Honeywell

Build Generic Custom Displays

L61210

AG

TotalPlant®

Notices and Trademarks

This module supports **TotalPlant** Solution (TPS) system network.

TPS is the evolution of TDC 3000^{X}.

Honeywell Inc.

Industrial Automation and Control

Automation College

2820 West Kelton Lane

Phoenix, AZ 85053-3028

1-800 852-3211

Table of Contents

Figures and Tables

Acronyms

AM..... Application Module
AMC.....Advanced Multifunction Controller
APM ..... Advanced Process Manager
CL .....Control Language
DDB ..... Universal Station Display Database
EXTIDLST.....list of External Point Identifiers in Area Group
HG .....Hiway Gateway
HM .....History Module
INTIDLST.....List of Internal Point Identifiers in Area Group
LOAD ..... Load User DDB
LOADEQ..... Load Equipment List DDB
MC .....Multifunction controller
NIM .....Network Interface Module
PC.....Personal Computer
PM..... Process Manager
PSDP ..... Processor Status Data Point
TIP .....Text Input Port
UNLOADEQ.....Unload Equipment List DDB
US..... Universal Station

Parameters

DATIMEn .....Date/Time Type Local DDB Variable
DATIMEnG .....Date/Time Type Global DDB Variable
ENMnn.....Enumeration Type Local DDB Variable
ENMnnG ..... Enumeration Type Global DDB Variable
ENTnn..... Entity Type Local DDB Variable
ENTnnG..... Entity Type Global DDB Variable
INTnn .....Integer Type Local DDB Variable
INTnnG ..... Integer Type Global DDB Variable
PV ..... Process Variable
REALnn.....Real Type Local DDB Variable
REALnnG.....Real Type Global DDB Variable
STRINGnn .....String Type Local DDB Variable
STRINGnnG .....String Type Global DDB Variable
TRENDnn .....Trend Type Local DDB Variable
TRENDnnG.....Trend Type Global DDB Variable
VARnn.....Variable Type Local DDB Variable
VARnnG..... Variable Type Global DDB Variable

References

Publication Title	Publication Number	Binder Title	Binder Number
Picture Editor Reference Manual	SW09-650	Implementation/Engineering Operations-2	TDC 3032-2
Actor's Manual	SW09-655	Implementation/Engineering Operations-2	TDC 3032-2
Equipment List Reference Manual	SW27-560	Implementation/Engineering Operations-3	TDC 3032-3
Text Editor Operation	SW11-506	Implementation/Engineering Operations-3	TDC 3032-3

INTRODUCTION

Module Overview

About this module

This course module describes the Universal Station display database and how it can be used to create generic custom displays.

For the purpose of this discussion, a generic display is a single display that can be used to view multiple sets of data, depending upon target action.

The Universal Station Display Database (DDB) is the main tool for building generic displays.

The following scenarios describe various ways you can implement generic custom displays; this course module discusses the first three:

- several loops, units, or areas can be viewed from the same schematic after the operator selects which item to display,
- a basic schematic fills in with tagnames and values upon callup,
- equipment lists are used to define aliases, and
- parameterized tagnames and values are used in subpictures.

Objectives

Given stated display requirements, build a custom display that uses the display database to switch the display's context upon target selection.

Test

This course module's Criterion Test is to build a display that allows viewing of tags in unit 1 or tags in unit 2, depending on a target selection. The display's INITIAL action should cause tags in unit 1 to appear upon display call up.

DISPLAY DATABASES

What are DDBs

Description

Display databases (DDBs) are areas of US memory dedicated for custom display use. A DDB is either a local area or a global area. Table 1 describes the difference between local and global DDB areas

Table 1 – Description of Local and Global DDB Area

DDB Area	Description
Local	The Local DDB contains variables that are available only to the current custom display. The US initializes these data to the defaults at each invocation; therefore, you must configure the custom display to store useful data at each invocation.
Global	Global variables are available to any custom display on a specific US. Global means that the data remains in the US memory and can be referenced by different custom displays. The US initializes these data to the defaults only when you reload the station.
NOTE: The defaults of the Display Database are shown in Tables A-1 and A-2 of the *Actor's Manual*.	

Types of DDBs

Table 2 lists the six distinct DDBs available in a US. There is only *one local DDB area in a US*. This differs from the division of global data, where the DDBs have separate areas.

Table 2 – DDB Types

DDB	DDB Area
Standard, User, Equipment List, and Collectors	Local
Standard (see note 1)	Global
User (see note 2)	Global
Node Status	Global
Change Zone	Global
Honeywell Developer (see note 3)	Global

NOTES:
1. The Standard DDB is also referred to as the System DDB
2. The User DDB is also referred to as the Customer DDB.
3. The Honeywell DDB is used only by Honeywell developers. It is accessed by using a special PE command.

How to access DDB

Figure 1 illustrates that custom displays can access data contained in *any* of the DDBs. You use these Picture Editor tools to access DDBs.

- ADD VALUE command to view DDB variables,
- Target actions to read from and write to DDB variables, and
- VARIANTS and CONDITIONS to test DDB variables.

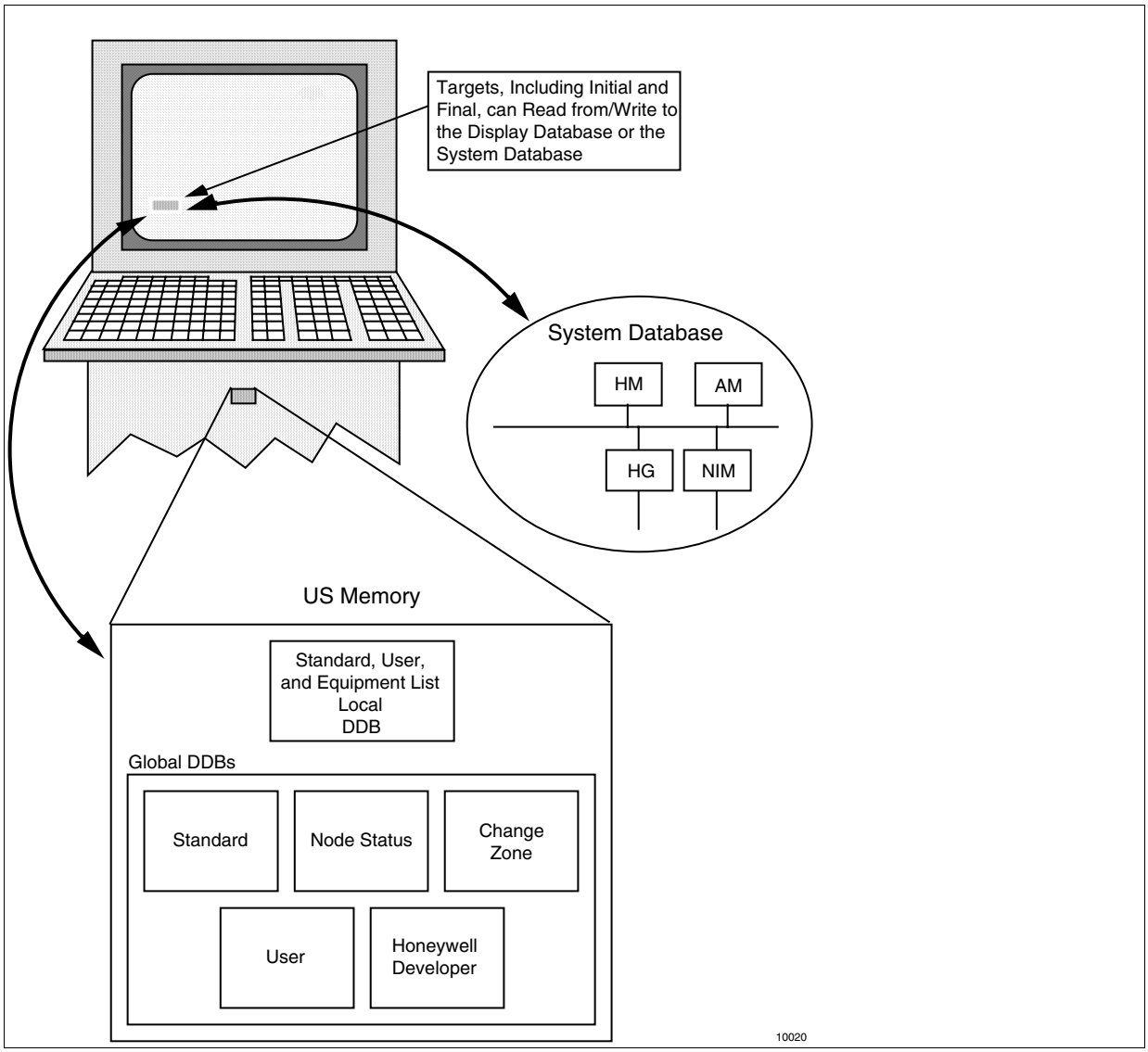


Figure 1 - Universal Station DDBs

DDB data types

Each local and global DDB area subdivides into separate areas for these different data types:

- an area containing *real* values,
- an area containing *integer* values,
- an area containing *boolean* values,
- an area containing *variable* values,
- an area containing character *strings*,
- an area containing *entity* names,
- an area containing *enumerations*, and
- an area containing *date and time* values.

User-defined DDBs

Several DDBs are predefined by Honeywell; the others are user-configurable:

User-defined

- User Local and User Global DDBs
- Equipment List Local DDB

Predefined

- Standard Local and Standard Global DDBs
- Change Zone Global DDB
- Node Status Global DDB

In the user-defined DDBs, the names and index numbers of the variables are defined by the user, within guidelines.

In the predefined DDBs, the names of the variables and index numbers are dedicated.

Index numbers

To locate a value in a user-defined DDB, you specify an index number.

User-defined DDB index numbers range from +100 to +32767.

Indexes can be negative numbers. This may seem unusual at first, but if you understand the index number range, it will not be a problem.

Although the predefined DDB variables have a known index number, the indexes are transparent to you when building a custom display. The only time you see the indexes of predefined DDB variables is

- in the output of a FIND NAMES search, or
- when using the Picture Editor PRINT command.

Layout of DDBs

Figures 2, 3, and 4 lay out the index numbers used by each DDB area.

