaction must at least have a processor and type defined for it. IM/Fast executes the correct transaction processor when it receives a valid transaction code.

1.4.11 TRANSACTION PROCESSOR

A transaction processor is what executes in the system to satisfy a request. Each transaction definition contains the transaction processor name. A transaction processor name is the name of a program, sub-routine, or a command. A processor of this name is called by IM/Fast when a request for it is received. The method used by IM/Fast to call a transaction processor determines its type. The transaction processor types supported by IM/Fast are sub-program, task, and SCL command.

The "sub-program" type of transaction processor is called by IM/Fast with the CALL/RETURN mechanism supported by the COBOL and FORTRAN languages to call sub-routines or

sub-programs.

The "task" type of transaction processor is called by IM/Fast with the NOS/VE execute task request. The name of the transaction processor is the name of a program which is executed in the task. This task executes in parallel to the IM/Fast task.

IM/Fast calls SCL (System Command Language) to process a "command-processor" type of transaction processor. The name of the transaction processor must be the name of a command in the user's command list or on the application object library.

IM/Fast calls the NOS/VE loader to load transaction processors. If the transaction processor is present on the application object library, it is loaded from there. Otherwise, the transaction processor may not be loaded or loaded from another library.

1.4.12 USER

Any NOS/VE user can be a user of one or more IM/Fast applications. The NOS/VE user name should be defined for every application the user is authorized to use. One user can be defined only once per application. This also defines the privilege, access level, and other pertinent information about the user.

1.4.13 USER-GROUP

IM/Fast lets you refer to a group of users by a unique name. This grouping is called a user-group. As many user-groups as needed can be defined. The use of user-groups is optional. A tree-like system of users and user-groups can be defined. A user or a user-group can belong to only one other user-group. However, a user-group may contain many users and other user-groups. A user-group may not belong to itself either directly or indirectly.

User-groups have the same attributes (privilege, access level, etc.) as users do. A user-group attribute value is used by IM/Fast for any attribute value required and not defined for the user. This allows you to define attributes for a group of users collectively, rather than having to define each user attribute individually for every user of the

group.

1.4.14 IM/FAST APPLICATION ENVIRONMENT

Users communicate with IM/Fast and the IM/Fast application via a terminal. The terminal is controlled by IM/Fast when the application comes up initially. The actual terminal used must be supported by NOS/VE's Screen Formatting Facility and the Terminal Manager.

IM/Fast uses the terminal in screen mode to communicate with a user. The terminal screen is partitioned into forms. Each form is used to display related information. IM/Fast uses the same set of forms for every user, though the information contained in the forms depends on the user and the application being used. (A form is an input/output entity supported by NOS/VE Screen Formatting Facility).

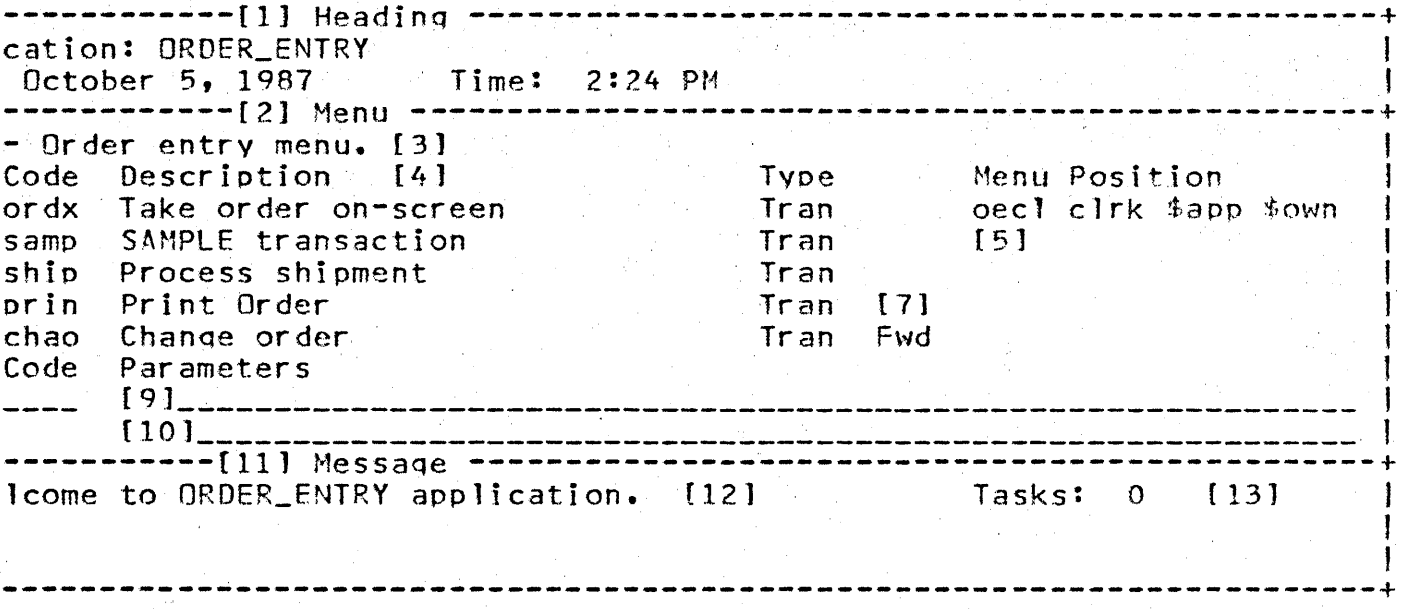
Function keys of the terminal are also active, and a table at the bottom of the screen shows the user which function keys perform what function.

The forms used by IM/Fast define the display environment for users. The same forms are used for every user. Notice that these forms are shown when IM/Fast is in control of the terminal; the terminal control switches between IM/Fast and the transaction processors. When a transaction processor is in control it may use its own forms. These forms may cover IM/Fast forms either partially or completely.

A picture of a terminal screen under normal IM/Fast operation is shown below. The screen used is a sample taken from the order_entry application.

Note: A terminal with CDC PC Connect version 1.2 was used to capture the forms. The same forms on a different terminal may appear different and may have different placement of function keys.

1.0 INTRODUCTION
1.4.14 IM/FAST APPLICATION ENVIRONMENT



Xecute Refrsh
f1 Ordx f2 Samp f3 Back f4 Help f5 Ship f6 Quit f7 Prin f8 Chao

IM/Fast Forms

This screen has three forms which are normally shown by IM/Fast on the terminal simultaneously. They are Heading form, Menu form and a Message form. The Numbers in [ ] mark an area of a form for its explanation.

IM/Fast uses other forms which are not shown in the picture above. These forms are used to provide help or details on error conditions.

The Heading form [1] shows the name of the application you are using and the current date and time. IM/Fast updates this information every time it interacts with the terminal. Most terminals prevent information on this form from being changed by the user. If it is changed, IM/Fast will not use the new data.

The Menu form [2] is the main form. It shows the title of the current menu [3] and the placement of this menu in the user's menu system [5]. Initially the user's Initial menu is the current menu.

Area [4] is a title line for the key, code, description,

type and menu position columns, which make up part of area [6]. Options in the current menu are shown in area [6]. The first column shows the function key for selecting the option, the second column shows the transaction or the menu code, the third column is a short description of the option, and the type column contains Tran if the option is a transaction or Menu if it is a menu.

Area [5] has a list of menus from the current menu to the user's root menu.

Area [7] indicates if there are more options in the current menu than will fit on the screen. A blank value indicates that all options are shown. If there are more options, a key label is displayed, use this key to look at the rest of the options. The terminal on which this screen was generated uses the Fwd and Bkw keys to page the options field forward and backward.

Data values in the menu form fields may not be altered by the user, with the exception of the data entry fields defined below.

Areas [8], [9], and [10] are for data entry. User requested transaction or menu code is entered in area [8]. This field has a title "Enter Code". Entering a code in this field and pressing the Xecute key causes execution of the transaction or selection of the menu. The requested code need not be an option of the current menu. Note that if the transaction code entered is unique or is an option in the current menu it is executed. However, if the transaction code appears in more than one menu and is not an option in the current menu, IM/Fast presents the user with a list of menus in which the transaction code appears. The user must select the desired menu.

Areas [9] and [10], represented by two lines on the screen, are used for entering data. These two fields make up a single string. No continuation ellipses are required at the end of the first field. If a parameter ends in the last column of the first field, include a separator (comma or blank) as the first column of the second field.

Enter the data on these lines before executing the transaction processor with the Xecute key or a function key. IM/Fast will pass any data in this field to the transaction processor program for processing.

The Message form [11] is used to display a status message.

Area [12] on this form is used to display IM/Fast or transaction processor messages, if and when necessary. This area has three lines. Area [13] shows the number of tasks which the user initiated which have not yet terminated.

The operator of an IM/Fast application can perform most functions with the press of a single key. So IM/Fast assigns displayed menu options to function keys on the terminal. Pressing a function key fx executes the option indicated by key labeled fx (x = 1 thru 8). The code field of the option appears in the function key label and the function key appears in the key field of the menu form. For example, pressing function key f1 labeled "ordx" in the sample display will run the order entry transaction ordx. Note, that you must enter any data required by the transaction processor in area [9] and [10], before pressing the function key to select a transaction. Labels on these keys will change as the current menu changes or the Fwd or Bkw key is used to display excess menu options.

Xecute, Back, Help, Refrsh, and Quit functions are always available to the user when this screen is displayed. The Xecute key selects the menu option whose code appears in area [8] of the menu form; remember to enter data before pressing this key. Using the Next or Return key on the terminal merely moves the cursor to the next data entry field.

Back key takes you one step back up the menu system to the parent menu, if one exists, or to the root menu.

Refrsh key causes IM/Fast to repaint the terminal screen. This clears up any garbage that may have appeared on the screen.

The Quit key terminates the user's session and returns to NOS/VE. If the user has activated one or more tasks which are still active, the user must choose one of three options: a) terminate as soon as all of the active tasks have terminated, b) terminate all active tasks and terminate the user's session, or c) do nothing. If the user chooses to do nothing, he could re-enter the Quit function or another function later to get an updated count of active tasks.

The Help key provides help on a topic selected by the cursor position at the time Help key is pressed. For example, if your cursor is within the Enter Code field and you press Help, the following help form is displayed:

1.0 INTRODUCTION
1.4.14 IM/FAST APPLICATION ENVIRONMENT

```
+---------------------- Heading ----------------------------------------------+
|  Application: ORDER_ENTRY                                                   |
|  Date: October 5, 1987       Time:  2:28 PM                                 |
+---------------------- Menu -------------------------------------------------+
|  OECL - Order entry menu.                                                   |
|  Key  Code  Description                        Type       Menu Position     |
|   F1  ordx  Take order on-screen               Tran       oecl clrk $app $own|
|   F2  samp  SAMPLE transaction                 Tran                         |
|   F5  ship  Process shipment                   Tran                         |
|   F7  prin  Print Order                        Tran                         |
|   F8  chao  Change order                       Tran  Fwd                    |
| Enter Code  Parameters                                                      |
|       ----  ----------                                                      |
+---------------------- Help -----------------------------------------------+
|  --  After you have exited HELP, enter a menu code or a transaction code in |
|  the Enter Code field. It will be executed when you press XECUTE. You may   |
|  enter a code which is not in the menu which is currently displayed.        |
|                                                                             |
|  Press NEXT to return to Fast. Press function key to get help for that key. |
+-----------------------------------------------------------------------------+
   Xecute                                                    Refrsh
f1  Ordx  f2  Samp  f3  Back  f4  Help  f5  Ship  f6  Quit  f7  Prin  f8  Chao
```

The Help form appears at the top or the bottom of the screen depending on the cursor position. The help form does not overwrite the cursor.

If you then press a function key, IM/Fast displays a message explaining the function. If the function corresponds to one of the menu items, the help message is the help text provided by the application developer; otherwise IM/Fast provides the message. For example, if you press f1, a help message for the ordx transaction is shown in the Help form:

1.0 INTRODUCTION
1.4.14 IM/FAST APPLICATION ENVIRONMENT

```
+---------------------- Heading ----------------------------------------------+
|  Application: ORDER_ENTRY                                                   |
|  Date: October 5, 1987       Time:  2:36 PM                                 |
+---------------------- Menu -------------------------------------------------+
|  OECL - Order entry menu.                                                   |
|  Key  Code  Description                        Type       Menu Position     |
|   F1  ordx  Take order on-screen               Tran       oecl clrk $app $own |
|   F2  samp  SAMPLE transaction                 Tran                         |
|   F5  ship  Process shipment                   Tran                         |
|   F7  prin  Print Order                        Tran                         |
|   F8  chao  Change order                       Tran  Fwd                    |
| Enter Code  Parameters                                                      |
|       ----  ------------------------------------------------------------    |
+---------------------- Help -------------------------------------------------+
| Execute ordx to take orders on screen. You may enter the order number as the |
| first 10 characters in the Parameters field.  If you do not enter an order   |
| number, the first order is displayed.  After taking orders, press Back.      |
|                                                                              |
|  Press NEXT to return to Fast. Press function key to get help for that key.  |
+------------------------------------------------------------------------------+

   Xecute                                                    Refrsh
f1  Ordx  f2  Samp  f3  Back  f4  Help  f5  Ship  f6  Quit  f7  Prin  f8  Chao
```

If you then press Quit, help for Quit function is provided:

1.0 INTRODUCTION
1.4.14 IM/FAST APPLICATION ENVIRONMENT

```
+---------------------- Heading ------------------------------------------------+
|  Application: ORDER_ENTRY                                                     |
|  Date: October 5, 1987        Time:  2:47 PM                                  |
+----------------------- Menu --------------------------------------------------+
|  OECL - Order entry menu.                                                     |
|  Key  Code  Description                        Type       Menu Position       |
|   F1  ordx  Take order on-screen               Tran       oecl clrk $app $own |
|   F2  samp  SAMPLE transaction                 Tran                           |
|   F5  ship  Process shipment                   Tran                           |
|   F7  prin  Print Order                        Tran                           |
|   F8  chao  Change order                       Tran  Fwd                      |
| Enter Code  Parameters                                                        |
|       ----  ------------------------------------------------------------      |
+----------------------- Help --------------------------------------------------+
|  --  QUIT terminates Fast and returns to the NOS/VE Operating System.         |
|                                                                               |
|                                                                               |
|                                                                               |
|  Press NEXT to return to Fast. Press function key to get help for that key.   |
+-------------------------------------------------------------------------------+

   Xecute                                                        Refrsh
f1  Ordx  f2  Samp  f3  Back  f4  Help  f5  Ship  f6  Quit  f7  Prin  f8  Chao

          Use the Next/Return key to exit help mode.
```

2.0 IM/FAST APPLICATION ADMINISTRATOR

2.0 IM/FAST APPLICATION ADMINISTRATOR

This section is for people who are responsible for creating and maintaining IM/Fast applications. Users with APPLICATION_OWNER or APPLICATION_ADMINISTRATOR privilege can maintain the application. There may be more than one application administrator per application.

If you are an application administrator you can add, change, or delete users, user groups, menus, and transactions.

2.1 SETTING UP A NEW APPLICATION

An IM/Fast application must be initialized (created) before it can be used. This is done by the FAST command. To create a new IM/Fast application, you must have permission to read and change the contents of the IM/Fast application directory file, $system.fast.tff$application_directory.

The following IM/Fast command is used to create an application. This command is also used to use an existing IM/Fast application.

```
FAST (
  application_identifier, ai: string (2) = $required
  application_name, an: name             = $required
  application_prolog, ap: file           = $optional
  application_epilog, ae: file           = $optional
  status: var of status                  = $optional)
```

Where:

APPLICATION_IDENTIFIER — This string uniquely identifies an IM/Fast application. It is used by IM/Fast as an index into the system application directory file to find the name of a permanent file catalog where the application definition is stored. This identifier is also used

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.1 SETTING UP A NEW APPLICATION

	as the "application identifier" field for generating status messages.
APPLICATION_NAME	This is the name given to the new application, if one is created. This name is matched with the name stored in the system application directory and is also used as a descriptive name.
APPLICATION_PROLOG	This parameter specifies the name of a NOS/VE file. SCL commands on this file, if one is specified, are executed at the start of the session. This parameter is used to override the application prolog specified in the application definition file.
APPLICATION_EPILOG	This parameter specifies the name of a NOS/VE file. SCL commands on this file, if one is specified, are executed at the end of the session. This parameter is used to override the application epilog specified in the application definition file.
STATUS	This parameter contains the status of IM/Fast command at completion.

If the requested application does not exist, IM/Fast will

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.1 SETTING UP A NEW APPLICATION

check with you to make sure that you do want to create a new application. This is the message it sends you:

> Application 'xx' does not exist. Enter YES or Y if this is a new application, otherwise enter NO. NO is the default entry.

Default entry is assumed if you just press the ENTER key, or an equivalent key to cause data entry. For example, on the CDC 721 terminal, press the Next key.

If you have entered YES, then IM/Fast has to know where you want to store the application definition files. IM/Fast stores the definition in NOS/VE permanent files, called Application Definition Files or ADF for short. IM/Fast lets you enter the name of a permanent file catalog. It asks you for the following information:

> Enter the name of a permanent file catalog in which to save the ADF files. $USER is the default catalog.

Your $USER catalog is used if you press the ENTER key without supplying a catalog name. However, IM/Fast must be able to define the ADF files in the catalog you specify. To do this the CYCLE permission to the specified catalog is needed. You have the CYCLE and other permissions if you own the catalog i.e., the catalog is a subcatalog of your $user catalog either directly or indirectly. If the catalog is not owned by you, then the owner of the catalog must give you permission by the CREATE_CATALOG_PERMIT SCL command.

IM/Fast creates the application definition for you and brings up the the administrator workbench. You can start adding to the definition right away. The application definition at this point contains mostly default values. You may want to use the default values for now and change them later, if needed.

The application is created with a single user (the creating user) authorized to use it. This user has APPLICATION_OWNER privilege. The next thing that you would want to do is to authorize other users and establish the APPLICATION OBJECT LIBRARY.

After that you can define Transactions, Menus, Users and User Groups. However, all this need not be done at one time.

2.2 APPLICATION_OWNER

An application owner is a user who is authorized to use the application and has "APPLICATION_OWNER" privilege. The user who creates the application definition is given this privilege automatically. Application owners will be called the "owner" in the rest of this document. There must always be at least one owner of an application.

As the owner you may have the $own menu as the root of your menu tree, which allows you access to the menus for application administrators, database administrators, application programmers, and application users. In addition you may also copy and delete the application. Only the owner may copy or delete the application.

If you have just created the application, by default, you have $own menu as your root menu and $adm as your initial menu.

NOTE: All menus with "$" in the first character position of the code are supplied by IM/Fast. To avoid conflict between IM/Fast and application menu codes, it is recommended that this character not be used for naming user developed menus. However, this restriction is not enforced by IM/Fast.

The $own menu is provided by IM/Fast and gives you the following options:

$OWN - Application owner menu.

Code	Description
$adm	Select to CHANGE application
$dba	Select to MAINTAIN Database
$prg	Select to DEVELOP Programs
$app	Select to USE application
$dea	DELETE application definition
$cop	COPY application definition

Each option in the $own menu is explained in detail below:

$adm Select to CHANGE application

This menu is provided by IM/Fast. The default options provided in this menu let you change the application definition.

You may add other options to this menu or remove

the default options. If the $cap (change application) option is removed you cannot use this menu to change the application definition.

$dba Select to MAINTAIN Database

This menu is provided by IM/Fast. There are no options provided in this menu by the system. This menu is included so that it can be used as a building block for operations performed by the database administrators. You could include in this menu calls to the DMDBA module of IM/DM or to IM/Control.

If this menu is not useful to you, it can be deleted. Caution, if you delete this menu, a user with only "DATABASE_ADMINISTRATOR" privilege cannot use the application.

$prg Select to DEVELOP Programs

This menu is provided by IM/Fast. Users with "APPLICATION_PROGRAMMER" privilege can use this menu or menus under it. Application programmers write, test, and maintain transaction processors. In NOS/VE, Professional Programming Environment (PPE) and Programming Environment (PE) support program development. IM/Fast lets you call either one. Or you may provide access to your own facilities to support transaction development.

