

ARRIS Internal

|  |  |  |
| --- | --- | --- |
| All information contained in this document is property of ARRIS Group, Inc. No license, expressed or implied, under any patent, copyright or trade secret right is granted or implied by the conveyance of this document. No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise without the prior written permission of ARRIS Group, Inc. (See Standard Document Security, 320190-000 for details.)  Copyright © **2018** ARRIS Group, Inc. All rights reserved. | | |
| **Security Level** | | 2 |
| DSR Remote Interface Protocol (DRIP) | | | |
| Document Number: | 365-095-05646 | | |
| Revision: | x.9 | | |
| Author(s): | Bob Conover | | |
| Revision Date: | 03/05/2018 | | |

|  |
| --- |
| ARRIS Group, Inc.  101 Tournament Drive  Horsham, PA 19044 |

Revision History

| Rev | Iteration | Description | Incorporated By | Date |
| --- | --- | --- | --- | --- |
| x.1 | 1 | DSR 5xx Diagnostic proposal | J. Nelson | 08/10/06 |
| x.1 | 2 | Renamed to DRIP. Switched to using UDP as the main agent for communication. | B. Conover | 1/26/07 |
| x.2 |  | Added type field for Response Message. Update table of contents. | B. Conover | 1/21/07 |
| x.3 |  | Added Diagnostics Message body fields. | B. Conover | 5/10/07 |
| x.4 |  | Added Reportback Status Message type. | B. Conover | 11/7/07 |
| x.5 |  | Added Diagnostics Screen A2 | B. Conover | 1/15/08 |
| x.6 |  | Added Channel Scan. | B. Conover | 2/12/08 |
| x.7 |  | Added Section Filter, Diagnostics Screen D3. | B. Conover | 8/5/08 |
| x.8 |  | Added Debug and Stats Message Types. | B. Conover | 8/25/17 |
| x.9 |  | Added Unsolicited Message Type. | B. Conover | 2/27/18 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

Section Title Page Number

1. Purpose 4

1.1. Scope 4

1.2. Applicable Documents 4

2.1. H/W Interface 4

2.2. F/W Interface 4

3.1. Request Message 5

3.2. Response Message 5

4. Client/Server Communication 6

4.1. Server Authorization 6

4.2. Setting the IP Address 6

4.3. Sending and Receiving Messages 6

4.4. Client Interface 7

5. Neptune Design 7

5.1. The Message Types 7

5.2. The Status Types 8

5.3. The Message Body 9

5.3.1. RMT 9

5.3.2. DIAG 10

5.3.3. IP\_ADDR 11

5.3.4. MSP\_MSG 12

5.3.5. FRONT\_PANEL 12

5.3.6. REPTBACK\_STATUS 13

5.3.7. STATS 14

5.3.8. DEBUG 14

5.3.9. NEPTUNE\_TEST 15

5.3.10. UNSOLICITED 15

6.1. Player (Drip Client) 15

6.2. DRIP DROP (Drip Client) 16

6.3. DRIP Logger (Drip Client) 16

6.4. Browser (Drip Client) 16

# Purpose

This document will serve as a proposal for a diagnostic interface design for the 5xx which will allow a 3rd party to connect an Automated Test System or PC via the Ethernet across the USB port and control the DSR unit and retrieve data.

## Scope

This document contains a proposal for the diagnostic interface for the DSR 5xx product line.

## Applicable Documents

Refer to the following for additional information:

1. 526713-001, DSR 5xx On Screen Diagnostic descriptions.
2. **SYSTEM ARCHITECTURE**

The following section gives a very high level overview of the systems in scope.

## H/W Interface

The hardware interface will consist of using an Ethernet cable to connect the STB to the test station. This has the advantage of

1. It is available on all of the designs

2. It can be connected without opening the box.

This will require a USB-to-Ethernet device on all boxes without an Ethernet port. Only the DSR500 currently has an Ethernet port on the back of the box, but all other IRDs have a USB port that will allow for a USB-to-Ethernet device to be attached to it. Connecting an Ethernet cable from here to the network will allow the host and target devices to be able to communicate with each other.