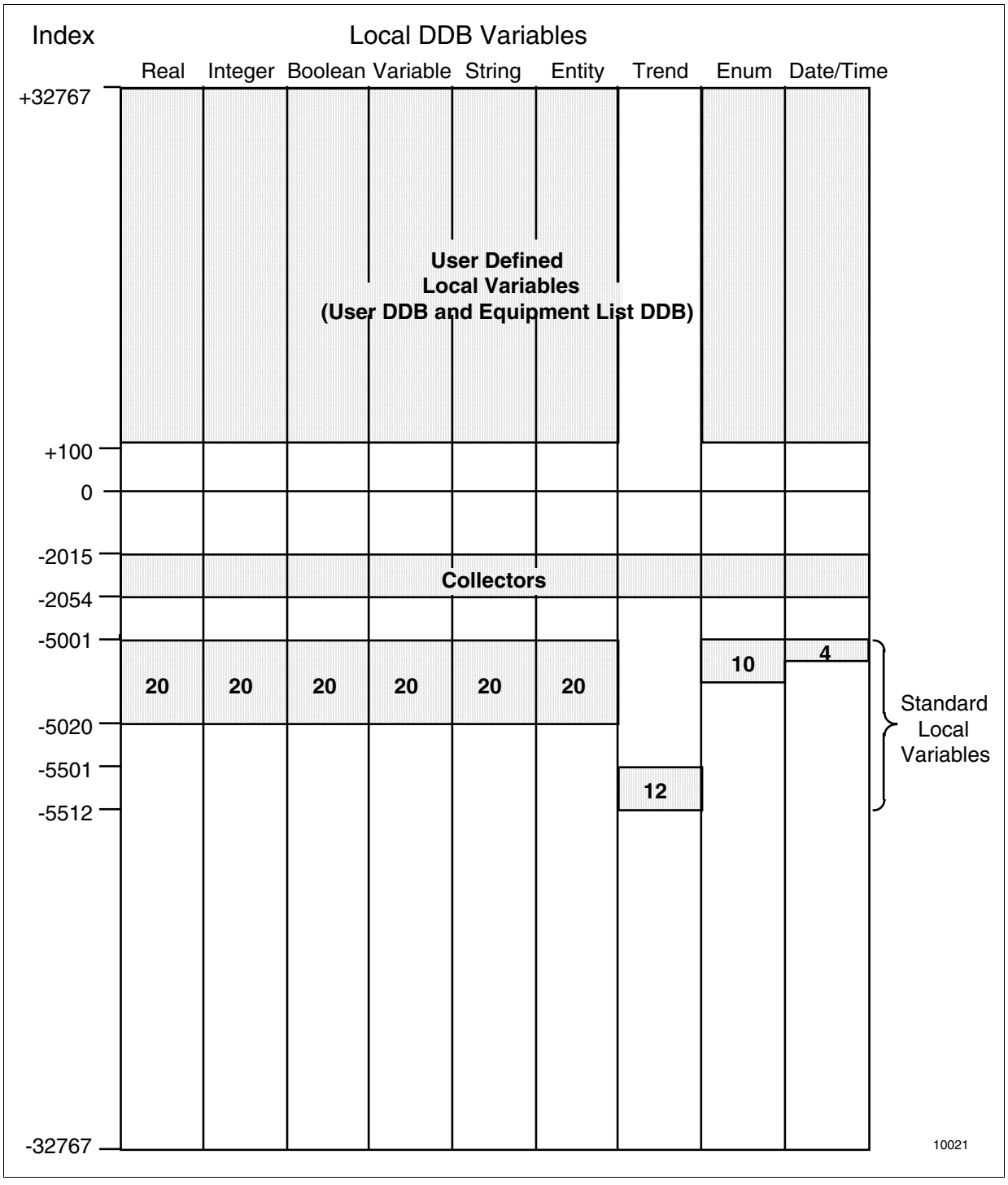


Figure 2 - Layout of Local DDB

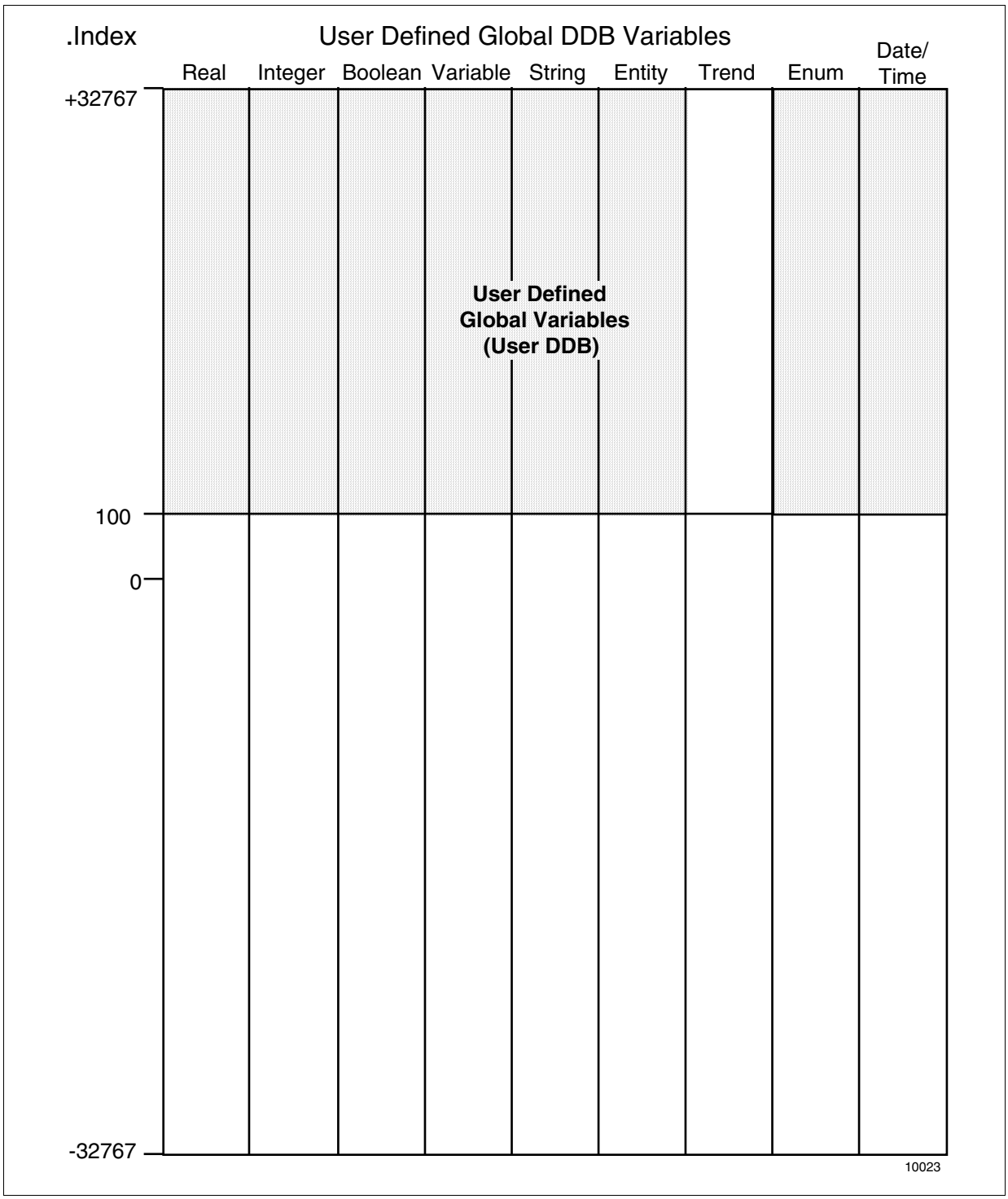


Figure 3 - Layout of User Global DDB Area

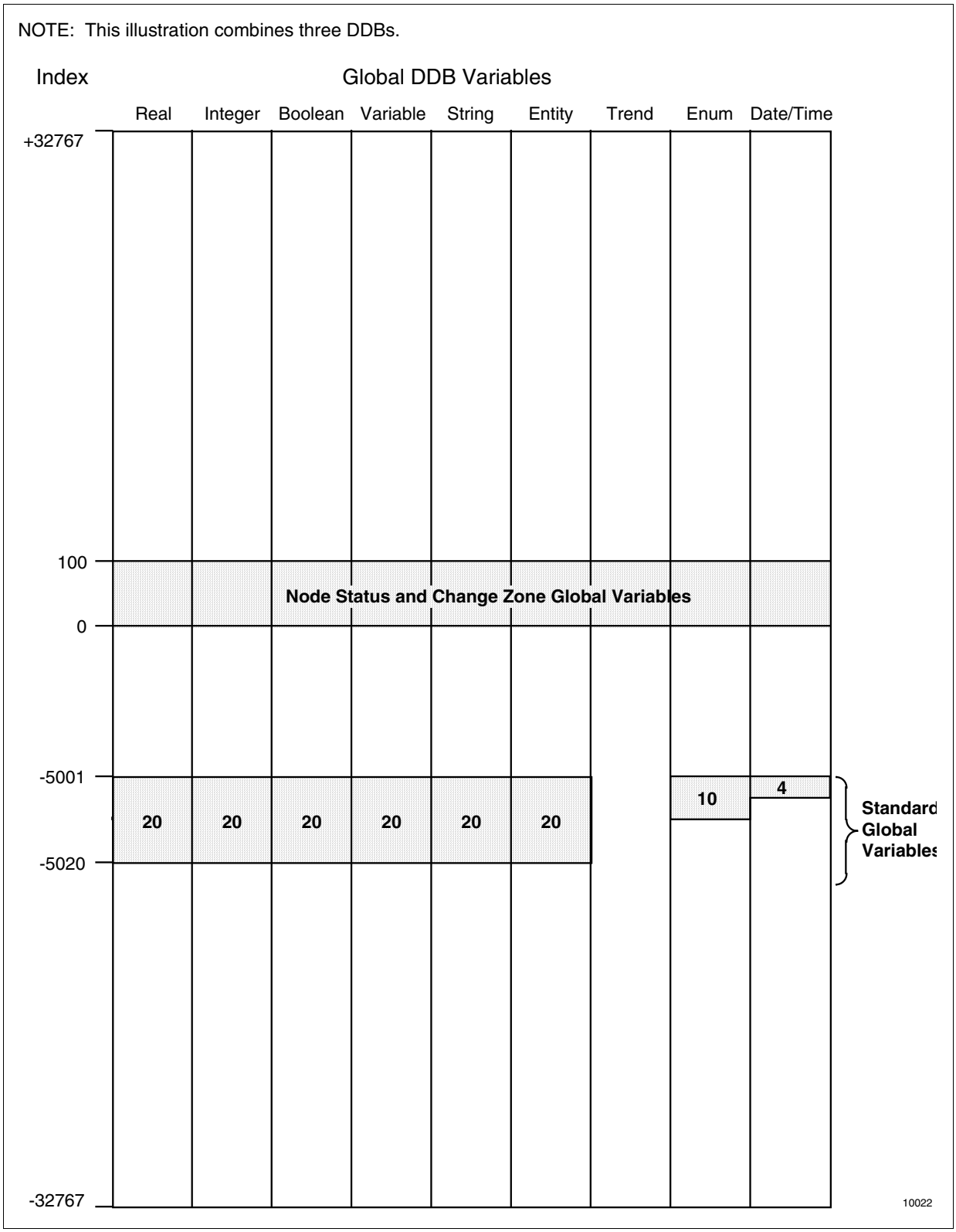


Figure 4 - Layout of Standard Global DDB Area

Standard DDB Overview

Variable names

The Standard DDB variable names have been defined by Honeywell.

Table 3 lists the Standard variable names (local and global).

Table 3 – Standard DDB Variable Names

Data Type	Standard *Local* DDB Variable Names	Standard *Global* DDB Variable Names
Real	REAL01 – REAL20	REAL01G – REAL20G
Integer	INT01 – INT20	INT01G – INT20G
Boolean	BOOL01 – BOOL20	BOOL01G – BOOL20G
Variable	VAR01 – VAR20	VAR01G – VAR20G
String	STRING01 – STRING20	STR01G – STR20G
Entity	ENT01 – ENT20	ENT01G – ENT20G
Trend	TREND01 – TREND12	(not applicable)
Enumeration	ENM01 – ENM10	ENM01G – ENM10G
Date/Time	DATIME1 – DATIME4	DATIME1G – DATIME4G

Global vs. local DDB

For practical purposes, the Standard Global and Standard Local DDBs are identical, with the following exceptions:

- they are separate DDBs,
- the Standard Global variable names end with a “G,”
- all custom displays within a US can use Standard Global variables, and
- the Standard Global DDB does not support the trend data type.

User DDB Overview

Description

Unlike the Standard DDB, which uses variable names defined by Honeywell, the User DDB is defined by the user. In fact, the User DDB is often referred to as the Customer DDB.

Index numbers

Regardless of data type, User DDB variables must have indexes within the range of 100 to 32767.

Defining

To define the User DDB, you create a text file. The text file contains statements declaring the names and indexes of the variables.

You can use either the TDC 3000^X Text File Editor or a PC editor and the Honeywell Text File Converter to create the text file.

The Picture Editor obtains the text file when you execute the LOAD command.

ATTENTION

ATTENTION—If you LOAD a User DDB declare file when building a custom display, the declare file is not unloaded unless you exit the Picture Editor. This may affect any other graphic you might edit during the same Picture Editor session.

Node Status and Change Zone DDB Variables

Variable names

For your reference, Table 4 lists the names of special read-only global variables located in the Node Status and Change Zone DDBs.

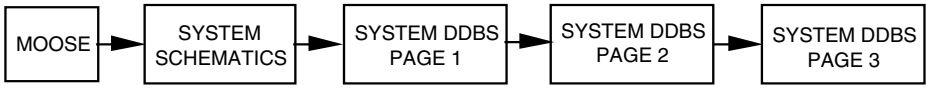
Table 4 – System DDB Variables

***Current* Area/Console Data**		
Variable	**Data Type**	**Description**
$AL_ENTY	Entity	Contains name of the last point selected from an Alarm Summary display or an Organizational Summary display (R500 and later).
$ARAID	String (8-char.)	Contains name of *current* area.
$ARADSC	String (24-char.)	Contains descriptor of *current* area.
$CONDSC	String (24-char.)	Contains descriptor of *current* console.
$CONNUM	Integer	Represents number of *current* console.
$DT_ENTY	Entity	Contains name of the last point for which a Detail display was invoked (R430 and later).
$KEYLEVL	Enumeration	Contains *current* keylock access level of US (OPR, SUP, or ENGR)
$MY_AREA	Integer	Represents area number on which *current* custom display is running.
$MY_PNA	Integer	Represents node number on which *current* custom display is running.
$PERSON	Enumeration	Contains current personality running in US. For example, USUNPR (universal personality).
$STNNUM	Integer	Represents the station number on which *current* custom display is running.
***Other* Area/Console Data**		
$ARAID01 – $ARAID10	String (8-char.)	Contains name of area specified by variable.
$ARADS01 – $ARADS10	String (24-char.)	Contains descriptor of area specified by variable.
$CONDS01– $CONDS10	String (24-char.)	Contains descriptor of console specified by variable.
$GRPBASE	Entity	Used to access group definitions.

Change Zone Data		
Variable	**Data Type**	**Description**
$CZ_ACT	Boolean	True = change zone Active, False = change zone Inactive.
$CZ_ENTY	Entity ID	Contains point name that is provided as a parameter to the CHG_ZONE or USER_CZ actor.
System-Wide Alarm Configuration Data		
$ALMCOLR	Integer	Specifies the alarm priority color option selected in the NCF. 0 = two color option (red, yellow) 1 = three color option (user selected colors)
$EALMCOLR $HALMCOLR $LALMCOLR	Integer Integer Integer	Specifies the alarm priority color option selected in the NCF for Emergency, High, and Low priority alarms (1=red, 2=green, 3=yellow, 4=blue, 5=magenta, 6=cyan, 7=white).
$REDYEL	Enumeration	Contains minimum alarm priority indicated by the color red. The enumeration set is ALPRIOR. It contains the members LOW, HIGH, and EMERGNCY. If $ALMCOLR=1, then this variable represents button lamps only.

Moose

These displays on your course cartridge summarize the system variables:



REFERENCING DDBS

Direct vs. Indirect References in Custom Displays

Types

Table 5 describes the two types of variable referencing that can be used in a custom display.

Table 5 – Types of Variable Referencing

Type	Description	Example
Direct *(specific)*	Direct references specify a system entity and parameter. They usually take this form: POINT.PARAMETER POINT—specific system point name PARAMETER—valid system parameter for point	`FIC100.PV`
Indirect *(generic)*	Indirect references point to a DDB location in the US memory instead of to a system entity. The DDB location contains a pointer to the data.	`ENT01.PV`

Direct references

When compiling a display that contains direct references, the Picture Editor does the following:

- obtains the internal identifiers (IDs) of the referenced points from the data owners, and
- binds (links) the internal IDs to the display.

