# DBGP Specification

© 2001 Sony Computer Entertainment Inc.

Publication date: October 2001

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052, Japan

Sony Computer Entertainment America
919 E. Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.

The *DBGP Specification* manual is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® license agreements.

The *DBGP Specification* manual is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® license agreements.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® license agreements.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *DBGP Specification* manual is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

and PlayStation are registered trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

# Table of Contents


**About This Manual** **v**
- Changes Since Last Release v
- Related Documentation v
- Typographic Conventions v
- Developer Support v

**Overview** **1**
- Message Format 1

**Messages** **5**
- Configuration (GETCONF/GETCONFR) 5
- Register (GETREG/GETREGR/PUTREG/PUTREGR) 7
- Memory (RDMEM/RDMEMR/WRMEM/WRMEMR) 8
- Breakpoint (GETBRKP/GETBRKPTR/PUTBRKPT/PUTBRKPTR) 9
- Execution (BREAK/BREAKR/CONTINUE/CONTINUER/RUN/RUNR) 10
- XGKICK Trace (XGKTCTL/XGKTCTLR/XGKTDATAR) 12
- Debug Control (DBGCTL/DBGCTLR) 13
- GS StoreImage (RDIMG/RDIMGR) 14
- Break Function(SETBPFUNC/SETBPFUNCR) 15
- Appendix A (IOP register kind, number) 16
- Appendix B (EE register kind, number) 16

# About This Manual

This manual is the Runtime Library Release 2.4 version of the *DBGP Specification*.

It describes the system-level and thread-level debugging protocols used on the IOP and the EE (issues related to threads are not currently documented).

## Changes Since Last Release

- In the "Message Format" section of "Overview", a description of target processors specified with id has been added.

## Related Documentation

**Note:** the Developer Support Web site posts current developments regarding the Libraries and also provides notice of future documentation releases and upgrades.

## Typographic Conventions

Certain Typographic Conventions are used throughout this manual to clarify the meaning of the text:

| Convention | Meaning |
|---|---|
| `courier` | Indicates literal program code. |
| *italic* | Indicates names of arguments and structure members (in structure/function definitions only). |
| **medium bold** | Indicates data types and structure/function names (in structure/function definitions only). |
| blue | Indicates a hyperlink. |

## Developer Support

#### Sony Computer Entertainment America (SCEA)

SCEA developer support is available to licensees in North America only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

| Order Information | Developer Support |
|---|---|
| *In North America:* | *In North America:* |
| Attn: Developer Tools Coordinator<br>Sony Computer Entertainment America<br>919 East Hillsdale Blvd.<br>Foster City, CA 94404, U.S.A.<br>Tel: (650) 655-8000 | E-mail: PS2_Support@playstation.sony.com<br>Web: http://www.devnet.scea.com/<br>Developer Support Hotline: (650) 655-5566<br>(Call Monday through Friday,<br>8 a.m. to 5 p.m., PST/PDT) |

**Sony Computer Entertainment Europe (SCEE)**

SCEE developer support is available to licensees in Europe only. You may obtain developer support or additional copies of this documentation by contacting the following addresses:

| Order Information | Developer Support |
|---|---|
| *In Europe:* | *In Europe:* |
| Attn: Production Coordinator<br>Sony Computer Entertainment Europe<br>30 Golden Square<br>London W1F 9LD, U.K.<br>Tel: +44 (0) 20 7859-5000 | E-mail: ps2_support@scee.net<br>Web: https://www.ps2-pro.com/<br>Developer Support Hotline:<br>+44 (0) 20 7859-5777<br>(Call Monday through Friday,<br>9 a.m. to 6 p.m., GMT) |

# Overview

This document describes the system-level and thread-level debugging protocols used on the IOP and the EE (issues related to threads are not currently documented).

The four debugging protocols shown below are available, along with their corresponding DBGP protocol numbers.

IOP System Debugger: ISDBGP = 0x0130

IOP Thread Debugger: ITDBGP = 0x0140 (reserved for future use)

EE System Debugger: ESDBGP = 0x0230

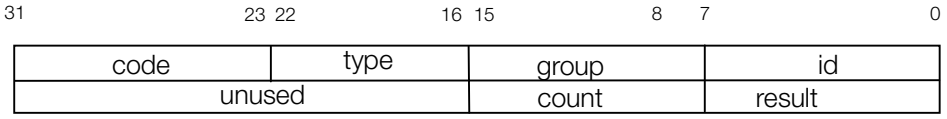
EE Thread Debugger: ETDBGP = 0x0240 (reserved but not planned for future use)

These four protocols together are referred to as DBGP.

## Message Format

In a DBGP message, the DECI2 header is always followed by a DBGP header. The data formats used after the DBGP header are dependent on the message type. The format of the DBGP header is shown below.

**Figure 1: The basic DBGP message header**




**id**

The target processors, indicating the action of the message.

For ESDBGP

| | |
|---|---|
| DBGP_CPUID_CPU | 0 // CPU (ESDBGP) |
| DBGP_CPUID_VU0 | 1 // VU0 (ESDBGP) |
| DBGP_CPUID_VU1 | 2 // VU1 (ESDBGP) |

For ISDBGP

| | |
|---|---|
| DBGP_CPUID_CPU | 0 // CPU (ISDBGP) |

**group**

The target group, indicating the action of the message.