## F/W Interface

The method of communication is using UDP across a user-defined port. UDP was chosen because

1. All boxes have UDP built-in to the vxWorks module.

2. Coding the server is rather straightforward and small.

3. UDP allows for broadcasting of messages, unlike TCP.

The host and the client will need communication software to be written. The host program should have a nice look-and-feel to it. This part will be used by testers, and all possible testing scenarios should be considered. Since testing needs will differ from group to group, it is expected that Starchoice will develop their own host program and Motorola will develop theirs.

The client firmware will reside in the Neptune and the application layers. Neptune will handle the basic receiving and sending of messages, and will handle some messages. If the message is not intended for Neptune, it will send the message to the application for processing.

1. **DSR Remote Interface Protocol**

The DSR Remote Interface Protocol (DRIP) will include a simple protocol that will allow for sending and receiving of data. The protocol is built on top of UDP. The protocol allows a computer to remotely control a DSR. The protocol can be used to simulate RCU input into the receiver, fetch diagnostic information, send messages, set parameters like the IP address, and many other uses. Functionality can be added by defining and adding request types. The DSR shall implement a DRIP server, while the PC shall implement a DRIP client.

## Request Message

This section describes the structure of an DRIP request. DRIP requests are transmitted in binary form within a UDP packet. The data within the packet is transmitted in network order, meaning that the most significant byte is transmitted first. A request packet has the following structure:

|  |  |  |
| --- | --- | --- |
| **Field** | **Bits** | **Description** |
| Type | 16 | The request type. Example values include RMT, RECORD, DIAG, etc. This field is used to “float” the request to the appropriate party (Neptune, OpenTV, IPG). Values from 0x0000 to 0x00FF are reserved for Neptune, while values from 0x0100 to 0xFFFF are reserved for the application. |
| Id | 16 | A unique number used to tie a request to a response. The client is required to increment this number after each new request. |
| Length | 16 | Length of the request body, in bytes. Maximum value is 1017. |
| Body | Length\*8 | Any additional data associated with the request type. The size (in bytes) of this field is specified by the length field. |
| Crc | 8 | Used to ensure the integrity of the message. This will be a two’s complement of the sum of the bytes of the entire message. |

## Response Message

For every Request Message received, there is a Response Message. The client matches a response to its associated request through the use of the id field.

|  |  |  |
| --- | --- | --- |
| Field | Bits | Description |
| Type | 16 | An echo of the Type field from the Request Message. |
| Id | 16 | An echo of the id field from the Request Message. |
| Status | 16 | The result of the request indicating either failure or success. Failure modes such as NOT\_AUTHORIZED (when the receiver is not DRIP authorized), and FAIL\_CRC (when the checksum fails), will be sent by the Neptune layer and will indicate the message has not been passed to the application layer. Other failure modes and success modes will be defined by the Neptune and application layers as needed. Values from 0x0000 to 0x00FF are reserved for Neptune, while values from 0x0100 to 0xFFFF are reserved for the application. |
| Length | 16 | Length of the response body, in bytes. Maximum value is 1017. |
| Body | Length\*8 | Any additional data associated with the response. This can be used to deliver any additional data. The contents of this field vary based on the request. The size of this field is specified by the length field. |
| Crc | 8 | Used to ensure the integrity of the message. This will be a two’s complement of the sum of the bytes of the entire message. |

# Client/Server Communication

Typically, a server will remain inactive until it is started up by a client. This will provide security against unwanted IP access into the IRD.

## Server Authorization

The server will not be running and, therefore, not accept messages unless it is authorized via a tier sent by the uplink to the box. When the appropriate tier is sent by the uplink, this will turn on the Neptune DRIP server. Turning off the tier will disable the DRIP server.

Another idea is to always have the server running and to return a NOT\_AUTHORIZED status to all DRIP messages. This has the advantage of informing the user that the connection is good, but just that the tier needs to be sent to authorize the box.