This requires you to *build and load all of the referenced points before compiling* the custom displays. At run time, the display directly accesses the data using the internal IDs.

If you delete and rebuild a point, its internal ID changes, its bound pointer in display objects becomes mismatched, and you must *recompile*. This is one of the disadvantages of using direct references in custom displays.

Indirect references

A reference to a DDB location is generic (as opposed to specific) because it can be changed at run time to point to different data.

Because a DDB address satisfies the compiler, the point data that the display will ultimately fetch *does not have to exist at compile time*.

Binding occurs at run time through a target action that initializes the DDB before it is displayed.

Example

ENT01 is the variable name of a Standard DDB address used as a substitute for a point name (alias). Figure 5 illustrates ENT01**.**PV as an indirect reference to FIC100.PV.

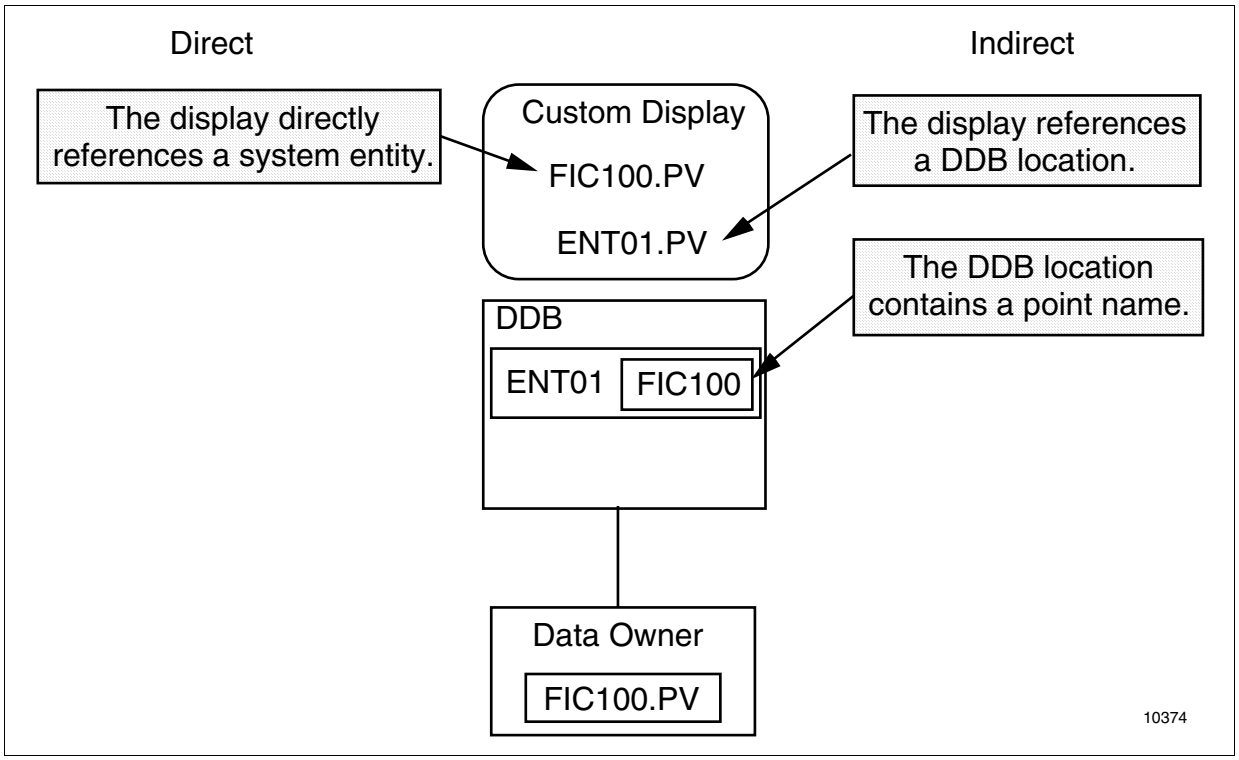


Figure 5 - Examples of Direct and Indirect Referencing

Summary

When deciding whether to use direct or indirect references, you should consider the advantages and disadvantages summarized in Table 6.

Table 6 – Advantages and Disadvantages of Direct vs. Indirect References

Type	Advantage	Disadvantage
Direct	*Reduced building time* required for a single custom display.	Before you can compile a display *you must build all referenced points and load them into their data owners*.
	Easier to analyze how a display works using the Picture Editor.	*If you delete and rebuild a point, you must recompile* the displays in which it is directly referenced. If many displays are affected, recompiling can be time-consuming.
	Less overhead at run time, because direct references are bound to the display at compile time. In a large display, this may noticeably increase invocation speed.	
Indirect	Can display *multiple sets* of data.	*Can be difficult to analyze* from the Picture Editor because of complexity.
	After deleting and rebuilding points, you do not have to recompile their displays if you used indirect references.	*Requires more overhead* at run time. Depending on the complexity of the display, it may be noticeably slower.
		You must be careful not to change DDB values that would conflict in another custom display.

When to Use Entity and Variable Data Types

Description

All of the different data types can be very useful in building generic custom displays; however, two data types deserve extra discussion because they are the ones used to collect system values (Point**.**Parameter) directly:

- Entity data type
- Variable data type

Entity data type

When you require an alias point name in the DDB, substitute Entity type variables for the point names.

Example:

The Picture Editor compiler accepts ENT01G**.**PV at compile time because ENT01G merely references a DDB address.

Of course this will not work if you do not initialize ENT01G to contain a system point name. To initialize the variables, you create a TARGET action that executes at run time. At run time, any place the custom display references ENT01G**.**PV, the display uses the current PV for the point assigned to ENT01G.

Variable data type

The *Variable* data type takes indirection a step further because it can substitute for the entire Point**.**Parameter used to fetch a system value.

The Picture Editor compiler accepts VAR01G as a substitute for a system value such as FIC100**.**PV.

A TARGET action must initialize the DDB variable to point to a system value.

One difference in the use of the VARIABLE data type is that if, for example, you assign VAR01G to FIC100**.**PV, you must reference the DDB value by using the form VAR01G**.** (note the period), which returns the current value of FIC100**.**PV.

Example:

If FIC100.PV = 25 and
VAR01G = FIC100.PV, then
VAR01G**.** = 25.

Actors Used to Manipulate DDB

Description

To read from or write to DDB variables, the display must execute an Actor or a series of Actors. You add Actors to a custom display by using

- the ADD TARGET command, or
- the DEFINE command.

After initializing DDB variables to contain correct data, you can use them to display VALUES, and test them with CONDITIONS and VARIANTS.

DEFINE command

You can use the DEFINE command to associate target actions with

- several definable keys on the operator's keyboard,
  - PAGE_FWD and PAGE_BACK
  - DISP_FWD and DISP_BACK
  - HELP and ASSOC_DISP
- invocation of the custom display (DEFINE INITIAL),
- exit from the custom display (DEFINE FINAL).

DEFINE INITIAL is useful for initializing a DDB before its display. DEFINE FINAL is useful for propagating data.

Store to DDB

Table 7 lists the Actors that can be used to store to DDB variables:

Table 7 – Actors Used to Store to DDB

Use this Actor...	To store this...
S_BOOL	Boolean data
S_DATE	a Date
S_DUR	a Time Duration
S_ENT	an Entity ID
S_INT	an Integer
S_REAL	a Real number
S_SENM	a Standard Enumeration
S_STR	a Character String
S_TIME	a Time
S_VAR	a Variable ID (Point.Parameter)

Read from DDB

Table 8 lists the Actors that can be used to read (GET) DDB variables:

Table 8 – Actors Used to Read from DDB

Use this Actor...	To get this...
G_BOOL	a Boolean value
G_DATIME	a Date and Time
G_ENT	a Entity ID
G_INT	an Integer value
G_REAL	a Real number
G_SENM	a Standard Enumeration
G_STR	a character String
G_VAR	a Variable ID (Point.Parameter)

Read and store

The RS_LOC actor reads data from a Text Input Port (TIP) and stores it in a specified DDB variable.

This actor cannot be used for self-defining enumerations.

Read and convert

Table 9 lists the actors used to read and convert data types (These do not have to be preceded by a GET.). These actors read the internal representation of a local DDB and convert it to its external representation.

Table 9 – Actors to Read and Convert Local DDB Data

Use this Actor...	To read this ...	And convert it to this...
IE_ENT	Entity ID	its external (String) representation
IE_VAR_E	Variable ID (Entity ID portion)	
IE_VAR_P	Variable ID (Parameter portion)	
IE_VAR	Variable ID (Point.Parameter)	

ATTENTION

ATTENTION—Actors used with configurable buttons and Variants used in Free Format Logs cannot reference the display database.

Examples

The following are examples of how to *indirectly* specify a Point**.**Parameter.

Example 1:

Use the actor S_ENT to store an Entity ID in the Standard Local DDB.

```
S_ENT(ENT02,FIC100)
```

Example 2:

Use the actor S_VAR to store a Variable ID in the Standard Local DDB.

```
S_VAR(VAR01,FIC100.PV)
```

NOTE: A display object (value, condition, variant, or target) that requires a variable ID like FIC100**.**PV accepts either VARnn**.** or ENTnn**.**PV for its Variable ID.

Additional information

For details on any of the Actors, see the *Actor's Manual.*

DEFINING THE USER DDB

Using Text Editor to Create a User DDB

Description

To define the names of variables in the User DDB, you create a text file referred to as the User DDB declare file.

After creating the text file you access the Picture Editor and execute the LOAD command, which reads the User DDB declare file into the Picture Editor and allows you to reference the DDB variables.

Procedure

Table 10 describes the procedure to define the User DDB.

Table 10 – Procedure to Define User DDB File

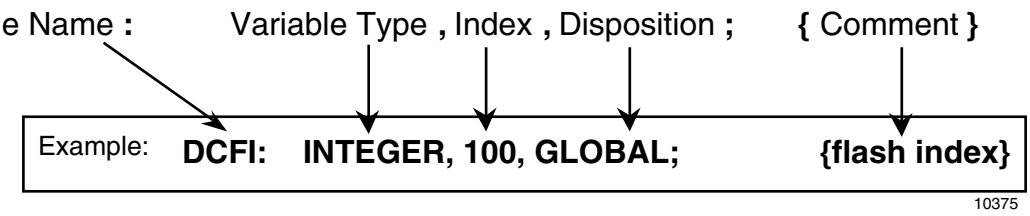
Step	Action
1	Access the Command Processor: Select COMMAND PROCESSOR target on Engineering Main Menu, or Press the [ESC] key while using the Picture Editor.
2	Call up the Text Editor: Type-in `EDIT` nnnnnnnn`.DF` then press [ENTER]. nnnnnnnn is the filename you are assigning to the DDB declare file.
3	Type in the statements to declare each variable you want to use in the custom display. Format: Variable Name: Variable Type, Index, Disposition; Example: `BOOL50: Boolean, 500, Local;`
4	QUIT, then EXIT the Text Editor.

The *Text Editor Operation* manual, SW11-506 explains how to use the Text Editor.

Syntax of User DDB Declare File

Syntax

Figure 6 shows the syntax for statements in a User DDB declare file.



Notes:

1. A statement must begin and end on the same line.
2. More than one statement can appear on a line.
3. Spaces can appear anywhere, except in identifiers such as the Variable Name.

Variable Name

- Up to eight characters (A-Z, a-z, 0-9, and underbar)
- First character must be a letter. Last character cannot be an underbar, and there cannot be two consecutive underbars.

Variable Type	**Synonyms**
• Integer	Int, I
• Number	Num, Real, R
• Boolean	B, Logical, L
• String	S
• Enum:<data type>	E
• Date_Time	Date/Time, Date
• Variable	Var, V
• Entity_ID	Entity, Ent

Index

- Index numbers **100 – 32767** are available for **each type** of global and local user DDB variables.
- 500 indexes maximum per declare file.

Disposition	**Synonyms**
• Local	Loc, L
• Global	Glob, G

(User globals are not initialized. There is a memory limit of 1000 words for user globals.)

Comment

- A-Z, a-z, 0-9, any special characters.
- Must be enclosed in braces { }. Must begin and end on the same line.
- Can appear before and/or after the declare statement.