If this menu is not useful to you, it can be deleted. Caution, if you delete this menu, a user with only "APPLICATION_PROGRAMMER" privilege cannot use the application.

$app Select to USE application

This menu is where you would add your application menus and options. This system provided menu has no options in it. This and other menus under it can be accessed by users who have "APPLICATION_USER" privilege.

If this menu is not useful to you, it can be deleted. Caution, if you delete this menu, a user with only "APPLICATION_USER" privilege cannot use the application.

$dea DELETE application definition

This menu option lets you delete the application definition files from the system and remove the application identifier from the application directory file. The application cannot be used after it has been deleted.

$cop COPY application definition

This menu option lets you copy the application definition under a new name.

Each user may have one or more privileges. The following chart shows the root menu for the user's privileged subset for users with exactly one permission.

Permission	Root Menu
application_owner	$own
application_administrator	$adm
database_administrator	$dba
application_programmer	$prg
application_user	$app

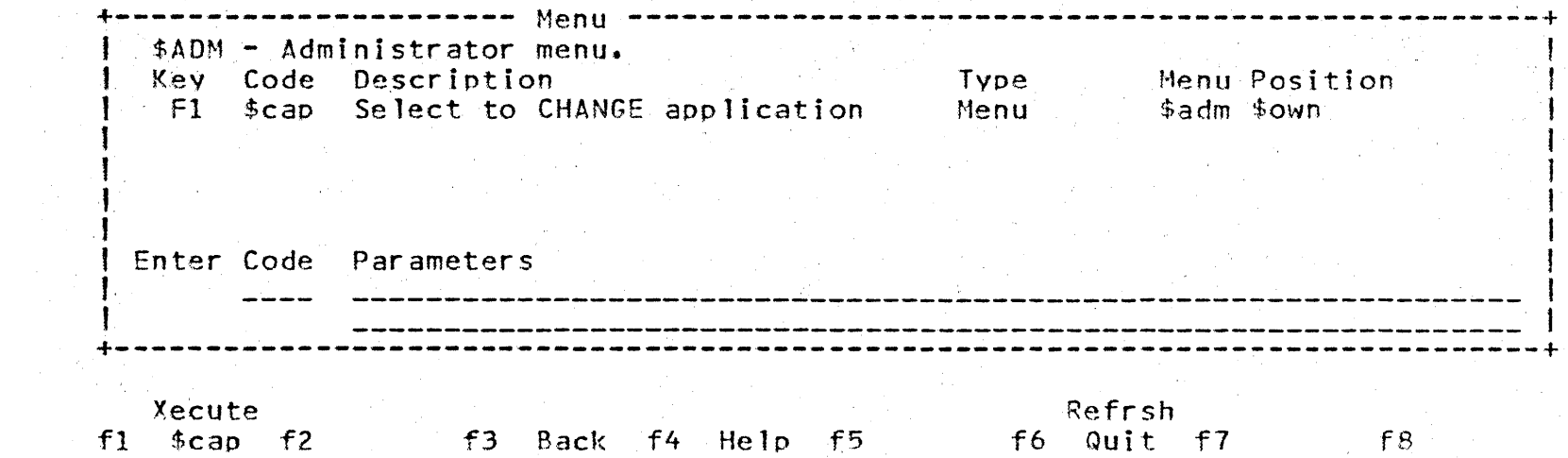
If a user has more than one permission, $own is the root menu of the user's privileged subset and $own contains those menus that the user has privilege to use. For example, $own would contain $adm and $dba for a user with application_administrator and database_administrator privilege.

2.3 APPLICATION_ADMINISTRATOR

An Application Administrator is a user defined in the IM/Fast application who has the "APPLICATION_ADMINISTRATOR" privilege. There can be as many application administrators per application as needed. A user with only "APPLICATION_ADMINISTRATOR" privilege can only use the $adm menu or menus which belong to it.

The $adm menu is provided by IM/Fast. The options included in this menu, as supplied, are shown in the picture below. This menu can be changed or deleted. Caution, if you delete this menu, a user with only "APPLICATION_ADMINISTRATOR" privilege cannot use the application.

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3 APPLICATION ADMINISTRATOR



$ADM - Application Administrator Menu

The option in this menu is:

$cap - Select to CHANGE application.

This is a system provided menu. You select this menu to either change or look at the application definition. Only one administrator can change the application at one time.

2.3.1 CHANGE APPLICATION OVERVIEW

The $cap menus gives you the options to change or look at the application definition. By choosing this menu you will be able to create, change, or delete other users, user groups, transactions, and menus.

Application definition is changed by permanently changing the Application Definition Files (ADFs). Since the ADF files are shared by all application users, the changes you make will affect all users. IM/Fast will use new values for a user session which is started after the changes have been made permanent.

Changing the ADF files (or changing application definition) is a three stage process. First, when you select the $cap - Change Application menu, a private copy of the (shared) ADF files is made for you in your $user catalog. Second, you will

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1 CHANGE APPLICATION OVERVIEW

change the contents of the private copy of the ADF files until you are ready to leave the $cap menu. At this time you will be asked if you want to make the changes permanent, and if you do, as the final step your changes will be copied on to the ADF files shared by all users, and the private copy of the ADF files discarded.

If you decide not to make your changes permanent, the system will keep the private ADF files around, so that you can resume changing from where you left off at a later time. However, the system will discard your changes, and notify you, if someone else has changed the ADFs since a copy of them was made for you. If that happens you will have to start over.

Each application may have more than one application administrator defined, and more than one administrator can use the application at the same time with the restriction that only one of them can change the application definition. Other active administrators may only look at the definition.

The system provided $cap menu is shown below:

```
+---------------------- Menu ------------------------------------------------+
|  $CAP - Change Application.                                                 |
|  Key  Code  Description                          Type      Menu Position    |
|   F1  $cha  CHANGE application attributes        Tran      $cap $adm $own   |
|   F2  $chg  ADD/CHANGE/DELETE groups             Tran                       |
|   F5  $chm  ADD/CHANGE/DELETE menus              Tran                       |
|   F7  $cht  ADD/CHANGE/DELETE transactions       Tran                       |
|   F8  $chu  ADD/CHANGE/DELETE users              Tran                       |
| Enter Code  Parameters                                                      |
|       ----  ------------------------------------------------------------  |
|             ------------------------------------------------------------  |
+-----------------------------------------------------------------------------+

   Xecute                                                        Refrsh
f1  $cha  f2  $chg  f3  Back  f4  Help  f5  $chm  f6  Quit  f7  $cht  f8  $chu
```

$CAP - Change Application Menu

Options in this menu are:

$cha CHANGE application attributes

This option runs a system transaction to let you change the attributes of the application.

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1 CHANGE APPLICATION OVERVIEW

$chg ADD/CHANGE/DELETE user groups

This option lets you change the user groups.

$chm ADD/CHANGE/DELETE menus

This option lets you change menus.

$cht ADD/CHANGE/DELETE transactions

This option lets you change transactions.

$chu ADD/CHANGE/DELETE users

This option lets you change users.

Remember, that "change" means that you can look at the definition records one at a time and make changes as you go along. Changing also includes adding new definition records or deleting ones not needed.

The Add, Change, and Delete function keys will be shown in alternate intensity when you enter the $cap menu to indicate that these functions are disabled, leaving you the option to look at the definition records. This will happen only when another active administrator of this application has entered the $cap menu before you and has the permission to change the definition.

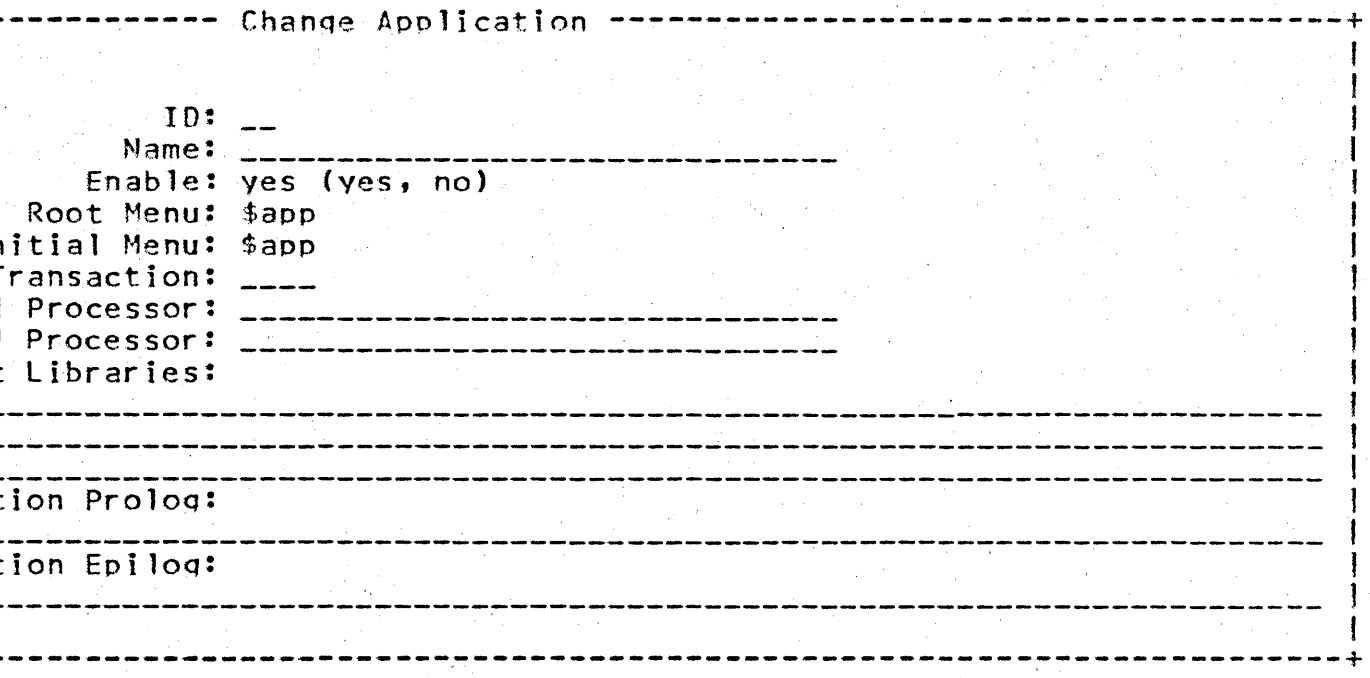
2.3.1.1 Change Application Attributes

When you select this option, the CHANGE APPLICATION form will be displayed on your terminal. The fields on this form show you the current values stored.

If you want to change a value, move the cursor to the field, enter the value you want to store and then press the Change function key. Note that this will store the new record in your copy of the definition files only. The application definition record can only be changed with this screen.

The "ID" and "Name" fields of this form cannot be changed. All other values may be changed as needed.

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1.1 Change Application Attributes



Change Refrsh
f1 f2 f3 Back f4 Help f5 Undo f6 f7 f8 NextEr

Enable

This field lets you enable or disable the application. A YES value in this field means the application is enabled. Enabled application can be used by any of its users. If the application is disabled, which is indicated by a value of NO in this field, then only administrators and owners can use it.

The next five fields, Root Menu, Initial Menu, Transaction, Initial Processor, and Final Processor, are application default values. These fields also appear on the Add/Delete/Change Group and Add/Delete/Change User screens. For each user, the value of each of these fields is taken from the user definition if nonblank, else from the group definition if nonblank, else from the application definition. If the value for the root menu is blank in all of these definitions, the user's root menu is the root menu of his privileged subset. If the value for the initial menu is blank in all of these definitions, the initial menu is the user's root menu. If the value for initial transaction, initial processor, or final processor is blank in all of these

definitions, there is no default; the corresponding processor is not executed.

Root Menu

This is the default root menu code for users and user groups that don't have root menu defined.

Initial Menu

This is the default initial menu code for users and user groups that don't have initial menu defined.

Transaction

This is the transaction code of the initial transaction for users and user groups for which no initial transaction is defined.

Initial Processor

This is the initial processor name for users and user groups for which no initial processor is named. Initial processor is automatically executed when a user starts to use the application.

Final Processor

This is the name of a processor for users and groups for which no final processor is named. Final processor is automatically executed when a user stops using the application.

Object Libraries

The full path to the Application Object Library file is specified in this field. There may be a maximum of three library files per application.

Application Prolog

This is the full file path of the file of SCL commands which is executed before the Initial Processor when a user starts to use the application.

Application Epilog

This is the full file path of the file of SCL

commands which is executed after the Final Processor when a user stops using the application.

A description of Change Application form function keys follows:

Back (f3)

This key returns you to IM/Fast. You are given the opportunity to decide whether to make the changes to the temporary definition files permanent.

Change (Shift f3)

This function is used to change the application attributes. When you press this key the attributes displayed on the terminal will be stored in the application definition.

Help (f4)

The Help key provides help. When you press it a brief message is displayed on your terminal to help you.

Undo (f5)

This key is used to undo the last valid Change Application function performed.

Refrsh (Shift f6)

This key is used to repaint the screen.

NextEr (f8)

This key moves the cursor to the next field which has an error. It may be used after a change operation that is not completed due to errors in one or more field.

2.3.1.2 Add/change/delete_user

IM/Fast keeps a record of each authorized user in the application definition files. Each user record assigns privilege, access level, and other attributes to a user. A user record is an authorization for the user to access the application with assigned attributes.

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1.2 Add/change/delete user

Users are known in IM/Fast by a valid NOS/VE user name. NOS/VE user names are assigned by the NOS/VE Family Administrator. A user name can be defined only once in an IM/Fast application definition. But the same user name can be used in different IM/Fast applications with the same or different attributes.

As an application administrator you can authorize new users for the application by adding their record to the ADF files. The NOS/VE user name is used on this record.

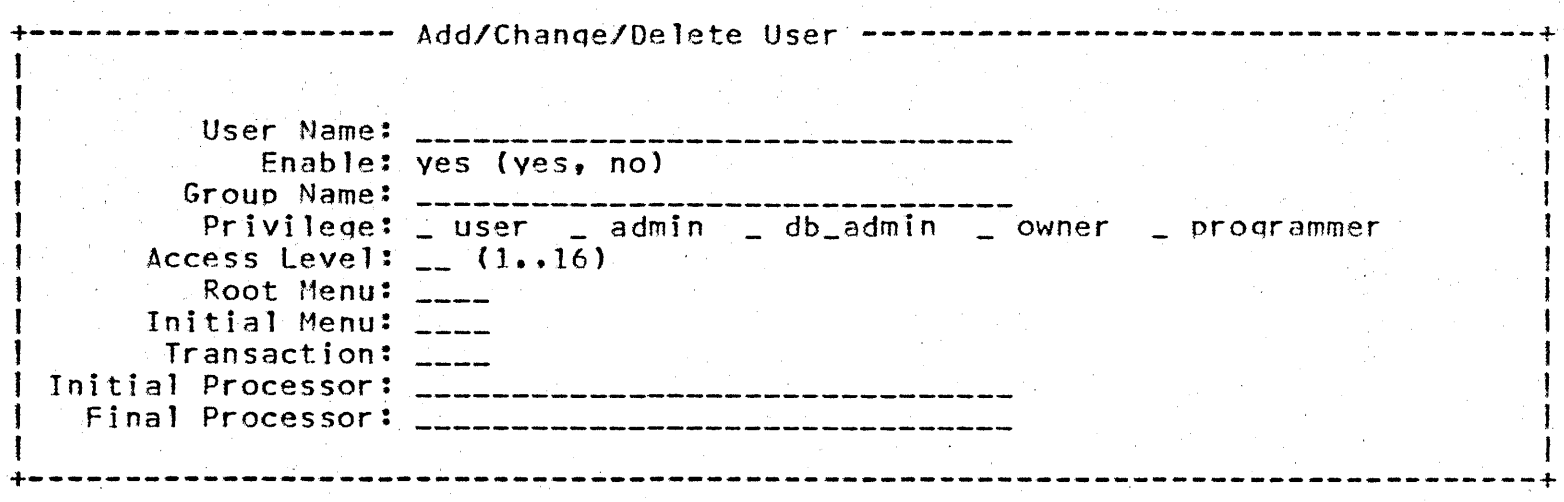
The ADD/CHANGE/DELETE USER option is selected when you want to work with the user authorization records to:

- o Look at authorization records of users
- o Add new users
- o Change existing users
- o Delete users

When you select this option a system transaction processor is executed. This processor lets you look at the user records, one at a time, either sequentially or by user name, and lets you add, change, or delete these records as you go along. This processor only works in screen mode. If you only have read access to the ADFs the Add, Change, and Delete function keys will be shown in low intensity, telling you that using these keys will have no affect on the definition.

The ADD/CHANGE/DELETE USER screen is used to display a user's record. The picture of this screen and the description of fields and function keys is given next. Remember that you are adding, changing, and deleting records from a private copy of the ADF files. The changes to the shared application definition are not made until you exit this menu.

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1.2 Add/change/delete user



FindUs Change Delete AppDef Refrsh ShowMn
f1 NextUs f2 Add f3 Back f4 Help f5 Undo f6 f7 ShowGr f8 NextEr

Description of ADD/DELETE/CHANGE USER form fields.

User Name

This field is used to enter or display a user name. Enter a user name here to add a new user record or to find a user record. The system displays the user name of the current user record in this field.

Enable

This field indicates if a defined user is enabled or not. If the value in this field is YES the user can access the application, and is said to be enabled. If the value is NO, access to the application will be denied to this user. You can enter a YES or a NO value in this field to temporarily disable users. Disabling an owner has no affect.

Group Name

This is the name of a group the user directly belongs to. A "blank" value in this field means the user does not belong to any group. The group membership of the user may be changed by entering the new group name here. When the user record is changed, the system checks the group name to ensure that the group record is defined. A user may belong to only one group.

Privilege

These fields indicate the user's current privileges. A non-blank value in the field in front of the privilege name indicates the user has that privilege. A user may have one or more privileges.

Access-level

This is the user's access level in this application. It can be changed to any value between 1 and 16.

Root Menu

This is the menu code of the user's root menu.

Initial Menu

This is the first menu the user will see when the application is first started if there is no initial transaction. It can be changed by entering a new value here.

Transaction

This is the initial transaction automatically started for the user when the application is first started. If no value is given here, in the user's group definition, or in the application definition, no transaction is started when the user starts the application.

Initial Processor

This field contains the name of an Initial processor for the user. Initial processor is optional. The Initial Processor Name must be a valid NOS/VE name.

Final Processor

This field contains the name of a Final processor for the user. Final processor is optional. The Final Processor name must be a valid NOS/VE name.

Description of ADD/DELETE/CHANGE USER form function keys.

NextUs (f1)

This function key displays the next user record, if one exists. The records are read by user name in alphabetical

order. A message is displayed to indicate the end of user records.

FindUs (Shift f1)

This function key is used to locate and show the attributes of a user. The user name in the "Name" field is used as the search key.

Add (f2)

This function key is used to add a new user's record to the definition file. The "Name" field is used as the key for the record, and other values shown on the terminal are stored in the record. The record is not written if any field has an error or if the user record already exists. The operating user is asked to correct any errors.

If you want to add a new user record which has the same attributes as an existing user, all you have to do is bring up the user record you want, enter the new user name in the "User Name" field and press the "Add" key. If a record with the same name doesn't already exist it will be added.

Back (f3)

This key returns you to IM/Fast. You are given the opportunity to decide whether to make the changes to the temporary definition files permanent.

Change (Shift f3)

This function is used to change a user record which is already present in the definition. When you press this key the attributes displayed on the terminal will be stored in the definition record for the user.

So to change any user record, bring it up on the terminal, enter the values as needed in desired fields and press the "Change" key.

Help (f4)

The Help key shows you a help message. The topic of help is determined by the cursor position when help key is pressed.

Delete (Shift f4)

This function key removes a user record. User record displayed on the terminal is deleted, if it is not the only user with application_owner privilege.

Undo (f5)

This key is used to undo the last valid Add, Change, or Delete User function performed.

