Global Development System Debugger Notes

June 2001

M??V??/??

1. Introduction

This document describes $DBUG, a modern, visual debugging tool for 32-bit Global applications. $DBUG is a major new component of the Global Development System.

$DBUG uses a full GUI interface, which shows the current source position of an executing program at all times. This allows the programmer to debug the source program directly, without needing to reference secondary listings and other files. The source program can be executed and trapped, resumed and advanced. To solve difficult problems, execution can proceed on a line-by-line (Single-Step) basis, giving the programmer total control of the debugging process.

Examining the contents of symbols needs only a single mouse click. Clicking on a symbol causes the symbol's assigned value to be shown in one of 4 watch windows. Any group variable (such as an I/O channel) is instantly dissected into its component fields. Equally, arrays are instantly dissected into their elemental fields. $DBUG retains a complete history of all inspected symbols within four separate watch windows, the current contents of which can be examined at all times.

Trap handling is also powerful and easy to use. **Code Traps** are set and un-set with a single click, and a special search facility makes these traps easy to maintain. **Conditional Code** traps can now be implemented using the new $BREAK statement, which allows the application program to halt execution under particular conditions.

Most importantly, a new **Value Trap** facility now allows traps to be set on Data items as well as on code lines. When a Value Trap is set, program execution halts when a given variable is modified, or attains a specified value. $DBUG then instantly displays the responsible source code line, allowing super-fast problem resolution.

$DBUG is the first Development Suite component of the NET-GSM environment. While impressive when used locally, perhaps its most outstanding feature is the ability to debug remote applications from ANYWHERE in the world. Supporting remote Server installations has never been easier, and can now be approached with confidence. All you'll need is access to the internet, and $DBUG will help you deliver the rest.

The remainder of this notice is aimed at developers of 32-bit Global applications. It summarises the operation of $DBUG, and provides installation and compatibility information.

2. Installation

$DBUG and its overlay, $DBUG1, are distributed with GSM Service Pack-5 (GSM SP-5). Both $DBUG and $DBUG1 are held in the P.$CMLB2 library.

3. Compatibility

$DBUG requires GSM or GSM-PM V8.1l Service Pack-5 to operate. If installed on an earlier version of GSM, the program will suffer fatal Loader Relocation Warnings when executed.

$DBUG must be running under Global Application Explorer (GX) V2.4, or later. While $DBUG itself will only operate on GX it can be used to debug a 32-bit program, on another user, that is not running under GX.

$DBUG only operates with 32-bit Global applications developed using $SDL32 compiler version V2.40, or later. $DBUG will not recognise the option SD source option for programs compiled with earlier versions of the compiler. Before running the debugger, we therefore recommend you ensure that all applications have been recompiled. $SDL32 V2.40 is included in the Global Development System service pack 5 (GDS-SP5).