Figure 6 - User DDB Declare File Syntax

Rules

Figure 7 illustrates the rules for a User DDB declare file.

```
{    Note that blank lines are legal      }

SEL_ENT : ENTITY , 200,  G ; { Spaces are allowed  }
SEL_ENT2: ENT,200,L;              { but not necessary. }

my_int: int, 100, local;          { Lower case is okay. }

REAL01: R,  500,  L;         { This will not be recognized  }
                             { because it is a duplicate of }
                             { a Standard DDB name.           }

BOOL50: Boolean, 500, L;     { This is okay. }
BOOL50: Boolean, 250, L;     { This will not be recognized }
                             { because it is a duplicate name. }

XYZZY : STRING,  400, G;     { This is okay. }
PLOVER: STRING,  400, G;     { This is an alias of XYZZY. }

My_Date:Date_Time, 700, L;    My_Int3:Integer, 5, GLOBAL;

{  This is okay  }    My_Enum :  ENUM:MODE,  450,  LOCAL;

{   All of the above are syntactically correct. }
{   All of the following have a syntax error.    }

_Int: INTEGER, 180, G;       { Cannot start with underbar. }
10Real: REAL,  190,  Glob;        { Cannot start with numeric. }
STR__5: S,  230,  G;              { Consecutive underbars are illegal. }
INTEGER20 : INTEGER, 260, L; { Name is longer than 8 characters. }
My_Var_1 : VAR,  330,  G          { Missing semicolon. }
My_Ent :  ENTITY,  Loc;           { Missing index. }
My_Bool: LOGICAL, {  } 400, G;    { Comments within a statement. }
My_Real: REAL,  50,   G;          { Index out of range.   }
```

Figure 7 - Rules For User DDB Declare File

Example

Figure 8 shows a portion of a declare file. This particular declare file is used for several schematics, as indicated by the comments in the file.

```
                                {description                    schematic}
                                {-------------                -------------}
PD1E:       ENTITY,121,GLOBAL;  {data point entity               XDPDEF0M}
DCFI:       INTEGER,121,GLOBAL; {flash index                     XPTC060M}
M11B:       BOOLEAN,121,LOCAL;  {record boolean                  XMAN030M}
:           DATE,121,LOCAL;     {record timestamp                XMAN030M}
M21R:       REAL,121,LOCAL;     {record value                    XMAN030M}
CF1S:       STRING,121,LOCAL;   {footnote string                 XCCHRT0M}
CH1B:       BOOLEAN,121,GLOBAL; {history selected upper          XCCHRT0M}

M21B:       BOOLEAN,122,LOCAL;  {record boolean                  XMAN030M}
ACVI:       INTEGER,122,GLOBAL; {variant logic index             XALRM60M}
M02D:       DATE,122,LOCAL;     {record timestamp                XMAN030M}
M22R:       REAL,122,LOCAL;     {record value                    XMAN030M}
CH2B:       BOOLEAN,122,GLOBAL; {history selected lower          XCCHRT0M}
```

Figure 8 - Example of User DDB Declare File

ATTENTION

ATTENTION—Limitations of the User DDB are described below:

- Index duplication within a data type and a disposition are not detected. This causes the names to be aliases for the same location.
- Name duplication within a DDB file is not detected; the first name encountered takes precedence.
- Duplication of names in the DDB file and *Standard* DDB names are not detected. The Standard name takes precedence.
- The maximum number of DDB Variable names per file is 500. Exceeding this number of DDB Variable names causes an error when the file is loaded.
- There is a memory limit of 1000 words for user defined global DDB variables. The size of each DDB variable is shown in Table A-4 of the *Actor's Manual*.

Honeywell DDB

Honeywell developers use a special statement (not shown in Figure 6) to allocate locations in the Honeywell DDB, which is independent and does not interfere with the user DDB.

Loading A User DDB

Loading

A User DDB file loaded into the Picture Editor remains there until you exit the Picture Editor or overwrite the file by loading another User DDB file.

ATTENTION

ATTENTION—The Picture Editor can contain only one User DDB file at a time, although it allows you to enter more than one LOAD command without giving an immediate error message.

The *last* User DDB file loaded is available to the Picture Editor.

When you compile the display, the Picture Editor gives an error message on references to variables not found in the currently loaded DDB file: `Point or Parameter Not Found.`

USING EQUIPMENT LIST IN CUSTOM DISPLAYS

Equipment List DDB Overview

Description

With R400, came a new way of referencing generic variables from a custom display, the Equipment List.

The Equipment List is a powerful way to do context switching. You are actually switching different sets of point names that have identical or similar functions.

The Equipment List DDB has a somewhat broader application than the other DDBs because control language (CL) programs in process-connected devices (such as the PM, APM, MC, and AMC) can also use the Equipment List. Equipment List provides a means to integrate CL programs and custom displays, allowing the user to switch between two or more different but similar Equipment Units that use the same batch recipes.

The Equipment List also provides additional capabilities for custom displays alone.

Index numbers

Equipment List DDB index numbers range from 100 through 32767.

The Equipment List DDB and the User Local DDB are one and the same DDB; therefore, take care to avoid index duplication if a custom display uses both.

The Equipment List files are

- Equipment List source file—**.**QS
- Equipment List object file—**.**QO

You define the Equipment List DDB in a text file (**.**QS source file), similar to the User DDB declare file.

The difference between the User DDB declare file and the Equipment List declare files is that

- the Equipment List declare file has different syntax, and
- you must compile the Equipment List declare file to produce an object file (**.**QO). The custom display references the object file to switch contexts at *run time*.

ATTENTION

ATTENTION—At build time, the Picture Editor references the Equipment List object file with the LOADEQ (LDQ) command.

You must compile the Equipment List DDB declare file in order to reference it with the Picture Editor LOADEQ command.

After you finish building a custom display that uses an Equipment List DDB, if you intend to build a new display with a different Equipment List DDB file or one with no Equipment List DDB file, enter the UNLOADEQ command to clear the DDB file from the Picture Editor.

Equipment List Concepts

Unit Class

The first part of the Equipment List declare file is called the Unit Class definition; it is analogous to the entire User DDB declare file.

The Unit Class associates User Local DDB indexes with variable names referred to as *aliases*. These aliases can be referenced by a custom display.

The Unit Class definition can stand alone as the only part of the declaration file, if context switching is to be done; for example, where a target action performs context switching by storing variables in the User Local DDB at run time.

Unit Instance

The second part of the Equipment List declare file is the Unit Instance definition; it has no counterpart in the User DDB declare file.

Equipment unit groupings called Unit Instances contain aliases to which the Unit Instance definition assigns point names.

Steps to implement

You must perform the following steps to implement an Equipment List DDB:

1. Create the Equipment List declare file (**.**QS) using the Text Editor,
2. Compile the declare file by using the Equipment List Builder command (ELB) in the Command Processor. Compiler errors are reported in the **.**QE file. A successful compile produces an object file (**.**QO).
3. Load the resulting **.**QO file into the Picture Editor using the LOADEQ command (To list all currently loaded object files, enter LISTEQ.).

4. Add the alias values to the display, if desired.
   (Examples: Alias Name**.**PV and Alias Name**.**NAME)
5. Add Target action or Initial action that includes the EQ_LIST actor to initialize the DDB as specified in the **.**QO file.

Limits

These limits apply to Equipment Lists:

- A custom display may be loaded with a maximum of **eight .**QO files.
- Each .QO file may contain:
  - 199 max. alias definitions, and
  - 32 max. Unit Instances

EQ_LIST actor

Instead of "stuffing" Entity type variables individually by target action, as in the case of User or Standard DDBs, a special actor (EQ_LIST) is provided to switch contexts between the specific Unit Instances (points, constants, and DDB variables) found in the Equipment List declare file:

```
EQ_LIST("object filename", "Unit Instance
                    name")
```

Unit Instances add a runtime capability to Custom displays that use them. Runtime efficiency is gained because only one actor needs to be executed to specify the object file and the selected instance.

Whereas "Stuffing" DDB variables requires a DDB store for each variable, and there could be a large number of them.

Search paths

ATTENTION—It is important to note that the Equipment List declare file is not bound to the custom display object, as are subpictures for example. Instead it is accessed at runtime and, therefore, must be located by using the Area Database Pathname Catalog as a search path. You must be aware of the fact that in the case of the Equipment List declare file (.QO), the US searches the catalog in reverse order.

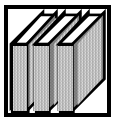
Selecting a target that references the Equipment List .QO file, causes the station to search the Area Database Pathname Catalog in reverse order.

If it is taking a considerable amount of time to display Equipment List custom displays, check the pathname catalog.

To prevent a long access time, you must configure your pathname catalog to accommodate the reverse search. Table 11 describes how to configure your pathname catalog.

Table 11 – Area Pathname catalog for Equipment List

IF …	THEN…
.QO and .DO files are in the *same* directory	Configure the directory at the beginning *and* at the end of the catalog.
.QO and .DO files are in the *different* directories.	Configure the **.**DO directory at the beginning and the **.**QO directory at the end of the catalog.



REFERENCE— For your future use, the *Equipment List Reference Manual* (SW27-560) provides a complete description of the Equipment List function and how to implement it. This manual is in Binder TDC 3032-3.

Equipment List Declare File

Format

Figure 9 shows the format of the Equipment List Declare File (.QS) as used with Custom Displays. There are other format considerations when the file is used in conjunction with Control Language Programs.

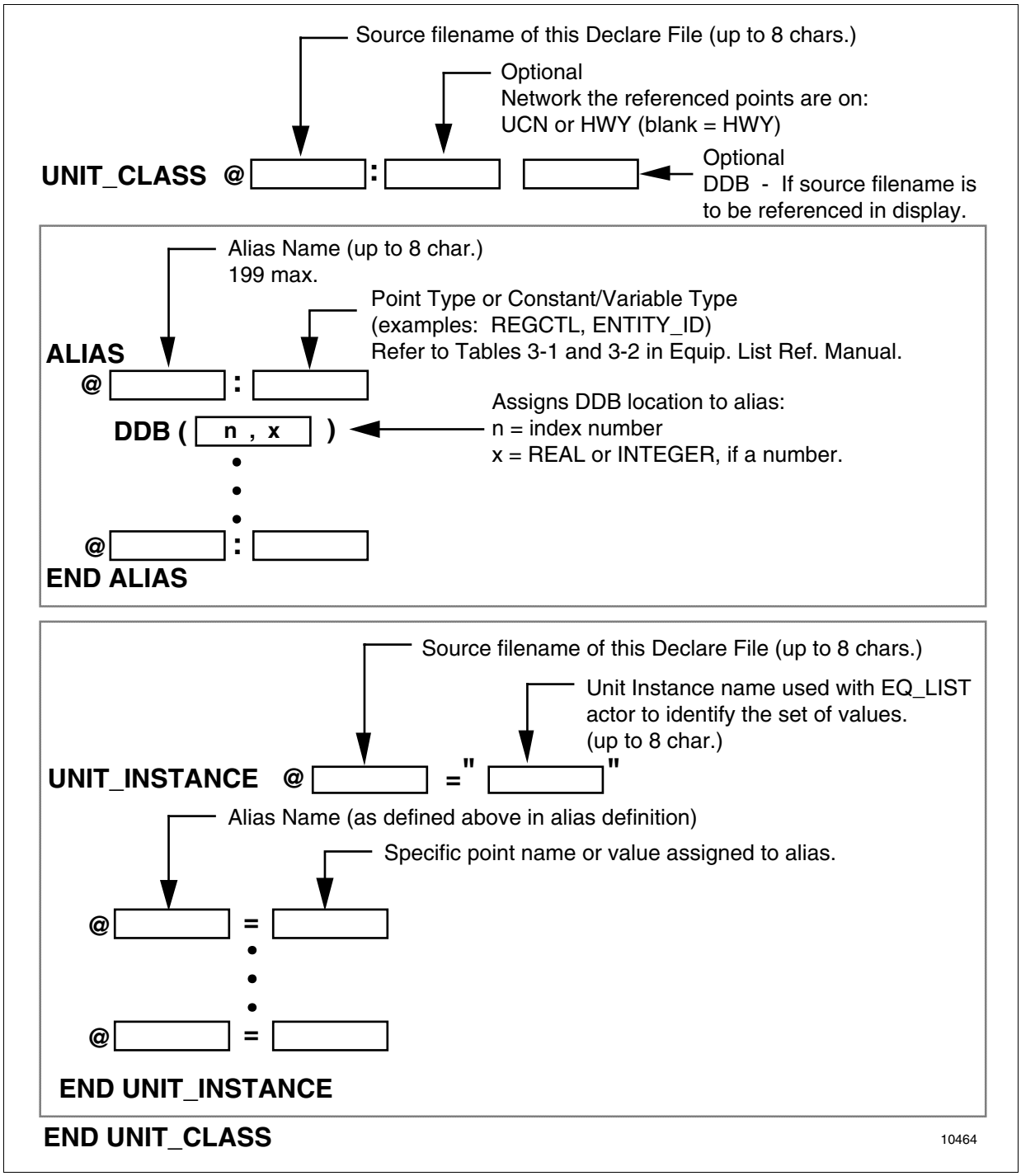


Figure 9 - Format of Equipment List Declare File For Custom Displays

Example

Figure 10 is an example of an Equipment List Declare File.

```
DDBLAB6.QS
10/11/93 16:24:51

UNIT_CLASS     @DDBLAB6: UCN

ALIAS

  @POINT9   : ENTITY_ID
  DDB(109)

  @POINT10  : ENTITY_ID
  DDB(110)

  @POINT11  : ENTITY_ID
  DDB(111)

  @POINT12  : ENTITY_ID
  DDB(112)

END ALIAS

UNIT_INSTANCE     @DDBLAB6 = "UNIT1"

  @POINT9  = RAMP331
  @POINT10 = RAMP332
  @POINT11 = RAMP333
  @POINT12 = RAMP334

END UNIT_INSTANCE

UNIT_INSTANCE     @DDBLAB6 = "UNIT2"

  @POINT9  = AGB331
  @POINT10 = AGB332
  @POINT11 = AGB333
  @POINT12 = AGB334

END UNIT_INSTANCE

END UNIT_CLASS
```

Figure 10 - Equipment List Declare File

USING UNIVERSAL STATION PSDP PARAMETERS IN CUSTOM DISPLAYS

Universal Station PSDP Parameters Overview

Description

With R610, additional PSDP parameters have been added to the US for system-wide access by any US or AM on the LCN. The parameters are read/write with an access level of Operator.

Parameter Types

Table 12 describes the new PSDP parameters.