AppDef (Shift f5)

This function key is provided to bring up a user record which has application default values in the following fields: root menu, initial menu, transaction, initial processor, and final processor. The application default values are obtained from the application definition. Blank fields that come up "blank" have no default value. You can change or enter values as needed. Be sure to enter a user name, and press the "add" key to add a new user record.

Refrsh (Shift f6)

This key is used to repaint the screen.

ShowGr (f7)

This function key is used when you want to look at a Group definition record. When this key is pressed the name for the "Owner Group Name" field of the form is used to select the group for display.

After you have examined the Group definition record, press Next to return to the Add/Change/Delete User screen.

ShowMn (Shift f7)

This function key is used when you want to look at a Menu definition record. When this key is pressed the menu code in the "Root Menu" or the "Initial Menu" field of the form is used to select the menu for display. If only the Root Menu or only the Initial Menu field has a menu code, (the other is "blank"), that is the menu record shown. If both Root and Initial are non-blank, place the cursor on the field to select the menu for display.

This key shows the menu definition record for the selected record, not the actual menu as it would appear on the screen when it is used. Press Next to return to the

Add/Change/Delete User screen.

NextEr (f8)

This key moves the cursor to the next field that is in error. It may be used after an update operation that is not completed due to errors in one or more field.

2.3.1.3 Add/change/delete (user) group

Every IM/Fast application definition has a set of group records. Each group record defines a group and assigns privilege, access level, and other attributes to a group of users. As many groups can be defined as needed.

A group is known in IM/Fast by a Group Name. A Group Name has the same form as a NOS/VE name. A group name can be defined only once in an IM/Fast application definition.

If you are the application administrator for an IM/Fast application and you want to establish a new group, all you have to do is to "add" the group definition record.

The ADD/CHANGE/DELETE GROUP option is selected when you want to work with the group records to:

- o Look at groups
- o Add new groups
- o Change existing groups
- o Delete groups

When you select this option a system transaction processor is executed. This processor lets you look at group records, one at a time, either sequentially or by group name, and lets you add, change, or delete records if you want. This processor only works in screen mode. And if you only have read access to the ADFs the Add, Change, and Delete function keys are shown in low intensity, telling you that using these keys will have no affect on the definition.

The ADD/CHANGE/DELETE GROUP screen is used to display a group definition record. The picture of this screen and the description of fields and function keys is given next. Remember, that you are only adding, changing, and deleting

records from a private copy of the ADF files. The changes to the shared application definition are not made until you exit this menu.

```
+------------------- Add/Change/Delete Group ---------------------------------+
|                                                                             |
|                                                                             |
|        Group Name: ______________________________                           |
|            Enable: yes (yes, no)                                            |
|  Owner Group Name: ______________________________                           |
|         Privilege: x user  _ admin  _ db_admin  _ owner  _ programmer        |
|       Access Level: 1_ (1..16)                                              |
|          Root Menu: ____                                                    |
|       Initial Menu: ____                                                    |
|        Transaction: ____                                                    |
| Initial Processor: ______________________________                           |
|   Final Processor: ______________________________                           |
|                                                                             |
+-----------------------------------------------------------------------------+

   FindGr                Change    Delete    AppDef    Refrsh    ShowMn
f1 NextGr f2  Add   f3 Back   f4  Help  f5  Undo  f6        f7 ShowOG f8 NextEr
```

Description of ADD/DELETE/CHANGE GROUP form fields.

Group Name

This field is used to enter or display a group name. Enter a group name here to add a new group or to find a group record. Ths system displays the group name of the current group record in this field.

Owner Group Name

This is the name of a group this group directly belongs to. A "blank" value in this field means the group does not belong to any group.

A User group definition has the same attribute fields, Enable, and Privilege through Final Processor, as a user definition. See Section 2.3.1.2 for a description of these fields. The value of each of these fields is taken from the user definition if nonblank, else from the group definition if nonblank, else from the owner group definition if nonblank, etc, else from the application definition.

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1.3 Add/change/delete (user) group

A description of ADD/DELETE/CHANGE GROUP form function keys follows:

NextGr (f1)

This function key displays the next group record, if one exists. The records are read by group name in alphabetical order. A message is displayed at the end of the group records.

FindGr (Shift f1)

This function key is used to locate and show a group definition record. The group name in the "Name" field is used as the search key.

Add (f2)

This function key is used to add a new group to the definition file. The "Name" field is used as the key for the record, and other values shown on the terminal are stored in the record.

If you want to add a new group record which has the same attributes as an existing group record, all you have to do is bring up the group record you want, enter the new group name in the "Name" field and press the "Add" key. If the group record with the same name doesn't already exist it will be added.

However, a record is not added if any field has an error or the group record already exists. The user is asked to correct the errors.

Back (f3)

This key returns you to IM/Fast. You are given the opportunity to decide whether to make the changes to the temporary definition files permanent.

Change (Shift f3)

This function is used to change a group record which is already present in the definition. When you press this key the attributes displayed on the terminal will be stored in the definition record for the group.

So to change any group record, bring it up on the

terminal, enter the values as needed in desired fields and press the "Change" key.

However, a record is not changed if any field has an error. The user is asked to correct any errors before the change can be made.

Help (f4)

Press the Help key to see a help message. The topic of help is determined by the cursor position when the help key is pressed.

Delete (Shift f4)

This function key removes a group definition record. The group whose record is on the terminal is deleted. Users belonging to a deleted group may not use the application.

Undo (f5)

This key is used to undo the last valid Add, Change, or Delete Group function performed.

AppDef (Shift f5)

This function key is provided to bring up a group record which has application default values in the following fields: root menu, initial menu, transaction, initial processor, and final processor. The application default values are obtained from the application definition. Fields that come up "blank" have no default value. You can change or enter values as needed, be sure to enter a group name, and press the "add" key to add a new group record.

Refrsh (Shift f6)

This function key repaints the screen.

ShowOG (f7)

This function key is used when you want to look at the owner group's definition record, if it exists. When this key is pressed the name from the "Owner Group Name" field is used to select the group for display.

After you have examined the owner group's definition record, press Next to return to the Add/Change/Delete Group screen.

ShowMn (Shift f7)

This function key is used when you want to look at a Menu definition record. When this key is pressed the menu code in the "Root Menu" or the "Initial Menu" field of the form is used to select the menu for display. If only the Root Menu or only the Initial Menu field has a menu code, (the other is "blank"), that is the menu record shown. If both Root and Initial are non-blank, place the cursor on the field to select the menu for display.

This key shows the menu definition record for the selected record, not the actual menu as it would appear on the screen when it is used. Press Next to return to the Add/Change/Delete Group screen.

NextEr (f8)

This key moves the cursor to the next field that is in error. It may be used after an update operation that is not completed due to errors in one or more field.

2.3.1.4 Add/change/delete transaction

Every IM/Fast application definition has a set of transaction records. Each transaction record defines an IM/Fast application transaction. As many transactions as needed can be defined.

A transactions is known in IM/Fast by a code. A transaction with the same code can be defined more than once. It is defined once for every menu that it appears in.

If you are the application administrator for an IM/Fast application and you want to establish a new transaction, all you have to do is to "add" the transaction definition record.

The ADD/CHANGE/DELETE TRANSACTION option is selected when you want to work with the transaction records to:

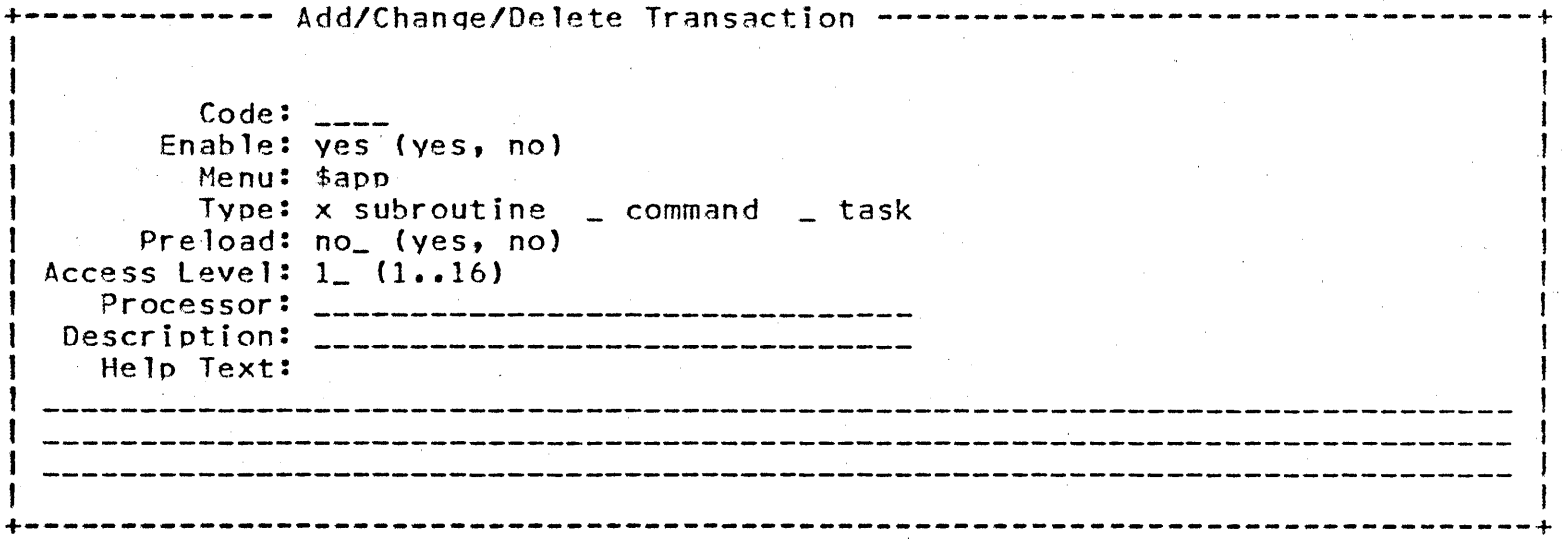
2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1.4 Add/change/delete transaction

- Look at transactions
- Add new transactions
- Change existing transactions
- Delete transactions

When you select this option a system transaction processor is executed. This processor lets you look at transaction records, one at a time, either sequentially or by transaction code, and lets you add, change, or delete records as you go along.

If proper access to change ADFs is not established for you, the Add, Change, and Delete function keys are shown in low intensity, telling you that using these keys will have no affect on the definition.

The ADD/CHANGE/DELETE TRANSACTION screen is used to display a transaction definition record. The picture of this screen and the description of fields and function keys is given next. Remember, that you are only adding, changing, and deleting records from a private copy of the ADF files. The changes to the shared application definition are not made until you exit this menu.



2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1.4 Add/change/delete transaction

A description of ADD/DELETE/CHANGE TRANSACTION form fields follows:

Code

The transaction code is a four character long NOS/VE name. It is used to name a transaction. More than one transaction can have the same code. More than one transaction with the same code cannot be added to the same menu, and a transaction code cannot be the same as a menu code.

Enable

This field indicates if the transaction is enabled. Only enabled transactions can be used. A YES value in this field indicates an enabled transaction.

Menu

This field has the menu in which this transaction will appear as an option. The transaction must belong to exactly one menu.

Type

These fields determine how a transaction is executed. A transaction can be executed as a subroutine, a command, or a NOS/VE task. A non-blank value in the field in front of the type name indicates the transaction has that type. A transaction may only have one type.

Preload

This field indicates if the transaction processor is to be loaded at the beginning of each user's session who has access to this transaction. A NO value in this field causes the processor to be loaded on the first request for it, causing a slight delay before execution begins.

Access Level

This field contains the transaction access level. Transaction access level is an integer between 1 and 16. The transaction access level and a user's access level determine if the user can run a transaction. If the user can't run a transaction it will not be shown in any of the user's menus. A user can run a transaction only if the transaction access level is less than or equal to the

user's access level.

Processor

This field contains the name of a processor. A processor of this name is executed when the transaction code is received by IM/Fast. A processor can be the name of a subroutine, program, or command. A processor name is required.

Description

This field contains a line of text. It can be used to describe the transaction. When the transaction code is displayed as a choice in a menu, the description is also displayed. See Section 1.4.14 for an example.

Help Text

This field contains the help information for the transaction. This message is shown when a user requests help and this transaction is the object of help. See Section 1.4.14 for an example.

A description of ADD/DELETE/CHANGE TRANSACTION form function keys follows:

NextTr (f1)

This function key displays the next transaction record, if one exists. The records are shown in the order they are defined. A message is displayed when the end of transaction records is reached.

FindTr (Shift f1)

This function key is used to find and show the desired transaction definition record. The transaction code in the "Code" field together with the menu code in the "Menu" field of the form are used to select the transaction record to display.

Add (f2)

This function key is used to add a new transaction to the definition file. The "Code" and "Menu" fields are used as the key for the record, and other values shown on the terminal are stored in the record.

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.1.4 Add/change/delete transaction

To add a new transaction record which has the same attributes as an existing transaction record, all you do is bring up the transaction record you want, enter the new transaction code in the "Code" field and press the "Add" key. If the transaction record doesn't already exist it will be added.

However, a record is not added if any field has an error or the transaction record already exists. The user is asked to correct the errors.

Back (f3)

This key returns you to IM/Fast. You are given the opportunity to decide whether to make the changes to the temporary definition files permanent.

Change (Shift f3)

This function is used to change a transaction record which is already present in the definition. When you press this key the attributes displayed on the terminal will be stored in the transaction record.

To change any transaction record, bring it up on the terminal, change the values as needed and press the "Change" key.

However, a record is not changed if any field has an error. The user is asked to correct any errors before a change can be made.

Help (f4)

Press the Help key to see a help message. The topic of help is selected by the cursor position when the key is pressed.

Delete (Shift f4)

This function key removes a transaction definition record. The transaction whose record is on the terminal is deleted.

Undo (f5)

This key is used to undo the last valid Add, Change, or Delete Transaction function performed.

Refrsh (Shift f6)

This function key repaints the screen.

ShowMn (Shift f7)

This function key is used to look at the owner menu record. When this key is pressed the name from the "Menu" field is used to select the menu record for display.

This key shows the menu definition record for the selected record, not the actual menu as it would appear on the screen when it is used. After you have examined the owner menu's definition record, press Next to return to the Add/Change/Delete Transaction screen.

NextEr (f8)

This function key moves the cursor to the next field in error.

2.3.1.5 Add/change/delete_menu

Every IM/Fast application definition has a set of menu records. Each menu record defines a menu in the IM/Fast application definition. One menu may contain a maximum of ten choices where each choice may be another menu or a transaction. The menu system may be a maximum of ten menus deep.

A menu is known in IM/Fast by a code. A menu code must not be the same as another menu code or a transaction code in the application.

The ADD/CHANGE/DELETE MENU option is selected when you want to work with menu records to:

- o Look at menus
- o Add new menus
- o Change existing menus
- o Delete menus

When you select this option a system menu processor is

executed. This processor lets you look at the menu records, one at a time, either sequentially or by menu code, and lets you add, change, or delete records as needed.

If proper access mode to change ADFs is not established for you, the Add, Change, and Delete function keys are shown in low intensity, telling you that using these keys will have no affect on the definition.

The ADD/CHANGE/DELETE MENU screen is used to display a menu definition record. The picture of this screen and the description of fields and function keys is given next. Remember, that you are only adding, changing, and deleting records from a private copy of the ADF files. The changes to the shared application definition are not made until you exit this menu.

```
+------------------- Add/Change/Delete Menu ---------------------------------+
|                                                                             |
|                                                                             |
|              Code: ____                                                     |
|            Enable: yes (yes, no)                                            |
|       Parent Menu: $app                                                     |
|             Title: ______________________________                           |
|      Access Level: 1  (1..16)                                               |
| Initial Processor: ______________________________                           |
|   Final Processor: ______________________________                           |
|       Description: ______________________________                           |
|         Help Text:                                                          |
| ----------------------------------------------------------------------------|
| ----------------------------------------------------------------------------|
| ----------------------------------------------------------------------------|
|                                                                             |
+-----------------------------------------------------------------------------+

   FindMn                Change    Delete                Refrsh    ShowPM
f1 NextMn f2  Add   f3  Back  f4  Help  f5  Undo  f6        f7        f8 NextEr
```

A description of ADD/CHANGE/DELETE MENU form fields follows:

Code

This is a 4 character NOS/VE name used for a menu. All menu codes are unique among menu and transaction codes of the application.

Enable

This field indicates if the menu can be used. A YES value in this field indicates that the menu (and all menus contained) may be used. If there is a NO value the menu cannot be used. A menu is not used if none of the options in it can be selected. $own menu may not be disabled.

Parent Menu

This is the menu code of the parent menu. Every menu except the $own menu must have a valid menu code in this field. A menu belongs to only one other menu. No cycles are allowed in the menu system.

Title

This is the title of this menu. Title is a text field which is shown on the menu form when it is displayed on the terminal.

Access Level

This field contains the access level of the menu.

Initial Processor

This field has the name of a menu Initial (prolog) processor. If blank there is no menu prolog processor.

Final Processor

This field has the name of a menu Final (epilog) processor. If blank there is no menu epilog processor.

Description

This field has a brief description of the menu. It is shown on the menu form when the parent menu is shown.

Help Text

This field has a help message for the menu. This message is shown when a user requests help and this menu is the object of help.

A description of ADD/CHANGE/DELETE MENU form function keys follows:

NextMn (f1)

This function key displays the next menu record, if one exists. The records are shown in the order of the menu code in alphabetical order.

FindMn (Shift f1)

This function key is used to find and show the desired menu definition record. The menu code in the "Code" field of the form is used to select the menu record to display.

Add (f2)

This function key is used to add a new menu to the definition file. The "Code" field is used as the key for the record, and other values shown on the terminal are stored in the record.

To add a new menu record with the same attributes as an existing menu record, all you do is bring up the menu record you want, enter the new menu code in the "Code" field and press the "Add" key. If the menu record doesn't already exist, it will be added.

However, a record is not added if any field has an error or the menu record already exists. The menu code must not be the same as another menu or transaction code. The user is asked to correct the errors.

Back (f3)

This key returns you to IM/Fast. You are given the opportunity to decide whether to make the changes to the temporary definition files permanent.

Change (Shift f3)

This function is used to change a menu record which already exists in the definition. When you press this key the attributes displayed on the terminal will be stored in the definition record for the menu.

To change any menu record, bring it up on the terminal, change the values as needed and press the "Change" key. The menu Code may not be changed.

However, a record is not changed if any field has an error. The user is asked to correct the errors.

Help (f4)

Use the Help key to see a help message. The topic of help is determined by the cursor position when the key is pressed.

Delete (Shift f4)

This function key removes a menu definition record. The menu record displayed on the terminal is deleted if it is not $own.

Undo (f5)

This key is used to undo the last valid Add, Change, or Delete Menu function performed.

Refrsh (Shift f6)

This function key repaints the screen.

ShowPM (Shift 7)

This function key is used when you want to look at the parent menu's definition record. When this key is pressed the name from the "Parent Menu" field is used to select the menu for display.

After you have examined the parent menu's definition record, press Next to return to the Add/Change/Delete Menu screen.

NextEr (f8)

This function key moves the cursor to the next field that has an error.

2.3.2 SETTING UP A NEW USER

Setting up a new user for an IM/Fast application requires that the following steps be followed:

o Assign a valid NOS/VE user name

This is done by the NOS/VE Family Administrator for your site. A NOS/VE user name and password is assigned and a permanent file catalog is created for the user. The user

name is used to log into NOS/VE.

o Add user to the application definition

Define the user to IM/Fast. Follow the procedure specified in section 2.3.1.2, Add/Change/Delete User, to add a new user record. Use the user name assigned by the NOS/VE family administrator as the user name under IM/Fast.

o Setup a PROLOG file in $USER catalog

The use of a user prolog file is optional. It may be used to define the terminal characteristics and invoke the IM/Fast application automatically. The SCL command necessary is SET_TERMINAL_ATTRIBUTES. Consult the SCL manual on the details of any command. As an example a user prolog may have the following commands:

```
SETTA TRM=cdc721....
FAST 'xx' yyyy
```

where xx is the application identifier and yyyy is the application name.

o Give user access to the application definition files

The user must be given proper permission so that IM/Fast may access the application definition files on behalf of the user. This may be done by allowing user access to the entire catalog that contains these files or permit each file individually to the user. Use the CREATE_FILE_PERMIT or the CREATE_CATALOG_PERMIT SCL commands for this.