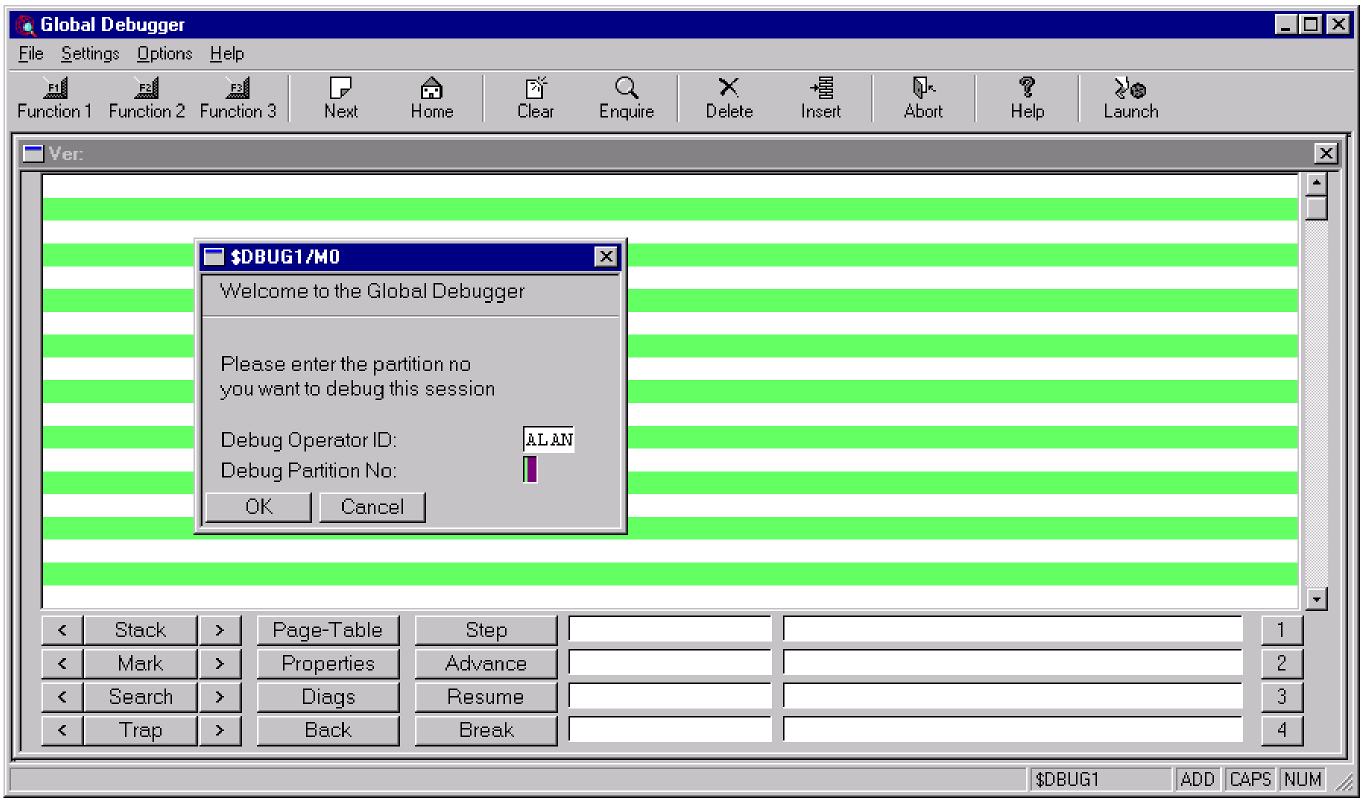
4. Running the Debugger

A Debug session runs in 2 Partitions. The debugger runs in one partition, which is known as the $DBUG partition. The Task that is being debugged runs in the other partition - the Target Partition.

The 2 partitions do not need to belong to the same user, but must both execute on the same Global client (i.e. the same instance of GLOBAL.EXE). The $DBUG partition MUST run under the control of Global Explorer, but the target partition can be connected to any supported display device, including serial screens, GUI-1, serial screen emulators or GX. The only requirement is that the target partition must be running a 32-bit program, developed using $SDL32. **$DBUG cannot be used to debug programs developed using $SDL or $COBOL.**

4.1 Starting the Debugger

$DBUG is invoked by keying $DBUG at the menu. This causes the $DBUG pre-amble window to be displayed:


Global Debugger
File Settings Options Help
Function 1 Function 2 Function 3 Next Home Clear Enquire Delete Insert Abort Help Launch
Ver:
Stack Mark Search Trap Page-Table Properties Diags Back Step Advance Resume Break 1 2 3 4
$DBUG1 ADD CAPS NUM

$DBUG1/M0

Welcome to the Global Debugger

Please enter the partition no
you want to debug this session

Debug Operator ID: ALAN
Debug Partition No:

OK Cancel

You start the debug process by keying in the operator-id and partition number of the target partition (i.e. the partition that is (or will) run the program you want to debug). Clicking OK causes $DBUG to halt execution of the target partition, which then displays the source code of the interrupted program at the current program position.

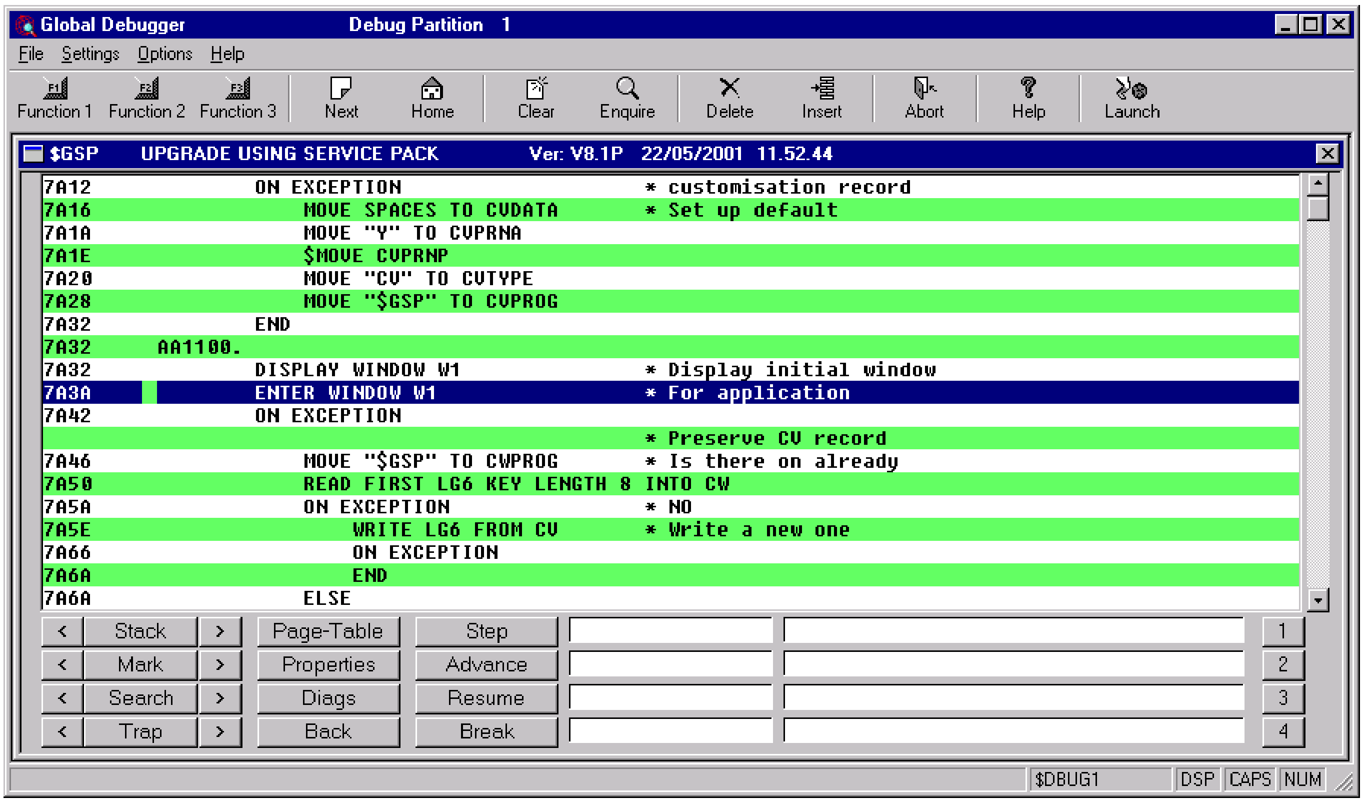
When a Program Break takes place, $DBUG always displays the source code at the Lowest (i.e. deepest) position within the call stack for which source debugging is available. If source debugging is not enabled at the lowest level, it simply displays the current source position at the preceding level in the Call-Stack (and so forth).