Table 12 – PSDP Parameters

Parameter Name (Index)	Data Type
UREAL (1…100)	Real
UINTEGER (1...100)	Integer
UBOOL (1...100)	Boolean
UENT (1...100)	Entity
USTRING (1...50)	String

Use in Custom Displays

The PSDP parameters are reference in custom display condition and value expressions using the syntax

```
$PRSTSnn.param (index)
```

where nn is the US TPS node number.

For example, `$PRSTS12.UREAL(20)` references element 20 in the UREAL array on US node 12. Since these are system level parameters on the Processor Status Data Point, they are written using the Store Into System Database Actors (SS_) and, within targets, their values are read using the Real Data from the System Database Actors (GS_).

LAB EXERCISE 1

Overview

Introduction

In this lab exercise you will review the MOOSE information on generic schematics and DDBs.

Duration

This lab exercise should take about 30 minutes

Instructions

1. Call up MOOSE and select GENERIC DISPLAYS .
2. Work through the menu of information.

When you have finished, continue to Lab Exercise 2.

LAB EXERCISE 2

Overview

Introduction

In this lab exercise

- you will explore the capabilities of the Find Names utility as it relates to DDBs, and
- you will interpret the declare files for the User DDB and the Equipment List DDB.

Duration

This lab exercise should take about 30 minutes to complete.

Lab Exercise 2—Using Find Names to Search DDBs

1. Find Names

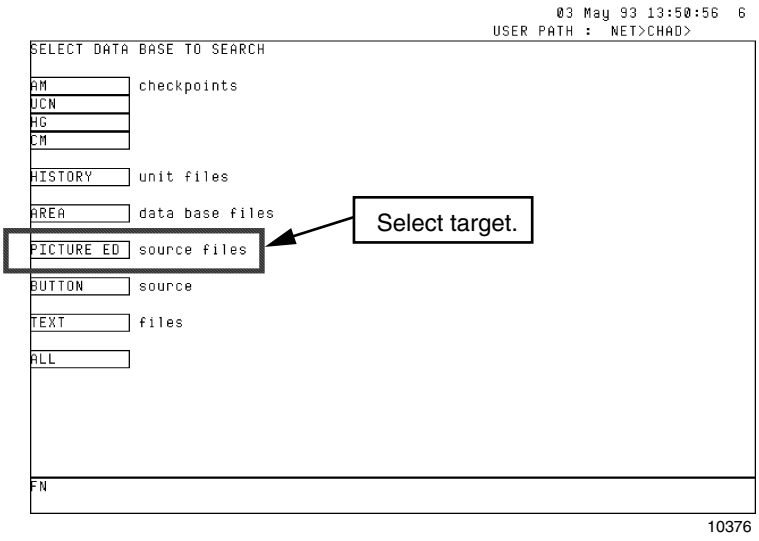
From the Command Processor, perform a Data Out command to a console printer, then call up the Find Names utility:

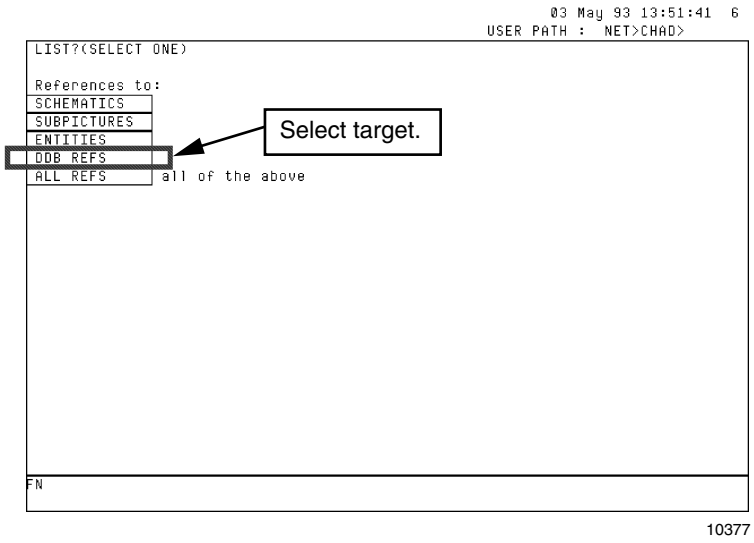
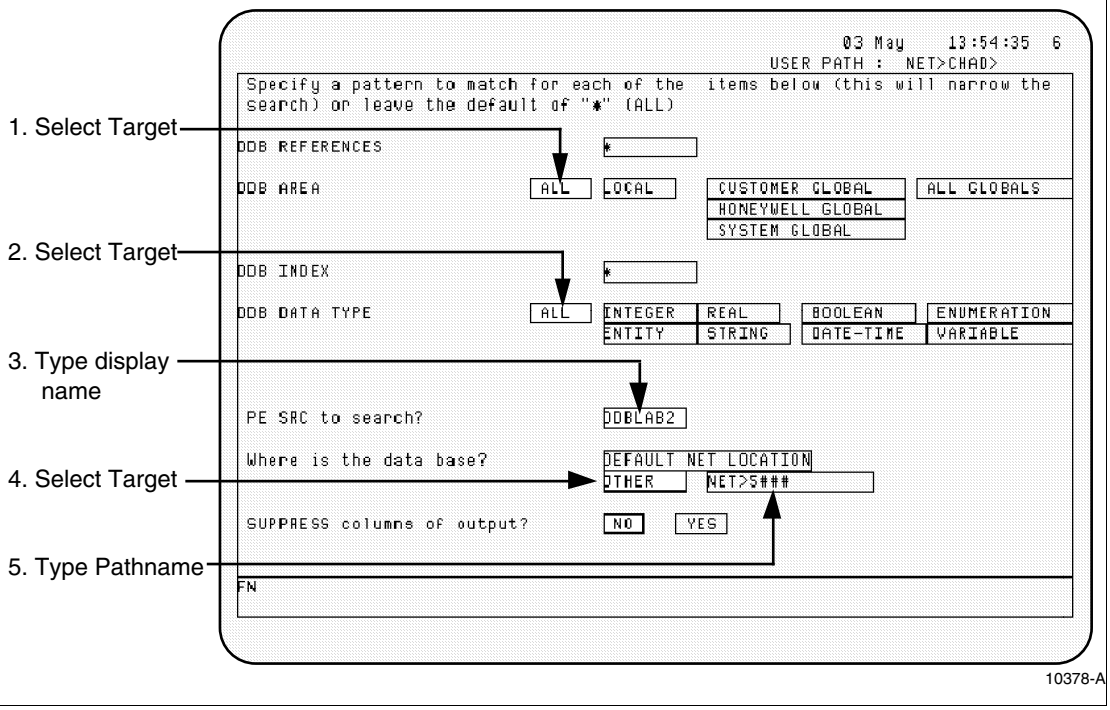
```
DO $Pn

FN
```

2. DDB references

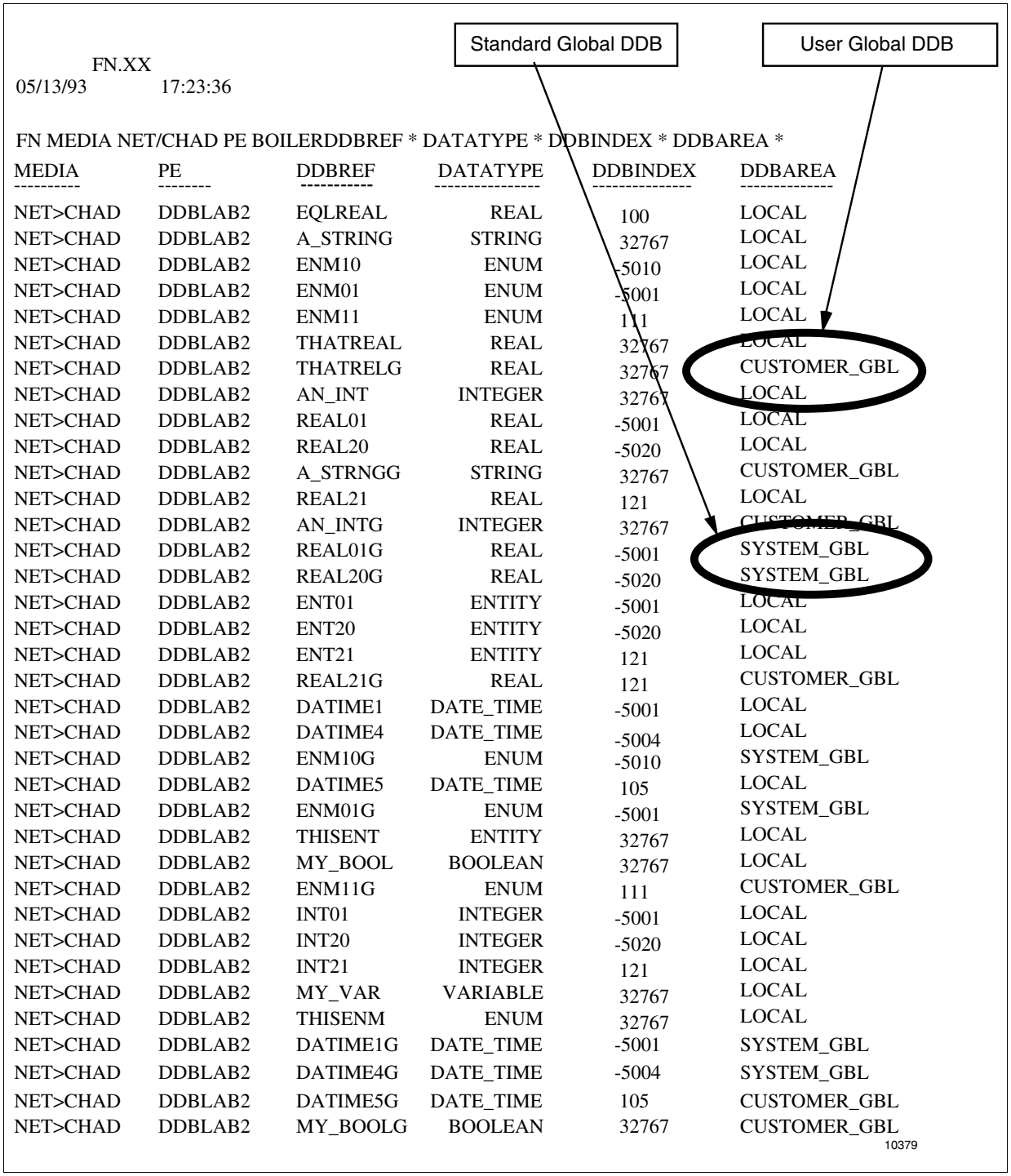
Perform the procedure in Table 13 to search the source file of DDBLAB2 for all DDB references.

Table 13 – Procedure to Use Find Names Utility for DDB Search

Step	Action
1	Select PICTURE ED 

Step	Action
2	Select DDB REFS 
3	Select these targets and type-in the information indicated: 1. DDB AREA: Select ALL 2. DDB DATA TYPE: Select ALL 3. PE SRC to search? Type display name DDBLAB2. 4. Where is the data base? Select OTHER 5. Type $Fn>GENP (n = right drive number). 

4	Press [ENTER]. RESULT: The Find Names utility searches display DDBLAB2 for DDB references and outputs its results to the printer. (Figure 11 shows an example of Find Names output.)



MEDIA	PE	DDBREF	DATATYPE	DDBINDEX	DDBAREA
NET>CHAD	DDBLAB2	EQLREAL	REAL	100	LOCAL
NET>CHAD	DDBLAB2	A_STRING	STRING	32767	LOCAL
NET>CHAD	DDBLAB2	ENM10	ENUM	-5010	LOCAL
NET>CHAD	DDBLAB2	ENM01	ENUM	-5001	LOCAL
NET>CHAD	DDBLAB2	ENM11	ENUM	111	LOCAL
NET>CHAD	DDBLAB2	THATREAL	REAL	32767	LOCAL
NET>CHAD	DDBLAB2	THATRELG	REAL	32767	CUSTOMER_GBL
NET>CHAD	DDBLAB2	AN_INT	INTEGER	32767	LOCAL
NET>CHAD	DDBLAB2	REAL01	REAL	-5001	LOCAL
NET>CHAD	DDBLAB2	REAL20	REAL	-5020	LOCAL
NET>CHAD	DDBLAB2	A_STRNGG	STRING	32767	CUSTOMER_GBL
NET>CHAD	DDBLAB2	REAL21	REAL	121	LOCAL
NET>CHAD	DDBLAB2	AN_INTG	INTEGER	32767	CUSTOMER_GBL
NET>CHAD	DDBLAB2	REAL01G	REAL	-5001	SYSTEM_GBL
NET>CHAD	DDBLAB2	REAL20G	REAL	-5020	SYSTEM_GBL
NET>CHAD	DDBLAB2	ENT01	ENTITY	-5001	LOCAL
NET>CHAD	DDBLAB2	ENT20	ENTITY	-5020	LOCAL
NET>CHAD	DDBLAB2	ENT21	ENTITY	121	LOCAL
NET>CHAD	DDBLAB2	REAL21G	REAL	121	CUSTOMER_GBL
NET>CHAD	DDBLAB2	DATIME1	DATE_TIME	-5001	LOCAL
NET>CHAD	DDBLAB2	DATIME4	DATE_TIME	-5004	LOCAL
NET>CHAD	DDBLAB2	ENM10G	ENUM	-5010	SYSTEM_GBL
NET>CHAD	DDBLAB2	DATIME5	DATE_TIME	105	LOCAL
NET>CHAD	DDBLAB2	ENM01G	ENUM	-5001	SYSTEM_GBL
NET>CHAD	DDBLAB2	THISENT	ENTITY	32767	LOCAL
NET>CHAD	DDBLAB2	MY_BOOL	BOOLEAN	32767	LOCAL
NET>CHAD	DDBLAB2	ENM11G	ENUM	111	CUSTOMER_GBL
NET>CHAD	DDBLAB2	INT01	INTEGER	-5001	LOCAL
NET>CHAD	DDBLAB2	INT20	INTEGER	-5020	LOCAL
NET>CHAD	DDBLAB2	INT21	INTEGER	121	LOCAL
NET>CHAD	DDBLAB2	MY_VAR	VARIABLE	32767	LOCAL
NET>CHAD	DDBLAB2	THISENM	ENUM	32767	LOCAL
NET>CHAD	DDBLAB2	DATIME1G	DATE_TIME	-5001	SYSTEM_GBL
NET>CHAD	DDBLAB2	DATIME4G	DATE_TIME	-5004	SYSTEM_GBL
NET>CHAD	DDBLAB2	DATIME5G	DATE_TIME	105	CUSTOMER_GBL
NET>CHAD	DDBLAB2	MY_BOOLG	BOOLEAN	32767	CUSTOMER_GBL

Figure 11 - Example of Find Names Output (DDB Search

Lab Exercise 2—Interpret Find Names Printout

3. Written exercise

Using your Find Names printout, Table 3, and Figures 1, 2, and 3 in this course module, answer the following questions about the DDB variables.