The access and share modes depend on the user privilege. User's with APPLICATION_OWNER and/or APPLICATION_ADMINISTRATOR privileges must be able to read and write on the files and all other users (of this application) must be able to read the files.

The file permissions are given only once per user. The permissions must be changed when user's privilege changes, and removed if the user's record is deleted. For users with the application_administrator privilege use the following commands:

```
CREFP tff$define_application am=(read write) sm=(read write)...
CREFP tff$define_group       am=(read write) sm=(read write)...
CREFP tff$define_menu        am=(read write) sm=(read write)...
CREFP tff$define_transaction am=(read write) sm=(read write)...
```

```
CREFP tff$define_user          am=(read write) sm=(read write)  .
```

For other users of the application use the following command:

```
CREFP tff$define_application am=read sm=all
CREFP tff$define_group       am=read sm=all
CREFP tff$define_menu        am=read sm=all
CREFP tff$define_transaction am=read sm=all
CREFP tff$define_user        am=(read write) sm=(read write)
```

NOTE: Commands shown above are not complete. You may also have to specify other parameters such as "USER", "GROUP" or "FAMILY".

o Give user access to the system application directory file

IM/Fast creates a directory of known applications on $system.fast.tff$application_directory. This file is accessed in READ and WRITE mode when an application is created and when an owner is deleting or copying an application. In all other cases this file is accessed in READ mode. A user with proper access to the $system catalog must create the necessary permissions.

o Give users access to the global data area file

The global data area file is also accessed by IM/Fast for every user. File permissions necessary must be created for every user with proper access and share modes. The access and share modes are determined by the application developers. IM/Fast open request for the file is consistent with the permissions given the user. As an example, for writers of the global data area use the SCL command, CREFP tff$global_data_area am=(read write), and for readers of the global data area use, CREFP tff$global_data_area am=read. It is up to the application developer to determine what share mode to specify.

o Give user access to the database files

Each user of IM/Fast is an independent user of the files and/or the database. So each user must be given necessary permissions to these files according to the data manager being used to access the file. For keyed files for instance, permanent file permissions must be defined using the CREATE_FILE_PERMIT or the CREATE_CATALOG_PERMIT SCL commands. For DM database follow the directions in section 3.1.5.1 Using IM/DM Database.

o Give user access to the Application Library file

IM/Fast accesses this file for each user to load and execute transaction processors, SCL procedures, forms, and messages. This file must be available for each user in read and execute access modes.

2.3.3 MONITOR APPLICATION

This feature is not available in the first release of IM/Fast.

2.3.4 APPLICATION SCHEDULING

The Application Scheduling feature of NOS/VE supports a Site Defined Service Class concept. This feature may also be used for IM/Fast applications for improving their performance.

An active NOS/VE job, interactive or batch, belongs to a Service Class. A Service Class is site defined or defined by the system, and the attributes associated with a Service Class, such as priority and working set size, determine the level of service the job receives. For interactive jobs, the Service Class is determined at login time based on the user's NOS/VE Job Class. This Service Class is normally applied to every task of the job. However, the Service Class of a task may be different from the Service Class of the job, if application developers use NOS/VE Application Scheduling and a Site Defined Service Class.

The Service Class of an IM/Fast task may be set to raise or lower the task's priority if proper steps are taken by application developers. To use Application Scheduling consult the NOS/VE Application Scheduling ERS, DCS # A7575.

Note, that if the Application Scheduling feature is not used to explicitly change the Service Class of an IM/Fast application task, it will receive the same level of service from the system as any other NOS/VE task or command executed on behalf of a user.

2.3.5 APPLICATION ACCOUNTING

This feature is not available in the first release of IM/Fast.

2.3.6 OBJECT LIBRARY MAINTENANCE

This section describes the creation and maintenance of IM/Fast Application Object Library file. It also describes the $chl command. $chl command is required to invoke the new copy of a transaction processor after the Object Library containing it is changed.

The Application Object Library file must contain all transaction processors, forms, and message templates developed for an application.

2.3.6.1 Creating_Object_Library

Every IM/Fast application has one to three Object Libraries, and they are created before anyone can use the application. The object libraries may or may not exist when creating the application definition.

There may be more than one cycle of this file, however, IM/Fast uses the highest cycle every time the application is invoked. IM/Fast must be able to access this file on behalf of every valid user of the application. After creating this file give proper permissions to each user. More information on how to do this is contained in Section 2.3.2, Setting Up a New User.

An Object Library file is created and maintained with the Object Code Management utility (OCM) of NOS/VE. A complete description of OCM is contained in Object Code Management Usage manual, CDC publication number 60464413. Follow the OCM commands described there to create the Application Object Library file. Follow these general guidelines:

- Always create a new library file (a file with the next higher cycle number or use a new file name). An existing Object Library file can be modified only when it is not in use.

o When combining several load modules (programs) into one (larger) module on the Object Library consider the affect it has on the time it takes to load the module and how much of the loaded text is likely to be used.

IM/Fast loads transaction processor programs from this file. At the start of each session IM/Fast attaches the application object library file and adds it to the program library list. Then it loads all Sub-program type processors, in the user's menu system, with the Preload flag set to YES. Sub-program type processors which have the Preload flag set to NO are loaded when used.

The NOS/VE loader searches all object library files in the Program Description when satisfying subsequent load requests. If there is more than one file the order in which the files are searched is defined by the loader. However, files can be added to the Library List (SET_PROGRAM_ATTRIBUTES ADD_LIBRARY=....) before starting IM/Fast. This may cause a transaction processor not on the Application Object Library to be loaded from other files that happen to be in the Library List at the time. As a general rule all transaction processors should be on the Application Object Library.

2.3.6.2 Changing_Object_Library

An Application Object Library must be changed when a module has to be added, deleted, or updated. For example, the application object library must be changed when a change is made to an existing transaction processor.

Object Code Management (OCM) utility of NOS/VE is used for this also. However, when making changes the old Application Object Library file is used as input and a new file is created, with required changes.

The new Application Object Library file can be created (with changes) while the old file is being accessed by other users. Changes to object libraries cannot be made if it is in use.

When changed, only the new file will contain the changes. Any users active at the time the new file is created will continue to use the old file and would not be affected by the changes. IM/Fast provides the $chl command to load transaction processors from a new object library without terminating a session. The $chl command has the following

2.0 IM/FAST APPLICATION ADMINISTRATOR
2.3.6.2 Changing Object Library

format:

$chl <code> <library_file_path>

Where:

code	Is the transaction or menu code. This code identifies the menu for which the prolog and/or epilog processors have been changed or a transaction for which the processor has been changed. IM/Fast will load the processor specified for the transaction code, or if it is a menu code the prolog and epilog processors of the menu will be loaded from the specified new object library file for subsequent use. This is limited to the processors specified here and does not apply to Task or Command type processors.
library_file_path	This is the path name of the new object library file. The specified processors are loaded from this file when needed subsequent to this command.

You can execute the $chl command when IM/Fast menu form is on the terminal. Put $chl in the Enter Code field and parameters in the Parameters field and press Xecute.

2.3.7 APPLICATION LOG PROCESSING

Application log processing is not available in the first release of IM/Fast.

3.0 IM/FAST APPLICATION PROGRAMMING

3.0 IM/FAST APPLICATION PROGRAMMING

In this section you are going to find out how to write programs for IM/Fast applications. Programs are needed mainly to process input requests from users. These programs are one type of transaction processors. However, as a programmer you will be responsible for developing Session processors, Menu processors, command processors, and SCL procedures.

You should be familiar with how to write and debug programs and procedures using NOS/VE. You should also be able to maintain source and object code with NOS/VE Source Code Utility (SCU) and Object Code Management utility (OCM), respectively. For more information on SCU or OCM, please consult the following NOS/VE manuals:

- o SCU - SCL for NOS/VE Source Code Management Usage
  CDC Publication # 60464313

- o OCM - SCL for NOS/VE Object Cod Management Usage
  CDC Publication # 60464413

Most programs are written in the COBOL or FORTRAN languages. However, programming is restricted to these languages only in a few cases. The restrictions and other development options are explained later in this section.

3.1 PROGRAMMING CONCEPTS

The purpose of program development activity in IM/Fast is to create the Application Object Library, (Object Library for short). The object library contains load modules of programs, procedures, and forms that a user develops. Object libraries also contain message templates. IM/Fast places the object library in every user's program attributes to ensure that special programs and procedures that you have developed become available to all users of the application. Furthermore, all programs used from this library share the same code. IM/Fast also provides the capability so that you can change program modules that are being shared by active users.

Building the Application Object Library is a three step development process. The steps necessary are, first write the programs, then debug the programs, and finally place functional copies of programs on the object library file. Out

of these three steps only debug requires IM/Fast, others can essentially be done without it. NOS/VE provides you with several options that can be used for program development, such as the Professional Programming Environment (PPE) and the Programming Environment (PE). It is up to you to choose the method that best suits your needs.

IM/Fast provides the capability to integrate your procedures for developing programs and maintaining object library into the application menus. The $prg menu is provided for this purpose. This "Integration" of development and production allows programmers to do the development using the IM/Fast menus. This is needed and allows testing and debugging programs on-line.

3.1.1 PARAMETER PASSING

Session processors, menu prologs and epilogs, and transaction subroutines are sub-programs that are called by IM/Fast with the standard call/return mechanism. They can be COBOL sub-programs or FORTRAN subroutines. The same parameter list is passed for all of these sub-programs.

The parameters are explained below in detail. The parameter descriptions describe the parameters as they are passed to transaction subroutines. Where differences exist for session processors and menu prolog and epilogs, the differences are explained in sections 3.1.2 and 3.1.3.

In describing the parameters, a standard format is used; each parameter is given a type which tells if the parameter is read only or read/write followed by a description of the parameter. An INPUT parameter contains input data for the processor and the value of the parameter should not be changed. An OUTPUT parameter is interpreted by IM/Fast after the transaction processor completes. The contents of an OUTPUT parameter are undefined on input to the processor, i.e. the processor cannot depend on the incoming value. An INPUT/OUTPUT parameter supplies a value to the processor and is also interpreted by IM/Fast at completion. INPUT parameter types can be modified by the processors, but results are undefined.

COBOL Parameter Name	FORTRAN Parameter Name	Type - Description

3.0 IM/FAST APPLICATION PROGRAMMING
3.1.1 PARAMETER PASSING

tfv-transaction-code	tfcode	INPUT - This parameter contains the requested transaction code.
tfv-application-id	tfapid	INPUT/OUTPUT This parameter contains the application id on input. On OUTPUT this parameter contains the message template id, if different from the application id. Normally this parameter is not changed on output.
tfv-application-name	tfapnm	INPUT - This parameter contains name of the application.
tfv-user-name	tfusnm	INPUT - This parameter contains the user name of the current user.
tfv-family-name	tffmnm	INPUT - This parameter contains the family name of the current user.
tfv-date	tfdate	INPUT - This parameter contains the date when the transaction request was received by the system.
tfv-time	tftime	INPUT - This parameter contains the time of the current transaction request.
tfv-parameter-lines	tfparm	INPUT - The first two occurrences of this parameter contain the contents of the Parameters field on the Menu form. See Section 1.4.14 for an example of the Menu form. The last two occurrences of this parameter contain blanks and are for future use.
tfv-message-lines	tfmsg	INPUT/OUTPUT - This parameter contains a message

		that is displayed in the message form area on the terminal.
tfv-local-data-area	tfldat	INPUT/OUTPUT - This parameter is a reference to the local-data-area. This parameter is not accessed by IM/Fast.
tfv-global-data-area	tfgdat	INPUT/OUTPUT - This parameter is the reference to the global data area. The format and contents of the global data area are application dependent. This parameter is not accessed by IM/Fast.
tfv-interaction-mode	tfimod	INPUT - This parameter specifies if the interaction mode is BATCH or INTERACTIVE. If the interaction-mode is INTERACTIVE, input transactions are received form the terminal and if it is BATCH, the input is received from a file. This parameter is for future use.
tfv-interaction-style	tfisty	INPUT - This parameter specifies the style of interaction when the tfv-interaction-mode is INTERACTIVE. Interaction-style may be LINE or SCREEN. This parameter is for future use.
tfv-recovering-flag	tfrecf	INPUT - A "T" value in this parameter indicates that the transaction processor is being recovered. This parameter is for future use.
tfv-user-form-id	tfuser	INPUT - This parameter is the form identifier of a application provided form.

		Which is combined with the forms IM/Fast used for application specific data. This parameter is for future use.
tfv-processing-option	tfprop	OUTPUT - This parameter is for future use. It indicates processing option if the tfv-status-condition value is set to 0. The value of this parameter determines further action taken by IM/Fast. 0 - Continue normal processing; wait for next transaction request. 1 - Call specified transaction processor. The tfv-transaction-code field must be set to the desired transaction processor code. The contents of parameter tfv-message-lines are passed to the called transaction processor. 2 - Change user's current menu. The tfv-transaction-code field must be set to the desired menu code. 3 - Exit automatic menu.
tfv-status-parameter	tfstpm	OUTPUT - This parameter is set to the status parameters on output. Refer to section 3.2.1.1 for further details on this parameter.
tfv-status-condition	tfstcn	OUTPUT - This parameter is set to the status code value on output. Refer to section 3.2.1.1 for further details on this parameter.

3.0 IM/FAST APPLICATION PROGRAMMING
3.1.1 PARAMETER PASSING

To assist you in coding your COBOL sub-programs and FORTRAN subroutines, decks are provided in the SCU source library $system.common.psf$external_interface_source. For more information about SCU, please consult SCL for NOS/VE Source Code Management Usage, CDC Publication # 60464313.

Most of the examples in this ERS are COBOL examples; hence, the description of the COBOL decks is provided here. Refer to section 3.2.1.3 for a description of the FORTRAN decks.

Maintain your COBOL program as a deck on an SCU source library and put the following statements in your COBOL program:

```
*copy tfd$cobol_linkage
*copy tfp$cobol_procedure
```

Here is an example of using SCU to expand your COBOL program:

```
SCU
  USE_LIBRARY LIBRARY=your_library
  EXPAND_DECK DECK=your_deck ..
    ALTERNATE_BASE=$system.common.psf$external_interface_source ..
    COMPILE=your_expanded_deck
END NO
```

Your expanded COBOL program will contain the following PROCEDURE and LINKAGE section definitions:

```
LINKAGE SECTION.
01 tfv-transaction-code PIC X(04).
01 tfv-application-id PIC X(02).
01 tfv-application-name PIC X(31).
01 tfv-user-name PIC X(31).
01 tfv-family-name PIC X(31).
01 tfv-date PIC X(18).
01 tfv-time PIC X(09).
01 tfv-parameter-lines.
    02 tfv-parameter-line PIC X(60) OCCURS 4 TIMES.
01 tfv-message-lines.
    02 tfv-message-line PIC X(50) OCCURS 3 TIMES.
01 tfv-local-data-area.
    02 tfv-local-data PIC X(01) OCCURS 1000 TIMES.
01 tfv-global-data-area.
    02 tfv-global-data PIC X(01) OCCURS 1000 TIMES.
01 tfv-interaction-mode PIC X(11).
01 tfv-interaction-style PIC X(6).
01 tfv-recovering-flag PIC X(1).
```

```
        01 tfv-user-form-id PIC 9(18) COMP SYNC LEFT.
        01 tfv-processing-option PIC 9(18) COMP SYNC LEFT.
        01 tfv-status-parameter PIC X(31).
        01 tfv-status-condition PIC 9(18) COMP SYNC LEFT.
        PROCEDURE DIVISION USING
                tfv-transaction-code,
                tfv-application-id,
                tfv-application-name,
                tfv-user-name,
                tfv-family-name,
                tfv-date,
                tfv-time,
                tfv-parameter-lines,
                tfv-message-lines,
                tfv-local-data-area,
                tfv-global-data-area,
                tfv-interaction-mode,
                tfv-interaction-style,
                tfv-recovering-flag,
                tfv-user-form-id,
                tfv-processing-option,
                tfv-status-parameter,
                tfv-status-condition
                .
```

Deck tfd$cobol_linkage actually contains:

```
*copy tfd$cobol_linkage_begin
*copy tfd$cobol_local_data_area
*copy tfd$cobol_global_data_area
*copy tfd$cobol_linkage_end
```

The contents of tfd$cobol_local_data_area are merely an example of how the local data area could be defined. The contents of tfd$cobol_global_data_area are merely an example how the global data area could be defined. An application developer may define these areas as he wishes.

Here is an example of how an application developer could define the data areas differently from the default definitions:

```
*copy tfd$cobol_linkage_begin
        01 tfv-local-data-area.
           02 local-number pic 9(6).
           02 local-code pic x(6).
        01 tfv-global-data-area.
           02 global-number pic 9(6).
           02 global-code pic x(6).
```

3.0 IM/FAST APPLICATION PROGRAMMING
3.1.1 PARAMETER PASSING

```
*copy tfd$cobol_linkage_end
```

The actual contents of tfp$cobol_procedure are:

```
       PROCEDURE DIVISION USING
*copy tfd$cobol_procedure_parameters
                .
```

An application developer would use tfd$cobol_procedure_parameters when coding the statements which call a transaction subprogram from another transaction subprogram.

3.1.2 SESSION PROCESSORS (INITIAL/FINAL)

There are two session processors available in IM/Fast applications. They are both optional, either one or both can be specified on a per user basis.

Session processors are like user exit processing routines. The Initial session processor, if specified, is executed by IM/Fast every time a user starts to use the application. The Final session processor is executed, if one is specified, when the session is terminated with a QUIT command or due to an error.

It is a matter of application design to determine the need for session processors, if any, on a user by user basis, and what processing is required in each. IM/Fast does not impose any restriction on what can be done by session processors.

However, the role of session processors is significant for making the system responsive and minimizing resource usage. Session processors may be used in such a way that certain things are done only once, at the beginning of a user's session, and undone at the end. For example if you are using an IM/DM database, you may want to sign on to the database with the initial session processor and sign off the database in the final session processor. If you are using keyed files, you can attach and open the files in the initial session processor.

If the initial session processor returns an abnormal status, IM/Fast displays the abnormal status and then terminates. If the final session processor returns an abnormal status, IM/Fast displays the abnormal status and then continues the termination process.

3.0 IM/FAST APPLICATION PROGRAMMING
3.1.2 SESSION PROCESSORS (INITIAL/FINAL)

Session processors are sub-programs that are called by IM/Fast with the standard call/return mechanism. They can be COBOL sub-programs or FORTRAN subroutines. Parameters are passed to/from a session processor using the standard calling sequence supported under NOS/VE.

Refer to section 3.1.1 Parameter Passing of the ERS on how to use the definitions provided by IM/Fast in your COBOL or FORTRAN session processors to declare the parameters. The deck tfp$cobol_procedure which may be used to declare the parameters in a COBOL session processor expands to:

```
PROCEDURE DIVISION USING
        tfv-transaction-code,
        tfv-application-id,
        tfv-application-name,
        tfv-user-name,
        tfv-family-name,
        tfv-date,
        tfv-time,
        tfv-parameter-lines,
        tfv-message-lines,
        tfv-local-data-area,
        tfv-global-data-area,
        tfv-interaction-mode,
        tfv-interaction-style,
        tfv-recovering-flag,
        tfv-user-form-id,
        tfv-processing-option,
        tfv-status-parameter,
        tfv-status-condition
        .
```

A parameter that is different for session processors than for transaction subroutines is explained below in detail:

tfv-transaction-code	tfcode	This is an INPUT parameter to the processor. It contains the value '$ini' to indicate initial call, or '$fin' to indicate final call.

3.1.3 MENU PROCESSORS (PROLOG/EPILOG)

IM/Fast menus and any user defined menus may have prolog and epilog processors defined. Menu prolog and epilog processors are optional. Menu Prolog and Epilog processors are sub-programs provided by the application developers.

