Tailorable Electronic Information Exchange System (TEIES) Forms Processor IBM - NJIT Joint Study

Document Number 87-09 V 1.50

July 28th, 1987

John L. Foster

Computerized Conferencing and Communications Center
New Jersey Institute of Technology
Newark, NJ 07102
(201) 596-EIES

Table of Contents

List of Illustrations

Forms Processor

Forms is the TEIES function that formats data for the presentation of information to the user via the user interface. While Forms provide the ability to organize display screens for input and output, they do not implement the control necessary to move around the user interface. Interface state changes are handled by the Transaction Manager and the Object Level Object-Action procedures. Therefore, Forms are a tool for the Transaction Manager and the Object Level. Together these levels form a User Interface Management Service (UIMS). This capability not only presents the user interface of TEIES services to the user, but eventually allows the user to redefine his or her interface to the system. Combining the capability of dynamic Forms definition and Object-Action procedures, how TEIES looks on the display screen may never be static.

Forms present output and request input from virtual devices assigned to a particular server; User, Master, Data Base, or Network. Typically a User Server will have at least one terminal device where information can be presented to a real user. However, the Terminal Interface model defines devices as object consistent with the object level. This way the Forms level does not know, nor is it concerned, if its output and input are coming from another data base object, a file or a real device. Given this approach the same techniques and methods for interface implementation are consistent across TEIES UIMS, and it matters not if there is a real user (as with a User Server) or a pseudo user (as with the Master Server or a virtual user task).

Consistent with the TI level, all Form output goes to a window on the target device (object). This will facilitate multi-window interfaces and a separation of Forms definitions from physical display locations and areas. The UIMS, together with Forms *profiles*, determine where to place windows. Window placement can not only be user dependent, but task oriented as well.

Once Forms send display output to their eventual destination and that destination is a real device, the TI takes over to handle real user input. The user interacts with the presentation, performing input to input fields, scrolling screens, moving the cursor, etc., until a defined signal is sent back indicating a change of state. A state change occurs with defined function keys or interrupt keys. At this point the Forms level sends the function requested and all the user maipulated input fields back to the UIMS. The UIMS then determines the proper action to take based on user input and its knowledge of state transitions. This is described in more detail in -- Heading id 'tm' unknown -- and -- Heading id 'obj' unknown --.

Forms are used to format menus, present objects and object lists, present input field formats, and display error messages. The proper Form to use is determined by the Object-Action combination that comes from the UIMS.

Forms exist either in the TEIES data base as the contents of objects, in native operating system files, or as in-memory structures. A Form is created as a document using a text editor by either a systems person or a sophisticated user. The initial Forms control language may not be simple enough for use by average users, but the interface will evolve with time. Forms connect the functionality provided by the Object Level for data base access, the TM for process and state control, and the Terminal Interface for display device access. Together these areas make up the UIMS. The Forms definition language is the Standard Generalized Markup Language (SGML) specified by ISO, with extensions. See "Appendix A. SGML Reference" on page 19 for a reference to SGML functions.

Mark-ups are used in TEIES for many reasons. Most prevalent are the needs of the system to distribute objects, display device independence, window manager independence, and to isolate any rendering-specific information from the definitions of Forms and Object contents. Eventually expert systems can be employed to analyze mark-ups entered by users. Mark-ups can easily be converted into display specifics compatible with any word processor or text formatter. On top of these technical features, mark-ups are simple to type in and easy to understand and utilize by users.

Each object definition specifies at least four default Forms: one for input, one to present the entire object, one to abstract the object, and one to output only the object contents. The default input Form formats a display for user input into an instance of the object. This occurs during a create or modify Action. The default output Form defines the format for presentation of an existing object with its fields filled out. In the user interface, users will be allowed to compose their own Forms

and override the default displays. The TM and Object Level Action Procedures are responsible for validating any data base transactions.

Forms Level Logic Components

The Forms level is broken into three distinct parts, each responsible for a separate function of the Forms processor.

Parser The forms parser locates tags in the text (contents fields of Object Level object instances) and determines if the tag and its associated qualifiers are valid. Qualifiers are the keywords and values entered in the mark-up to request various processing options associated with the mark-up. Tags and qualifiers are stored in two separate dictionaries.

Functions Once a tag has been identified the code to implement the function of the tag is invoked. Currently all forms code must be implemented in the C language. The function code for tags can take advantage of tools and services throughout TEIES.

Forms Profile Before a Forms can be processed, a profile must be defined. The profile sets all initial values of the qualifies. Initial formatting options are set and the Terminal Interface is queried to determine any special window and device considerations. Users can tailor their profiles. Each task has associated with it a profile. This way multiple invokations of the Forms on multiple windows, or even multiple devices, can have separate profiles.

Form Mark-Up Tags

A mark-up tag is a sequence of characters that occur in the text contents of an object that activate a special function or feature to be applied to textual data that follows. Mark-ups can occur anywhere in the text. Many mark-ups can be nested inside of other mark-up tags that have been activated. Mark-ups are not case sensitive. Therefore tags can occur in upper case, mixed case, or all lower case throughout the text. Tags and qualifiers can be abbreviated according to alphabetic match rules. The first alphabetic match for a tag against the table of all available tags is used. The same is true for tag qualifiers.