For ESDBGP

| | |
|---|---|
| DBGP_GROUP_ENTIRE_SYSTEM | 0 |
| DBGP_GROUP_EE_THREAD | 1 |

For ISDBGP

| | |
|---|---|
| DBGP_GROUP_ENTIRE_SYSTEM | 0 |
| DBGP_GROUP_IOP_THREAD | 128 |
| DBGP_GROUP_IOP_MODULE | 129 |

The descriptions of type, code and unused below pertain to DBGP_GROUP_ENTIRE_SYSTEM. DBGP_GROUP_EE_THREAD is described in a separate document.

**type**

The message type, indicating what the message is for (Memory, Register, Execution) as well as whether the message is a request or a response.

**code**

Codes are defined by specific message types. For example, in execution commands, the code indicates Step, Next, etc. The following types and codes are available.

**Table 1: Available types and codes**

| Type/Code | Value | Description |
|---|---|---|
| DBGP_TYPE_GETCONF | 0x00 | // Get Configuration Request |
| DBGP_TYPE_GETCONFR | 0x01 | // Get Configuration Reply |
| DBGP_TYPE_GETREG | 0x04 | // Get Register Request |
| DBGP_TYPE_GETREGR | 0x05 | // Get Register Reply |
| DBGP_TYPE_PUTREG | 0x06 | // Put Register Request |
| DBGP_TYPE_PUTREGR | 0x07 | // Put Register Reply |
| DBGP_TYPE_RDMEM | 0x08 | // Read Memory Request |
| DBGP_TYPE_RDMEMR | 0x09 | // Read Memory Reply |
| DBGP_TYPE_WRMEM | 0x0a | // Write Memory Request |
| DBGP_TYPE_WRMEMR | 0x0b | // Write Memory Reply |
| DBGP_TYPE_GETBRKPT | 0x10 | // Get Breakpoint Request |
| DBGP_TYPE_GETBRKPTR | 0x11 | // Get Breakpoint Reply |
| DBGP_TYPE_PUTBRKPT | 0x12 | // Put Breakpoint Request |
| DBGP_TYPE_PUTBRKPTR | 0x13 | // Put Breakpoint Reply |
| DBGP_TYPE_BREAK | 0x14 | // Break Request |
| DBGP_TYPE_BREAKR | 0x15 | // Break Reply |
| DBGP_CODE_OTHER | 0xff | // not-DBGP_TYPE_CONTINUE |
| DBGP_TYPE_CONTINUE | 0x16 | // Continue Request |
| DBGP_CODE_CONT | 0x0 | // Continue |
| DBGP_CODE_STEP | 0x1 | // Step |
| DBGP_CODE_NEXT | 0x2 | // Next |
| DBGP_TYPE_CONTINUER | 0x17 | // Continue Reply |
| DBGP_TYPE_RUN | 0x18 | // Run Request (ESDBGP only) |
| DBGP_TYPE_RUNR | 0x19 | // Run Reply (ESDBGP only) |
| DBGP_TYPE_XGKTCTL | 0x20 | // XGKICK Trace Control Request (ESDBGP) |
| DBGP_TYPE_XGKTCTLR | 0x21 | // XGKICK Trace Control Reply (ESDBGP) |
| DBGP_TYPE_XGKTDATAR | 0x23 | // XGKICK Trace Data Reply (ESDBGP) |
| DBGP_TYPE_DBGCTL | 0x24 | // Debug Control Request (ESDBGP) |
| DBGP_TYPE_DBGCTLR | 0x25 | // Debug Control Reply (ESDBGP) |
| DBGP_TYPE_RDIMG | 0x28 | // read GS image request (ESDBGP) |
| DBGP_TYPE_RDIMGR | 0x29 | // read GS image request (ESDBGP) |
| DBGP_TYPE_SETBPFUNC | 0x2e | // set break point function request |
| DBGP_TYPE_SETBPFUNCR | 0x2f | // set break point function request |

**result**

The result codes are set as shown in the table below and are used in messages from the target to the host. Result is always set to 0 for messages sent from the host to the target.

**Table 2: Result codes**

| Result Code | Value | Description |
|---|---|---|
| Common to all groups | | |
| DBGP_RESULT_GOOD | 0x00 | // Good |
| DBGP_RESULT_INVALREQ | 0x01 | // Invalid Request |
| DBGP_RESULT_UNIMPREQ | 0x02 | // Unimplemented Request |
| DBGP_RESULT_ERROR | 0x03 | // Error |
| DBGP_ENTIRE_SYSTEM_GROUP only | | |
| DBGP_RESULT_INVALCONT | 0x04 | // Invalid Continue |
| DBGP_RESULT_TLBERR | 0x10 | // TLB mod/load/store while cmd exec |
| DBGP_RESULT_ADRERR | 0x11 | // Address Error for WRMEM/RDMEM |
| DBGP_RESULT_BUSERR | 0x12 | // Bus Error for WRMEM/RDMEM |
| DBGP_RESULT_INVALSTATE | 0x20 | // Invalid State |
| DBGP_RESULT_BREAKED | 0x21 | // Breaked |
| DBGP_RESULT_BRKPT | 0x22 | // Breakpoint |
| DBGP_RESULT_STEPNEXT | 0x23 | // Step or Next |
| DBGP_RESULT_EXCEPTION | 0x24 | // Exception |
| DBGP_RESULT_PROGEND | 0x25 | // normal end of program (EE) |
| DBGP_RESULT_BUSYSTATE | 0x26 | // busy/critical state |
| DBGP_RESULT_DEBUG_EXCEPTION | 0x27 | // Debug Exception |
| DBGP_RESULT_TIMEOUT | 0x28 | // timeout |

If the message type cannot be recognized by the target, the target sends a message of the same type as the received message but with result=DBGP_RESULT_INVALREQ. The only time the target returns a response message having the same type as the request type is when an "unrecognized type" is received.