If none of the programs in the call stack have option SD source available, then $DBUG will not be able to display any source information. This may happen, for example, if you break into the operation of the menu handler. In this event, you should set an Overlay Trap using the DIAGS button, and resume the partition. This will cause a Break to take place when that program begins execution.

In all other circumstances $DBUG will now display the source program showing the current program position in the Source Program Window. All further debugging operations are then controlled using this window.

4.2 The Source Program Window

This window is the heart of the debugger. From here you control the operation of the target partition, set traps, resume and step the program, inspect variables and so forth.



```
$GSP    UPGRADE USING SERVICE PACK            Ver: V8.1P   22/05/2001  11.52.44
7A12                 ON EXCEPTION                      * customisation record
7A16                     MOVE SPACES TO CVDATA         * Set up default
7A1A                     MOVE "Y" TO CVPRNA
7A1E                     $MOVE CVPRNP
7A20                     MOVE "CV" TO CVTYPE
7A28                     MOVE "$GSP" TO CVPROG
7A32                 END
7A32         AA1100.
7A32                 DISPLAY WINDOW W1                 * Display initial window
7A3A                 ENTER WINDOW W1                   * For application
7A42                 ON EXCEPTION
                                                       * Preserve CV record
7A46                     MOVE "$GSP" TO CWPROG         * Is there on already
7A50                     READ FIRST LG6 KEY LENGTH 8 INTO CW
7A5A                     ON EXCEPTION                  * NO
7A5E                         WRITE LG6 FROM CV         * Write a new one
7A66                         ON EXCEPTION
7A6A                         END
7A6A                     ELSE
```

The Window heading provides you with the Name and Title of the program you are currently debugging. The source code of that program is shown in the main body of the window. When a Break or Program Exception is triggered, the source is shown at the current source position within the Call Stack, with the current line highlighted.

The Main Source Window has three banks of buttons on the left hand side. These buttons allow you to view various tables, such as the Call-Stack and the Page-Table, and allow you to control execution of the target partition.

To the right-hand side of these buttons there are 4 Watch Symbol slots within the Watch Sub-Window, in which you can view a selection of symbols from the current Watch Window. You select the Watch Window you want to view by clicking a Watch Window Buttons, which are numbered "1" "2" "3" and "4". It is convenient to use each of these for a different purpose, for example you can use Watch 1 for general symbol inspection, Watch 2 for Value Traps, Watch 3 for a particular IO channel, and so forth.

The quickest way to view a symbol is to click on its symbol name within the displayed source. $DBUG then displays both the symbol's name and value in the next available watch-window slot, which it scrolls if necessary. You can also click on any Hex address field within the source, which causes a 'manual' symbol to be created at the clicked offset. This allows you to get a quick hex view of the contents of any symbol or address within a program page.

Whenever you add a symbol, it is added to end of the list of symbols for that Watch Window. Clicking on any Symbol name in the Watch Sub-Window, causes the full Watch Window to be displayed, which allows you to amend, delete or add symbols to that Watch Window. Amongst many other functions, the full Watch Window allows you choose which 4 symbols you want to display in the Main Source window. The full Watch Window is fully described in section 6.

