PDT

By Yossi Leon, Project Leader.

December 2006
Table of Contents

1. Introduction .......................................................................................................................... 4

2. Definitions ............................................................................................................................ 4

3. Debug Session ......................................................................................................................... 4
   3.1. Initializing New Debug Sessions ....................................................................................... 4
   3.2. HTTP Parameters ............................................................................................................. 4

4. Messages ................................................................................................................................. 5
   4.1. Data Types: ....................................................................................................................... 5
   4.2. Message Packet Structure ................................................................................................. 5
   4.3. The Request - Response Mechanism .................................................................................. 6
   4.4. Notification Messages ........................................................................................................ 6
       4.4.1. MSG_SESS_START .................................................................................................... 6
       4.4.2. MSG_SCRIPT_END .................................................................................................... 6
       4.4.3. MSG_READY ............................................................................................................. 6
       4.4.4. MSG_OUTPUT ............................................................................................................ 7
       4.4.5. MSG_HEADER_OUTPUT .......................................................................................... 7
       4.4.6. MSG_PHP_ERROR ..................................................................................................... 7
       4.4.7. MSG_ERROR .............................................................................................................. 7
   4.5. Request Messages ............................................................................................................ 8
       4.5.1. MSG_START .............................................................................................................. 8
       4.5.2. MSG_STOP ................................................................................................................ 8
       4.5.3. MSG_SESS_CLOSE ..................................................................................................... 8
       4.5.4. MSG_SET_OPTIONS ................................................................................................. 8
       4.5.5. MSG_STEP_INTO ....................................................................................................... 9
       4.5.6. MSG_STEP_OVER ..................................................................................................... 9
       4.5.7. MSG_STEP_OUT ....................................................................................................... 9
       4.5.8. MSG_GO ................................................................................................................... 9
       4.5.9. MSG_ADD_BREAKPOINT .......................................................................................... 9
       4.5.10. MSG_DEL_BREAKPOINT ....................................................................................... 10
       4.5.11. MSG_DEL_ALL_BREAKPOINTS ............................................................................. 10
       4.5.12. MSG_EVAL ............................................................................................................. 10
       4.5.13. MSG_GET_VAR ........................................................................................................ 10
       4.5.14. MSG_ASSIGN_VAR ................................................................................................. 11
       4.5.15. MSG_GET_CALL_STACK ....................................................................................... 11
       4.5.16. MSG_GET_STACK_VAR .......................................................................................... 11
       4.5.17. MSG_SET_PROTOCOL ............................................................................................. 11
   4.6. Response Messages .......................................................................................................... 12
       4.6.1. MSG_DONT_UNDERSTAND_R ................................................................................... 12
       4.6.2. MSG_START_R ........................................................................................................... 12
4.6.3.  MSG_STOP_R ................................................................. 12
4.6.4.  MSG_SESS_CLOSE_R ...................................................... 12
4.6.5.  MSG_SET_OPTIONS_R ..................................................... 12
4.6.6.  MSG_STEP_INTO_R ........................................................ 13
4.6.7.  MSG_STEP_OVER_R ....................................................... 13
4.6.8.  MSG_STEP_OUT_R ........................................................ 13
4.6.9.  MSG_GO_R .................................................................. 13
4.6.10. MSG_ADD_BREAKPOINT_R .............................................. 13
4.6.11. MSG_DEL_BREAKPOINT_R .............................................. 14
4.6.12. MSG_DEL_ALL_BREAKPOINTS_R .................................... 14
4.6.13. MSG_EVAL_R .............................................................. 14
4.6.14. MSG_GET_VAR_R ........................................................ 14
4.6.15. MSG_ASSIGN_VAR_R ................................................... 14
4.6.16. MSG_GET_CALL_STACK_R .............................................. 15
4.6.17. MSG_SET_PROTOCOL_R .................................................. 15
4.6.18. MSG_GET_STACK_VAR_R .............................................. 15
1. **Introduction**

This document describes the debug communication protocol used in the PDT project as of version 2006040701.

2. **Definitions**

**Client:** The debugger component in the PDT project that implements the client side of the debug protocol, specified in this document.

**Server:** A PHP extension that implements the debug protocol specified in this document. The server side debugger is not part of the PDT project.