Menu prologs and epilogs are executed in hierarchical fashion. A menu prolog, if one is specified, is executed when the menu is selected for display when you are moving down in the menu system (away from the root menu). The menu epilog of the current menu is executed, if one is specified, when the current menu is left to display a different menu and you are moving up in the menu system (toward the root menu).

IM/Fast allows you to jump from one menu to another. IM/Fast automatically executes all required menu epilogs as it moves up in the menu system to a common node, then it automatically executes all required menu prologs as it moves down in the menu system to the new menu.

On initialization, IM/Fast automatically executes all required menu prologs from $own through the initial menu. On termination, IM/Fast automatically executes all required menu epilogs from the currently displayed menu through the $own menu.

If a menu prolog returns an abnormal status, the abnormal status is displayed, the process of moving from one menu to another is stopped, and the menu which is the parent of the menu whose prolog returned the abnormal status is displayed. If a menu epilog returns an abnormal status, the abnormal status is displayed. After the user acknowledges the abnormal status, the process of moving from one menu to another is continued.

Once again, IM/Fast does not restrict the processing that can be done by menu processors.

Menu processors, like session processors, can be used for any desired purpose determined by the application developers. Menu processors may be used to set up the execution environment for transaction processors that can be executed from it. For example, you may want to put transaction processors that access the same data file(s) in a given menu, open that file when the menu prolog is executed and close it when menu epilog is processed. By doing so the file is opened only when it is most likely to get used, and resources

required to support the open are tied up for the shortest possible duration.

Menu processors are sub-programs which are called by IM/Fast with the standard call/return mechanism. They can be COBOL sub-programs, or FORTRAN subroutines. Parameters are passed to/from a menu processor using the standard calling sequence supported under NOS/VE.

Refer to section 3.1.1 Parameter Passing of the ERS on how to use the definitions provided by IM/Fast in your COBOL or FORTRAN session processors to declare the parameters. The deck tfp$cobol_procedure which may be used to declare the parameters in a COBOL session processor expands to:

```
PROCEDURE DIVISION USING
        tfv-transaction-code,
        tfv-application-id,
        tfv-application-name,
        tfv-user-name,
        tfv-family-name,
        tfv-date,
        tfv-time,
        tfv-parameter-lines,
        tfv-message-lines,
        tfv-local-data-area,
        tfv-global-data-area,
        tfv-interaction-mode,
        tfv-interaction-style,
        tfv-recovering-flag,
        tfv-user-form-id,
        tfv-processing-option,
        tfv-status-parameter,
        tfv-status-condition
        .
```

A parameter that is different for menu processors than for transaction subroutines is explained below in detail:

tfv-transaction-code	tfcode	This is an INPUT parameter to the processor. It contains the value '$pro' to indicate prolog call, or '$epi' to indicate epilog processing call.

3.0 IM/FAST APPLICATION PROGRAMMING
3.1.3 MENU PROCESSORS (PROLOG/EPILOG)

IM/Fast pushes its forms by calling the fdp$push_forms NOS/VE Screen Formatting request before it executes a menu prolog or epilog. It pops its forms by calling fdp$pop_forms after it executes a menu prolog or epilog.

Menu prologs and epilogs must leave their application forms in such a state that IM/Fast's fdp$pop_forms call makes its forms available. CAUTION: If this is not done, no further processing is possible, and the terminal may hang.

3.1.4 USING GLOBAL SHARED DATA AREA

IM/Fast attaches the permanent file on which the global data area is stored with intent to read and write, if possible. The actual access mode granted by the system depends on the permissions that the user has to this file. Application developers must determine and give each user appropriate access and share modes (permission)to this file. As many users as needed may be given access to read and write on this file. So there can be several users with read/write permission running at once, and it is up to the application developer to handle how they control who can write on this file. The application cannot be used by a user who has no access to this file.

An attempt to access this file without proper permission causes IM/Fast to terminate the processor attempting to access the file and to display the current IM/Fast menu. An error message is displayed and must be acknowledged. The user may then execute another processor or execute the QUIT function.

3.1.5 DBMS CONSIDERATIONS

You can either use IM/DM, keyed files, or sequential files to implement your application database. The flexibility to use any data manager is available because IM/Fast does not interpret or process any calls you make from your program, including calls to the DBMS.

As a consequence of this some good and some bad things happen. The good things are:

- You have total control over the interface, and you may tailor it to your needs.

o There is only one program interface to work with; that which is provided by the data manager and not IM/Fast.

o You can use all the capabilities provided by the data manager, not just a subset.

The bad thing about this interface is that the developers have to do more work; they have to determine when each file should be opened and closed and then code the requests, to do so, in the appropriate processor.

3.1.5.1 Using_IM/DM_database

This section is to be written. The issues that will be addressed in this section are:

- o How to SIGNON to the IM/DM kernel
- o How to open/close the IM/DM database
- o How to use start/finish requests
- o How to read/write data
- o How to use transaction recovery
- o Compiling programs
- o A complete example showing all of the above

3.1.5.2 Using_Keyed_files

If your database includes keyed files, then you will be responsible for the following actions:

- o Attaching files for users in proper access modes.
- o Opening and closing files.
- o Reading/writing data on files.
- o Initializing keyed files

A brief discussion of each issue is given in this section. In the discussion that follows it is assumed that you are familiar with the NOS/VE permanent file system, and keyed files.

If you are not familiar with or just need more information on either the NOS/VE permanent file system or keyed files, please consult the following NOS/VE manuals:

- o Keyed file - CYBIL for NOS/VE Keyed file and Sort/Merge Interface.

CDC Publication # 60464117E

o Permanent file - CYBIL for NOS/VE File Management
CDC Publication # 60464114

ATTACHING/DETACHING KEYED FILES

A file must be attached first before it can be opened. Keyed files are attached for every user who needs them. Inquiry only transaction processors need access to read a file and update transaction processors need access to read and update a file. The mode of access when a file is attached must be consistent with the mode of access required by the available transaction processor(s).

Based on access requirements, file permissions must be granted to users for each file. This is done by creating the proper file permit entries. File permit entries may be created by System Command Language command (SCL) CREATE_FILE_PERMIT, described in the SCL Reference Manual.

Finally, files can be attached for users. There are four possible places where files may be attached: in the application prolog, in the session initial processor, the menu prolog processor, or the transaction processor. It is the responsibility of the developers to determine when and how to attach and detach files.

As an example, the following ATTACH_FILE command may be used to attach a file (aamfile) for a user (user-a) of the application with the read and write access modes:

```
ATTACH_FILE aamfile access_mode = (read modify)
```

Before attaching the file the CREATE_FILE_PERMIT command must be used to give the user the necessary permissions to this file.

```
CREATE_FILE_PERMIT aamfile user = user_a access_mode = (read modify)
```

The commands used in this simple example have more parameters than are shown above. To get more information on these commands and parameters consult the SCL Manual set.

OPENING AND CLOSING FILES

After a file is attached it must be opened to read and write the data in it. The file must be closed when it is

no longer needed. COBOL and FORTRAN languages provide statements to open and close files. An open request opens one file for one user with the specified access modes. Access modes at the time of open must be consistent with the access modes when the file is attached. For example, a file which is attached with only the read access mode cannot be opened with access mode of update.

Files can be opened and closed in either session processors, menu processors, or the transaction processors. Again, the developers must make the decision on where to open the files and what language to use. The language a file is opened in must be the same as the language in which it will be accessed. However, if necessary a file may be opened more than once.

READ/WRITE FILES

Data records are accessed by transaction processors mainly, and if needed, may also be accessed by session processors and menu processors also. The access method depends on the language of your processor, and the language that is used for opening the file. If the processor is in COBOL, then all standard COBOL statements described in the COBOL Usage manual may be used, and if the processor is in FORTRAN, standard statements described in the FORTRAN Usage manual may be used.

This also applies to other processing needed for files, such as explicit record/file locking and parcelling.

Next, there are examples that show how files are used in IM/Fast. The NOS/VE Source Code Utility (SCU) was used to store the various pieces of code contained in these examples. The term "deck" refers to a named unit of code managed by SCU.

USING KEYED FILE THROUGH COBOL - EXAMPLE

This example shows the use of a keyed file, called CUSTOMER, for an IM/Fast application. The file is used from COBOL transaction processors, so the entire application is in COBOL.

The example shows the SELECT clause used for the file first. The SELECT clause is a "standard" select clause, there is nothing special about it. This SELECT clause is used in all COBOL procedures that use the file. It is stored in the deck select_customer and is copied in a

program that uses it with the SCU command COPY.

The SELECT_CUSTOMER deck contains the following text:

```
SELECT customer-file ASSIGN TO "customer"
    ACCESS MODE IS dynamic
    ORGANIZATION IS indexed
    RECORD KEY IS cust-code OF customer
    FILE STATUS IS cust-file-status
.
```

The File Descriptor (FD) entry, which is also used in each procedure that uses this file, is shown next. The EXTERNAL verb used in the FD entry allows the instance of open for the file to be shared by several independently compiled and executed procedures. The text of the FD entry is stored in deck fd_customer and is copied in the program that uses it with the SCU command copyc.

The FD clause for CUSTOMER file is:

```
FD customer-file
    EXTERNAL
    LABEL RECORD IS omitted
    DATA RECORD IS customer
    RECORD CONTAINS 125 CHARACTERS
.

01  customer.
    03  cust-code              PIC  X(005).
    03  cust-name              PIC  X(030).
    03  cust-address           OCCURS 2 TIMES.
        05  cust-address-line  PIC X(030).
    03  cust-account           PIC  X(010).
    03  cust-credit-rating     PIC  X(010).
    03  cust-order             PIC  X(010).
```

The next thing shown is the procedure that opens and closes the CUSTOMER file. The procedure opens other files in the database also. But only customer file processing is included here. This procedure was declared as a menu prolog and epilog processor for one of the application menus.

The open/close procedure is:

3.0 IM/FAST APPLICATION PROGRAMMING
3.1.5.2 Using Keyed files

```
        IDENTIFICATION DIVISION.

        PROGRAM-ID.    OCOECL.
        AUTHOR.        Ravi Tavakley.
        DATE-WRITTEN.  04/30/86.
*******DATE-modify.   08/27/87.  Frank Chow

        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SOURCE-COMPUTER. CYBER-180.
        OBJECT-COMPUTER. CYBER-180.
        SPECIAL-NAMES.

        INPUT-OUTPUT SECTION.
        FILE-CONTROL.


        SELECT customer-file ASSIGN TO "customer"
            ACCESS MODE IS dynamic
            ORGANIZATION IS indexed
            RECORD KEY IS cust-code OF customer
            FILE STATUS IS cust-file-status
        .

        SELECT order-file ASSIGN TO "order"
            ACCESS MODE IS dynamic
            ORGANIZATION IS indexed
            RECORD KEY IS ordr-number OF orders
            FILE STATUS IS ordr-file-status
        .


        SELECT inventory-file ASSIGN TO "inventory"
            ACCESS MODE IS dynamic
            ORGANIZATION IS indexed
            RECORD KEY IS inve-number OF inventory
            FILE STATUS IS inve-file-status
        .

        DATA DIVISION.
        FILE SECTION.

        FD customer-file
            EXTERNAL
            LABEL RECORD IS omitted
            DATA RECORD IS customer
            RECORD CONTAINS 125 CHARACTERS
        .
```

```
01  customer.
    03  cust-code              PIC  X(005).
    03  cust-name              PIC  X(030).
    03  cust-address           OCCURS 2 TIMES.
        05  cust-address-line  PIC X(030).
    03  cust-account           PIC  X(010).
    03  cust-credit-rating     PIC  X(010).
    03  cust-order             PIC  X(010).

FD order-file
    EXTERNAL
    LABEL RECORD IS omitted
    DATA RECORD IS orders
    RECORD IS VARYING IN SIZE
      FROM 90 TO 1592 CHARACTERS
.

01  orders.
    03  ordr-number            PIC  X(010).
    03  ordr-cust-code         PIC  X(005).
    03  ordr-date              PIC  X(018).
    03  ordr-time              PIC  X(010).
    03  ordr-status            PIC  X(010).
    03  ordr-taker.
        05  shop-id            PIC  X(015).
        05  station-id         PIC  X(012).
        05  clerk-id           PIC  X(010).
    03  ordr-lines             PIC  9(002) COMP.
    03  ordr-line-items        OCCURS 0 TO 99 TIMES
                                 DEPENDING ON ordr-lines.
        05  ordr-inventory     PIC X(010).
        05  ordr-quantity      PIC X(005).


FD inventory-file
    EXTERNAL
    LABEL RECORD IS omitted
    DATA RECORD IS inventory
    RECORD CONTAINS 62 CHARACTERS
.

01  inventory.
    03  inve-number            PIC  X(010).
    03  inve-description       PIC  X(030).
    03  inve-unit-cost         PIC  X(010).
    03  inve-on-hand           PIC  X(006).
    03  inve-on-order          PIC  X(006).
```

```
WORKING-STORAGE SECTION.

01  file-status.
    03  ordr-file-status     PIC X(002).
        88  ordr-file-normal  VALUES ARE "00" THRU "02".
    03  cust-file-status     PIC X(002).
        88  cust-file-normal  VALUES ARE "00" THRU "02".
    03  inve-file-status     PIC X(002).
        88  inve-file-normal  VALUES ARE "00" THRU "02".
01  ve-command-line.
    03  attach-file          pic x(004) value "attf".
    03  filler               pic x(003) value " f=".
    03  attf-path            pic x(026) value  spaces.
    03  lfn-parameter        pic x(015) value spaces.
    03  am-parameter         pic x(007) value
                                          " am=all".
    03  sm-parameter         pic x(007) value
                                          " sm=all".
    03  filler               pic x(004) value spaces.

01  ve-command-line-2.
    03  command              pic x(004) value "detf".
    03  file-parameter       pic x(003) value ",f=".
    03  detf-path            pic x(007) value "$local.".
    03  file-name            pic x(015) value spaces.
    03  filler               pic x(005) value spaces.
01  status-variable.
    03  status-val           pic 9(001) usage comp value 0.
        88  it-worked                                value 1.
        88  it-failed                                value 0.
    03  product-code         pic x(002) value spaces.
    03  diag-no              pic 9(006) usage comp value 0.
    03  filler               pic x(256) value spaces.


LINKAGE SECTION.
01  tfv-transaction-code          PIC X(04).
01  tfv-application-id            PIC X(02).
01  tfv-application-name          PIC X(31).
01  tfv-user-name                 PIC X(31).
01  tfv-family-name               PIC X(31).
01  tfv-date                      PIC X(18).
01  tfv-time                      PIC X(09).
01  tfv-parameter-lines.
    02  tfv-parameter-line        PIC X(60) OCCURS 4 TIMES.
01  tfv-message-lines.
    02  tfv-message-line          PIC X(50) OCCURS 3 TIMES.
01  tfv-local-data-area.
    02  tfv-local-data            PIC X(01) OCCURS 1000 TIMES.
```

```
        01  tfv-global-data-area.
            02  tfv-global-data          PIC X(01) OCCURS 1000 TIMES.
        01  tfv-interaction-mode         PIC X(11).
        01  tfv-interaction-style        PIC X(6).
        01  tfv-recovering-flag          PIC X(1).
        01  tfv-user-form-id             PIC 9(18) COMP SYNC LEFT.
        01  tfv-processing-option        PIC 9(18) COMP SYNC LEFT.
        01  tfv-status-parameter         PIC X(31).
        01  tfv-status-condition         PIC 9(18) COMP SYNC LEFT.
        PROCEDURE DIVISION USING
                tfv-transaction-code,
                tfv-application-id,
                tfv-application-name,
                tfv-user-name,
                tfv-family-name,
                tfv-date,
                tfv-time,
                tfv-parameter-lines,
                tfv-message-lines,
                tfv-local-data-area,
                tfv-global-data-area,
                tfv-interaction-mode,
                tfv-interaction-style,
                tfv-recovering-flag,
                tfv-user-form-id,
                tfv-processing-option,
                tfv-status-parameter,
                tfv-status-condition
                .
        DECLARATIVES.

        CUSTOMER-TROUBLE SECTION.

        USE AFTER STANDARD EXCEPTION PROCEDURE on customer-file.
       *
       * This part traps all I/O errors which might occur during
       * processing of file CUSTOMER-FILE.
       *


        Return-I-O-status-of-cust.
          MOVE cust-file-status TO tfv-status-condition.
          MOVE "CUSTOMER"       TO tfv-status-parameter.
          Enter "CBP$IO_ERROR_NO_ABORT".
1006      EXIT PROGRAM.

        INVENTORY-TROUBLE SECTION.

        USE AFTER STANDARD EXCEPTION PROCEDURE on inventory-file.
```

```
      *
      * This part traps any error situation which might occur during
      * processing of file INVENTORY-FILE.
      *

       Return-I-O-status-of-inve.
         MOVE inve-file-status TO tfv-status-condition.
         MOVE "INVENTORY"      TO tfv-status-parameter.
         Enter "CBP$IO_ERROR_NO_ABORT".
1006     EXIT PROGRAM.

       ORDER-TROUBLE SECTION.

       USE AFTER STANDARD EXCEPTION PROCEDURE on order-file.
      *
      * This part traps any error situation which might occur during
      * processing of file ORDER-FILE.
      *

       Return-I-O-status-of-ordr.
         MOVE ordr-file-status    TO tfv-status-condition.
         MOVE "ORDER"             TO tfv-status-parameter.
         Enter "CBP$IO_ERROR_NO_ABORT".
1006     EXIT PROGRAM.

       END DECLARATIVES.

         MAIN-LINE.
           IF tfv-transaction-code IS EQUAL TO "$pro" THEN
tfc          MOVE " Welcome to ORDER menu system "    TO
tfc                                      tfv-message-line ( 2 )
tfc*****     display " open file "  tfv-local-data-area ( 41 : 20 )
             PERFORM 1000-open-files
                thru 1000-end-open-files.
           IF tfv-transaction-code IS EQUAL TO "$epi" THEN
tfc          MOVE " Exit from ORDER menu system "    TO
tfc                                      tfv-message-line ( 1 )
tfc*****     display " close file " tfv-local-data-area ( 41 : 20 )
             PERFORM 2000-close-files
                 thru 2000-end-close-files.
           EXIT PROGRAM.
         END-MAIN-LINE.

         1000-open-files.
           PERFORM 1000-open-order-file
              thru 1000-end-open-order-file
           IF not ordr-file-normal THEN
             GO TO 1000-end-open-files.
           PERFORM 1000-open-customer-file
```

```
              thru 1000-end-open-customer-file
            IF not cust-file-normal THEN
              GO TO 1000-end-open-files.
            PERFORM 1000-open-inventory-file
              thru 1000-end-open-inventory-file
          1000-end-open-files.
            EXIT.


          1000-open-order-file.
        *
        *  Detach the local file
        *
            move "order" to file-name.
            ENTER "clp$scan_command_line" using ve-command-line-2
                                                status-variable.
        *
        *  The file needs to be attached before open it
        *
  tfc******  move ".frankc.fast_dbf.order" to attf-path.
            move ".fast.prototype.order" to attf-path.
            move " lfn=order" to lfn-parameter.

            ENTER "clp$scan_command_line" using ve-command-line
                                                status-variable.

            OPEN I-O order-file.
          1000-end-open-order-file.
            EXIT.

          1000-open-customer-file.
        *
        *  Detach the local file
        *
            move "customer" to file-name.
            ENTER "clp$scan_command_line" using ve-command-line-2
                                                status-variable.

        *
        *  The file needs to be attached before open it
        *
  tfc******  move ".frankc.fast_dbf.customer" to attf-path.
            move ".fast.prototype.customer" to attf-path.
            move " lfn=customer" to lfn-parameter.

            ENTER "clp$scan_command_line" using ve-command-line
                                                status-variable.

            OPEN I-O   customer-file.
```

```
           1000-end-open-customer-file.
             EXIT.

           1000-open-inventory-file.
       *
       *   Detach the local file
       *
             move "inventory" to file-name.
             ENTER "clp$scan_command_line" using ve-command-line-2
                                                 status-variable.

       *
       *   Attach the inventory file
       *

             move ".fast.prototype.inventory" to attf-path.
  tfc******  move ".frankc.fast_dbf.inventory" to attf-path.
             move " lfn=inventory" to lfn-parameter.
             ENTER "clp$scan_command_line" using ve-command-line
                                                 status-variable.

             OPEN I-O   inventory-file.
           1000-end-open-inventory-file.
             EXIT.

           2000-close-files.
             PERFORM 2000-close-order-file
                thru 2000-end-close-order-file.
             IF not ordr-file-normal THEN
               GO TO 2000-end-close-files.
             PERFORM 2000-close-customer-file
                thru 2000-end-close-customer-file.
             IF not cust-file-normal THEN
               GO TO 2000-end-close-files.
             PERFORM 2000-close-inventory-file
                thru 2000-end-close-inventory-file.
           2000-end-close-files.
             EXIT.

           2000-close-order-file.
             CLOSE order-file.
           2000-end-close-order-file.
             EXIT.

           2000-close-customer-file.
             CLOSE customer-file.
           2000-end-close-customer-file.
             EXIT.
```

```
           2000-close-inventory-file.
             CLOSE inventory-file.

           2000-end-close-inventory-file.
             EXIT.

       END PROGRAM OCOECL.
```

The CUSTOMER file is opened (for reading and writing) when the menu is selected for display and the file is closed by this procedure when it is called to do menu epilog processing. Note that menu prolog and epilog processing can be done by a single program. This file is available for input/output processing to any processor that is executed by one of the transaction options in the menu. Also, transaction processors that declare this file as EXTERNAL may access this file, without opening it again.