If the target recognizes a type but determines that the request is invalid or if the type can be recognized but has not been implemented yet, a reply message having the appropriate type is returned.

**count**

A count value to be used for the operation indicated in the message. For example, if the operation is to act on a register, count would indicate the register block count. If the message is for execution, then count would indicate the number of times execution is to be performed. count is an unsigned value in the range 0-255. In some cases, count may simply be indicated as N.

Fields that are not explicitly described in this document must be set to 0 by the sender. If possible, receipt of non-zero values at the receiving end should be treated as an error.

# Messages

## Configuration (GETCONF/GETCONFR)

This message is for reading the debugging function settings of the target. The message includes the following headers and data.

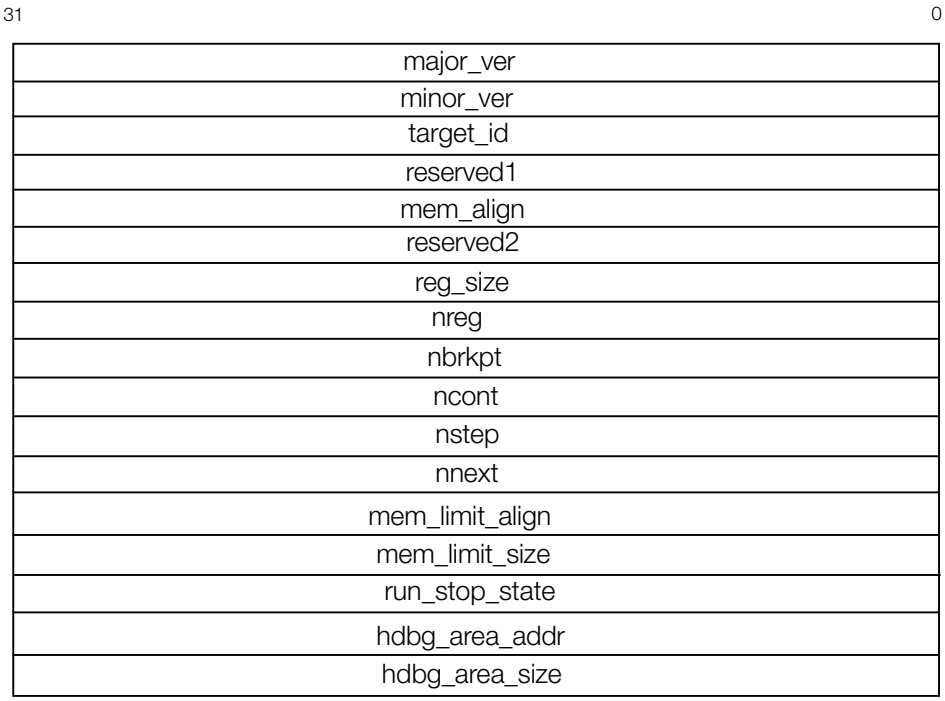
DBGP_TYPE_GETCONF: DECI2 header + DBGP header + DBGP_CONF_DATA

DBGP_TYPE_GETCONFR: DECI2 header + DBGP header + DBGP_CONF_DATA

The settings may differ depending on the version of the debugging module on the target side, so the host-side must always read the version from the settings, and issue commands that correspond to that version. Limiting the host to a minimal feature set is not recommended since this may result in a significant drop in performance, e.g., reduced data transfer speed.

Even if a host application works only with a specific major_ver, the code for checking the size of DBGP_CONF_DATA should check for values "greater than" rather than "equal to" the data size stated in this document. Otherwise, there may be compatibility problems in the event of future extensions to the specification.

**Figure 2: DBGP Configuration Data (DBGP_CONF_DATA)**




**major_ver**

The major version number for the DBGP protocol (always 3 for the current specification). This value should be referenced to determine if the format of DBGP_CONF_DATA is extended in the future.

**minor_ver**

The minor version number on the target side. In this document, the initial minor version number is defined as 0 and no other definitions are made. The application can display this value but must not perform any checking.

**target_id**

The DBGP protocol number.

**mem_align**

The memory alignments supported by the target for memory-related messages. The value is expressed as a combination together with the align value in DBGP_MEM, which is described later. For example, if the value is 0x07, then the three alignments of byte, half-word, and word are supported. At the very least, ((mem_align & 1) == 1) will be true.

**reserved2**

Always set to 1 to maintain compatibility.

**reg_size**

Set to register data size for messages relating to registers. Set to 5 for the IOP and 7 for the EE.

```
DBGP_CF_REG_SIZE_WORD        5   // (1 << 5) =  32bit (IOP)
DBGP_CF_REG_SIZE_QUAD        7   // (1 << 7) = 128bit (EE)
```

**nreg**

The maximum register block count in register-related messages. The range is 1-255.

**nbrkpt**

The maximum address for breakpoint-related messages. The range is 1-255.

**ncont**

The maximum number of executions for continued execution. Always set to 1.