The size of the Watch Sub-Window is necessarily restricted. To get more details of a displayed symbol, you can click on the Symbol's Value field. If the field is an indexed field, this causes each of the array elements to be displayed in a subsidiary window. If the field is a Group Field, clicking on it's value causes the field to be exploded into its elementary items, which are again displayed in a subsidiary window. If the field is a Entry-point, Section, or Label, then clicking on it causes $DBUG to display the source code of that entry, which allows for a very quick view of a called routine.

All other $DBUG operations are controlled by one of the following buttons:

Stack	Displays the Call Stack Window
Mark	Allows you Mark the Source code for later Redisplay
Search	Allows you to search the source for text strings
Trap	Sets and Clears Code Traps
Page-Table	Displays the Page Table Window
Properties	Displays the Source Properties Window
Diags	Displays the Diagnostics Window
Back	Allows you to back-track through the source
Step	Allows you to execute the current source line
Advance	Allows you to Advance to the next source line
Resume	Resumes the execution of the program
Break	Allows you to halt execution at any time

The **Stack** button is used to display the Call-Stack Window. When clicked, the Call-Stack window is displayed, showing the current source position in terms of the call hierarchy. The **<** and **>** Buttons allow you to travel up and down the Call stack, displaying the current source position at each level. **Important note**: Any calls on programs not compiled with option SD are ignored by these buttons.

The **Mark** button is used to Mark a text line to make it easy to return to later. When marked, the source code line is displayed with a "!". You can then use the **<** and **>** buttons to reposition on the marked source code at any time. This facility works with multiple source programs, so you can use to flip from program to program.

The **Search** button is used to perform text searches within the current page. When clicked, it causes a Search Window to be displayed, which allows you to specify a search string. You can specify whether the search should be case sensitive, and whether the search needs to take place from the current position. The **<** and **>** buttons allow you to repeat the current search from the current source position going either backwards or forwards from that point.

The **Trap** button allows you to set or clear a trap on the current source code line, which must contain procedure code (and thus have an address). The **<** and **>** buttons allow you to show the source for the preceding or next trap within the current program.

The **Page-Table** button causes the Page Table Window to be displayed. This table lists all of the 32-bit pages currently loaded within the target partition, and shows the total memory used by the application. All programs that support Source Debugging are shown within the Page table with a check, and you can view the source of these by clicking on the appropriate page.

The **Properties** button causes the Program Properties window to be displayed. This window lists all of the source components (including all copy libraries and Dictionaries) that were used to compile the displayed source program. The window also displays the source file name and line number of the currently displayed source position.

The **Diags** button causes the Diagnostics Window to be displayed. This displays the current program status, with last file access information. This window can also be used to Set or Clear the Overlay Trap field. This trap allows you to cause a break whenever any instruction within the entered program name is executed. Note that this trap stays set until you reset it, so before resuming an Overlay Trapped program, you must first use this function to clear the trap.

The **Back** button is used to go back to the last viewed source positions. It simply takes you back to the last (or prior) position that was displayed. When single stepping through a program, for example, this button allows you to retrace your steps. Note that it does NOT cause the program to be executed in reverse direction, and the values of inspected variables are therefore not affected.

The remaining 4 buttons are used to control the execution of the target program. Note that execution of a program is permitted only up to the point when a very serious condition (such as a stop code) is detected. After that, any attempt to resume the program will have no effect.

The **Step** button is used to execute the target program in line-by-line mode. When the next instruction is a Call or Perform, the operation will show the next source line executed, even if that source code line is in a different program page. Thus if you use STEP to execute a CALL statement, the next source line seen will be the entrypoint of the called routine. Note that the step operation will return control only in program pages that have been compiled using option SD.