Portions of a transaction processor that reads and writes records from this file is shown next.

The procedure is shown below:

```
       IDENTIFICATION DIVISION.

       PROGRAM-ID.    CUSTOMER.
       AUTHOR.        Ravi Tavakley.
       DATE-WRITTEN.  10/14/86.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. CYBER-180.
       OBJECT-COMPUTER. CYBER-180.
       SPECIAL-NAMES.

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.


       SELECT customer-file ASSIGN TO "customer"
           ACCESS MODE IS dynamic
           ORGANIZATION IS indexed
           RECORD KEY IS cust-code OF customer
           FILE STATUS IS cust-file-status
       .

       DATA DIVISION.
```

3.0 IM/FAST APPLICATION PROGRAMMING
3.1.5.2 Using Keyed files

```
FILE SECTION.

FD customer-file
    EXTERNAL
    LABEL RECORD IS omitted
    DATA RECORD IS customer
    RECORD CONTAINS 125 CHARACTERS
.

01  customer.
    03  cust-code              PIC  X(005).
    03  cust-name              PIC  X(030).
    03  cust-address           OCCURS 2 TIMES.
        05  cust-address-line  PIC X(030).
    03  cust-account           PIC  X(010).
    03  cust-credit-rating     PIC  X(010).
    03  cust-order             PIC  X(010).

WORKING-STORAGE SECTION.

77 at-end-value                PIC 9(001) VALUE IS 1.
77 not-at-end-value            PIC 9(001) VALUE IS 0.

77 quit-value                  PIC 9(001) VALUE IS 1.
77 not-quit-value              PIC 9(001) VALUE IS 0.

77 event-type-abnormal         PIC 9(001) VALUE IS 0.
77 event-type-normal           PIC 9(001) VALUE IS 1.

77 new-customer-value          PIC 9(001) VALUE IS 0.
77 old-customer-value          PIC 9(001) VALUE IS 1.

77 seq-read-value              PIC 9(001) VALUE IS 0.
77 ran-read-value              PIC 9(001) VALUE IS 1.

01  at-end-condition           PIC 9(001) VALUE IS 0.
    88 not-at-end              VALUE IS 0.
    88 at-end                  VALUE IS 1.

01  customer-type              PIC 9(001) VALUE IS 0.
    88 new-customer            VALUE IS 0.
    88 old-customer            VALUE IS 1.

01  quit-condition             PIC 9(001) VALUE IS 0.
    88 quit                    VALUE IS 1.
    88 not-quit                VALUE IS 0.

01  read-condition             PIC 9(001) VALUE IS 0.
    88 seq-read                VALUE IS 0.
```

```
            88 ran-read                  VALUE IS 1.

        01  form-event-type              PIC x(001).
            88 event-normal              VALUE IS "T".
            88 event-abnormal            VALUE IS "F".

        01  form-event-name              PIC X(031).
            88 event-next                VALUE IS
                                   "NEXT                           ".
            88 event-quit                VALUE IS
                                   "QUIT                           ".
            88 event-add                 VALUE IS
                                   "ADD                            ".
            88 event-change              VALUE IS
                                   "CHANGE                         ".
            88 event-delete              VALUE IS
                                   "DELETE                         ".
            88 event-next-r              VALUE IS
                                   "NEXT_RECORD                    ".
            88 event-prev-r              VALUE IS
                                   "PREVIOUS_RECORD                ".
            88 event-help                VALUE IS
                                   "HELP                           ".
            88 event-show                VALUE IS
                                   "SHOW                           ".

        01  form-name                    PIC X(031) VALUE IS
                         "CUSTOMER_FORM                  ".
        01  form-identifier              PIC 9(018) COMP SYNC LEFT
                                           VALUE 0.
  tfc   01  form-identifier-m            PIC 9(018) COMP SYNC LEFT
  tfc                                      VALUE 0.

        01  form-status                  PIC 9(018) COMP SYNC LEFT.
            88 form-status-normal        VALUE 0.

        01  form-xposition               PIC 9(018) COMP SYNC LEFT
                                           VALUE 1.
        01  form-yposition               PIC 9(018) COMP SYNC LEFT
                                           VALUE 0.
        01  form-object-xposition        PIC 9(018) COMP SYNC LEFT
                                           VALUE 1.
        01  form-object-yposition        PIC 9(018) COMP SYNC LEFT
                                           VALUE 0.
        01  screen-xposition             PIC 9(018) COMP SYNC LEFT
                                           VALUE 0.
        01  screen-yposition             PIC 9(018) COMP SYNC LEFT
                                           VALUE 0.
        01  form-event-location          PIC X(031).
```

```
            88 screen-event              VALUE IS
                                        "SCREEN                          ".
            88 variable-event            VALUE IS
                                        "VARIABLE                        ".
            88 command-event             VALUE IS
                                        "COMMAND                         ".
        01  form-event-object             PIC X(031).
        01  form-object-type              PIC 9(018) COMP SYNC LEFT.
            88  box-event                 VALUE IS 0.
            88  constant-text-event       VALUE IS 1.
            88  constant-text-box-event   VALUE IS 2.
            88  line-event                VALUE IS 3.
            88  variable-text-event       VALUE IS 4.
            88  variable-text-box-event   VALUE IS 5.

        01  form-event-occurrence         PIC 9(018) COMP SYNC LEFT.
        01  form-event-char-position      PIC 9(018) COMP SYNC LEFT.
        01  form-last-event-flag          PIC X(001).
            88  last-event                VALUE IS "T".
        01  form-validate-flag            PIC X(001) VALUE IS "T".
        01  form-unconverted-flag         PIC X(001).
        01  form-variable-status          PIC 9(018) COMP SYNC LEFT
                                            VALUE 0.
            88  variable-normal           VALUE IS 0.

        01  form-object-name              PIC X(031).

        01  form-object-occurrence        PIC 9(018) COMP SYNC LEFT
                                            VALUE IS 1.
        01  form-character-position       PIC 9(018) COMP SYNC LEFT
                                            VALUE IS 1.
        01  home-xposition                PIC 9(018) COMP SYNC LEFT
tfc                                         VALUE 5.
        01  home-yposition                PIC 9(018) COMP SYNC LEFT
tfc                                         VALUE 7.

        01  file-status.
            03  cust-file-status          PIC X(002).
                88  cust-file-normal      VALUES ARE "00" THRU "07".
            03  cust-file-status-no       REDEFINES cust-file-status
                                          pic 99.

        01  global-customer-code          PIC X(005) VALUE "     ".

        01  customer-form-open-flag       PIC X(001) VALUE "F".

        01  customer-form-added-flag      PIC X(001) VALUE "F".

        01  address-lines                 PIC 9(001) COMP VALUE 0.
```

```
01  first-cust-code                 PIC X(05).

01  save-cust-rec.
    03  save-cust-code              PIC X(05).
    03  save-info                   PIC X(120).

01  prev-cust-rec                   PIC X(125).
01  CUSTOMER-FORM                     .
    03 CUST-ADDRESS                      OCCURS  2.
       05  CUST-ADDRESS-LINE                 PIC X(30).
    03 CUST-ACCOUNT                     PIC X(10).
    03 CUST-CODE                        PIC X(5).
    03 CUST-CREDIT-RATING               PIC X(10).
    03 CUST-NAME                        PIC X(30).
    03 CUST-ORDER                       PIC X(10).

01  local-data-area.
    03  local-parameter             PIC X(040).
    03  local-message-form-no       PIC 9(018).
    03  local-message-form-flag     PIC X.
        88  message-form-added      value "T".
    03  local-user-name             PIC X(011).

LINKAGE SECTION.
01  tfv-transaction-code            PIC X(04).
01  tfv-application-id              PIC X(02).
01  tfv-application-name            PIC X(31).
01  tfv-user-name                   PIC X(31).
01  tfv-family-name                 PIC X(31).
01  tfv-date                        PIC X(18).
01  tfv-time                        PIC X(09).
01  tfv-parameter-lines.
    02  tfv-parameter-line          PIC X(60) OCCURS 4 TIMES.
01  tfv-message-lines.
    02  tfv-message-line            PIC X(50) OCCURS 3 TIMES.
01  tfv-local-data-area.
    02  tfv-local-data              PIC X(01) OCCURS 1000 TIMES.
01  tfv-global-data-area.
    02  tfv-global-data             PIC X(01) OCCURS 1000 TIMES.
01  tfv-interaction-mode            PIC X(11).
01  tfv-interaction-style           PIC X(6).
01  tfv-recovering-flag             PIC X(1).
01  tfv-user-form-id                PIC 9(18) COMP SYNC LEFT.
01  tfv-processing-option           PIC 9(18) COMP SYNC LEFT.
01  tfv-status-parameter            PIC X(31).
01  tfv-status-condition            PIC 9(18) COMP SYNC LEFT.
PROCEDURE DIVISION USING
        tfv-transaction-code,
```

```
                tfv-application-id,
                tfv-application-name,
                tfv-user-name,
                tfv-family-name,
                tfv-date,
                tfv-time,
                tfv-parameter-lines,
                tfv-message-lines,
                tfv-local-data-area,
                tfv-global-data-area,
                tfv-interaction-mode,
                tfv-interaction-style,
                tfv-recovering-flag,
                tfv-user-form-id,
                tfv-processing-option,
                tfv-status-parameter,
                tfv-status-condition
                .
        DECLARATIVES.

        TROUBLE SECTION.

            USE AFTER STANDARD EXCEPTION PROCEDURE on customer-file
       *
       * This part traps any error situation which might occur during
       * processing of file CUSTOMER-FILE.
       *

        Return-I-O-status.
TFC****     MOVE ordr-file-status       TO file-error-code.
tfc****     MOVE file-error-message     TO tfv-status-parameter.
            MOVE "CUSTOMER"             TO tfv-status-parameter.
1006        ADD 200 cust-file-status-no GIVING tfv-status-condition.
1006        PERFORM 4000-close-forms
            Enter "CBP$IO_ERROR_NO_ABORT".
            EXIT PROGRAM.

        END DECLARATIVES.

        1000-transaction SECTION.

          1000-begin-transaction.

            PERFORM 1000-start-up
            PERFORM 2000-customer-processing thru 2000-exit
            PERFORM 4000-delete-forms
            EXIT PROGRAM
            .
```

```
           1000-start-up.
               MOVE not-at-end-value     TO at-end-condition
               PERFORM 4000-prepare-form
               IF tfv-parameter-line (1) (1 : 5) IS NOT EQUAL space THEN
                  MOVE tfv-parameter-line (1) ( 1 : 10 )
                                         TO global-customer-code
                 MOVE global-customer-code TO cust-code OF customer
                 PERFORM 5000-read-customer-by-key
               .

         2000-customer SECTION.

           2000-customer-processing.
             MOVE not-quit-value TO quit-condition
             PERFORM 4000-set-form-defaults
tfc          MOVE space                    TO tfv-message-lines
tfc          MOVE " Begin CUSTOMER processing.."
tfc                                        TO tfv-message-line (1)
             IF old-customer THEN
               PERFORM 3000-move-record-to-form
             ELSE
               MOVE global-customer-code TO
                       cust-code OF customer-form
               PERFORM 5000-rewind-customers
               PERFORM 5000-read-next
               PERFORM 3000-move-record-to-form
               MOVE cust-code of customer TO first-cust-code
             .
             PERFORM 3000-process-customer UNTIL quit
             .

           2000-exit.
             EXIT.

           3000-process-customer.
             move 0 to file-status.
             PERFORM 4000-show-form.
             PERFORM 3000-process-events
             .

           3000-process-events.
             IF event-next   THEN PERFORM 3000-next-event.
             IF event-quit   THEN PERFORM 3000-quit-event.
             IF event-help   THEN PERFORM 3000-help-event.
             IF event-add    THEN PERFORM 3000-add-event.
             IF event-change THEN PERFORM 3000-change-event.
             IF event-delete THEN PERFORM 3000-delete-event.
             IF event-next-r THEN PERFORM 3000-next-record-event.
             IF event-prev-r THEN PERFORM 3000-prev-record-event.
```

```
              IF event-show   THEN PERFORM 3000-show-record-event.

            3000-next-event.
              MOVE not-quit-value TO quit-condition
              .

            3000-next-record-event.
              MOVE cust-code OF customer-form TO
                                   cust-code of customer
              MOVE cust-code OF customer-form TO
                                   global-customer-code
              PERFORM 4000-set-form-defaults
              IF not-at-end THEN
                PERFORM 5000-read-next
                IF at-end THEN
                  MOVE "At end of customer file     " TO
                                         tfv-message-line (3)
                ELSE
                  PERFORM 3000-move-record-to-form
                  MOVE "Next record..Select function by key." TO
                                         tfv-message-line (3)
                END-IF
              .

            3000-quit-event.
  tfc         MOVE  space                      TO tfv-message-lines
              MOVE "CUSTOMER - Processing complete.."
                                         TO tfv-message-line (3)
              MOVE quit-value TO quit-condition
              .

            3000-add-event.
              PERFORM 3000-move-form-to-record
              PERFORM 5000-write-customer-record
              IF NOT cust-file-normal THEN
                MOVE "Error detected ..." TO tfv-message-line (3)
              ELSE
                MOVE "Customer record added." TO tfv-message-line (3)
              .
              MOVE not-quit-value TO quit-condition
              .

            3000-change-event.
              PERFORM 3000-move-form-to-record
              PERFORM 5000-rewrite-customer-record
              IF NOT cust-file-normal THEN
                MOVE "Error detected ..." TO tfv-message-line (3)
              ELSE
                MOVE "Customer record changed." TO tfv-message-line (3)
```

```
      .
      MOVE not-quit-value TO quit-condition
      .

    3000-delete-event.
      PERFORM 5000-delete-customer-record
      IF NOT cust-file-normal THEN
        MOVE "Error detected ..." TO tfv-message-line (3)
      ELSE
        MOVE "Customer record deleted." TO tfv-message-line (3)
      .
      MOVE not-quit-value TO quit-condition
      .

    3000-help-event.
      MOVE "CUST HELP not available yet." TO
            tfv-message-line (3)
      MOVE not-quit-value TO quit-condition
      .

    3000-prev-record-event.
      IF ran-read THEN
        MOVE customer TO save-cust-rec
        MOVE LOW-VALUES to cust-code of customer
        PERFORM 5000-rewind-customers
        PERFORM 5000-read-next
          WITH TEST AFTER
            UNTIL (cust-code of customer = save-cust-code)
      END-IF
      .
      IF cust-code of customer = first-cust-code
        MOVE "This is the first record of the file  .. " TO
                  tfv-message-line (3)
      ELSE
        MOVE prev-cust-rec TO customer
        PERFORM 5000-read-customer-by-key
        PERFORM 3000-move-record-to-form
        MOVE ran-read-value to read-condition
        MOVE not-quit-value to quit-condition
        MOVE "Previous record found .. " TO
                        tfv-message-line (3)
      END-IF
        .


    3000-show-record-event.
      MOVE cust-code OF customer-form TO
                           cust-code of customer
      MOVE cust-code OF customer-form TO
```

```
                                global-customer-code
      PERFORM 4000-set-form-defaults
      PERFORM 5000-read-customer-by-key
      IF old-customer THEN
        PERFORM 3000-move-record-to-form
        MOVE "Customer record found..." TO
                tfv-message-line (3)
      ELSE
        MOVE "Customer record not found.." TO
                tfv-message-line (3)
        .
      MOVE not-quit-value TO quit-condition
      .

    3000-move-record-to-form.
      MOVE CORRESPONDING customer TO customer-form
      PERFORM 3000-move-address-to-form
              WITH TEST AFTER
                VARYING address-lines FROM 1 BY 1
                  UNTIL address-lines EQUALS 2
      .
      MOVE cust-code OF customer TO global-customer-code
      .
      MOVE "                                      " TO
                 tfv-message-line (3)
      .

    3000-move-address-to-form.
      MOVE cust-address-line OF customer (address-lines)
        TO cust-address-line OF customer-form (address-lines)
      .

    3000-move-form-to-record.
      MOVE CORRESPONDING customer-form TO customer
      PERFORM 3000-move-address-to-record
              WITH TEST AFTER
                VARYING address-lines FROM 1 BY 1
                  UNTIL address-lines EQUALS 2
      MOVE cust-code OF customer-form TO global-customer-code
      .

    3000-move-address-to-record.
      MOVE cust-address-line OF customer-form (address-lines)
        TO cust-address-line OF customer (address-lines)
      .

  4000-customer-form SECTION.

    4000-prepare-form.
```

```
              MOVE tfv-local-data-area    TO local-data-area.

              IF customer-form-open-flag IS EQUAL TO "F" THEN
                PERFORM 4000-open-form
                .
              IF customer-form-added-flag IS EQUAL TO "F" THEN
                PERFORM 4000-add-form
                .
              PERFORM 4000-position-form
              .

            4000-add-form.
tfc           MOVE  local-message-form-no  TO form-identifier-m.
tfc******     display " code = " tfv-transaction-code
tfc           IF  NOT  message-form-added
                  CALL "ADDMSF"    USING
                  tfv-transaction-code,
                  tfv-application-id,
                  tfv-application-name,
                  tfv-user-name,
                  tfv-family-name,
                  tfv-date,
                  tfv-time,
                  tfv-parameter-lines,
                  tfv-message-lines,
                  tfv-local-data-area,
                  tfv-global-data-area,
                  tfv-interaction-mode,
                  tfv-interaction-style,
                  tfv-recovering-flag,
                  tfv-user-form-id,
                  tfv-processing-option,
                  tfv-status-parameter,
                  tfv-status-condition
tfc           END-IF

              CALL "FDP$XADD_FORM" USING form-identifier, form-status
              IF NOT form-status-normal THEN
                MOVE "F" TO customer-form-added-flag,
                  GO TO 9000-abnormal-form-add
              ELSE
                MOVE "T" TO customer-form-added-flag
              .

            4000-close-forms.
              IF customer-form-added-flag EQUALS "T" THEN

tfc             PERFORM 4000-delete-forms
```

```
                CALL "FDP$XCLOSE_FORM" USING form-identifier, form-  atus
                IF NOT form-status-normal THEN
                  GO TO 9000-abnormal-form-close
                END-IF

                MOVE "F" TO customer-form-open-flag
              END-IF
              .

            4000-delete-forms.
              IF customer-form-added-flag EQUALS "T" THEN
                CALL "FDP$XDELETE_FORM" USING form-identifier, form-status
                IF NOT form-status-normal THEN
                    GO TO 9000-abnormal-form-delete
                END-IF
                MOVE "F" TO customer-form-added-flag

tfc             IF  tfv-transaction-code  =  "cust"
tfc               CALL "ADDMSF"    USING
                  tfv-transaction-code,
                  tfv-application-id,
                  tfv-application-name,
                  tfv-user-name,
                  tfv-family-name,
                  tfv-date,
                  tfv-time,
                  tfv-parameter-lines,
                  tfv-message-lines,
                  tfv-local-data-area,
                  tfv-global-data-area,
                  tfv-interaction-mode,
                  tfv-interaction-style,
                  tfv-recovering-flag,
                  tfv-user-form-id,
                  tfv-processing-option,
                  tfv-status-parameter,
                  tfv-status-condition
tfc             END-IF

              END-IF
              .

            4000-open-form.
              CALL "FDP$XOPEN_FORM" USING form-name,
                                          form-identifier, form-status
              IF NOT form-status-normal THEN
                MOVE "F" TO customer-form-open-flag,
                  GO TO 9000-abnormal-form-open
              ELSE
```

```
              MOVE "T" TO customer-form-open-flag
            END-IF
            .

          4000-position-cursor.
          4000-end-position-cursor.

          4000-position-form.
            CALL "FDP$XPOSITION_FORM" USING form-identifier,
                   home-xposition, home-yposition, form-status
            IF NOT form-status-normal THEN
              GO TO 9000-abnormal-form-position
            END-IF
            .

          4000-show-form.
            CALL "FDP$XREPLACE_RECORD" USING
  tfc                                 form-identifier-m,
                                       tfv-message-lines,
                                        form-variable-status,
                                         form-status
            IF NOT form-status-normal THEN
              GO TO 9000-abnormal-form-replace
              .
            CALL "FDP$XREPLACE_RECORD" USING
                                      form-identifier,
                                       customer-form,
                                        form-variable-status,
                                         form-status
            IF NOT form-status-normal THEN
              GO TO 9000-abnormal-form-replace
              .
            IF NOT variable-normal THEN
              GO TO 9000-abnormal-form-variable
              .

            MOVE "CUST_CODE                    " TO form-object-name
            MOVE 1 TO form-object-occurrence, form-character-position
            CALL "FDP$XSET_CURSOR_POSITION" USING
                  form-identifier,
                    form-object-name, form-object-occurrence,
                      form-character-position, form-status
            IF NOT form-status-normal THEN
              GO TO 9000-abnormal-form-cursor
              .

            CALL "FDP$XREAD_FORMS"  USING form-status
            IF NOT form-status-normal THEN
              GO TO 9000-abnormal-form-read
```

```
            .
          CALL "FDP$XGET_NEXT_EVENT" USING
            form-event-name, form-event-type,
              screen-xposition, screen-yposition,
                form-identifier, form-xposition, form-yposition,
                  form-event-location, form-event-object,
                    form-event-occurrence, form-event-char-position,
                      form-object-type, form-object-xposition,
                        form-object-yposition, form-last-event-flag,
                          form-status
          IF NOT form-status-normal THEN
            GO TO 9000-abnormal-form-event
            .
          IF event-normal THEN
            CALL "FDP$XGET_RECORD" USING form-identifier,
                                         customer-form,
                                          form-variable-status,
                                           form-status
            IF NOT form-status-normal THEN
              GO TO 9000-abnormal-form-get
              .
            IF NOT variable-normal THEN
              GO TO 9000-abnormal-form-get
            .
        4000-set-form-defaults.
          CALL "FDP$XRESET_FORM" USING form-identifier, form-status
          IF NOT form-status-normal THEN
            GO TO 9000-abnormal-form-reset
            .
          CALL "FDP$XGET_RECORD" USING form-identifier, customer-form,
                                       form-variable-status, form-status
          IF NOT form-status-normal THEN
            GO TO 9000-abnormal-form-get
            .
          IF NOT variable-normal THEN
            GO TO 9000-abnormal-form-get
          .
      5000-file SECTION.
        5000-rewind-customers.
          START customer-file KEY IS NOT LESS THAN
                cust-code of customer
                                INVALID KEY MOVE "customer file empty "
                                          to tfv-message-line (3)
            .
```

```
       5000-read-customer-by-key.
         MOVE old-customer-value TO customer-type.
         MOVE not-at-end-value TO at-end-condition
         READ customer-file RECORD
           INVALID KEY MOVE new-customer-value TO customer-type
         MOVE ran-read-value TO read-condition
           .

       5000-read-next.
         MOVE customer to prev-cust-rec.
         READ customer-file NEXT RECORD
                                  AT END PERFORM 5000-read-end-of-file
         MOVE seq-read-value TO read-condition


       5000-read-end-of-file.
         MOVE at-end-value TO at-end-condition
         .

       5000-delete-customer-record.
         DELETE customer-file INVALID KEY
           MOVE "Record cannot be deleted or found "
                               to tfv-message-line (3)
           MOVE 50 to cust-file-status
         .

       5000-rewrite-customer-record.
         REWRITE customer INVALID KEY
           MOVE "Record cannot be rewrited or found "
                               to tfv-message-line (3)
           MOVE 50 to cust-file-status
         .
       5000-write-customer-record.
         WRITE customer INVALID KEY
           MOVE "INVALID KEY error detected "
                               to tfv-message-line (3)
           MOVE 50 to cust-file-status
           .

     9000-termination SECTION.

       9000-abnormal-form.
         MOVE 1 TO tfv-status-condition
         MOVE "Form initialization error - customer-form"
           TO tfv-message-line (3)
         PERFORM 4000-close-forms.
         EXIT PROGRAM.
```

```
9000-abnormal-form-add.
  MOVE 2 TO tfv-status-condition
  MOVE "Form ADD error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-close.
  MOVE 3 TO tfv-status-condition
  MOVE "Form CLOSE error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-cursor.
  MOVE 4 TO tfv-status-condition
  MOVE "Form CURSOR error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-delete.
  MOVE 5 TO tfv-status-condition
  MOVE "Form DELETE error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-event.
  MOVE 6 TO tfv-status-condition
  MOVE "Form EVENT error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-open.
  MOVE 7 TO tfv-status-condition
  MOVE "Form OPEN error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-position.
  MOVE 8 TO tfv-status-condition
  MOVE "Form POSITIONING error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.
```

```
9000-abnormal-form-read.
  MOVE 9 TO tfv-status-condition
  MOVE "Form READ error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-replace.
  MOVE 10 TO tfv-status-condition
  MOVE "Form REPLACE error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-reset.
  MOVE 11 TO tfv-status-condition
  MOVE "Form RESET error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-variable.
  MOVE 12 TO tfv-status-condition
  MOVE "Form VARIABLE error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-form-get.
  MOVE 13 TO tfv-status-condition
  MOVE "Form GET error - customer-form"
    TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-file.
  MOVE 50 TO tfv-status-condition
  MOVE "File initialization error"
                              TO tfv-message-line (3)
  PERFORM 4000-close-forms.
  EXIT PROGRAM.

9000-abnormal-delete.
  MOVE 51 TO tfv-status-condition
  MOVE "Record does not exist, or error on DELETE."
                              TO tfv-message-line (3)
  PERFORM 4000-close-forms
  EXIT PROGRAM.
```

```
           9000-abnormal-file-next.
             MOVE 52 TO tfv-status-condition
             MOVE "End-of-file, or error on READ-NEXT"
                                          TO tfv-message-line (3)
             PERFORM 4000-close-forms
             EXIT PROGRAM.

           9000-abnormal-file-read.
             MOVE 53 TO tfv-status-condition
             MOVE "Record does not exist, or error on READ"
                                          TO tfv-message-line (3)
             PERFORM 4000-close-forms
             EXIT PROGRAM.

           9000-abnormal-rewrite.
             MOVE 54 TO tfv-status-condition
             MOVE "Record does not exist, or error on REWRITE"
                                          TO tfv-message-line (3)
             PERFORM 4000-close-forms
             EXIT PROGRAM.

           9000-abnormal-write.
             MOVE 55 TO tfv-status-condition
             MOVE "Record already exist, or error on WRITE"
                                          TO tfv-message-line (3)
             PERFORM 4000-close-forms
             EXIT PROGRAM.
         END PROGRAM CUSTOMER.
```