In general, a mark-up consists of three distinct parts:

begin tag

data

end tag

Begin and end tags are further broken down into their sub-parts:

- open tag
- tag name
- qualifiers
- close tag

A subset of the SGML, with some TEIES extentions, are used in the Phase I system. The tags are:

< This is the *open tag* lead-in character. The Forms parser is coded in such a way that any sequence of characters may be used to indicate where tags are in the text. The < character is the default.

</ The default open end tag, must be matched with an associated begin tag. For example, a begin simple list tag should be terminated with an end simple list tag, with the contents of the list in between.

```
<sl>
<li>this is list item number one
          .
          .
          .
</sl>
```

The open end tag may be redefined to any sequence of characters.

> The default *close tag* character. Just as is possible with the open tag indicator, the close tag can be redefined to any sequence of characters. This delimits the begin or end tags and separates the tag from the text.

Note: If what follows an open tag is not a known mark up then it and the tag open and close indicators are treated as part of the text.

< p > Indicates the start of a paragraph. Paragraph text formatting features are inherited from profile settings set before the form is processed. All form functions and features are resolved consistent to the current window and viewport settings.

< center break = " > Centers the text within the margin settings for the current window. The text is centered on its own line. The optional break keyword indicates a string of characters to use as a line delimiter to center text parts over multiple lines. The center tag is ended by an end-of-record or an end center tag.

< table break = " columns = j#s,... > Define a tabular specification to line information in columns. The break = keyword indicates the character or characters used to determine where to break the column data. The columns = keyword defines column widths. Column specifications are entered separated by any valid delimeter. The column specification is made up of 3 parts, the data's justification, the field size, and the space units of the field. Justification has three options:

left Left justify the field within the space assigned.

right Right justify

center Center justification within the field boundries.

Following the field justification is a number describing the size of the column boundries. Following this number is the type of space units the number describes. Column width is specified in 9 ways, with device units as the default:

inch Indicates spacing is in inches. Example: 12i means left justify in a column 2 inches wide (left 2 inches could also be used).

mm Millimeters, can also be spelled out.

cm Centimeters, can also be spelled out.

pica Pica font measurements. Pica is a standard printer's measurement that corresponds to 0.1663 inches. Picas are further subdivided into points. There are 12 points to a Pica and 72 points per inch. The Pica space unit is mostly used in conjunction with variable precision character sets.

cicero Cicero is the standard printer's measurement for countries other than the United States and Great Britain. A Cicero is 0.1776 inches and further divided with 12 Didot points per Cicero.

characters Corresponds to the size of the space character in the current font.

ems A number of ems, where an em is the width or height of the character M in the current font. This is also dependent on the pitch used for the current font.

device units The number of horizontal device units. A device unit is a whole number indicating the device specific addressing scheme. This is the least prefered specification since it is device dependent. Can be abbreviated 'du'.

An example table definition may look like this:

```
<table break=';' column=15char right15ems cent 20 du>
.
.  table information in the text
.
</table>
```

The above example specifies a table where column information is delimited in the text by semi-colons. There are 3 column field definitions, one left justified in a width of 5 characters, another right justified in a field of 15 M characters, and a third center justified in 20 device units.

< space amount units > To place blank space in the output. The space tag without any qualifiers will place an amount of space corresponding to a single vertical spacing of the current font. Vertical space units may be specified following the same guidelines for the tables markup column definition.

< attach name = 'itemname' reference = " > The attach tag is a direct reference to another object to be processed as a form. It is up to the local system (object level) to resolve the item name. The reference = qualifier can be used to indicate special considerations for referencing the item. Possibilities are:

abstract: to place the abstract information (header) of the item.

name: to only place the name of the item

notseen: to reference the item only if this user has no privilege resolution to it.

all: to reference the entire item.

< sl > Start the definition of a simple list.

< ol > Start the definition of an ordered list. Ordered list elements have their order number placed in the left column.

< ul > Unordered list. Here each list element has a bullet in the left column.

< li > Denotes the text to follow is a list item entry.

< /sl > End of simple list. :i2/end simple list

< /ol > End of ordered list.

< /ul > End of unordered list.

< dl tsize = " break compact headhi = " termhi = " > Definition lists differ from the other list types because they have headings, the left column space is definable, and the entries are divided up into terms and definitions. Five attribute controls can be specified with the definition list tag:

tsize = to specify the amount of column space for the term entry. Use the space unit specifications described under the table mark up. The default tsize is 10 characters.

break to specify the description entry should start on the next line if the term entry is longer that the term entry field size (specified with tsize).

compact used if a blank line is not to be placed between definition entries. The default is one space between entries.

headhi = to specify the highlight level for the headings. The default is level 3. See the highlight phrase tag below.

termhi = to indicate the highlight level for the term entries. The default is level 2.

< dt > what follows is a definition term entry.

< dd > what follows is a definition list term definition, matched with the most recent term specified with the term entry mark up tag.

< /dd > Indicates the end of a definition. The start of another definition term by default will end the previous definition.

< /dl > End of definition list.

< dthd > The text following is the definition list term heading.

< ddhd > The text following is the definition list definition heading.