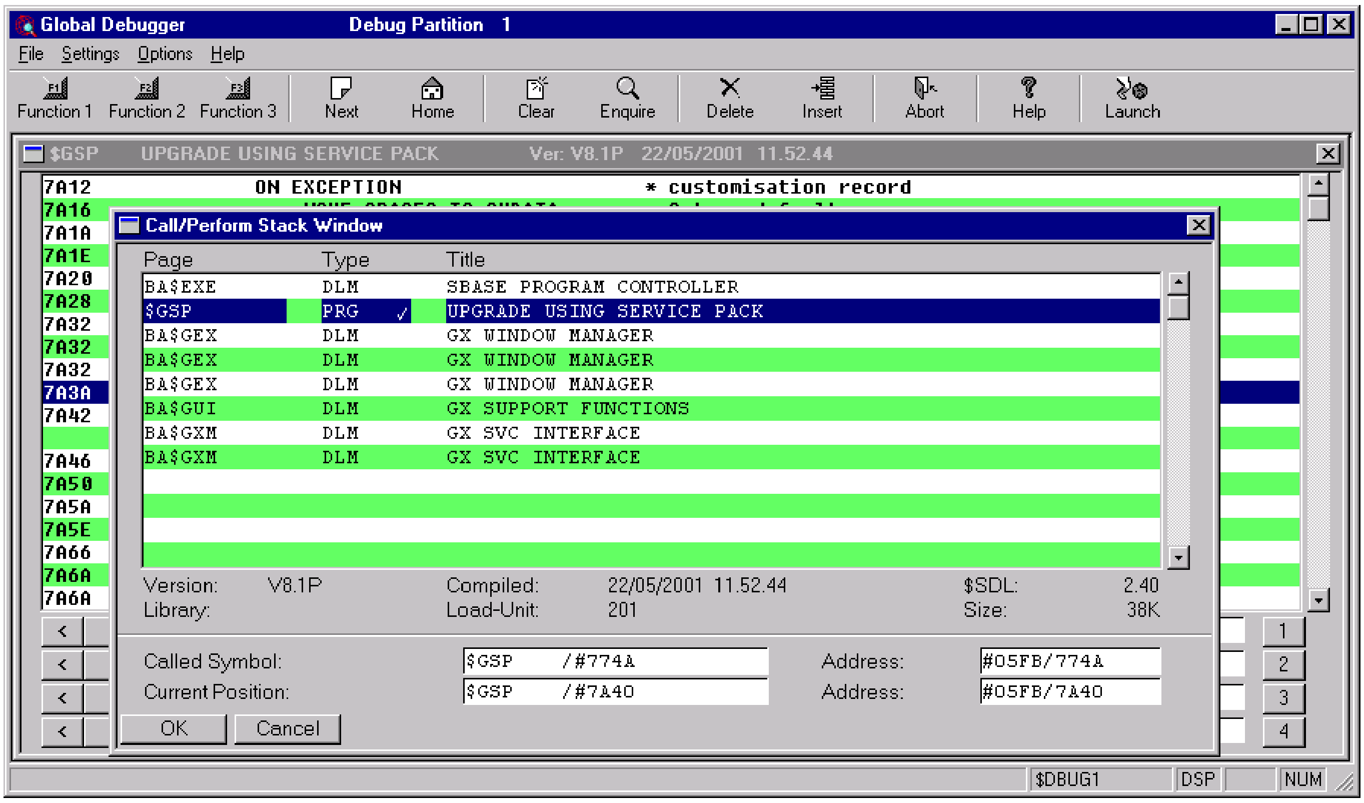
The **Advance** button is also used to execute the program in single-step mode. You use it when you are not interested in stepping through code at a lower call level. Thus if you use Advance to execute a CALL statement, the next source line seen will be the next physical line in the same program.

The **Resume** button is used to resume execution of the target partition. Execution of the program will then continue until the program hits a Procedure or Value Traps, suffers a hard exception, or until you use the Break Button to temporarily halt execution.

The **Break** button is used to halt execution of the target partition. If the target program is already halted, then the button has no effect. The button is used to break into an executing process. When clicked on, $DBUG displays the current source line of the lowest possible call level that supports Source Debugging.

4.3 The Call Stack Window

This window lists the call path details from the current Call-Stack, giving the name and title of the program at each call level, showing the Current Address as well as the originally called Address. The Program version, Compiler Version, Program Type, and Compilation Date and time are also displayed in this window.



```
$GSP    UPGRADE USING SERVICE PACK          Ver: V8.1P  22/05/2001  11.52.44
7A12             ON EXCEPTION                     * customisation record
7A16
7A1A
7A1E
7A20
7A28
7A32
7A32
7A3A
7A42
7A46
7A50
7A5A
7A5E
7A66
7A6A
7A6A
```

Page	Type	Title
BA$EXE	DLM	SBASE PROGRAM CONTROLLER
$GSP	PRG ✓	UPGRADE USING SERVICE PACK
BA$GEX	DLM	GX WINDOW MANAGER
BA$GEX	DLM	GX WINDOW MANAGER
BA$GEX	DLM	GX WINDOW MANAGER
BA$GUI	DLM	GX SUPPORT FUNCTIONS
BA$GXM	DLM	GX SVC INTERFACE
BA$GXM	DLM	GX SVC INTERFACE

Version:	V8.1P	Compiled:	22/05/2001 11.52.44	$SDL:	2.40
Library:		Load-Unit:	201	Size:	38K