For example, ENT01 is a Standard (System) Local DDB name.
Its index number is -5001.

Variable	Which DDB?	Index Number
ENT01	Standard (System)	-5001
ENT21		
EQLENT		
THISENT		
REAL21		
REAL21G		
EQLREAL		
BOOL21G		
MY_BOOL		
INT21		
EQLINT		
AN_INTG		
STRING21		
A_STRING		
DATIME5G		
TIMENOW		

Lab Exercise 2—Interpret Find Names Printout

Solutions

Compare your answers to the table below.

How did you do?

If there are discrepancies that you cannot resolve, ask your course manager for assistance.

Keep your Find Names printout to show your course manager for the test.

Variable	Which DDB?		Index Number
ENT01	Standard (System)		-5001
ENT21	User	Local	121
EQLENT	User	Local	100
THISENT	User	Local	32767
REAL21	User	Local	121
REAL21G	User	Global	121
EQLREAL	User	Local	100
BOOL21G	User	Global	121
MY_BOOL	User	Local	32767
INT21	User	Local	121
EQLINT	User	Local	100
AN_INTG	User	Global	32767
STRING21	User	Local	121
A_STRING	User	Local	32767
DATIME5G	User	Global	105
TIMENOW	User	Global	32767

Lab Exercise 2—Interpret DDB Declare Files

4. Print declare files

From the Command Processor print these two DDB declare files on a console printer:

$Fn>GENP	where n = right drive
DDBLAB2.DF	(text file that declares User DDB variables)
DDBLAB2.QS	(text file that declares Equipment List DDB variables)

5. Interpret printout

Use these printouts to locate the same information about the DDB variables as specified in the table on previous page.

6. Cancel Data Out

Cancel the Data Out command:

```
DO
```

End of Lab 2

LAB EXERCISES 3 – 6

Overview

Introduction

In the following four lab exercises you will build generic custom displays using different DDBs to accomplish the same task.

Lab exercises 3 through 6 use this model:

Assume you must build a custom display that is generic to two units, Unit 1 and Unit 2. These units, for practical purposes, are identical except that the point names differ.

The custom display must allow the operator to switch between units by touching targets.

The display will allow the operator to view parameters for either Unit 1 or Unit 2.

To keep the labs simple, build the custom display to reference only four points per Unit (eight points total).

Lab exercises

Each lab exercise asks you to use different DDB variables:

- Lab Exercise 3—Standard Global DDB
- Lab Exercise 4—User Global DDB
- Lab Exercise 5—User Local DDB
- Lab Exercise 6—Equipment List Local DDB

Duration

Each lab exercise should take about 30 minutes to complete.

Lab Exercise 3—Using the Standard Global DDB

1. Build display

Build a display similar to Figure 12, using the following guidelines:

Targets

a. The display should have two targets, one associated with Unit 1, and the other associated with Unit 2.

   The Unit 1 target should initialize the *Standard Global DDB* entity variables to the following four point names:

   – RAMP331 – RAMP332
   – RAMP333 – RAMP334

   The Unit 2 target initializes the same variables to:

   – AGB331 – AGB332
   – AGB333 – AGB334

Condition

b. The box around the last selected unit target should turn green . Use a local DDB boolean variable to accomplish this. The other box should be low intensity cyan.

Values

c. Display the point names. They should dynamically change to display the currently initialized variables; that is, the Unit 1 or Unit 2 points from step a.

d. Display the current PVs for each point to the right of the point name.

Initial Action

e. The display should always come up at invocation with Unit 1 variables displayed. Use DEF INIT to accomplish this. This also triggers the condition from step b.

Text

Put a title on the display indicating the type of DDB variables used.

Related Displays

f. The [HELP] key should call up any display.

g. The [ASSOC DISP] key should call up any display.

Optional

h. Configure your help and associated displays to return to the display that called them when you press the [HELP] or [ASSOC DISP] key again (HINT: Make the return generic.)

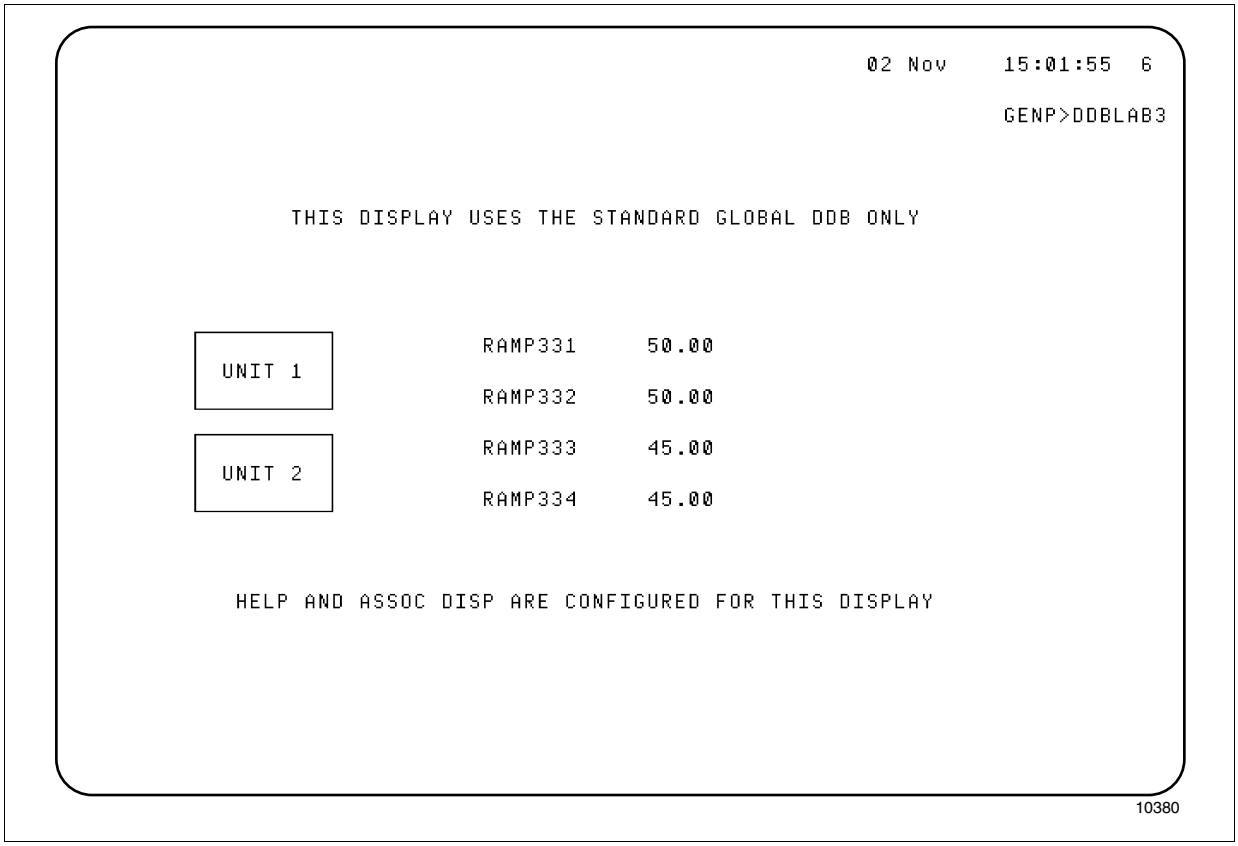


Figure 12 - Display for Lab 3—Standard Global DDB

2. Compile & Test

Compile and test your display.

When you are satisfied with the display, use the Picture Editor print command to print the targets and initial action. Keep the printouts to show your course manager for the end of module test.

End of Lab 3

Lab Exercise 4—Using the User Global DDB

1. Build display

Build a custom display that looks similar to Figure 13.

Requirements:

a. This display should operate the same as the display built in Lab Exercise 3, except that instead of Standard Global DDB variables, the Unit 1 and Unit 2 targets should initialize the *User Global DDB* entity variables to the point names specified in Lab Exercise 3.

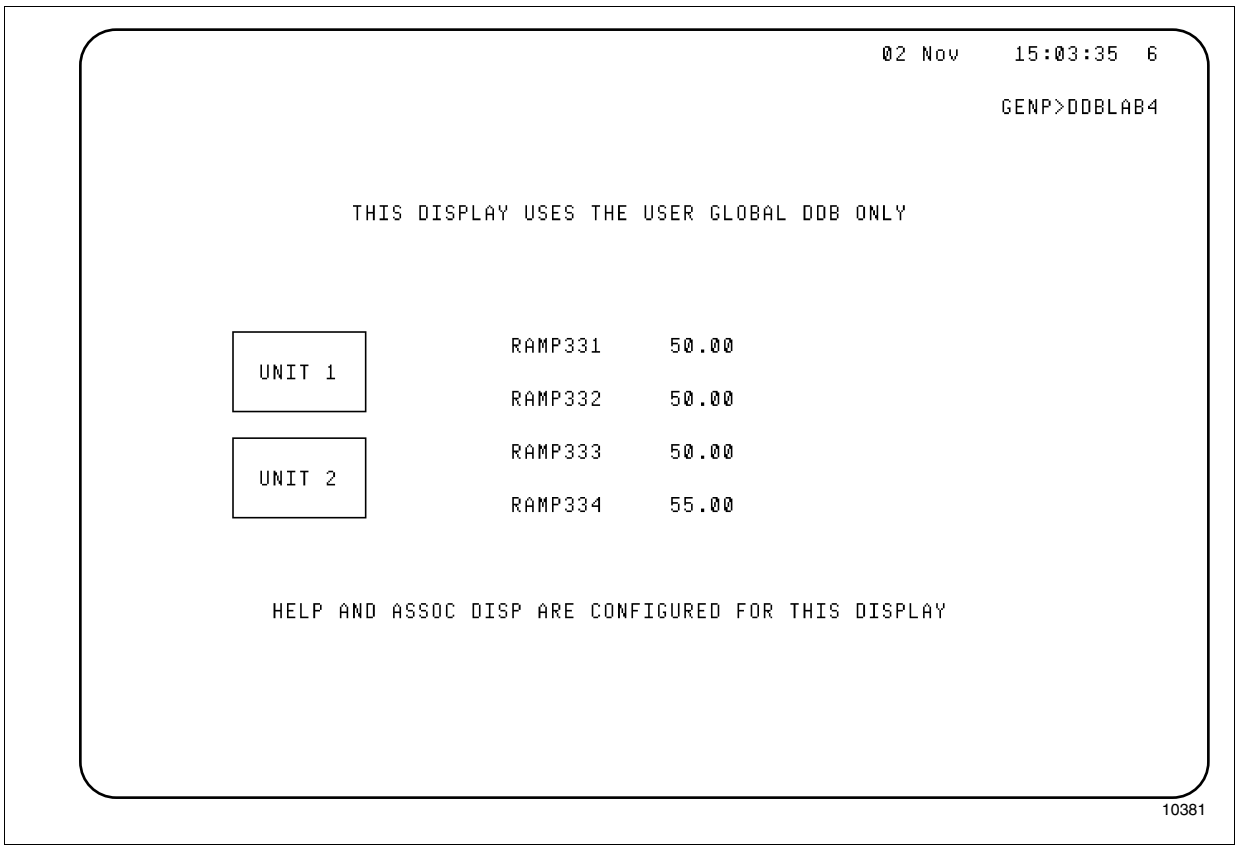


Figure 13 - Display for Lab 4—User Global DDB

2. Compile & Test

Compile and test your display.

When you are satisfied with the display, use the Picture Editor print command to print the targets and initial action. Keep the printouts to show your course manager for the end of module test.

End of Lab 4

Lab Exercise 5—Using the User Local DDB

1. Build display

Build a custom display that looks similar to Figure 14.

Requirements:

a. This display should operate the same as the display built in Lab Exercise 3, except that instead of Standard Global DDB variables, the Unit 1 and Unit 2 targets should initialize the *User Local DDB* entity variables to the point names specified in Lab Exercise 3.