< h# id = " stitle = " > Define a heading level. Levels go from 0 (most important, highest level) to level 6 (least important). The id= qualifier specifies a cross reference value for referencing to this heading. The stitle= may be used to specify a short title to be used in place of the heading for special purposes.

< hp# color = " > Begin highlighted phrase. Highlight levels are defined in the forms profile. If no level is specified in the tag level 0 is assumed. Typically level 1 turns on italics, level 2 bold face, level 3 bold face italics, level 4 uppercase bold face italics, and so on. Highlighted phrases may be nested. The color= qualifier may denote the color the phrase is to be printed. For example:

```
<hp color=blue>This may be displayed in blue</hp>
```

This will print the phrase in the color blue, if supported by the target device, in highlight 0, or the normal character highlight.

< fig id = " place = " width = " frame = " depth = " > Figures are composed of 5 elements: the figure definition, the figure contents, the figure caption, the figure description, and the end figure indicator. The figure definition contains 5 qualifiers as follows:

id = assign an identification string to this figure for reference with the reference tag.

place = used to indicate where the figure is to go, such as

- **top** float the figure to the top of the next page
- **bottom** float the figure to the bottom of next page
- **inline** place the figure where entered with respect to the surrounding text.

width = to specify the width of the figure in space units.

frame = to specify how the figure is to be framed.

- **rule** place a ruler around the figure
- **box** place the figure inside a box
- **none** do not frame the figure
- **'string'** specify a string of characters to use as the frame

depth = to indicate an amount of vertical space for a figure paste-in, specified in space units.

< figcap > The figure caption tag automatically numbers the figure and highlights the caption in level 3 highlight.

This is an example figure, defined as:

< fig place = inline frame = box id = example1 >

Defaults are taken with the depth and width qualifiers. Note the handling of the figure caption and description.

Figure 1. An Example of a Figure Definition: This figure is defined using the place, frame, and id qualifiers.

< figdesc > define the figure description. This text is formatted after the figure caption and in the frame of the figure definition.

< /fig > End of figure definition.

< xmp > Begin example. Examples are similar to figures, only they do not contain captions or descriptions and they are placed in-line with the text. Examples also do not appear in a figure reference list. All text appearing within the example tags is placed on the display in a fixed precision font, usually in device addressing.

< /xmp > End example.

< field id = " name = " attributes ... > The field tag is provided to define fields for interactive display devices and to associate data base and runtime information with the form.

objecttype = the field name refers to an object definition identifier for the object implied by the Forms execution call. This declaration references an object handle to easily access and validate an object definition and its fields (see -- Heading id 'obj' unknown --). If the object type is not specified and there is an ambiguous OCD reference the field will resolve to the first object with this OCD in the list of open objects made with the call to Forms.

id = " The pseudo name assigned to identify the field attributes with the field in the form. This allows to use a different and shorter name in the mark-up tags than the actual data variable name known to the Object or TM levels. If this qualifier is not specified the id will default to the name = qualifier. Multiple field definitions with the exact same qualifiers can be specified in one field mark-up with each field name enclosed in quotes and separated by commas (or any other delimeter).

name = " Associate the field with a variable name for use not associated to actual Object OCD elements. If the var qualifier is used then this name is a

variable name to be used in the TM code or Object-Action code. If the var qualifier is not used then this name is an OCD for the object related to this field.

variable the field ID is a variable. Variables are not associated to object fields. These entities can be associated to data values assigned by the TM, or they can be data values created in the Form and passed back to the TM or Object-Action code procedure. These variables are local to the current Form.

index = " assign array or index information from an object to the entity_name. If the entity name is an array, the key is the array element. If the entity name is an index, the key is the search key to retrieve the information by. To specify an arbitrary key, the quotes are mandatory. If no quotes are used the string following the qualifier delimeter will be used as a function to be called to return a string to be used as the key.

prompt = " assign characters to indicate start and end of a field. Absence of this attribute or assigning blank space indicates no prompt characters are to be used.

input an input field is to be set up when this entity is referenced in the Form. If any information is already contained in the field (for example default values) it will automatically be placed in the field when the Form is displayed.

output when this entity is referenced, the information contained in its object instance is to be output. Output is the default if either input or output is not specified.

len = x override the field length as defined in the object definition. The length is specified in space units.

pad = x indicates a pad character to use to fill in unused field space. Default pad character is a white space.

justify = to specify field justification; L for left, R for right, C for center justification. These can be spelled out.

default = sets an input field to a default contents either a constant value enclosed in quotes or the results of a function.

< topm > begin definition of top matter material. Top matter is the information to appear at the top of each window, screen or page of output. Top matter is active until another top matter definition changes the definition. To turn top matter off specify a top matter and end top matter with **no** text between the tags.

< bottomm > Begin the definition of bottom matter material. Bottom matter appears at the bottom of each screen or page.

< if condition > Conditional form processing. If the condition is true, form parsing continues with the text immediately following the end-tag sequence of characters. If the condition is not true, forms parsing continues with the text immediately following the **else** tag or, if else is not found, the **endif** tag.

< else condition > Matched with an **if** tag.

< endif > Determines the end of if-else logic.