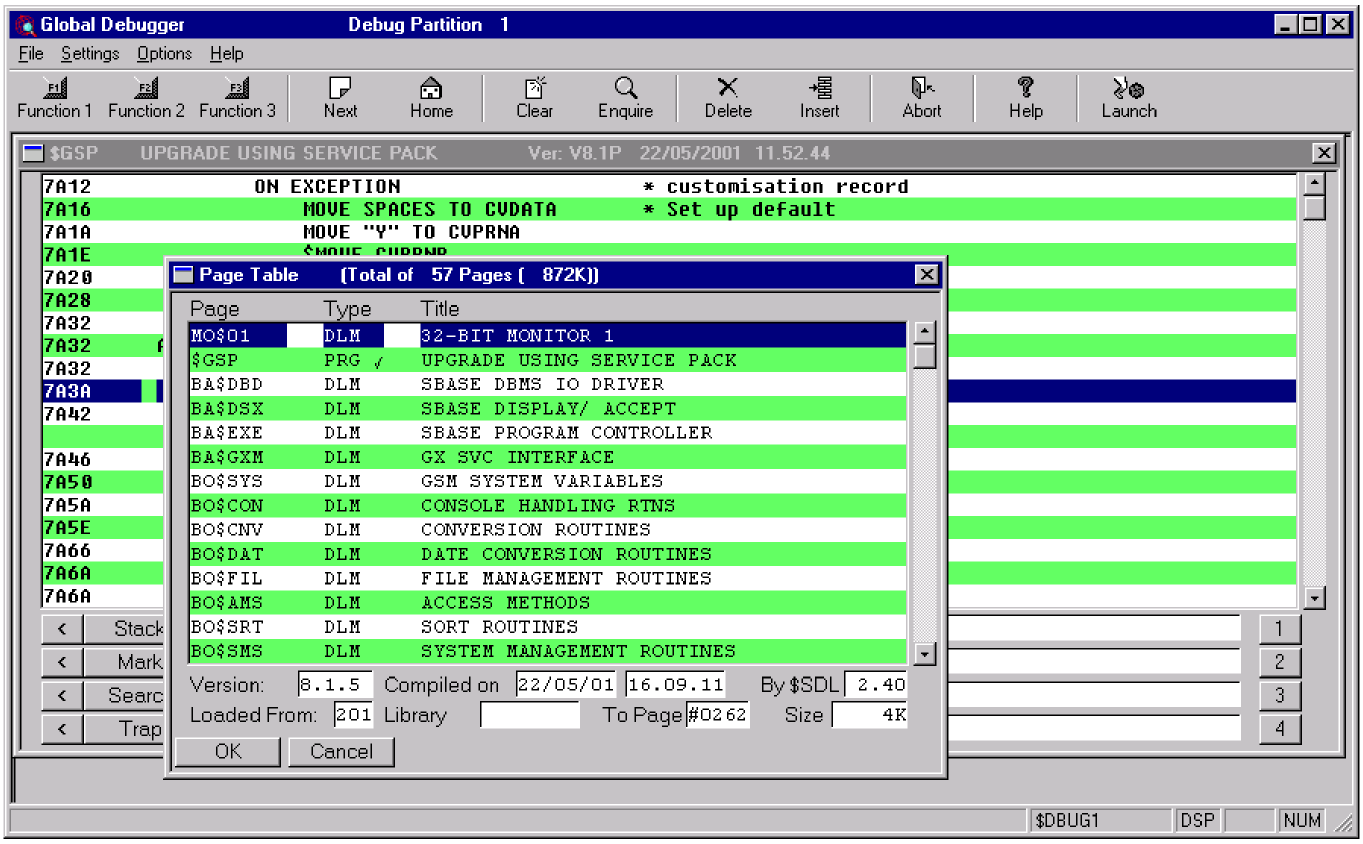
Called Symbol:	$GSP /#774A	Address:	#05FB/774A
Current Position:	$GSP /#7A40	Address:	#05FB/7A40

Each Program in the call path may be displayed with a Check next to the Program type. When Checked, this indicates that the program supports source debugging (i.e. it was compiled with compiler option SD by $SDL version 2.40, or later, **AND** the program has not subsequently had its Source Debug information removed using the $LIB NSD option).

Selecting a Checked Call level in this window causes the Source Window to be entered at the Current Position at the selected Call Level, and allows you to debug the program at that level of control. A subsequent Step or Advance operation will then bypass the display of any source code for any lower levels within the Call-Stack.

4.4 The Page-Table Window

This window displays details of all Program and Data pages currently used by the Target Partition, giving the Page-Name, Title and Type of each page. When the page is a program page, the window also shows the Program version, Compiler Version, Program Type, and Compilation Date and time.



$GSP UPGRADE USING SERVICE PACK Ver: V8.1P 22/05/2001 11.52.44

```
7A12            ON EXCEPTION                * customisation record
7A16                MOVE SPACES TO CVDATA   * Set up default
7A1A                MOVE "Y" TO CVPRNA
7A1E                $MOVE CVPRNB
7A20
7A28
7A32
7A32
7A32
7A3A
7A42

7A46
7A50
7A5A
7A5E
7A66
7A6A
7A6A
```

Page Table (Total of 57 Pages (872K))