**nstep**

The maximum number of instructions to step for single-step execution. The range is 0-255.

**nnext**

The maximum number of instructions to execute for next execution. The range is 0-255.

**mem_limit_align**
**mem_limit_size**

For memory-related messages, the packet size may be restricted by align.

mem_limit_align is used to specify alignment combinations for messages which have size restrictions (see mem_align, above, for a description of the method used to specify alignment combinations).

mem_limit_size is used to specify the maximum number of bytes in DECI2 packets for messages which have size restrictions, including DECI2, DBGP, and DBGP_MEM headers.

For example, if mem_limit_align=0x1f and mem_limit_size=0x400, the maximum packet size for memory-related messages whose alignment is 16 bytes or less is 1 KBytes. The maximum size for other alignments is the DECI2 packet maximum: (64K - 1) bytes.

**run_stop_state**

The run, stop state of the target. run_stop_state is used by the host to determine the initial state of the target. No matter what state the target is in, the "current state" is set in this field for DBGP_TYPE_GETCONFR. However, once DBGP_TYPE_GETCONF has been issued, the host should determine the target's state using sent and received packets.

```
DBGP_CF_RSS_RUNNING        1  // running
```

DBGP_CF_RSS_STOPPED 2 // stopped

**hdbg_area_addr**
**hdbg_area_size**

Memory area on the target that can be freely used by the host-side debugger. hdbg_area_addr is a word-aligned address that can be specified as a virtual address in a memory-related message.

When the CPU is in kernel mode, this address can be used for instruction fetches and data accesses. hdbg_area_size is a byte count with a minimum non-zero value of 256 bytes.

The purpose of this memory area is to provide a workspace that allows high-speed execution of pattern fills and pattern searches from the host, by executing very small modules that are sent to the target.

## Register (GETREG/GETREGR/PUTREG/PUTREGR)

Messages for reading and setting up target registers and associated reply messages. The messages use the following headers and data.

| | |
|---|---|
| DBGP_TYPE_GETREG: | DECI2 header + DBGP header + (DBGP_REG + <reg>) * N |
| DBGP_TYPE_GETREGR: | DECI2 header + DBGP header + (DBGP_REG + <reg>) * N |
| DBGP_TYPE_PUTREG: | DECI2 header + DBGP header + (DBGP_REG + <reg>) * N |
| DBGP_TYPE_PUTREGR: | DECI2 header + DBGP header + (DBGP_REG + <reg>) * N |

<reg> is register data that follows the DBGP_REG header. The size is not the size of the register, but a fixed size determined for each target using reg_size of DBGP_CONF_DATA. This value is always 32 bits for the IOP and 128 bits for the EE.

If the actual number of bits in the register is less than reg_size, valid data must be placed in the lower-order bits. The invalid section should be set to 0, but this is not required.

When accessing the VU0, VU1 registers, id in the DBGP header must indicate the corresponding processor ID. Thus, a single message cannot be used to simultaneously access EE registers and VU0 registers.

In DBGP_TYPE_GETREG messages, the <reg> value is not used by the target, but instead, dummy data is sent to eliminate the need to allocate an area on the target. The <reg> value sent by the host should be set to 0, but this is not required.

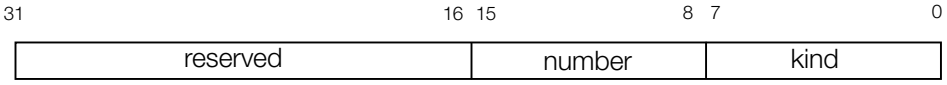
In DBGP_TYPE_PUTREG messages, the target loads the specified register with the value in <reg>, then sends back the DBGP_TYPE_PUTREGR message with the <reg> value intact.

If a read request for a non-existent, undefined register is received, the target returns a 0 and simply ignores any write requests. In neither case is an error returned.

The maximum number of registers that can be operated on in a single DBGP_TYPE_GETREG/DBGP_TYPE_PUTREG message is defined by nreg and depends on the target-side implementation. nreg is guaranteed to be 1 or greater.

The DBGP_REG header is as shown below.

**Figure 3: DBGP Register Header (DBGP_REG)**




**kind**

The register type code. Registers having identical names in the EE and the IOP will have different codes.

**number**

The register number.

The specific values for kind and number are described in the appendices to this document (Appendix A, B).

## Memory (RDMEM/RDMEMR/WRMEM/WRMEMR)

Messages for reading and writing from and to the target memory and associated reply messages. The messages use the headers and data shown below:

| | |
|---|---|
| DBGP_TYPE_RDMEM: | DECI2 header + DBGP header + DBGP_MEM |
| DBGP_TYPE_RDMEMR: | DECI2 header + DBGP header + DBGP_MEM + <pad> + <data> |
| DBGP_TYPE_WRMEM: | DECI2 header + DBGP header + DBGP_MEM + <pad> + <data> |
| DBGP_TYPE_WRMEMR: | DECI2 header + DBGP header + DBGP_MEM |

<data> is the actual memory data to be written out by these messages. <data> must be aligned from the start of the packet based on the align field specified in the DBGP_MEM header.

<pad> is padding data used for alignment.