Example Form Definition

The example in Figure 2 on page 8 demonstrates what a Form will look like as it is entered by its creator and stored as the contents of an object, not as it will be displayed when executed.

```
<FIELD NAME=full_name ID=name INPUT PROMPT='><' >
<FIELD NAME=address   INPUT PROMPT='><' >
<FIELD NAME=city      INPUT PROMPT='><' >
<FIELD NAME=state     INPUT PROMPT='' PAD='_' >
<FIELD NAME=zip       INPUT PROMPT='' PAD='_' >
<FIELD NAME=area_code INPUT PROMPT='()'
<FIELD NAME=phone     INPUT PAD='_' LEN=8char >

<h2>Standard package request</h>
<p> Enter your full name and address and we will
send you a TEIES information package via US Mail.</p>


Full Name  &name;
Address    &address;
City       &city;
State      &state;       Zip  &zip_code;

<p>Enter your phone number:</p>
          ( &area_code; ) &phone;
```

Figure 2. Example Object-Create Form Definition: This example shows a possible form definition to format a display, request user input, and place the resulting input into the data base.

The Form definition in Figure 2 may be presented, depending upon what attributes were in effect at the time of execution, to a user as shown in Figure 3.

```
STANDARD PACKAGE REQUEST

Enter your full name and address and we will send
you a TEIES information package via US Mail.


Full Name  > _                          <
Address    >                                  <
City       >                 <
State __            Zip _____


Enter your phone number:
          (    ) _______
```

Figure 3. Example Form Rendition

```
<topm>
TEIES Casual          Screen (&SCRNO;) of (&TOTSCR)     S:&STATE;
<\topm>
<H3>View <hp2>new<¢hp> Material:<\H> NEW
<field id='NOTIF','MAIL', INPUT,VAR,LEN=5,PAD='_',DEFAULT='yes'>
      Notifications          #  ?&NOTIF;
      Mail                   #  ?&MAIL;
  <H3>Conference with new:<\H>
<field ID='CMTS','ACTIV',IN,VAR,LEN=5,PAD='_',DEFAULT='yes'>
       Comments              #  ?&CMTS;
       Activities            #  ?&ACTIV;
<H3>ACTIONS:              If YES, fill in:<\H>
<field ID='ENTRJTCF','CRTNOTIF','MESS','CMT','DRAFT','DIRUP',
 INPUT,VAR,LEN=5,PAD='-',DEF='no'>
<field ID='CONF','WHOM','RMESS','ENTRY',INPUT,VAR,LEN=13,PAD'_'>
<field ID='SCRAT',INPUT,VAR,LEN=4,PAD='-',DEFAULT='#'>
     Enter/Join Conference? &ENTRJTCF;  Conference ( # / name ) &CONF;
     Create Notifications?  &CRTNOTIF;  To Whom ( # / name )?   &WHOM
            Message?        &MESS;      Reply to Message ( # )? &RMESS;
            Comments?       &CMT;       Conference ( # / name )? &CONF;
            Draft?          &DRAFT;     Scratchpad ( # )?        &SCRAT;
     Directory Update?      &DIRUP;     Entry ( # / name )?      &ENTRY;
<H3> Find: <\H>                           A Member?
<field ID='YRCONF','PUBCONF','YRGRP',IN,VAR,LEN=5,PAD='_',DEF='no'>
     Your Conferences?   &YRCONF;          Your Groups?   &YRGRP;
     Public Conferences? &PUBCONF;         Public Groups? &PUBGRP;
<bottomm>
<H3>Confirm (F2)   Finished (+)    Homebase (++)    Print Screen (F3)
Process (ENTER) Cancel   (-)     Quit      (--)    Help (F1/?/?word)
<field ID='OPTION',INPUT,VAR,LEN=29,PAD='_'>
Command Option?&OPTION;                   Screen control (F7/F8/F9/10)
<\bottomm>
```

Figure 4. Example of the Casual Interface Form Definition

Forms Operation

The contents of a Form may contain any data type the Basic Storage Element can hold, such as text and graphics; commands to operate on the contents, such as markups and display parameters; input fields; and variables to be imbedded into the contents.

Additional information, such as the binding of data base variables, are requested through the UIMS. Results of the parsing intended for display are placed in a buffer controlled by attributes of the target display device. The output is passed on to TI for window management and display attributes.

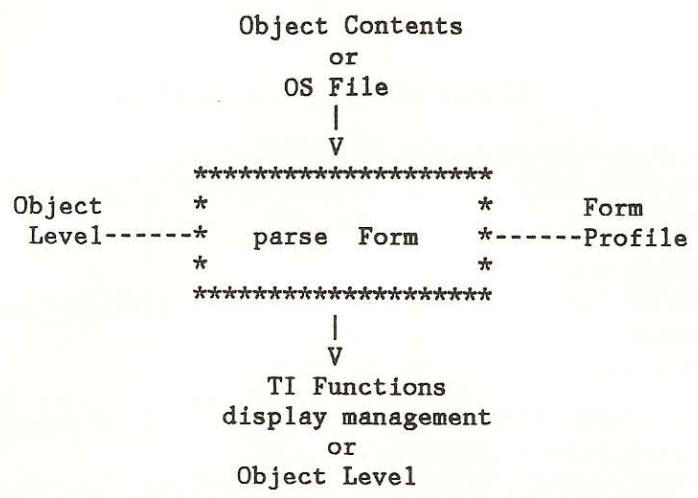

Figure 5. Forms Processor Data Flow: The Forms Processor takes a Form from a TEIES object and includes information from the object level and user input to format a display area.