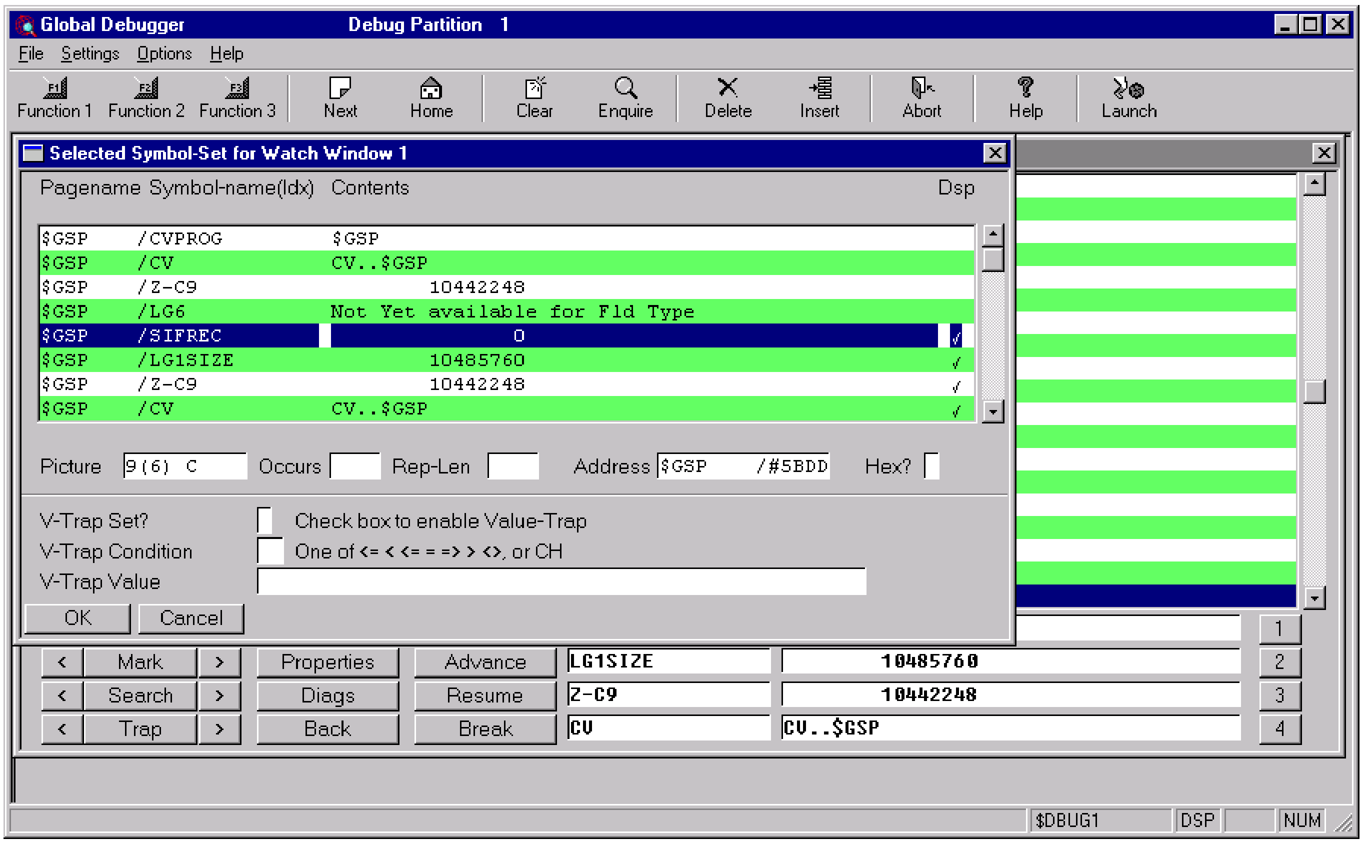
Page	Type	Title
MO$01	DLM	32-BIT MONITOR 1
$GSP	PRG ✓	UPGRADE USING SERVICE PACK
BA$DBD	DLM	SBASE DBMS IO DRIVER
BA$DSX	DLM	SBASE DISPLAY/ ACCEPT
BA$EXE	DLM	SBASE PROGRAM CONTROLLER
BA$GXM	DLM	GX SVC INTERFACE
BO$SYS	DLM	GSM SYSTEM VARIABLES
BO$CON	DLM	CONSOLE HANDLING RTNS
BO$CNV	DLM	CONVERSION ROUTINES
BO$DAT	DLM	DATE CONVERSION ROUTINES
BO$FIL	DLM	FILE MANAGEMENT ROUTINES
BO$AMS	DLM	ACCESS METHODS
BO$SRT	DLM	SORT ROUTINES
BO$SMS	DLM	SYSTEM MANAGEMENT ROUTINES

Version: 8.1.5 Compiled on 22/05/01 16.09.11 By $SDL 2.40
Loaded From: 201 Library To Page #0262 Size 4K

Program pages may be displayed with a Check next to the Program type. When Checked, this indicates that the program supports source debugging. Selecting a Checked Program in this window causes the selected source to be displayed in the Source Window, allowing you to debug that source.

4.5 The Watch Window

This window displays full details of all symbols contained within the currently selected Watch window. There are 4 watch windows symbol sets, corresponding to the buttons marked "1" through "4" on the Source Program Window. You select the symbol set you want by clicking on one of these buttons, which causes the first 4 symbols from that watch window to be displayed in the Source window. To enter Watch Window for that symbol set, you then click on any of the Symbol Names fields. The following window is then displayed:



Selected Symbol-Set for Watch Window 1

Pagename	Symbol-name(Idx)	Contents	Dsp
$GSP	/CVPROG	$GSP	
$GSP	/CV	CV..$GSP	
$GSP	/Z-C9	10442248	
$GSP	/LG6	Not Yet available for Fld Type	
$GSP	/SIFREC	0	✓
$GSP	/LG1SIZE	10485760	✓
$GSP	/Z-C9	10442248	✓
$GSP	/CV	CV..$GSP	✓

Picture 9(6) C Occurs Rep-Len Address $GSP /#5BDD Hex?