Note, the example programs included above may not be compiled and used without modification. In order to focus attention on a given area, important parts of code had to be removed. However, the examples are extracts of real live procedures and files, in use in an IM/Fast application at the time of this writing.

3.1.5.3 Using Screen Formatting

This section will be added. The following information will be included:

- o User defined form
- o Defining and using other forms

3.2 DEVELOPING TRANSACTION PROCESSORS

The transaction processors do most of your application oriented processing, such as "taking orders" in an order entry application, or "taking a deposit" in a banking application. The processing done by a transaction processor is a function of the application.

Usually, a transaction processor is designed to do just one thing. The operation performed represents a logical unit of work which must be identified, tracked, and accounted for.

IM/Fast transaction processors are named when transactions are defined. The same transaction processor can be named on multiple transaction definitions. Each transaction processor is of a particular type.

As an example consider the definition of a transaction for order entry purpose. Let's call it ORDX. The definition of ORDX is shown below. The definition assigns values to different attributes of the transaction. The description of all attributes is included in section 2.3.1.4.

```
+------------- Add/Change/Delete Transaction ---------------------------------+
|                                                                              |
|         Code: ordx                                                           |
|       Enable: yes (yes, no)                                                  |
|         Menu: oecl                                                           |
|         Type: x subroutine  _ command  _ task                                |
|      Preload: yes (yes, no)                                                  |
| Access Level: 1_ (1..16)                                                     |
|    Processor: orderx_________________________                                |
|  Description: Take Order on-screen___________                                |
|    Help Text:                                                                |
| Execute ordx to take orders on screen. You may enter the order number as the |
| first 10 characters in the Parameters field. If you do not enter an order___ |
| number, the first order is displayed.  After taking orders, press Back._____ |
|                                                                              |
+------------------------------------------------------------------------------+
```

The transaction code ORDX identifies the transaction. ORDX will appear as an option in the "owner" menu OECL, along with a brief description of it provided on the "description" parameter. Any user who has access to the OECL menu could run ORDX transaction, by typing the transaction code ORDX.

When IM/Fast receives this request, it locates the transaction definition for ORDX (which is shown above). From

the definition it picks up the name of the processor and it's type. In this example the processor name is ORDERX and it is of type sub-routine. This tells IM/Fast to call processor ORDERX with a standard sub-routine call. Other processor types are COMMAND and TASK. For COMMAND type of processor, IM/Fast calls an SCL command, and uses the processor name as the command name. For TASK type of processor, IM/Fast calls a NOS/VE task. In this case the processor name is used as the entry point name for the task.

Before actually calling the processor, IM/Fast optionally logs the beginning of the transaction processor if necessary. When the processor completes, the completion event is also optionally recorded on the log. Transaction logging is not supported in the first release of IM/Fast.

The interface between IM/Fast and transaction processors, and other processing constraints for each type of processor is the topic of discussion next.

3.2.1 SUBROUTINE TYPE TRANSACTION PROCESSOR

A transaction processor of TYPE subroutine is executed by IM/Fast with a procedure (sub-program) call statement. The procedure call that IM/Fast uses works for COBOL and FORTRAN alike.

After completing the designated processing, a sub-program returns control back to IM/Fast with the equivalent of a RETURN call. Normally, this takes on different forms in different languages. So the actual call is left for the language specific discussion. COBOL processors are described in section 3.2.1.2. FORTRAN processors are described in section 3.2.1.3.

3.2.1.1 Status_processing_and_message_handling

NOS/VE provides a general mechanism for message handling. This message handling capability is used throughout NOS/VE and is described in Chapter 6 of the CYBIL for NOS/VE System Interface manual, CDC publication number 60464115. IM/Fast uses this facility also to process messages required by an application.

One place where this facility is invoked is when a

transaction processor of the sub-program type returns to IM/Fast, and indicates the need to construct and display a message, i.e., when a non zero value stored in a status condition parameter. Actually three parameter values are used by IM/Fast to construct a complete message. Their COBOL names are tfv-application-id, tfv-status-condition, and tfv-status-parameter; their FORTRAN names are tfapid, tfstpm, and tfstcn.

IM/Fast constructs a general status record with these values, with a call to OSP$SET_STATUS_ABNORMAL. Parameter tfv-application-id is used as the identifier, and tfv-status-condition as the condition code.If any status parameter is specified by the tfv-status-parameter it is appended to the text field of the status record by calling OSP$APPEND_STATUS_PARAMETER.

A message is generated according to the current setting of the message level parameter of the job. IM/Fast calls OSP$FORMAT_MESSAGE to format the message and displays the formatted message. The user must press NEXT to acknowledge the message, then he may continue executing transactions or displaying menus.

To make it possible for IM/Fast to generate application dependent messages the application developers must create a Message Module and store it on the Application Object Library. The message module must contain a message template for every condition code and application identifier used. If an application defined message template is not found a message is generated according to NOS/VE standards. A complete description of how to create message module is available in CYBIL for NOS/VE System Interface mentioned above.

Note: To avoid conflict in condition codes used by IM/Fast and any IM/Fast application, it is recommended that applications use condition codes in the range 10,000 to 19,999.

If a transaction subroutine aborts, IM/Fast displays an error message describing the abort, and the user must press NEXT to acknowledge the message. Then he may continue executing transactions or displaying menus. An example of a subroutine abort is a divide fault (division by zero). Depending on the cause of the abort, other transactions may also abort. If that occurs, the user should execute the QUIT function.

3.2.1.2 COBOL sub-program

The LINKAGE SECTION and PROCEDURE DIVISION statements of a COBOL sub-program should be as prescribed by IM/Fast here.

The LINKAGE SECTION is defined in deck tfd$cobol_linkage and the PROCEDURE DIVISION is defined in deck tfp$cobol_procedure. The example below shows the source statements required in a COBOL sub-program.

```
       IDENTIFICATION DIVISION.

       PROGRAM-ID.    SAMPLE.
       AUTHOR.        Programmer.
       DATE-WRITTEN.  10/14/87.

       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. CYBER-180.
       OBJECT-COMPUTER. CYBER-180.
       SPECIAL-NAMES.

       INPUT-OUTPUT SECTION.
       FILE-CONTROL.

       DATA DIVISION.
       FILE SECTION.

       WORKING-STORAGE SECTION.


 *copy tfd$cobol_linkage
 *copy tfp$cobol_procedure

******* Put your program statements here

           EXIT PROGRAM.
       END PROGRAM SAMPLE.
```

A detailed description of decks and their use and a detailed description of the parameters is provided in section 3.1.1, Parameter Passing. The deck tfp$cobol_procedure expands to:

```
       PROCEDURE DIVISION USING
               tfv-transaction-code,
               tfv-application-id,
               tfv-application-name,
```

```
                tfv-user-name,
                tfv-family-name,
                tfv-date,
                tfv-time,
                tfv-parameter-lines,
                tfv-message-lines,
                tfv-local-data-area,
                tfv-global-data-area,
                tfv-interaction-mode,
                tfv-interaction-style,
                tfv-recovering-flag,
                tfv-user-form-id,
                tfv-processing-option,
                tfv-status-parameter,
                tfv-status-condition
                .
```

Processing considerations for COBOL sub-programs are:

o The sub-program must terminate processing with an EXIT PROGRAM statement.

o When the sub-program is compiled the sp parameter must be set to TRUE on the COBOL command.

o IM/Fast pushes its forms by calling the fdp$push_forms NOS/VE Screen Formatting request before it executes a sub-program. It pops its forms by calling fdp$pop_forms after it executes a sub-program.

Sub-programs must leave their application forms in such a state that IM/Fast's fdp$pop_forms call makes its forms available. CAUTION: If this is not done, no further processing is possible, and the terminal may hang.

3.2.1.3 FORTRAN sub-program

Here is an example of a FORTRAN subroutine which could be used as a session processor, menu processor, or transaction subroutine.

```
      subroutine sample
     1  (tfcode, tfapid, tfapnm, tfusnm, tffmnm, tfdate, tftime,
     2   tfparm, tfmsg,  tfldat, tfpdat, tfimod, tfisty, tfrecf,
     3   tfuser, tfprop, tfstpm, tfstcn)
      character*4    tfcode
```

```
      character*2     tfapid
      character*31    tfapnm
      character*31    tfusnm
      character*31    tffmnm
      character*18    tfdate
      character*9     tftime
      character*60    tfparm(4)
      character*50    tfmsg (3)
      character*11    tfimod
      character*6     tfisty
      character*1     tfrecf
      integer         tfuser
      integer         tfprop
      character*31    tfstpm
      integer         tfstcn
      character*1000  tfldat
      character*1000  tfgdat

C     Put your executable statements here

      return
      end
```

The following deck was maintained on an SCU source library:

```
      subroutine sample
*copy tfp$fortran_subroutine
*copy tfd$fortran_variables

C Put your executable statements here

      return
      end
```

Here is an example of using SCU to expand this program:

```
SCU
  USE_LIBRARY LIBRARY=your_library
  EXPAND_DECK DECK=your_deck ..
    ALTERNATE_BASE=$system.common.psf$external_interface_source ..
    COMPILE=your_expanded_deck
END NO
```

The actual contents of tfd$fortran_variables are:

```
*copy tfd$fortran_partial_variables
*copy tfd$fortran_local_data_area
*copy tfd$fortran_global_data_area
```

The contents of tfd$fortran_local_data_area are merely an example of how the local data area could be defined. The contents of tfd$fortran_global_data_area are merely an example of how the global data area could be defined. An application developer may define these areas as he wishes.

Here is an example of how an application developer could define the data areas differently from the default definitions:

```
*copy tfd$fortran_partial_variables
      integer*50 tfldat
      integer*75 tfqdat
```

3.3 DEVELOPING TASKS

Only NOS/VE programs can be used as a TASK type of transaction processor. A NOS/VE program is a module or modules, with at least one starting procedure specified on a PROGRAM statement. IM/Fast initiates the task with a PMP$EXECUTE call. A complete description of this request is contained in Chapter 3 of CYBIL for NOS/VE System Usage manual (CDC Publication Number 60464115). You should be familiar with this request and the program execution concepts described there.

Parameters of the PMP$EXECUTE request are initialized by IM/Fast. The name of the processor, specified by the transaction definition, is used as the name of the starting procedure in the program attribute record.

The called task executes in parallel to the IM/Fast task; i.e the WAIT parameter of the execute call is set to OSC$NOWAIT.

Application developers may require that some data be provided when a Task type of transaction is executed. This data may be entered on the Parameter field of IM/Fast menu form. Refer to section 1.4.13 for more details on entering data and requesting tasks. Any data present in the Parameter field is passed to the called task. The parameter called PARAMETER will contain this data. The status parameter is also passed.

All other program attributes of the calling task are inherited by the called task. This ensures that the called task has the same program attributes (with the exceptions

noted here), the same object library list, the same object file list, and the same module list.

The maximum number of tasks that can be activated by a user is subject to a limit defined in NOS/VE. Two IM/Fast tasks are included in the count. When the limit is exceeded, no further attempts to use task transactions will be honored. The number of tasks is further subject to a NOS/VE system wide limit. You should talk to your NOS/VE System Administrator to determine the limits of your system.

The task called by IM/Fast may call other task(s) subject to limits mentioned above. IM/Fast also has a limit of 10 active tasks per user.

IM/Fast maintains a count of active tasks on the main screen. This count is incremented as tasks are activated and decremented as tasks terminate. The IM/Fast task cannot terminate until all tasks have terminated. If you press QUIT and there are active tasks, the user must choose one of three options: a) terminate as soon as all active tasks have terminated, b) terminate all active tasks and IM/Fast, or c) do nothing. If the user chooses to do nothing, he could re-enter the Quit function or another function later to get an updated count of active tasks.

No task, either called directly or indirectly by IM/Fast, may communicate with the terminal. Any attempt to use the terminal from a called task may produce unpredictable output at the terminal and may cause the job to abort.