A Form will not specify its window and target display requirements. A Form is invoked with the UMIS declaring and initializing the virtual device and window mappings. Then the Form profile sets up all initial forms variables. Finally the Form can be displayed.

Function Binding

The Forms processor can be invoked from C using the FORM() function.

```
rc = form( form_id, object_lu );

  sint            rc;
  string         *form_id;
  OCB            *object_lu;
```

The Form function will use a string, form_id, containing an identifier to open a handle to the content fields of the Form. This is a string that is resolved to an OID (see -- Heading id 'obj' unknown --), an OS file, or a memory buffer. The object_lu parameter is a pointer a list of object control blocks indicating the object instances to be used for the Form. An object handle is linked before calling the Forms processor. A return code is passed back from the Forms processor. Currently this value has no meaning. Output of the parser goes to the currently active display device via processing through GKS and VDI. If any errors occur during the processing of a Form, the Exception Handler will be called to resolve the problem. If the exception has to post messages to the display device, it will format a string and place it in a process area for Forms to display the next time a screen update is done. The Forms level provides the f_error() function to post an error message.

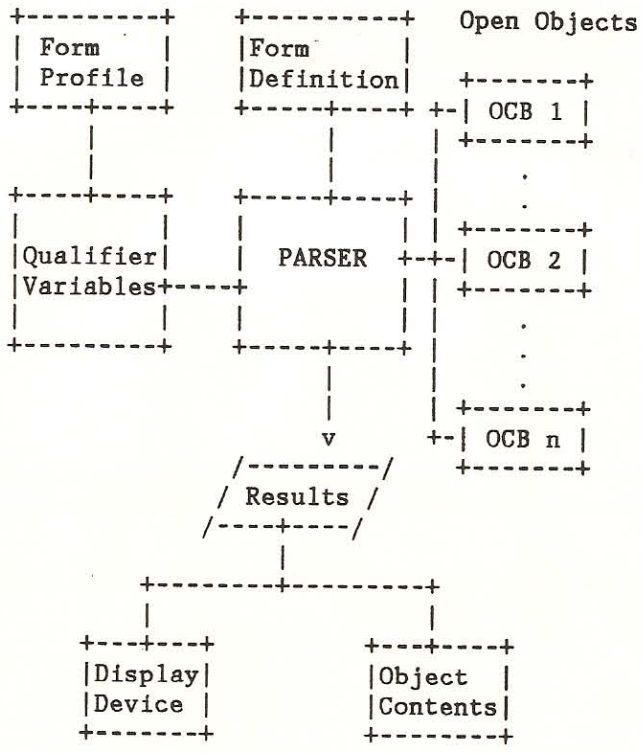

Figure 6. Input Elements to Process a Form: The Forms processor requires qualifiers initialized from the Forms profile and altered with mark-ups in the Form. Object fields and User Interface variables are taken from the UIMS (TM and Object levels). The results are sent to the Terminal Interface (TI) for eventual display or placement back into the data base.

```
rc = f_error( message, SE );

  sint            rc;
  string         *message;
  sched_struct   *SE;
```

For the entity reference mark up, the OBJECT keyword specifies an object will be referenced. The Object Level needs to be invoked with the proper object reference definition and operation to be performed. The FIELD keyword identifies a specific field of an object. The attributes of the field, as defined in the object definition, are imported if attribute keywords are not specified. The INPUT keyword specifies that input is to be performed when that field is referenced. The default is output. If the object instance has data in certain fields the information is displayed in the input field on the display.

Program Function Key Association

At the Forms level program funcition keys (PFK's) are associated to strings of characters. This provides an application abstraction so the function can be matched, not the PFK on the device.

Doing this gives the user the ability to redefine function keys and disassociates application functions with specific keyboard device layouts.

Three functions are provided: f_setkey(), f_key() and f_cmd(). f_setkey() associates application function strings to keyboard function keys. An in-memory table of strings is kept for each active device. f_key() is called to return the string of the function most recently requested. f_cmd() determines if the command line was edited by the user.

```
rc = f_setkey( key_def_vector );

   string      *key_def_vector;
```

```
rc = f_cmd( );

   sint      rc;
```

The argument to f_setkey() is a string pointer to the first simple string to be associated to the first PFK. Processing continues until a null string is encountered. An error code is returned if something goes wrong.

f_key() has no arguments and returns the string associated to the PFK most recently pressed. If no PFK has been encountered, a null string is returned.

```
key = f_key();

    string    *key;
```

Most Forms displayed on an interactive device will contain a command line on the Form. This allows the user to enter free form and override the context of the system input expectations. This line is taken from the Form after an interrupt key is pressed by the user, formatted as a TEIES string, and passed to the expression evaluator. The f_key function returns a true value if any information was entered on the command line when an interrupt key was pressed. If the line was not edited by the user, zero (false) is returned. This command line is pointed to in the Transaction Manager PID structure (refer to -- Figure id 'tmfig1' unknown -- under the -- Heading id 'tm' unknown --).

```
rc = f_cmd();

   sint    rc;
```

Exception Handler Interface

To associate messages generated by exceptions there must be coordination between the Exception Handler, Transaction Manager, and the Forms processor. When an exception message must be displayed, the target Form should have an area defined to place the information. Any display attributes can also be set by both the Form definition and the exception handler.