3. **Debug Session**

3.1. **Initializing New Debug Sessions**

1. The client opens a port and waits for connection from the server.
2. The client sends an HTTP request to open the debugged page. The request should include additional HTTP parameters such as: “start_debug” and “debug_port” (see next section for full list of HTTP parameters)
3. The server connects to the debug port that was specified in the HTTP request and waits for requests.
4. Upon connection with the server, the client can start sending requests.

3.2. **HTTP Parameters**
4. **Messages**

Communication between the Client and the Server is based on three types of messages:

1. **Notification**: A one-way message with no reply (Usually sent from the server to the client).
2. **Request**: A request for information. A request is always followed by a response to the requestor.
3. **Response**: A reply to a request.

### 4.1. Data Types:

The data types used in the debug protocol are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_debug=1</td>
<td>Start debug</td>
</tr>
<tr>
<td>Debug_host</td>
<td>Address to return to</td>
</tr>
<tr>
<td>Debug_port</td>
<td>Port to return to</td>
</tr>
<tr>
<td>send_sess_end=1</td>
<td>Tells the server to send a session end message</td>
</tr>
<tr>
<td>Debug_no_cache</td>
<td>Used to avoid caching problem</td>
</tr>
<tr>
<td>Debug_stop=1</td>
<td>Stop at first line on each debugged file.</td>
</tr>
<tr>
<td>original_url</td>
<td>Original debugged URL string</td>
</tr>
<tr>
<td>debug_session_id</td>
<td>Debug session id in the Eclipse</td>
</tr>
<tr>
<td>debug_cont_session</td>
<td>Continuous debugging</td>
</tr>
<tr>
<td>debug_start_url</td>
<td>Starting debug from this url</td>
</tr>
</tbody>
</table>

- **BYTE**: single byte field
- **SHORT**: 2-byte integer in network order
- **INT**: 4-byte integer in network order
- **STRING**: INT number, representing string length, and then the string contents

### 4.2. Message Packet Structure

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>Data length</td>
</tr>
<tr>
<td>SHORT</td>
<td>Message ID (see below)</td>
</tr>
</tbody>
</table>

Note: The data length includes the Message ID but not the data length field itself.
4.3. The Request - Response Mechanism

Each Request & each Response has a request-id field.

When the client sends a request, it specifies the request-id. When returning a response, the replier sets the request-id of the response with the request-id of the corresponding request.

The requests must be handled in a FIFO order.

4.4. Notification Messages

Notifications are sent from the server to the client when something happens inside the PHP or the server that the client has to know about.

4.4.1. MSG_SESS_START

First message in the session, sent immediately after the server connects to the client.
Message ID = 2005
Message structure:

- protocol_id INT The protocol version the server uses
- filename STRING The file name for the running script
- uri STRING The URI of the running script
- query STRING Query string
- mode STRING Parameters of session options

4.4.2. MSG_SCRIPT_END

Notifies the client that the session has ended. The client can then request some data and then it should send: MSG_SESS_CLOSE.
Message ID = 2002
Message structure:

- status INT Always 0 for now

4.4.3. MSG_READY

'Ready' message for client, is generated whenever the server stops on breakpoint or for any other reason.
Message ID = 2003
Message structure:

- filename STRING Name of the current file
- lineno INT The line number in the current file
4.4.4. **MSG_OUTPUT**

Notifies the client on output generated by the PHP script.
Message ID = 2004
Message structure:

```
text  STRING  output text
```

4.4.5. **MSG_HEADER_OUTPUT**

Notifies the client on HTTP header generated by the PHP script.
Message ID = 2008
Message structure:

```
text  STRING  The text of the header (includes the mandatory \
\n)
```

4.4.6. **MSG_PHP_ERROR**

Notifies the client on PHP error generated during the script run.
Message ID = 2006
Message structure:

```
type    INT      Type of the error
filename STRING  File where it happened
lineno   INT      Line of the error
error    STRING   Error text
```

4.4.7. **MSG_ERROR**

Server error message (as opposed to MSG_PHP_ERROR, which is caused by PHP code).
Message ID = 2007
Message structure:

```
Message  STRING  Text of the error message
```

4.5. Request Messages

These messages are issued when the server needs to fetch some data from the client or the client needs to fetch some data from the server. The server will answer each request message with respective * _R response message (e.g. MSG_START will be answered with MSG_START_R).