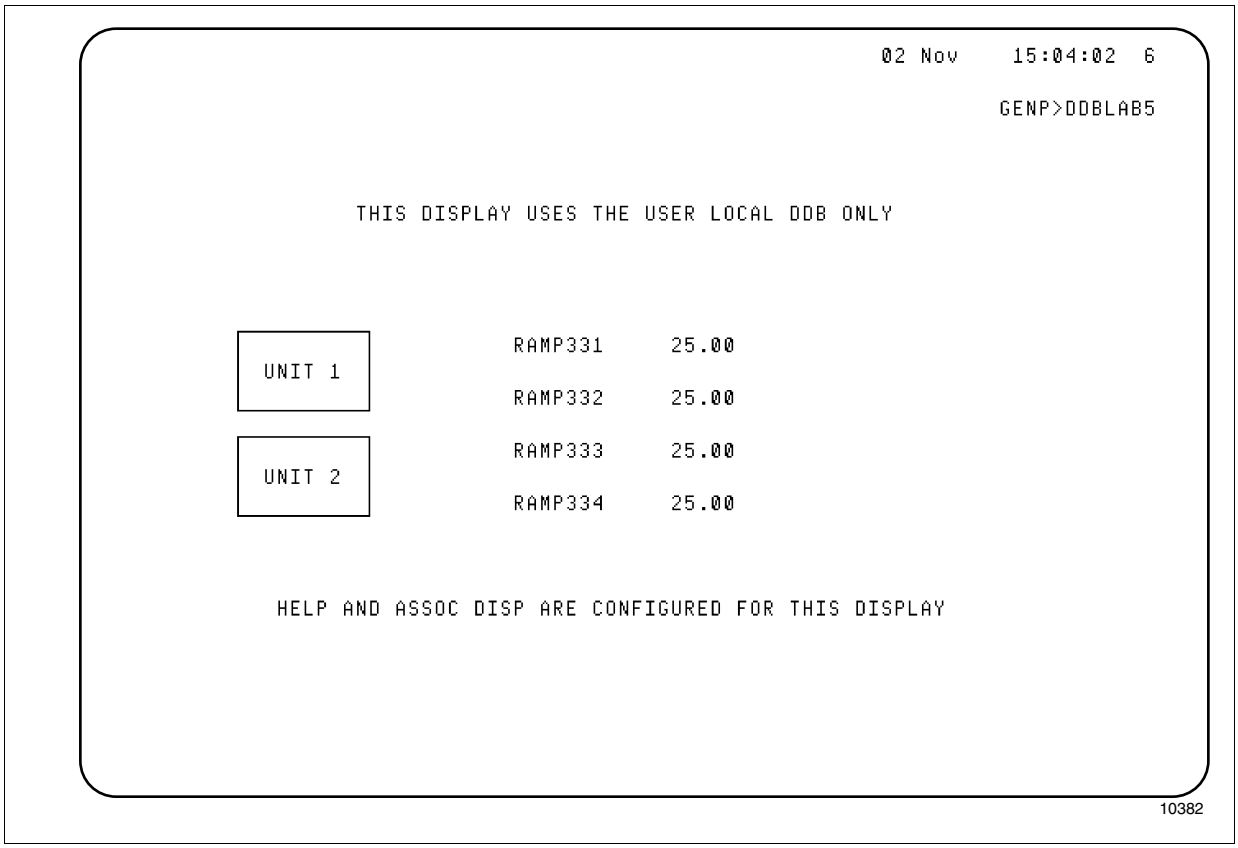


Figure 14 - Display for Lab 5—User Local DDB

2. Compile & Test

Compile and test your display.

When you are satisfied with the display, use the Picture Editor print command to print the targets, initial action, and declare file. Keep the printouts to show your course manager for the end of module test.

End of Lab 5

Lab Exercise 6 — Using the Equipment List Local DDB

1. Build display

Build a custom display that looks similar to Figure 15.

Requirements:

a. This display should operate the same as the display built in Lab Exercise 3, except that instead of Standard Global DDB variables, the Unit 1 and Unit 2 targets should initialize the *Equipment List Local DDB* entity variables to the point names specified in Lab Exercise 3.

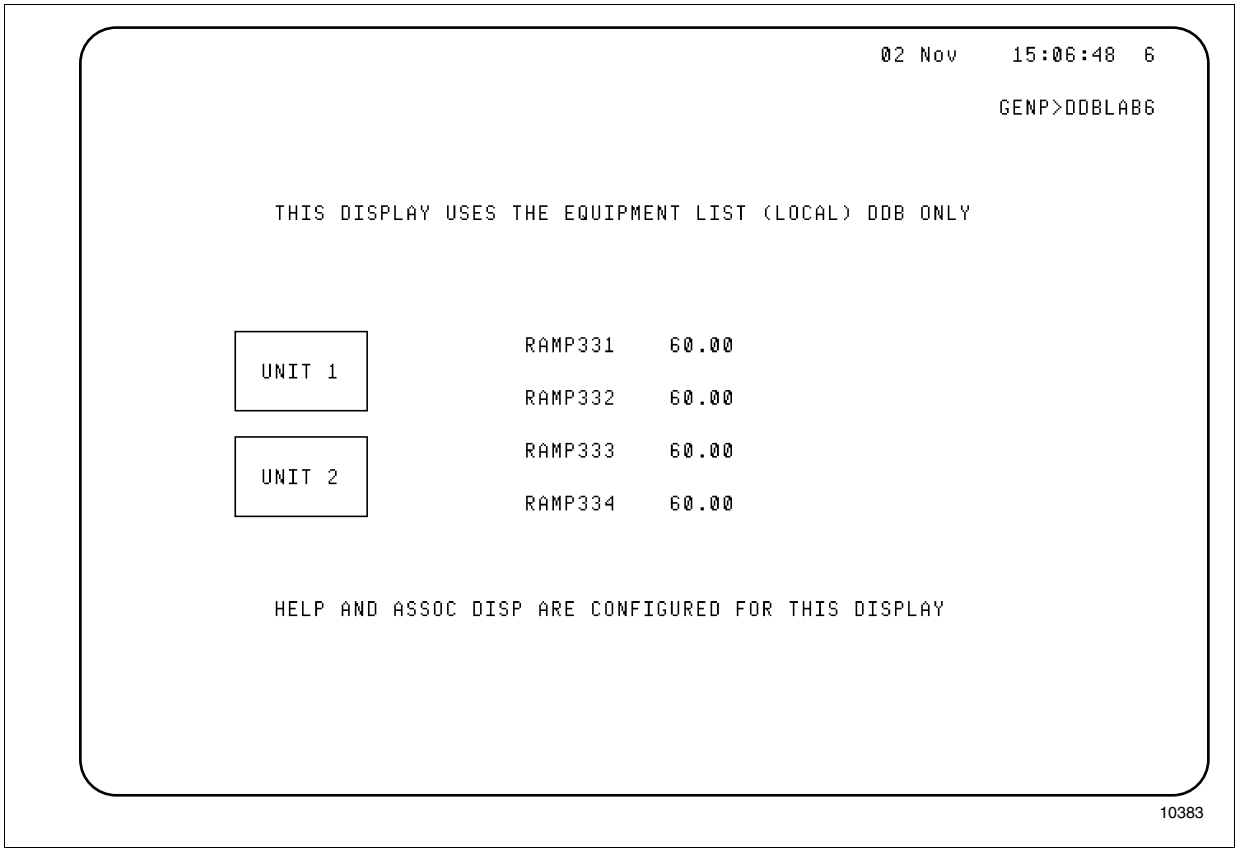


Figure 15 - Display for Lab 6—Equipment List DDB

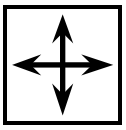
2. Compile & Test

Compile and test your display.

When you are satisfied with the display, use the Picture Editor print command to print the targets, initial action, and declare file. Keep the printouts to show your course manager for the end of module test.

End of Lab 6

Directions



DIRECTIONS—This is the end of the study material for this module. Discuss questions concerning the study material or the lab activities with a colleague or a course manager

If you are satisfied that you have achieved the objectives of this module, continue with the next section, the Student Proficiency Evaluation.

PROFICIENCY EVALUATION

Criterion Test

Instructions

In the lab exercises, you built displays that allowed viewing of different points, depending on a target selection. The display's INITIAL action caused specific points to appear upon display call up.

Successful completion of lab exercises 1 – 6 satisfies the test requirement.

To demonstrate that you successfully completed the lab exercise:

- give your course manager the printouts of the targets, initial action, and declare files.
- show your course manager that selecting a target on each of your displays switches the points that are displayed.

Self-Evaluation

INITIAL action

The INITIAL action should be similar to Figure 16.

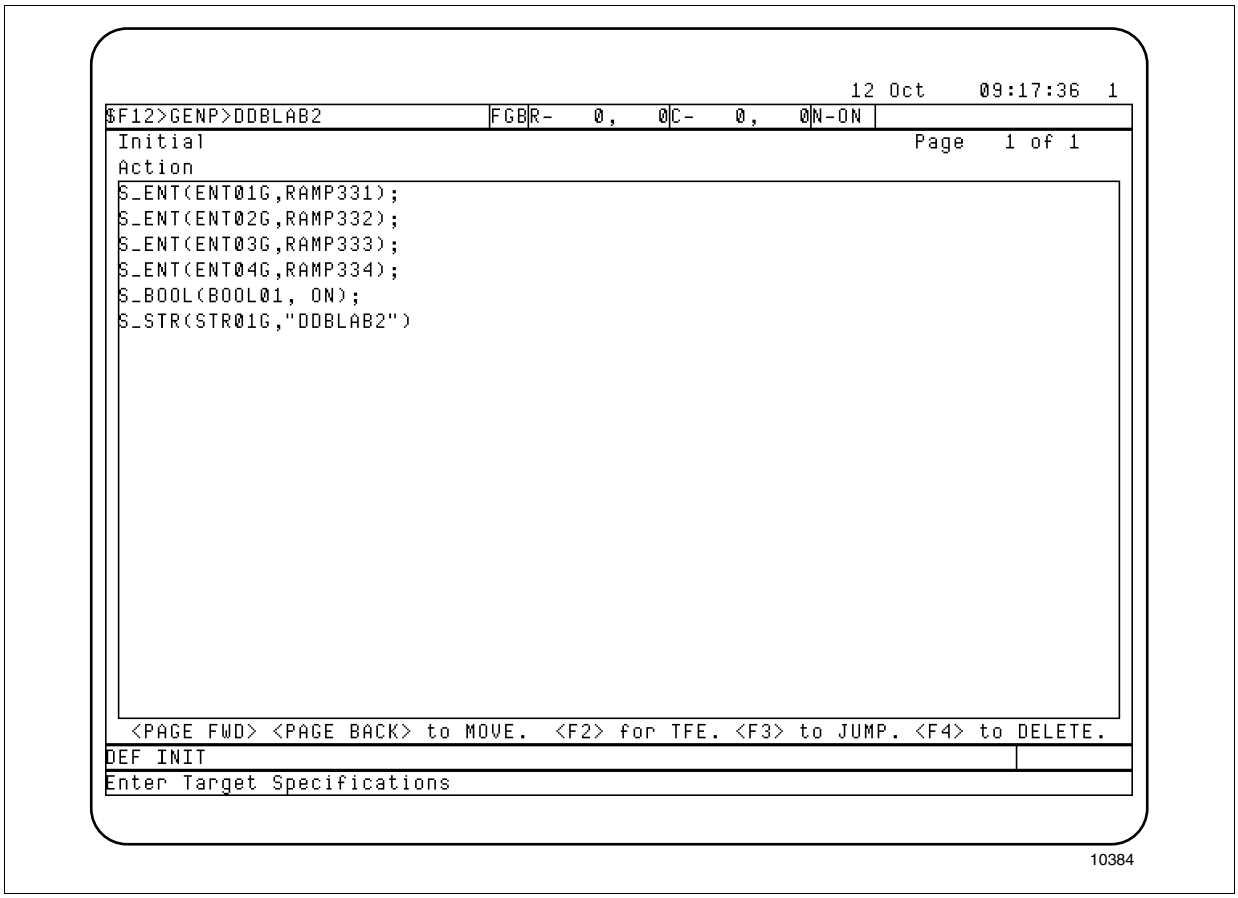


Figure 16 - INITIAL Action

Return from Help or Associated display

Figure 17 shows the configuration for help and associated displays to return to the display that calls them when you press the [HELP] or [ASSOC DISP] key again.