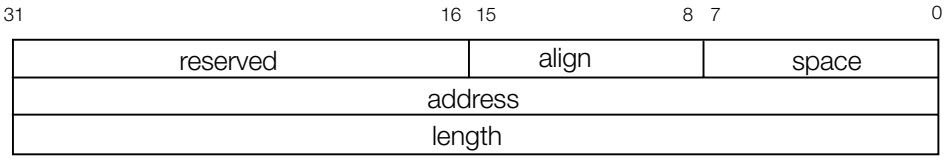
Since DECI2, DBGP, and DBGP_MEM headers take up a total of 28 bytes, <pad> will have a size of 0 if align is BYTE, HALF, or WORD. If align is 1KB, the size of <pad> will be 1024-28.

If the target returns an error for a DBGP_TYPE_RDMEMR message, the <pad> + <data> section may be omitted.

The data size that can be read or written in a single operation will be either the size determined by mem_limit_align and mem_limit_size of DBGP_CONF_DATA or the maximum for a DECI2 packet (64K - 1) (see the corresponding section on DBGP_CONF_DATA).

The count field in the DBGP header must always be set to 1.

**Figure 4: DBGP Memory Header (DBGP_MEM)**




**space**

The memory address space code. The following values are available.

| | | |
|---|---|---|
| DBGP_SPACE_MAIN_MEMORY | 0 | // main memory |
| DBGP_SPACE_VU_MEM | 1 | // VU memory(ESDBGP) |
| DBGP_SPACE_VU_UMEM | 2 | // VU micro memory(ESDBGP) |

If DBGP_SPACE_VU_MEM or DBGP_SPACE_VU_UMEM is specified, the processor ID for the read or write (in this case, DBGP_CPUID_VU0 or DBGP_CPUID_VU1) must be specified in the CPU ID field of the DBGP header.

Reads and writes to main memory can be performed at any time, but for DBGP_SPACE_VU_MEM or DBGP_SPACE_VU_UMEM, the target-side debugger determines whether the read or write operation is allowed when the packet is received. If a reads or writes are not allowed, a DBGP_RESULT_BUSYSTATE is returned.

**align**

An alignment value specifying how the target should access the memory space. Byte=0, half-word=1, word=2, double=3, quad-word=4, 1KB=10.

If the corresponding bit in DBGP_CONF_DATA is off, a DBGP_RESULT_INVALREQ error is returned.

If DBGP_SPACE_VU_MEM is specified for space, the field must be set to 4. If DBGP_SPACE_VU_UMEM is specified, the field must be set to 3.

**address**

The memory address, in bytes. If the address is not aligned according to the align field, a DBGP_RESULT_INVALREQ error is returned.

**length**

The memory data size, in bytes. If the size is not aligned according to align, a DBGP_RESULT_INVALREQ error is returned.

## Breakpoint (GETBRKP/GETBRKPTR/PUTBRKPT/PUTBRKPTR)

Messages for setting up breakpoints and reading the current settings, as well as associated reply messages. The messages use the headers and data shown below.

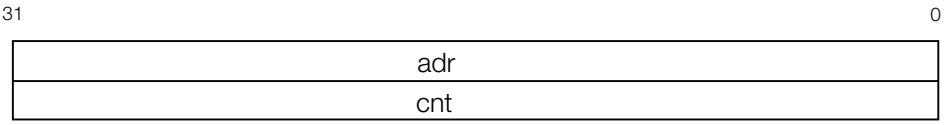
DBGP_TYPE_GETBRKPT: DECI2 header + DBGP header

DBGP_TYPE_GETBRKPTR: DECI2 header + DBGP header + (<adr> + <cnt>) * N

DBGP_TYPE_PUTBRKPT: DECI2 header + DBGP header + (<adr> + <cnt>) * N

DBGP_TYPE_PUTBRKPTR: DECI2 header + DBGP header

**Figure 5: DBGP Breakpoint Data**




<adr> is a breakpoint address.

<cnt> is the number of times <adr> is to be passed. An initial value is entered for DBGP_TYPE_PUTBRKPT messages. For DBGP_TYPE_GETBRKPTR messages, the target calculates and returns the number of times the breakpoint was passed.

<cnt> is an unsigned 32-bit value and is decremented each time <adr> is passed. When <cnt> changes from 1 to 0, the target is halted and a DBGP_TYPE_BREAKR (DBGP_RESULT_BRKPT) message is returned.

If <cnt> is set to the special value of 0xffffffff, the target is halted each time <adr> is passed without decrementing the pass count, and a DBGP_TYPE_BREAKR (DBGP_RESULT_BRKPT) message is returned.

If the current value of <cnt> for a breakpoint is 0, target assumes that the breakpoint is in an inactive state.

The maximum number of breakpoints that can be set is the value of nbrkpt in DBGP_CONF_DATA. If the count value (=N) in the DBGP header of the DBGP_TYPE_PUTBRKPT exceeds nbrkpt, the target returns a DBGP_RESULT_INVALREQ error.

The DBGP_TYPE_PUTBRKPT message cannot be used to make partial additions/removals of breakpoints. The settings in the message will update all settings. To delete all breakpoints, N must be set to 0.

The target will not return partial success/error indications in response to settings requested by DBGP_TYPE_PUTBRKPT. The results will be either completely successful or an error will be returned.

For the DBGP_TYPE_GETBRKPTR message, the count value (=N) in the DBGP header returned by the target is the actual setting.

For both the DBGP_TYPE_GETBRKPT and the DBGP_TYPE_PUTBRKPTR messages, countmust be set to 0 in the DBGP header.