4.5.1. MSG_START

Start or continue running the program. This message is used to allow the server to start running program after MSG_SESS_START.
Message ID = 1
Message structure:

req_id INT  Request ID. Debugger sends response with this ID

4.5.2. MSG_STOP

Stop running program immediately (i.e., act as if the next statement had a breakpoint on it).
Message ID = 2
Message structure:

req_id INT  Request ID. Debugger sends response with this ID

4.5.3. MSG_SESS_CLOSE

Closes the session.
Message ID = 3
Message structure:

status INT  Always 0 for now

4.5.4. MSG_SET_OPTIONS

Sends the debug session options bitmask. Should be sent before the MSG_START message
Message ID = 4
Message structure:

req_id INT  Request ID. Debugger sends a response with this ID
options INT  Send bitmask options to the server:
  - 0 bit (1) - send SCRIPT_END command and wait for SESS_CLOSE before closing the session
  - 1 bit (2) - return to PDT when there was an error in running the script ("stop on error")
  - 2 bit (4) - return to PDT when there was an exception (for PHP 5 only)
4.5.5. **MSG_STEP_INTO**
Step one statement with going into functions.
Message ID = 11
Message structure:

```plaintext
req_id  INT  Request ID.Debugger sends response with this ID
```

4.5.6. **MSG_STEP_OVER**
Step one statement without going into functions.
Message ID = 12
Message structure:

```plaintext
req_id  INT  Request ID. Debugger sends response with this ID
```

4.5.7. **MSG_STEP_OUT**
Run until end of the current function.
Message ID = 13
Message structure:

```plaintext
req_id  INT  Request ID. Debugger sends response with this ID
```

4.5.8. **MSG_GO**
Run the script, reset stepping settings.
Message ID = 14
Message structure:

```plaintext
req_id  INT  Request ID. Debugger sends response with this ID
```

4.5.9. **MSG_ADD_BREAKPOINT**
Add breakpoint.
Message ID = 21
Message structure:

```plaintext
req_id  INT  Request ID. Debugger sends response with this ID
type    INT  Breakpoint type:
          - 1: static breakpoint
          - 2: conditional breakpoint

Lifetime  INT  Breakpoint lifetime
          - 1: onetime breakpoint
          - 2: permanent breakpoint

For conditional breakpoints:
condition  STRING  Expression on which to break

For static breakpoints:
```
4.5.10. MSG_DEL_BREAKPOINT
Delete breakpoint.
Message ID = 22
Message structure:

  req_id  INT  Request ID. Debugger sends a response with this ID
  bp_id   INT  The breakpoint id (it was returned when this breakpoint was added)

4.5.11. MSG_DEL_ALL_BREAKPOINTS
Delete all breakpoints.
Message ID = 23
Message structure:

  req_id  INT  Request ID. Debugger sends response with this ID

4.5.12. MSG_EVAL
Evaluate an expression.
Message ID = 31
Message structure:

  req_id  INT  Request ID. Debugger sends response with this ID
  expr    STRING  Expression to evaluate

4.5.13. MSG_GET_VAR
Get the content of the variable or part of it, defined by the path (list of the elements to descend).
Message ID = 32
Message structure:

  req_id  INT  Request ID. Debugger sends a response with this ID
  var_expression STRING  Variable expression
  Depth    INT  Recursion depth
  path_len  INT  Length of the path
  path_len* {   path_el STRING  Path element
  }

Note: Depth is the depth of elements that will be put in a response if the value returned is an array or an object. E.g. if the depth is 2, then the contents of the variable itself and all contained elements will be sent, but not the contents of elements contained in those elements. One then may use another MSG_GET_VAR message with path to open these elements.
4.5.14. **MSG_ASSIGN_VAR**
Assign a value inside a variable.
Message ID = 33
Message structure:

- **Req_id** INT Request ID. Debugger sends response with this ID
- **Var_expression** STRING Variable expression
- **Val_expression** STRING Value to assign
- **depth** INT Recursion depth
- **path_len** INT Length of the path
- **path_len** [ ]
  - **path_el** STRING Path element

4.5.15. **MSG_GET_CALL_STACK**
Get the current call stack.
Message ID = 34
Message structure:

- **req_id** INT Request ID. Debugger sends response with this ID

4.5.16. **MSG_GET_STACK_VAR**
Get the variable from the stack or a part of it.
Message ID = 35
Message structure:

- **req_id** INT Request ID. Debugger sends response with this ID
- **Stack_depth** STRING Depth on the stack (current is 0, caller is 1, etc.)
- **var_name** STRING Name of the variable to fetch
- **Depth** INT Recursion depth. See **MSG_GET_VAR** for explanation of this option.
- **path_len** INT Length of the path
- **path_len** [ ]
  - **path_el** STRING Path element

4.5.17. **MSG_SET_PROTOCOL**
Request to set the protocol version
Message ID = 10000
Message structure:

- **req_id** INT Request ID. Debugger sends response with this ID
- **Protocol_id** INT The protocol version
4.6.  Response Messages

Responses are sent to client or the server as a reply to some request.