Here is an example of a COBOL task which accepts parameters:

```
       IDENTIFICATION  DIVISION.

       PROGRAM-ID. getparm.

      ************************************************************
      *  THIS IS A main PROGRAM to access execution parameter    *
      ************************************************************

       ENVIRONMENT     DIVISION.

       CONFIGURATION   SECTION.

       SPECIAL-NAMES.
       JOB-LOG   is jlog.
```

```
       DATA DIVISION.

       WORKING-STORAGE SECTION.
       01  index-line.
           03                          PIC X(10)  value "  ****** ".
           03  indx                    PIC 9(10).
           03                          PIC X(10)  value "  ****** ".
           03  index-name              PIC X(10)  value "  BATCH  ".

       01  error-line.
           03                          PIC X(10)  value "  *error ".
           03  display-status          PIC 9(10).
           03                          PIC X(10)  value "  error* ".

       01  user-information.
           03  user-time-for-wait      PIC 99.
           03  user-time-for-loop      PIC 99.
           03  user-giving-name        PIC x(10).
           03  filler                  PIC x(68).

       01  ve-command-line.
           03  command             pic x(005) value "wait ".
           03  command-seconds     pic 9(002) VALUE 5.
           03  filler              pic x(003) value "000".

       01  status-variable.
           03  status-val          pic 9(001) usage comp value 0.
               88  it-worked                              value 1.
               88  it-failed                              value 0.
           03  product-code        pic x(002) value spaces.
           03  diag-no             pic 9(006) usage comp value 0.
           03  filler              pic x(256) value spaces.

       PROCEDURE DIVISION.
       FIRST-PARAGRAPH.
           DISPLAY "  ******* Begin run of GETPARM **Batch" UPON jlog.
**tfc**      DISPLAY "  ******* Begin run of GETPARM *******" .
           MOVE space                    TO user-information.
           CALL "cbp$get_execution_parameters"
                                      USING user-information.
           IF  user-information  =  space
               DISPLAY " Please type parameters as following: "
               DISPLAY "      seconds-for-wait     99, "
               DISPLAY "         time-for-loop     99, "
               DISPLAY "      user-giving-name    x(10) "
               STOP RUN
           END-IF.
           DISPLAY " User Data = " user-information UPON jlog.
```

```
**tfc**      DISPLAY " User Data = " user-information.

             IF  user-time-for-wait       IS NUMERIC
                 MOVE user-time-for-wait TO command-seconds
             END-IF
             IF  user-giving-name    NOT =  spaces
                 MOVE user-giving-name   TO index-name
             END-IF
             IF  user-time-for-loop       IS NOT NUMERIC
                 MOVE 5                   TO user-time-for-loop
             END-IF
             MOVE 1                       TO indx
             PERFORM  user-time-for-loop TIMES
                 DISPLAY    index-line   UPON jlog
**tfc**            DISPLAY    index-line
                 PERFORM    wait-rtn
                 ADD  1                   TO indx
             END-PERFORM
             DISPLAY "  ********** End of GETPARM ****Batch" UPON jlog.
**tfc**       DISPLAY "  ********** End of GETPARM **********" .
             STOP RUN.

         WAIT-RTN.
             ENTER "clp$scan_command_line"
                                          USING ve-command-line
                                                status-variable.
             IF  it-failed                THEN
                 MOVE diag-no             TO  display-status
                 DISPLAY error-line       UPON jlog
                 STOP RUN
             END-IF
```

3.4 DEVELOPING COMMANDS

IM/Fast allows you to use commands as transaction processors. You can define new commands or use commands provided by NOS/VE. Command Language processing is described in detail in Chapter 9 of CYBIL for NOS/VE System Usage manual (CDC Publication Number 60464115). A brief description of system commands is in SCL for NOS/VE Quick Reference (CDC Publication Number 60464018).

A new command processor that you develop can either be an SCL procedure or a program module. The CYBIL for NOS/VE System Usage manual shows you in detail what you need to do to develop command processor programs.

Store all new command processors that you develop on the

Application Object Library. To make these commands available to users this library is also added to the user's command list when the IM/Fast application is called.

IM/Fast executes the command processor via the CLP$INCLUDE_COMMAND program interface call. The CLP$INCLUDE_COMMAND parameters COMMAND and status are supplied by IM/Fast. The COMMAND parameter is set to the command line to be interpreted by the SCL command processor. It has the following format:

command_name parameter

where command_name is the processor name declared in the transaction definition. Any data typed in by the user (in the Parameters field of the menu form) when selecting this transaction is passed on as parameter(s) to the command.

Commands are processed in series; i.e., the IM/Fast task processing is interrupted, the command process is executed, and when the command processor terminates, the interrupted IM/Fast task is restarted from the point of interruption.

3.5 TESTING IM/FAST TRANSACTION PROCESSORS

This section briefly describes the use of the NOS/VE Interactive Debug Facility to test your transaction processor programs.

There are other debugging facilities available in NOS/VE that are language dependent. If needed, these facilities may also be used for debugging. For example, you may wish to use the debugging statements of the COBOL language in some cases.

The interactive debug facility lets you debug your programs at the source code level. The Debug for NOS/VE Usage manual (CDC Publication Number 60488213) contains a detailed description of the concepts and commands.

3.5.1 TESTING SUB-ROUTINES

The processor you want to debug using this approach must be compiled with debug turned on, so that the compile time environment is available to debug to allow symbolic debugging. Create your Application Object Library with the object module

of the processor on it. For example, if you want to test a processor SAMP which is the name of a COBOL program, compile it with the following COBOL command:

```
COBOL,i=input,...,debug_aids=all, b=binary, l=listing, sp=yes
```

Add the object module of SAMP to the application library. The following commands may be used to do this:

```
CREOL
  ADDM application_object_library
  ADDM binary samp
  GENL application_object_library
  QUIT
```

Next, using the Application Administrator Workbench, define a transaction that specifies the sub-program you want to test as the processor, and add it to a menu. The programmer testing SAMP must have the menu in the set of menus available to him/her. If such a menu does not already exist, create a new one. A definition of the menu and the test transaction could be:

```
+------------------- Add/Change/Delete Menu ----------------------------------+
|                                                                             |
|             Code: test                                                      |
|           Enable: yes (yes, no)                                             |
|      Parent Menu: $prg                                                      |
|       .                                                                     |
|       .                                                                     |
|       .                                                                     |
+-------------- Add/Change/Delete Transaction ----------------------------------+
|                                                                             |
|         Code: samp                                                          |
|       Enable: yes (yes, no)                                                 |
|         Menu: test                                                          |
|         Type: x subroutine  _ command  _ task                               |
|      Preload: no_ (yes, no)                                                 |
| Access Level: 1_ (1..16)                                                    |
|    Processor: samp_____________________________                             |
|       .                                                                     |
|       .                                                                     |
|       .                                                                     |
```

With the definition of the SAMP transaction added to the TEST menu you are ready for debugging. The user validated for this menu must log-in and bring up this IM/Fast application. But before typing the FAST command be sure to set the debug

mode in the program attributes to ON. The command to do this is:

```
SET_PROGRAM_ATTRIBUTES DEBUG_MODE=ON
```

You will then be given a message from the DEBUG utility. This lets you know that debug is ready to accept your commands. When you type in the RUN command, the initialization task of IM/Fast executes. It initializes the second task, the run time processor of IM/Fast. You are given a message from the DEBUG utility to let you know that debug is ready to accept your commands in the run time processor.

At this point set a breakpoint and run IM/Fast with the following debug commands:

```
SET_BREAKPOINT EP=tfp$execute_subroutine
RUN
```

IM/Fast displays the initial menu for the user, and waits for the user input. Now when you run the SAMP transaction, debug stops execution at the breakpoint in IM/Fast, where it is ready to call the SAMP sub-processor.

Now you can set a breakpoint in the SAMP module anywhere you desire and debug the SAMP sub-program. Here is an example of setting the breakpoint on entry to the processor:

```
SET_BREAKPOINT m=SAMP p=SAMP
RUN
```

Execution will stop upon entry to the SAMP processor. You may now debug the program with any of the debug commands.

NOTE: You must be in line mode to initialize IM/Fast in debug mode as described above. You do this by entering the following command before entering IM/Fast:

```
CHANGE_INTERACTION_STYLE LINE
```

If you forget to do this, you will have to execute Debug's DEAS function (Deactivate Screen) to initialize IM/Fast. If you wish to debug your subroutine in full screen mode, you may enter the Debug ACTIVATE_SCREEN command when line mode Debug has stopped at the tfp$execute_subroutine breakpoint.

3.5.2 TESTING COMMANDS

To test commands, the method you use depends on the processor. If the processor is an SCL procedure, you can test it by using the procedure under control of SCL; there is no need to execute the command in IM/Fast. This requires that the command is added to the command list by the SET_COMMAND_LIST SCL command, and then executing the command repeatedly. Examine the results and correct the procedure as necessary.

If the command processor is a NOS/VE program or a procedure with a call to a program, you may use the NOS/VE Debug facility. This testing may be done with or without IM/Fast. The testing method is the same as outlined in section 3.5.1. First compile the program with the Debug option of the compiler used turned on. Put the processor on the object library and make sure you can run the processor, either under IM/Fast or under SCL, and set the debug mode on. NOS/VE debug facility will get control and stop when the first statement of the program is reached. You may then use the debug commands as necessary.

3.5.3 TESTING TASKS

Testing task transaction processors can be done under control of SCL. Use the same method as testing command programs outlined in section 3.5.2 above. But this time use the SCL command EXECUTE_TASK to start up the task. Debug facility will get control, and you may then use debug commands as necessary to test the program.

4.0 DATABASE_ADMINISTRATION

This section will mainly contain information for the IM/DM database administrators.

5.0 ORDER_ENTRY - AN IM/FAST APPLICATION MODEL

5.0 ORDER_ENTRY_-_AN_IM/FAST_APPLICATION_MODEL

5.1 OVERVIEW

This application was implemented under IM/Fast for testing and for showing IM/Fast concepts in a practical way. The application is called 'order entry' or 'oe' for short ('oe' is also used as the application identifier field). This is not a comprehensive model of a real application for taking customer orders.

5.2 APPLICATION_DEFINITION

```
identifier='oe' name=ORDER_ENTRY ..
  object_library=.ccccc.order_entry_library..
  state=ENABLE
```

5.2.1 USER DEFINITION

```
name=usera menu=($own, $adm,      ) privilege=owner access_level=2
name=userb menu=(incl, incl, inve) privilege=user
```

5.2.2 GROUP DEFINITION

```
name=manager_clerk pr=admin
name=order_clerk pr=user
```

5.2.3 TRANSACTION DEFINITION

```
ordx owner=oecl processor=orderx preload=yes..
  d='Take Order on-screen           ' access_level=1
samp owner=oecl processor=sample preload=yes..
  d='SAMPLE transaction             ' access_level=1
ship owner=oecl processor=ship..
  d='Process shipment               '
prin owner=oecl processor=print..
  d='PRINT Order                    '
cust owner=cucl p=customer pi=m     pl=yes..
  d='Process Customer record        ' al=1
chao owner=oecl p=orders pi=1       pl=yes..
  d='TEST transaction 2             ' al=1
sub1 owner=oecl p=sub1 pi=1..
  pl=no  d='SUB1 processor                  ' al=1
sub2 owner=oecl p=sub2 pi=1..
```

5.0 ORDER_ENTRY - AN IM/FAST APPLICATION MODEL
5.2.3 TRANSACTION DEFINITION

```
    pl=yes d='SUB2 processor                    ' al=1
  sub3 owner=oecl p=sub3 pi=l..
    d='SUB3 processor                  ' al=1
  inve owner=incl p=inventory          pl=yes..
    d='INVENTORY processing            ' al=1
  edic owner=$pap p=edic                  t=c..
    d='EDIT_CATALOG                    ' al=1
  entp owner=$pap p=entpe                 t=c..
    d='ENTER Programming Environment   ' al=1
  edif owner=$pap p=edit_file             t=c..
    d='Call NOS/VE editor              ' al=1
  desf owner=$pap p=design_forms          t=c..
    d='DESIGN Forms                    ' al=1
  edim owner=$pwb p=ed                    t=c..
    d='EDIT module                     ' al=1
  comm owner=$pwb p=cd                    t=c..
    d='COMPILE module                  ' al=1
  prim owner=$pwb p=prim                  t=c..
    d='PRINT module Listing            ' al=1
  edif owner=$pwb p=edit_file             t=c..
    d='Call NOS/VE editor              ' al=1
  task owner=$pts p=tasking               t=t..
    d='CALL parallel task              ' al=1
  $dea owner=$own p=app$delete_application..
    d='DELETE application definition   ' al=1
  $coa owner=$own p=app$copy_application..
    d='COPY application_definition     ' al=1
  $cha owner=$cap p=app$change_application_proc..
    d='CHANGE application attributes   ' al=1
  $eda owner=$cap p=edit_definition       t=c..
    d='Edif Application definition     ' al=1
  quik owner=$dba p=quick                 t=c..
    d='IM/QUICK                        ' al=1
```

5.2.4 MENU DEFINITION

```
  code=$own title='$OWN - Application owner menu. ' al=1 ..
    d='Select to DELETE application   ' state=enable
  code=$adm owner=$own title='$ADM - Administrator menu.     ' al=1 ..
    d='Select to CHANGE application   ' state=enable
  code=$cap owner=$adm title='$CAP - CHANGE application      ' al=1 ..
    d='Select to CHANGE application   ' state=enable ..
    p=app$change_menu_prolog e=app$change_menu_epilog
  code=$dba owner=$own title='$DBA - DB Administrator menu.  ' al=1 ..
    d='Select to MAINTAIN Database    ' state=enable
  code=$prg owner=$own title='$PRG - Programming menu.       ' al=1 ..
    d='Select to DEVELOP Programs     ' state=enable
  code=$pap owner=$prg title='$PAP - Program application     ' al=1 ..
```

5.0 ORDER_ENTRY - AN IM/FAST APPLICATION MODEL
5.2.4 MENU DEFINITION

```
      d='Select to PROGRAM application  ' state=enable
    code=$pwb owner=$prg title='$PWB - Program workbench       ' al=1 ..
      d='Select to PROGRAM IM/Fast      ' state=enable
    code=$pts owner=$prg title='$PTS - TEST program            ' al=1 ..
      d='Select to TEST programs        ' state=enable
    code=$app owner=$own title='$APP - Application user menu.  ' al=1 ..
      d='Select to USE application      ' state=enable
    code=mngr owner=$app title='MNGR - Manager options menu.   ' al=1 ..
      d='Select to USE Manager options. ' state=enable
    code=clrk owner=$app title='CLRK - Clerical options menu.  ' al=1 ..
              d='Select to USE Clerk options.   ' state=enable
    code=oecl owner=clrk title='OECL - Order entry menu.       ' al=1 ..
      prolog=ocoecl epilog=ocoecl d='Select to PROCESS ORDERS      '..
      state=enable
    code=cucl owner=clrk title='CUCL - Customer menu.          ' al=1 ..
      prolog=occucl epilog=occucl d='Select to PROCESS CUSTOMERS   '
    code=incl owner=clrk title='INCL - Inventory menu.         ' al=1 ..
      prolog=ocincl epilog=ocincl d='Select to PROCESS INVENTORY   '
```

5.3 APPLICATION_USAGE

```
+---------------------- Heading ----------------------------------------------+
|  Application: ORDER_ENTRY                                                   |
|  Date: October 5, 1987       Time:  2:28 PM                                 |
+----------------------- Menu ------------------------------------------------+
|  OECL - Order entry menu.                                                   |
|  Key  Code  Description                        Type       Menu Position     |
|   F1  ordx  Take order on-screen               Tran       oecl clrk $app $own |
|   F2  samp  SAMPLE transaction                 Tran                         |
|   F5  ship  Process shipment                   Tran                         |
|   F7  prin  Print Order                        Tran                         |
|   F8  chao  Change order                       Tran  Fwd                    |
| Enter Code  Parameters                                                      |
|       ----  ------------------------------------------------------------    |
|             ------------------------------------------------------------    |
+----------------------- Message -------------------------------------------- +
| --  Welcome to ORDER_ENTRY application.                 Tasks:  0           |
|                                                                             |
|                                                                             |
+-----------------------------------------------------------------------------+
   Xecute                                                  Refrsh
f1  Ordx  f2  Samp  f3  Back  f4  Help  f5  Ship  f6  Quit  f7  Prin  f8  Chao
```

5.0 ORDER_ENTRY - AN IM/FAST APPLICATION MODEL
5.3 APPLICATION USAGE

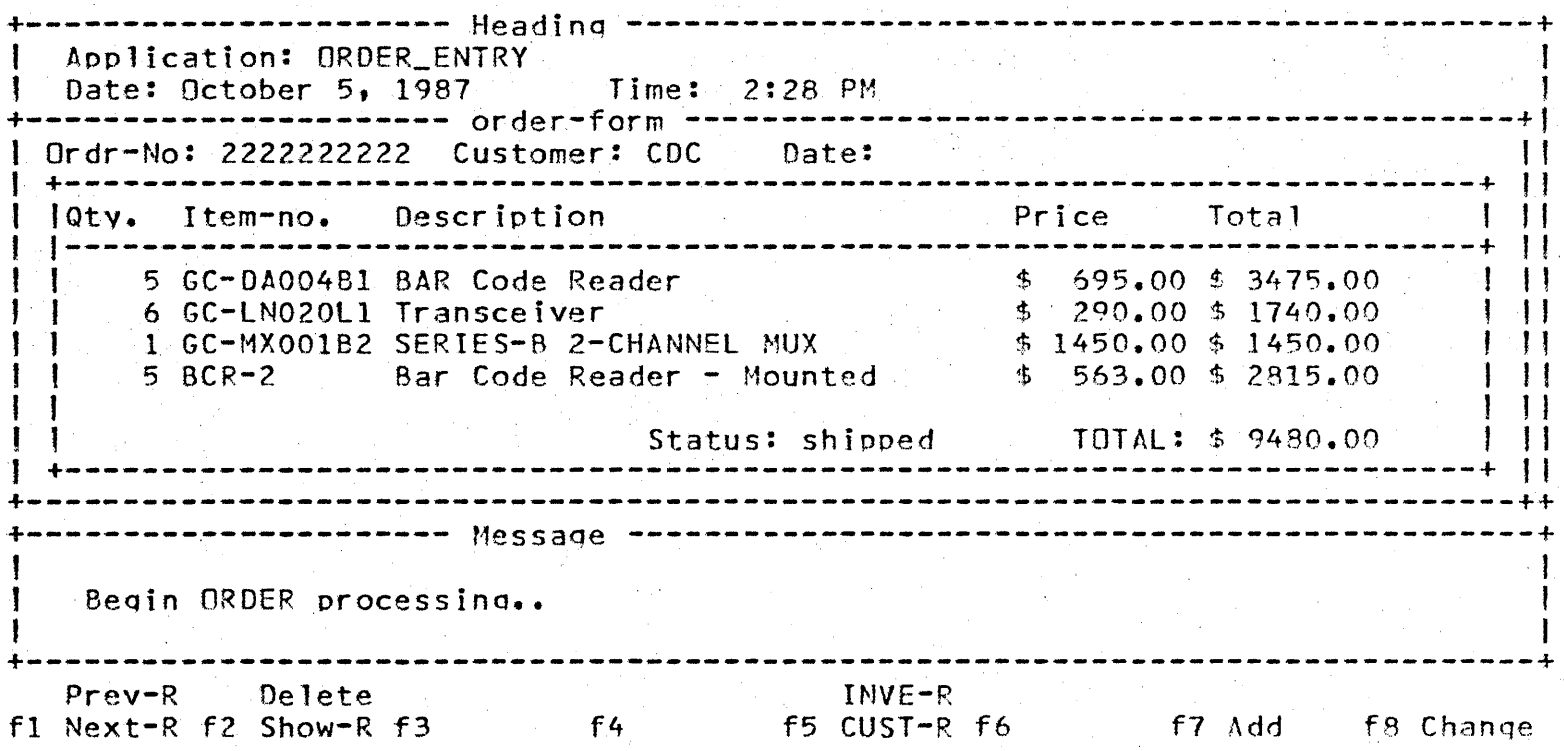


Table of Contents