## Execution (BREAK/BREAKR/CONTINUE/CONTINUER/RUN/RUNR)

Messages to start and stop target program execution as well as associated reply messages. The messages use the headers and data shown below.

**Table 3: Execution messages**

| Header | Data |
|---|---|
| DBGP_TYPE_BREAK: | DECI2 header + DBGP header |
| DBGP_TYPE_BREAKR: | DECI2 header + DBGP header |
| DBGP_TYPE_CONTINUE: | DECI2 header + DBGP header |
| DBGP_TYPE_CONTINUER: | DECI2 header + DBGP header |
| DBGP_TYPE_RUN: | DECI2 header + DBGP header + DBGP_EERUN + <args>... |
| DBGP_TYPE_RUNR: | DECI2 header + DBGP header |

DBGP_TYPE_BREAK is a message requesting a break in target program execution. If running, the target returns DBGP_TYPE_BREAKR (DBGP_RESULT_BREAKED). If already stopped, the target returns DBGP_TYPE_BREAKR (DBGP_RESULT_INVALSTATE).

Depending on the code field, the DBGP_TYPE_CONTINUE message can be a continue, step, or next request. When a DBGP_TYPE_CONTINUE is received, a DBGP_TYPE_CONTINUER is returned if there is no error and the operation described below is begun. If an error occurs, the error is indicated using DBGP_TYPE_CONTINUER. The target will not start running and will stay in a stopped state.

- For DBGP_CODE_CONT, continued execution. N is always 1.
- For DBGP_CODE_STEP, stepped execution of instructions. N is the number of steps (number of instructions).
- For DBGP_CODE_NEXT, run next instruction. N is the next count (number of instructions).

The target returns DBGP_TYPE_BREAKR (DBGP_RESULT_STEPNEXT) once the step/next execution completes without exception errors, etc.

With a DBGP_TYPE_BREAKR message, the value of result that is returned will not be DBGP_RESULT_GOOD. When the DBGP_TYPE_BREAKR message is returned, the target program will always be halted and will be waiting for a command from the host.

If a DBGP_TYPE_BREAKR message is sent in response to a DBGP_TYPE_CONTINUE message, the count value will be the set to the count value in the specified DBGP_TYPE_CONTINUE minus the actual number of operations completed.

At the same time, the code field will be set to the same code value as in the DBGP_TYPE_CONTINUE message. When sending a DBGP_TYPE_BREAKR message such as for a break right after a reset, the code field is set to DBGP_CODE_OTHER to indicate that "execution was not started by DBGP_TYPE_CONTINUE".

If completion of step/next execution (after execution of the step/next instruction count) occurs at the same time as a breakpoint condition (after the specified number of passes has taken place), the target will return

a DBGP_TYPE_BREAKR (DBGP_RESULT_BRKPT) message and the code field will be set to DBGP_CODE_STEP/DBGP_CODE_NEXT.

The following examples show the DBGP_TYPE_BREAKR messages returned by the target.

Example 1:
When count=3 is specified by DBGP_TYPE_CONTINUE, DBGP_CODE_STEP, and a bus error is generated during stepped execution of the first instruction:

-> DBGP_TYPE_BREAKR, DBGP_RESULT_EXCEPTION, DBGP_CODE_STEP, count=3

Example 2:
When count=8 is specified by DBGP_TYPE_CONTINUE, DBGP_CODE_STEP and a breakpoint (<cnt>=1) is encountered after stepped execution of 1 instruction:

-> DBGP_TYPE_BREAKR, DBGP_RESULT_BRKPT, DBGP_CODE_STEP, count=7

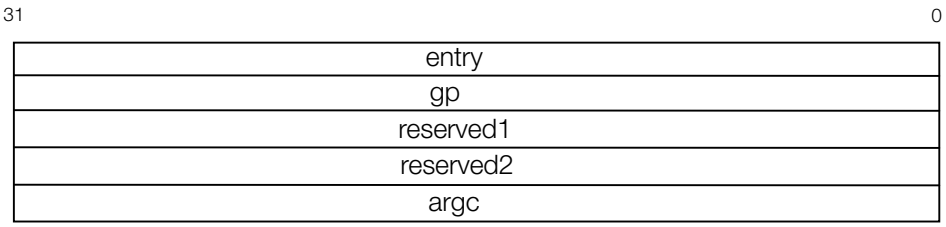
Example 3:
When count=2 is specified by DBGP_TYPE_CONTINUE, DBGP_CODE_NEXT and a breakpoint (<cnt>=1) is encountered after next execution of two instructions:

-> DBGP_TYPE_BREAKR, DBGP_RESULT_BRKPT, DBGP_CODE_NEXT, count=0

If execution is interrupted by a D-bit interrupt from VU0 or VU1, the target returns a DBGP_TYPE_BREAKR (DBGP_RESULT_EXCEPTION) with CPUID in the DBGP header set to DBGP_CPUID_VU0 or DBGP_CPUID_VU1.

DBGP_TYPE_RUN is a message requesting execution of a target program on the EE. DBGP_TYPE_RUN uses the following header.

**Figure 6 : DBGP EE Run Header (DBGP_EERUN)**




**entry**

The starting address of the program.

**gp**

The GP register value.

**argc**

The number of arguments.