V-Trap Set? Check box to enable Value-Trap

V-Trap Condition One of <= < <= = => > <>, or CH

V-Trap Value

		1
LG1SIZE	10485760	2
Z-C9	10442248	3
CV	CV..$GSP	4

The Window displays full details for each symbol currently contained within the selected watch symbol set, giving details of the symbol with it's currently assigned contents (the Symbol's Value). The window allows you to alter, add, and delete symbols. It allows you to define which of the symbols are to be displayed in the Source Window, and allows you to set Data Value Traps. The window can be used to modify the symbol definition, and can alter the assigned contents of a given symbol.

The following information is displayed for each symbol:

Pagename	This is the name of the Program Page in which the symbol is either declared or referenced. Note that a symbol is declared in only one page, but when Global, may be referenced in many pages. While it is usual to specify the Declared Page for a symbol, $DBUG is able to resolve symbols equally well when a referencing page is specified.
Symbol-name	This is the name of the symbol you wish to inspect. You may alternatively enter a Hex address in the form #hhhh to reference a specific memory location.
(Idx)	When the symbol is indexed (i.e. it forms part of an array), you may enter a Subscript value here to specify which element you want to inspect.
Contents	This shows the current contents of the symbol. The Symbols assigned value is displayed in its expanded display form, or in Hex format. When "$Memory Violation" is displayed in this field, then the current address is invalid, usually because the target page is not currently loaded, or (if the symbol is based) because the base pointer does not point to a valid address.

DSP	This is the Display Check-Box, which allows you to specify if the symbols is to be displayed within the Source Window. The Source Window can display up to 4 symbols simultaneously, and can therefore check up to 4 symbols within the list.
Picture	This shows the symbol's Picture Clause in standard format. You can modify the picture clause by clicking on this field. If you do this, you should note that the picture clause will be changed ONLY for Display purposes, the execution of the Program will be unaffected.
Occurs	This field allows you to specify or alter the number of occurrences of an array element.
Rep-Len	This field allows you to specify or alter the repeat length of an occurring item, which is useful when a subscripted item is an elementary field within an occurring group. You will generally only need to alter this field when you create manual symbols that are indexed.
Address	This shows the dereferenced address of the symbol, as affected by any subscript and Repeat-length you may have entered. The address is shown in the Page/Offset form, and specifies the memory in which the Symbol's value assignment is stored. Clicking on this field causes $DBUG to display the relevant source program at the indicated address. You will want to use this feature, its an extremely quick way of moving between the programs procedure and it's related Data Declarations.
Hex?	This Check-box allows you to specify that the field's contents are to be displayed in Hex as opposed to display format.

The next three fields are used to set a Data Value trap on the displayed symbol.

Vtrap-Set?	Is a Check-Box which you must check in order to supply any other Value-Trap details. You may disable an existing Value-Trap by unchecking this box at any time.
Vtrap-Condn	This field allows you to specify the Condition under which a Value Trap is to take place. It may contain any of the following relational conditions: "<", "<=", "=", "=>", ">", and "<>" or "CH" Setting Value Traps is described later in this section.
Vtrap-Value	This field allows you to specify the Reference Value used to trap the field. It is entered only if the condition is relational. When the field is trapped on change ("CH") it is not required.

4.5.1 Setting Value-Traps

There are 2 Value-Trap Types: **Modification Data** Traps and **Relational Data** Traps. Both trap types cause execution of the target program to break when a Symbols contents change in a certain way.

In a **Modification Data Trap**, you tell $DBUG to Break execution of the target program whenever the specified symbol is changed. You specify this trap by Checking the Vtrap-Set? Check-box, and entering "CH" for the Condition. For a modification trap, you do not need to enter a Reference value.

From this point on, execution will Break whenever the Symbol's Contents change. If the symbol is displayed in the Main Source Window (i.e. the DSP check-box is checked), then the field's data value is shown highlighted. Otherwise, the symbol will be displayed in the appropriate Watch Window.

In a **Relational Data Trap**, you ask $DBUG to Break execution of the program when a symbols attains a reference value you have entered into the Vtrap-Value field. Thus entering:

```
"=>"    "TR05"
```

Will cause the interpreter to break execution of the program when the symbol's value is equal or greater than the value "TR05".

You should note that the trap remains set until you unset it. To resume the program following a relational Value-Trap you should therefore disable the trap first. To make this easy to do, $DBUG always displays the Watch Window highlighting the Value-Trapped Symbol when a relational Value Trap is triggered. To allow the program to be continued, you should either now turn the trap off, or change the trap condition or value so that it will not trap out on when the next instruction is executed.