The function to associate exception messages to a Form for a particular process is f_message(). This function updates the Transaction Manager process flag for pending message ready and places

a pointer to the message to be displayed in the process msg field (-- Figure id 'tmfig1' unknown --).

```
rc = f_message( TM_PID, message );

   tint         TM_PID;
   string      *message;
```

The message will appear on the Form on the next display update.

Interface to GKS

Graphical Kernel System functions handle the presentation of TEIES information and binding it to display viewports. Forms are defined to determine how the information is to be formatted. By calling upon the GKS to present this formatted information, a well defined and flexible interface is maintained.

Typically Forms are concerned with formatting information from the TEIES data base for presentation to the user independent of display device. Another concern is retrieving information from the user in a formatted fashion to be placed back in the data base or control the state of the user interface. GKS defines a set of functions to accomplish the device access and input/output aspects of these tasks. Through definitions of World Coordinate Systems, Window and Viewport transformations of these World systems on to display devices, and Pick Identifiers for input selection recognition, a high level set of tools are used by Forms.

A generalization of a display surface represented by a terminal presenting a TEIES Form may appear as shown in Figure 7 on page 14.

The example shows four separate Forms bound together to present one screen of display information. One Form is used to define the information window. This Form formats state and control information from the World Coordinate system to a viewport mapped to the top line of the display. Another Form defines a prompt string and input field to be mapped to a viewport at the bottom of the display surface. A third Form creates a set of GKS Pick Identifiers for menu choices and formats the associated function strings in a viewport on the line above the command line. Finally one repeating Form defines how a list entry will appear and that information is formatted into a world coordinate space occupying the remainder of the display surface.

System control Forms are formatted in a world coordinate system separate from the world coordinate system used for target interaction information, such as Object Lists and Item Texts. This way GKS can easily perform window and viewport transformations on the Form data as it is formatted into the intended world coordinates. Using this method, the contents of a single item would be formatted into one world. This world could have a window mapped at any location onto the item, and a viewport mapped to any area of a display device. The same holds true for a list of object headers. They are formatted into a world, and a window is defined to determine which area of that list is mapped to the display. Using this method, display surface size, technology, and format are separated from the Forms. This also provides the possibility of the user interface rearranging the viewports dynamically, so windows can change in size and location during interaction with TEIES.