The argument data following the DBGP_EERUN header is in the following format:

array indicates the length of each argument (4 byte * argc)
(length includes terminating nulls of each argument)

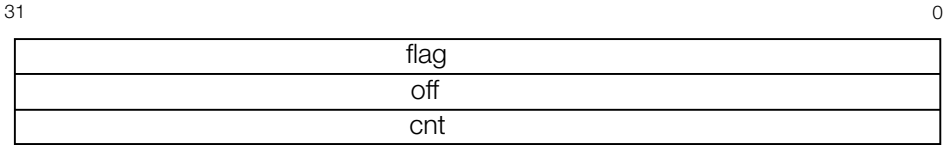
null-terminated arguments (variable number of bytes * argc)

## XGKICK Trace (XGKTCTL/XGKTCTLR/XGKTDATAR)

Messages for executing and suspending the tracing of GIF packet data when an XGKICK command is executed on VU1, and associated reply messages. The messages use the headers and data shown below.

DBGP_TYPE_XGKTCTL: DECI2 header + DBGP header + DBGP_XGKT_CTL
DBGP_TYPE_XGKTCTLR: DECI2 header + DBGP header + DBGP_XGKT_CTL
DBGP_TYPE_XGKTDATAR: DECI2 header + DBGP header + DBGP_XGKT_DATA

**Figure 7: DBGP XGKICK Trace Control (DBGP_XGKT_CTL)**




The DBGP_TYPE_XGKTCTL message requests the start of an XGKICK trace. GIF packet data is traced by tracing <cnt> XGKICK instructions from the specified offset <off>. (If <off>=0, then tracing begins at the next XGKICK instruction. If <off>=1, tracing begins at the second XGKICK instruction).

When performing XGKICK tracing, the D-bit for the upper instruction in the XGKICK instruction must be set beforehand. Also, the id fields in all DBGP headers of XGKICK trace-related packets must be set to DBGP_CPUID_VU1.

### flag

Indicates various options and is the result of ORing the following values.

```
DBGX_VBS0       0x00000001    // VU0  busy
//DBGX_VIF0     0x00000010    // VIF0 busy   (reserved)
//DBGX_VBS1     0x00000100    // VU1  busy   (reserved)
DBGX_VIF1       0x00001000    // VIF1 busy
//DBGX_GIF      0x00010000    // GIF  busy   (reserved)
```

If the specified bit(s) of <flag> are OFF, the target will wait indefinitely until the corresponding condition(s) are no longer valid.

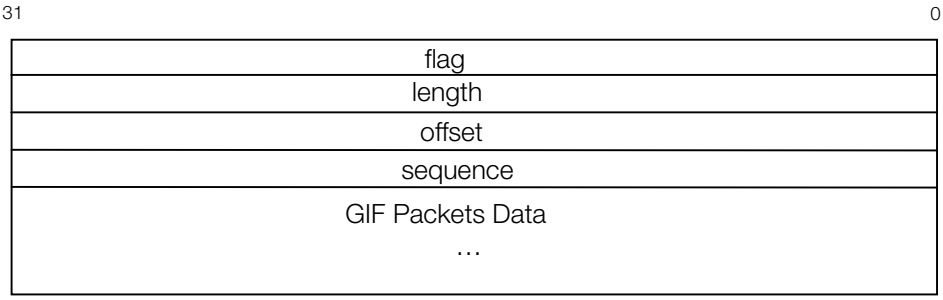
If the specified bit(s) of <flag> are ON and the corresponding condition(s) are valid, the trace operation will not be performed. Instead, the reason the operation was not performed will be indicated by setting the corresponding flag bits, and a DBG_TYPE_XGKTDATAR message with <length>=0 will be sent to the host.

The DBGP_TYPE_XGKTCTLR message is a reply to a start XGKICK trace request.

<flag>, <off>, <cnt> return the same values as DBGP_TYPE_XGKTCTL.

The DBGP_TYPE_XGKTDATAR message contains trace data sent from the target to the host.

**Figure 8: DBGP XGKICK Trace Data (DBGP_XGKT_DATA)**




**flag**

If any of the <flag> bits in DBGP_TYPE_XGKTCTL is ON and the corresponding condition is valid, then if subsequent trace operations are not performed, the corresponding bit will be turned ON (in this case, length is always 0).

**length**

Number of bytes in GIF packet data.Total number of bytes in the GIF packet data sent with a single <offset>.

**offset**

XGKICK offset number of the GIF packet data sequence, starting from 0. For example, if <off> of DBGP_TYPE_XGKTCTL is 2, the starting <offset> in DBGP_XGKT_DATA will be 2.

**sequence**

In some cases, the GIF packet data for an XGKICK instruction can span multiple DECI2 packets. Sequence is a sequence number starting at 0 (for the second and subsequent packets, flag, length, and offset will have the same value as the first packet).

**GIF Packets Data**

The traced GIF packet data.

## Debug Control (DBGCTL/DBGCTLR)

Message to turn target debugging mode on and off for individual processors and the associated reply message. The messages use the following headers and data.

DBGP_TYPE_DBGCTL: DECI2 header + DBGP header + flag(32bit)

DBGP_TYPE_DBGCTLR: DECI2 header + DBGP header + flag(32bit)

The processor to be controlled is indicated in the id field of the DBGP header. If on, the D-bit of VU0, VU1 is enabled and interrupts are enabled. If off, the D-bit is disabled. No action is performed when the id field is DBGP_CPUID_CPU.

**flag**

1 if on.

0 if off.