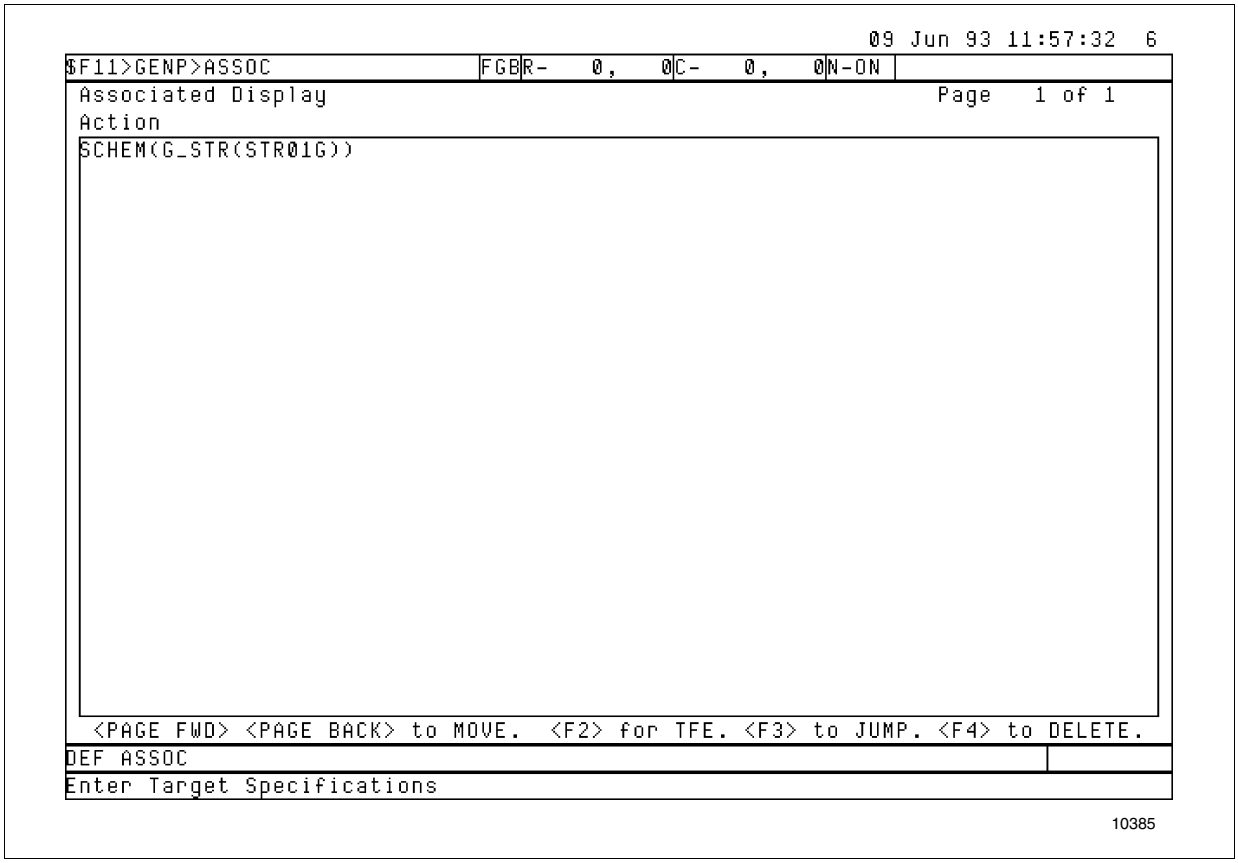


Figure 17 - Return from Help or Associated Display

Help display

Figure 18 shows the action to configure your custom display to call up another display when the [HELP] key is pressed. In this example, the other display is named HELP.

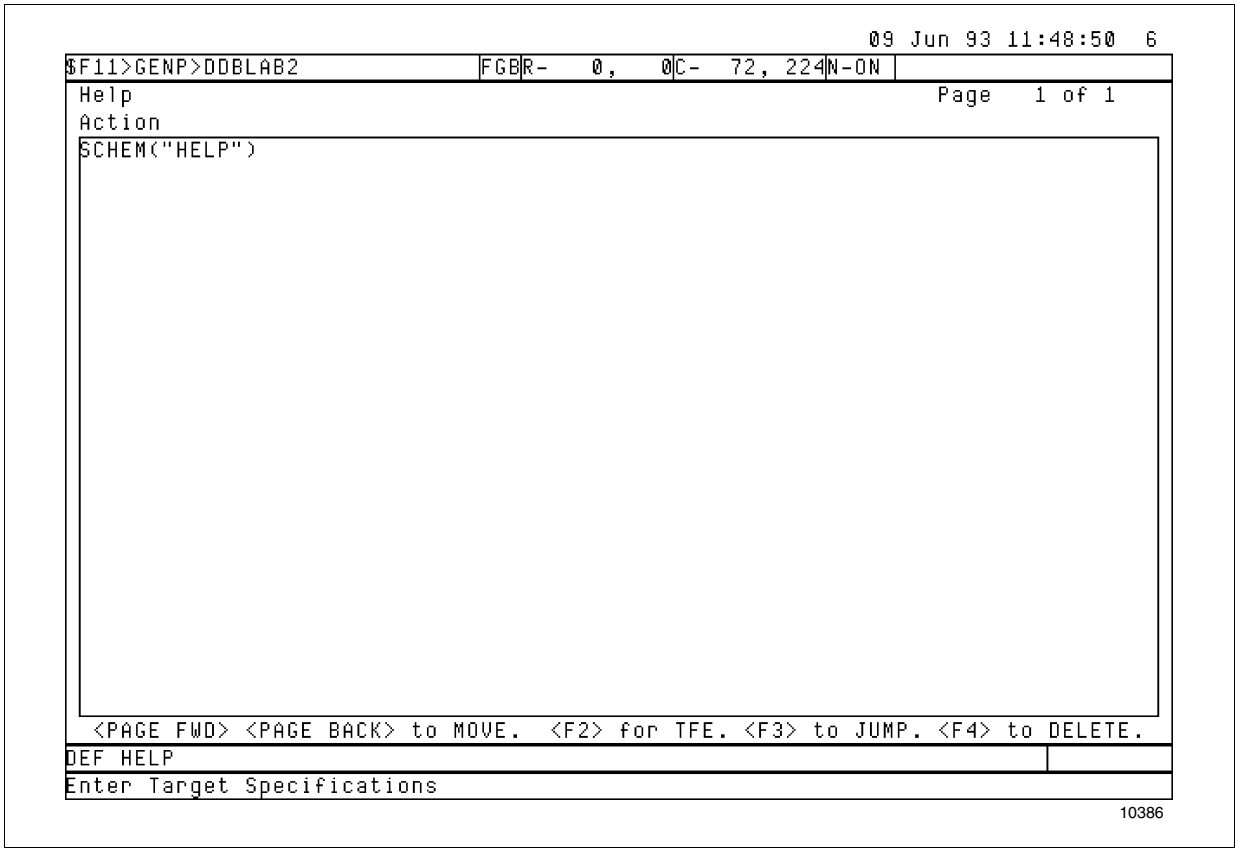


Figure 18 – Help Display

Targets

The action of your Unit 1 and Unit 2 targets should appear similar to Figure 19.

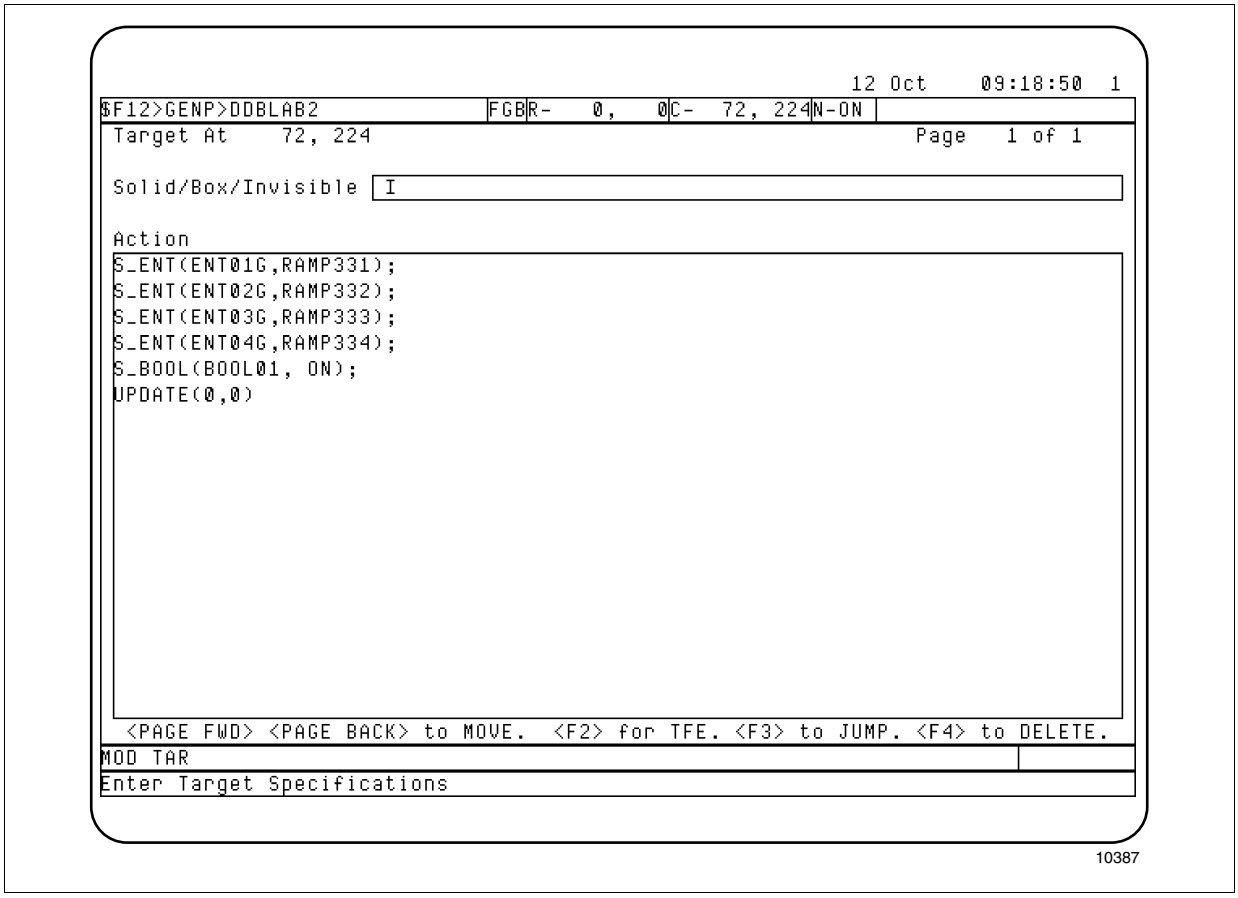


Figure 19 - Context Switching Targets - System Global DDB

User DDB

Figure 20 shows the declare files for the user DDB variables.

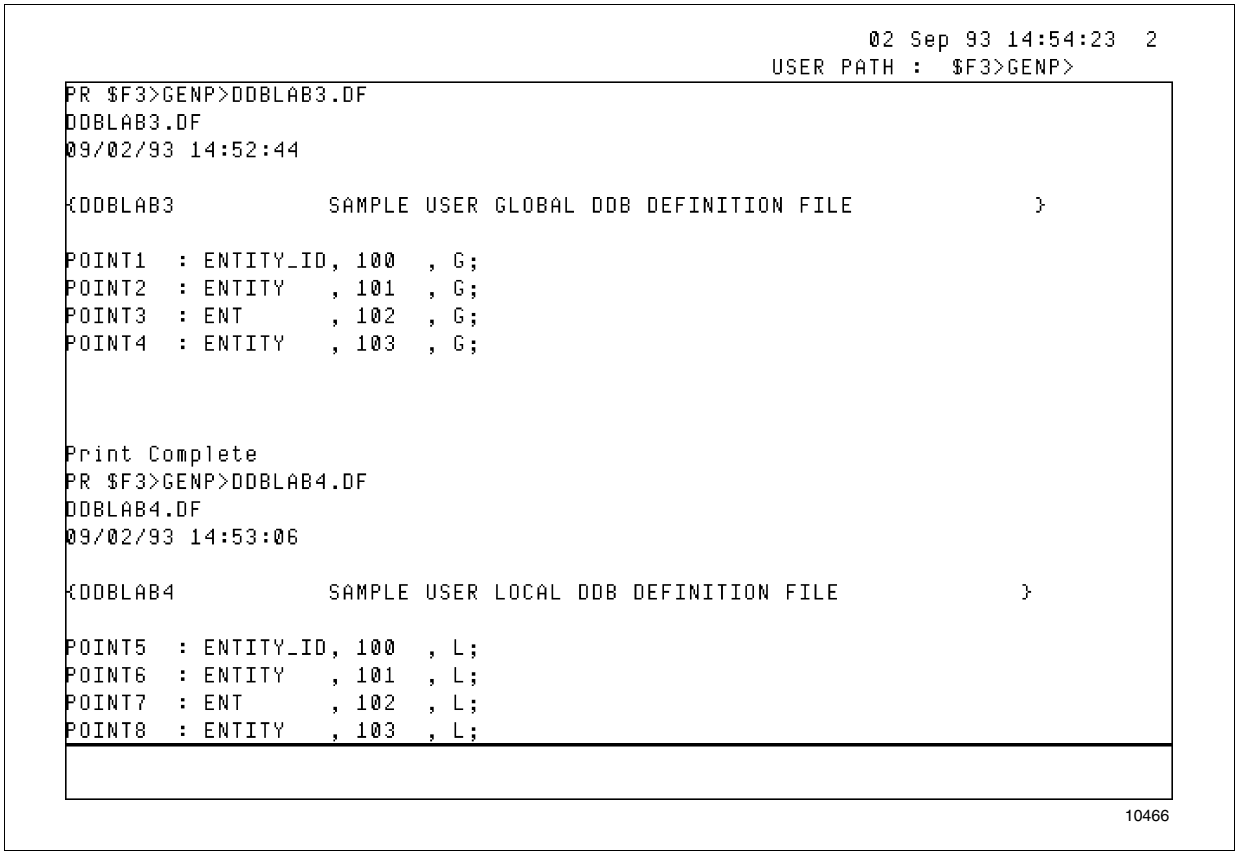
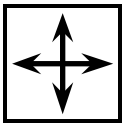


Figure 20 - User DDB Declare Files

Equipment List

Figure 10 shows the declare file for the equipment list DDB variables.

Directions



DIRECTIONS—This is the end of this module.

Use your course map to

- Get your course manager to sign off this module.
- Choose your next assigned module.

If you have a question, ask your course manager.