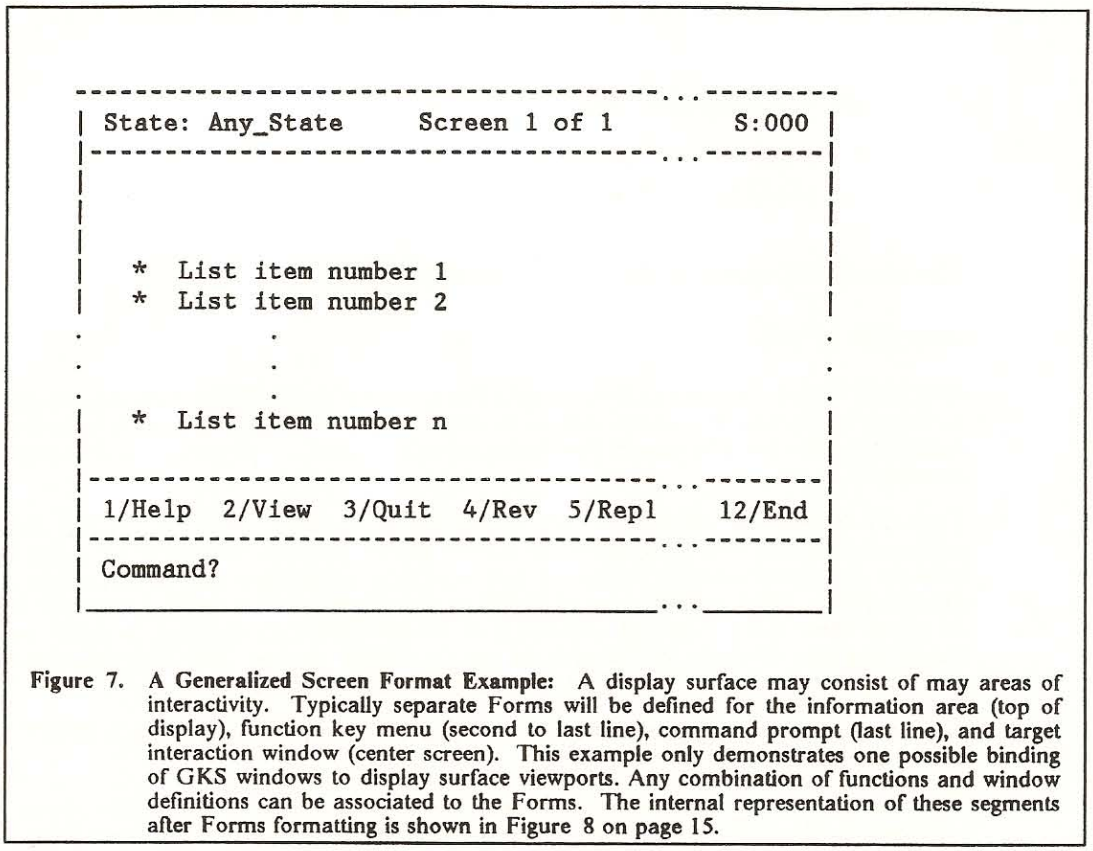

Figure 7. A Generalized Screen Format Example: A display surface may consist of may areas of interactivity. Typically separate Forms will be defined for the information area (top of display), function key menu (second to last line), command prompt (last line), and target interaction window (center screen). This example only demonstrates one possible binding of GKS windows to display surface viewports. Any combination of functions and window definitions can be associated to the Forms. The internal representation of these segments after Forms formatting is shown in Figure 8 on page 15.

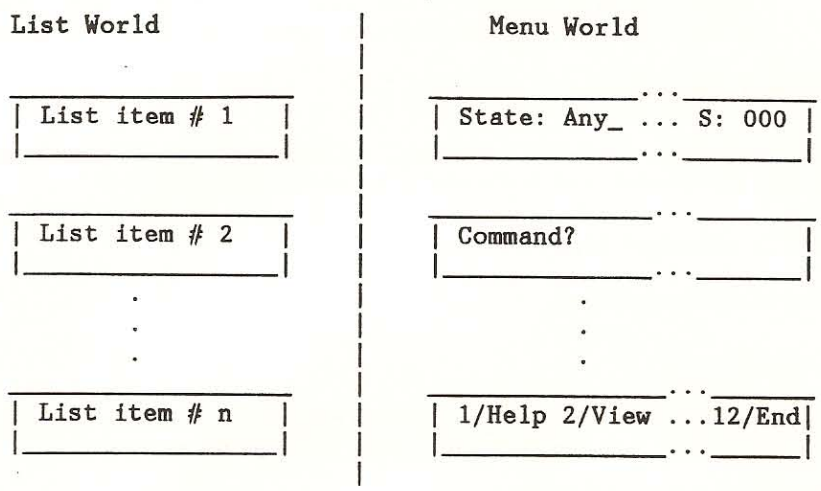

Figure 8. Binding of Forms into Segments in their respective worlds.: Item headers are fomatted into the "List World" Coordinate space and then mapped to a display viewport through window/viewport transformations. The same is done for menu and information areas of the display. They are formatted in their world coordinate space and then transformed to their respective viewport.

Form Field Linkage

All of the input fields for the current form are linked together in a linked list. There are also three additional pointers maintained: a pointer to the first field displayed, the current field, and the last field displayed.

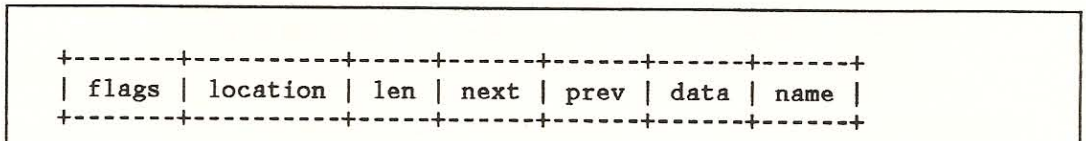

Figure 9. Structure of an Entry in the Form Field List: A linked list of fields is maintained by the Forms Processor for the current form. An entry of this structure exists for each input field.

Each field has associated with it a structure as shown in Figure 9. The elements of this structure take on the following meanings:

flags two bytes of bit flags used for form and user interface information. There are no display or terminal dependent meanings here. Currently, the following masks are defined:

- **x0001** Do not display if rendering on a line device.
- **x0002** Do not display if rendering on a page device.

location two full words denoting the world coordinates of this field (x, y).

len a two byte value indicating the length, in characters, of this input field.

next a pointer to the next field in the linked list.

prev a pointer to the previous field in the linked list.

data a pointer to where the data can be stored or retreived for this field.

name a pointer into the name table corresponding to this field in the variable name table for this form. This variable name table is compiled by the Form Processor.

Object Level Interface

We need to outline the object level commands and how an Object-Action will call a form. How does the form get data out of an object and then put data into the object?

Transaction Manager Interface

I think this is already explained.

Form Profile

This section describes how the forms processor binds display attributes by inquiring from GKS and associating tag qualifier default values. Information in the profile consists of:

- Paragraph spacing and formatting;
- Heading level font;
- Heading level spacing;
- Tag qualifier default values;
- Any margin spacing necessary;
- Special features;

Tag Dictionary

Here we should go on to describe what is the memory format of the mark-up tag dictionary. This is the structure s_parse() can go to look up tags. Once a tag is found in this dictionary a pointer to the qualifier dictionary and the code to execute the tag should be there.