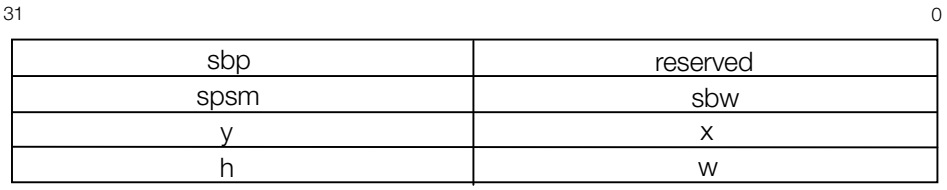
## GS StoreImage (RDIMG/RDIMGR)

Message which performs a read of image data from GS local memory and sends a response. The structure of the header and data are shown below:

DBGP_TYPE_RDIMG: DECI2 header + DBGP header + DBGP_RDIMG

DBGP_TYPE_RDIMGR: DECI2 header + DBGP header + DBGP_RDIMG_DATA

**Figure 9: DBGP StoreImage header (DBGP_RDIMG)**




The DBGP_TYPE_RDIMG message requests and reads GS image data.

The image data size (w x h x pixel size) must be a multiple of 16 bytes, and it must be less than 32767 X 16 bytes. Also, when the pixel size is 8 bits, x and w must both be multiples of 2. Similarly, when the pixel size is 4 bits, x and w must both be multiples of 4.

**sbp**

Address of transfer buffer (actual address is sbp x 64)

**sbw**

Width of transfer buffer (actual width is sbw x 64)

**spsm**

Pixel format of transfer data

0:PSMCT32 (Pixel size:32bit)
1 : PSMCT24 (Pixel size:24bit)
2 : PSMCT16 (Pixel size:16bit)
10: PSMCT16S (Pixel size:16bit)
19: PSMT8 (Pixel size:8bit)
20: PSMT4 (Pixel size:4bit)
27: PSMT8H (Pixel size:8bit)
36: PSMT4HL (Pixel size:4bit)
44: PSMT4HH (Pixel size:4bit)
48: PSMZ32 (Pixel size:32bit)
49: PSMZ24 (Pixel size:24bit)
50: PSMZ16 (Pixel size:16bit)
58: PSMZ16S (Pixel size:16bit)

**x, y**

Upper left point of transfer area

**w, h**

Width and height of transfer area (in pixels)

The DBGP_TYPE_RDIMGR message contains image data that is sent from the target to the host. When the image data size is larger than the buffer in the debugger running on the target, the image data will be split into multiple packets when it is sent.

**Figure 10: DBGP StoreImage Data (DBGP_RDIMG_DATA)**


**length**

Total no. of bytes in GS image data.

**sequence**

GS image data can be broken up into multiple DECI2 packets. The initial division number starts from sequence 0 (the length of the second and later divisions is always the same as the first division.)

**GS Image Data**

The GS image data that is read.

## Break Function(SETBPFUNC/SETBPFUNCR)

A message which sets the function that will be executed when a breakpoint is passed, and the associated response message. The structure of the header and data are shown below.

DBGP_TYPE_SETBPFUNC: DECI2 header + DBGP header + <adr>
DBGP_TYPE_SETBPFUNCR: DECI2 header + DBGP header + <adr>

The DBGP_TYPE_SETBPFUNC message requests that the specified function be executed when the breakpoint is passed. For example, if a function exists that saves the value of a certain variable to memory, when that function is specified using this message, the value of the variable can be traced when the breakpoint is passed <cnt> times.
If this message is used in the program, a function can be called from an arbitrary point without have to recompile the program, which would be the case if the function call were statically embedded.

Specify the address of the function you want to call in <adr>. If <adr> is set to 0, the setting is cleared.

## Appendix A (IOP register kind, number)

```
ISDBGP_KIND_HL      1     // hi,lo
    ISDBGP_NUM_LO   0
    ISDBGP_NUM_HI   1
ISDBGP_KIND_GPR     2     // general purpose register
ISDBGP_KIND_SCC     3     // system control register
ISDBGP_KIND_GTER    6     // GTE register
ISDBGP_KIND_GTEC    7     // GTE control register
```

## Appendix B (EE register kind, number)

```
ESDBGP_KIND_GPR     0     // general purpose register (128bit x 32)
ESDBGP_KIND_HLS     1     // hi,lo,sa
    ESDBGP_NUM_HI     0   //   hi  (64bit)
    ESDBGP_NUM_LO     1   //   lo  (64bit)
    ESDBGP_NUM_HI1    2   //   hi1 (64bit)
    ESDBGP_NUM_LO1    3   //   lo1 (64bit)
    ESDBGP_NUM_SA     4   // shift amount (32bit)
ESDBGP_KIND_SCR     2     // system control register (32bit x 32)
ESDBGP_KIND_PCR     3     // performance counter (32bit x 3)
ESDBGP_KIND_HDR     4     // hardware debug register (32bit x 8)
ESDBGP_KIND_FPR     5     // floating point register (32bit x 32)
ESDBGP_KIND_FPC     6     // floating point control (32bit x 2)
ESDBGP_KIND_V0F     7     // VU0 floating point register (128bit x 32)
ESDBGP_KIND_V0I     8     // VU0 integer/control register (32bit x 32)
ESDBGP_KIND_V1F     9     // VU1 floating point register (128bit x 32)
ESDBGP_KIND_V1I     10    // VU1 integer/control register (32bit x 32)
```