4.6.1.  MSG_DONT_UNDERSTAND_R

Response to any request the debugger does not understand.
Message ID = 1000
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>type</td>
<td>INT</td>
<td>Unknown message type as received</td>
</tr>
</tbody>
</table>

4.6.2.  MSG_START_R

‘Run’ (‘Start’) response.
Message ID = 1001
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>

4.6.3.  MSG_STOP_R

‘Stop’ response.
Message ID = 1002
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>

4.6.4.  MSG_SESS_CLOSE_R

Closes the session.
Message ID = 1003
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Always 0 for now</td>
</tr>
</tbody>
</table>

4.6.5.  MSG_SET_OPTIONS_R

‘Set options’ response.
Message ID = 1004
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>
4.6.6. MSG_STEP_INTO_R
'Step Into' response.
Message ID = 1011
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>

4.6.7. MSG_STEP_OVER_R
'Step over' response.
Message ID = 1012
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>

4.6.8. MSG_STEP_OUT_R
'Step out' response.
Message ID = 1013
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>

4.6.9. MSG_GO_R
'Go' response.
Message ID = 1014
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>

4.6.10. MSG_ADD_BREAKPOINT_R
'Add breakpoint' response.
Message ID = 1021
Message structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
<tr>
<td>breakpoint_id</td>
<td>INT</td>
<td>ID of the new breakpoint inside the Debugger</td>
</tr>
</tbody>
</table>
4.6.11. **MSG_DEL_BREAKPOINT_R**

'Delete breakpoint' response.
Message ID = 1022
Message structure:

```plaintext
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>
```

4.6.12. **MSG_DEL_ALL_BREAKPOINTS_R**

'Delete all breakpoints' response.
Message ID = 1023
Message structure:

```plaintext
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>
```

4.6.13. **MSG_EVAL_R**

'Eval' response.
Message ID = 1031
Message structure:

```plaintext
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
<tr>
<td>result</td>
<td>STRING</td>
<td>The eval result, converted to string</td>
</tr>
</tbody>
</table>
```

4.6.14. **MSG_GET_VAR_R**

'Get variable' response. Returns serialized variable contents.
Message ID = 1032
Message structure:

```plaintext
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
<tr>
<td>variable</td>
<td>STRING</td>
<td>Serialized variable contents</td>
</tr>
</tbody>
</table>
```

4.6.15. **MSG_ASSIGN_VAR_R**

'Assign var' response.
Message ID = 1033
Message structure:

```plaintext
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_id</td>
<td>INT</td>
<td>Request ID. As received from the client</td>
</tr>
<tr>
<td>status</td>
<td>INT</td>
<td>Status (0 on success, -1 on failure)</td>
</tr>
</tbody>
</table>
```
4.6.16. **MSG_GET_CALL_STACK_R**

'Get call stack' response. Returns the current call stack. The deepest level goes first.
Message ID = 1034
Message structure:

```plaintext
req_id          INT Request ID. As received from the client
depth           INT Depth of the call stack
depth * {
  caller_filename STRING Name of the file in which the function was called
  caller_lineno   INT Line in the file in which the function was called
  caller_function STRING Function name in which the function was called (can be empty)
  called_filename STRING Name of the file where the function is located
  called_lineno   INT Line in the file where the function starts
  called_function STRING Function name (can be empty)
  params          INT Function parameter count
  params*{
    name STRING Name of the variable
    value STRING Serialized (1-level) variable
  }
}
```

4.6.17. **MSG_SET_PROTOCOL_R**

Response for requesting to set protocol version
Message ID = 11000
Message structure:

```plaintext
req_id          INT Request ID. As received from the client
Protocol_id     INT The new protocol version or -1
```

4.6.18. **MSG_GET_STACK_VAR_R**

'Get variable' response. Returns serialized variable contents.
Message ID = 1035
Message structure:

```plaintext
req_id          INT Request ID. As received from the client
status          INT Status (0 on success, -1 on failure)
variable        STRING Serialized variable contents
```