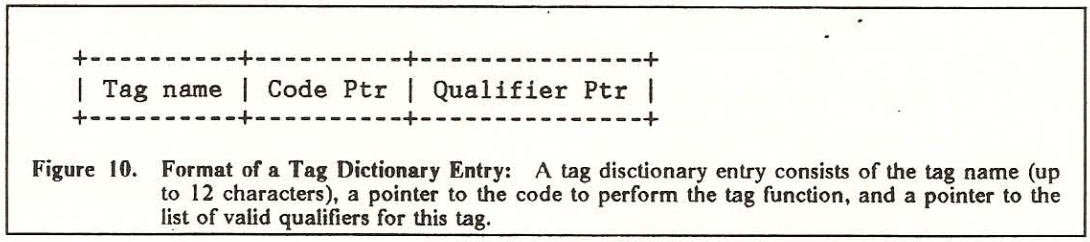

Figure 10. Format of a Tag Dictionary Entry: A tag disctionary entry consists of the tag name (up to 12 characters), a pointer to the code to perform the tag function, and a pointer to the list of valid qualifiers for this tag.

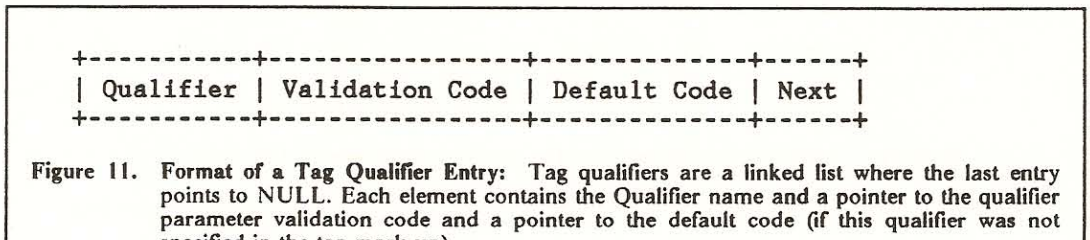

Figure 11. Format of a Tag Qualifier Entry: Tag qualifiers are a linked list where the last entry points to NULL. Each element contains the Qualifier name and a pointer to the qualifier parameter validation code and a pointer to the default code (if this qualifier was not specified in the tag mark-up).

Line Mode Considerations

It is not clear at this time what the line mode considerations are. For example, multiple input fields on a single line cannot be supported in line mode. Possibly they are presented one line at a time. Also how multiprocessed windows are handled is unexplainable at this time.

Full Duplex Considerations

Full duplex interfaces should be handled totally by the device drivers (VDI) so there are no special considerations by the Forms level. The major difference is the possibility of immediate feedback on user input by the keystroke. This can be handled by the design presented in this section, as well as understanding the Virtual Device Interface concepts.

Batch Mode Considerations

As with the full duplex interface style, the Forms interface is not concerned with batch screen vs. full duplex screen styles. The Forms, in conjuncion with GKS window and workstation management, is independent of interface style.

Appendix A. SGML Reference

Standard Generalized Markup Language (SGML)

< - default start tag open. Indicates the begining of a tag.

> - default start tag close. Indicated the end of a tag.

</ - default end tag open. Indicates the begining of a tag that ends a preeviously open tag.

<! - default markup declaration open. Used to define a declaration.

<? - default processing instruction open. Used to indicate the begining of a processing instruction definition.

& - default entity reference open. Indicates where an entity reference is to be resolved.

; - default entity reference close.

<p#> - start paragraph. # indicates a number describing the paragraph level, used for nesting. If # is not specified it defaults to 1.

</p> - end of paragraph.

<q> - start quotation. Used instead of quotation marks or

<lq> - begin long quotation. highlighting.

</q> - end of quotation.

<h#> - indicates heading. # indicates the heading level, defaulting to 1.

 - begin ordered list.

 - list item. Indicates an item to be formatted in the list.

<sl> - begin simple list. A simple list is not numbered and contains no bullets.

 - begin unordered list. An unordered list uses a special symbol to bullet each item.

</sl>   - end list. Lists may be nested.

 - end of list item.

<dl> - begin definition list.

</dl> - end of definition list.

<dthd> - definition list term heading.

<ddhd> - definition list description heading.

<dt> - definition list term.

<dd> - definition list definition.

< gl > - start glossary list.

< gt > - glossary term.

< gd > - glossary definition.

< lp > - list part.

< xmp > - begin example. Examples are processed verbaitum.

< /xmp > - end of example.

< pc > - continue paragraph.

< fig > - begin figure.

< figcap > - figure caption.

< figdesc > - begin figure description.

< hp > - begin highlighted phrase. 4 types of highlighting available.

< cit > - begin citation.

< frontm > - begin front matter.

< titlep > - begin title page.

< backm > - begin back matter.

< abstract > - begin abstract material.

< toc > - place table of contents.

< preface > - begin preface material.

< figlist > - place figure list.

< body > - begin document body (implies < /frontm >).

< appendix > - begin new appendix.

< index > - place index.

< glossary > -

< gdoc > - defines general document.

< /doc > - end of document.

< form > - define form.

< /form > - end of form.

< !ENTITY entity_reference_name entity_definition > - assigns a definition value to a reference for use in the document. Can be referenced later by &entity_reference_name;.

< !ENTITY entity_reference_name SYSTEM > - declares entity_reference_name as an object outside the SGML document.

< !ENTITY entity_reference_name SYSTEM "additional_info" > - if the entity_reference_name is not sufficient to bind its value.

Index