## Setting the IP Address

Normally, the IRD will not have an IP address defined. In this state, the IRD can receive broadcasted messages, but it can not receive IP-addressed messages. It is possible to set the IP address by sending a broadcast UDP message, using a command of type SET\_IP\_ADDRESS, and putting the Unit Address and IP address in the Message Body. The DRIP server will receive the broadcast message, will check the Unit Address in the message, compare to see if it matches, and set the IP address. This way the IP address can be set remotely for a particular unit.

## Sending and Receiving Messages

To communicate between the client and the server, a common port must be defined. The server will use UDP port 5002 for receiving messages, and will reply to the client port that was used to send the message.

Message size shall not exceed 1024 bytes of data. That should be more than enough space for most messages. If some messages require larger sizes, they can be broken up into 1024-byte chunks of data and sent across the interface.

The client will initiate all requests. The server will act as a listener and always reply whenever it receives a message. The client will send a message and listen for a response that may or may not come. Because UDP is not guaranteed to be sent without loss of packets, a UDP message may get lost. Therefore, the if the client does not receive a response within a certain amount of time, a timeout of say 5 seconds, it may assume that the packet is either lost, the server is suspended, or the IRD has crashed. At this time the client will take the appropriate measures such as retrying the message, logging a failure, etc.

The server will receive messages and act on them. The server always listens on port 5002, so all messages must be sent to that port. Messages may request the IRD to do a certain command or to fetch data. In either case, the server will always send a response back to the client. The id field of the response message will echo the id field of the message sent which will allow the client to know what message the response was for. The status field will indicate either failure or success, and this field can indicate some measure of the success or failure. For instance, it may indicate TEXT if it is sending back diagnostics info to the client in text format. Or it may indicate FAIL\_CRC if the checksum was invalid. The server replies back to the same port that the client used to send the message. This potentially allows for each client to be on a unique port and to receive messages only for messages that it sent and not for messages that other clients sent.

## Client Interface

The client interface for Neptune will be built separately from that of the application. This is because the application client software will send its own unique commands to run application testing, and the Neptune client will be designed to test at the Neptune level. Of course, there will be some features, such as RCU commands, that the Neptune client will have and that the application client will also wish to have, so some overlap is to be expected.

As to the look and feel, that remains to be decided. A command line interface would probably be easiest to code, but a GUI interface would be more user-friendly. The client interface will run on either Windows or Unix with probably Windows being the favored platform. The types of commands to be supported are:

RMT – Remote control command

FP – Front Panel

LRB – Little Red Button

DIAG\_XXX – Various diagnostics info, such as DIAG\_PPV for the pay-per-view, DIAG\_AUTH for authorization info, etc.

MSP\_MSG – Used to send an MSP-type message (Message Stream Protocol).

Various other commands can be programmed as the need arises. It should also display the replies in a readable fashion and log the replies to a file. Also, it should be able to support a method of scripting so that, for instance, channel change testing can be done overnight.

# Neptune Design

The following describes the Message Types, Status Types,  and Message Body for all supported Neptune DRIP messages.

## The Message Types

 Message Types from 0 to 0xFF are reserved for Neptune STS layer.  The following Message Types are defined:

|  |  |  |
| --- | --- | --- |
| Type | Value | Description |
| RMT | 1 | This means the message is a Remote Control messge.  It has a body length of 1, and a body of one byte |
| DIAG | 2 | This is a command to get some diagnostics data, such as Unit Address and IP Address.  No body. |
| IP\_ADDR | 3 | This command sets the IP address.  Body contains the Unit Address and IP address. |
| MSP\_MSG | 4 | This command sends a MSP message to Section Filter.  Can be used to put up TAMs, send UIMs, etc. |
| REBOOT | 5 | Sends a reboot request. |
| BOOT\_PARAMS | 6 | The message body contains the same information as in the bootup parameters, and this will replace that file. |
| FRONT\_PANEL | 7 | This message controls front panel requests. |
| RPTBACK\_STAT | 8 | Requests the Last Reportback Status along with time and date of last **successful** reportback. |
| STATS | 9 | Calls various “Stats” commands such as VcoStats, ChanStats, DmStats, etc. Output gets send to the serial port. |
| DEBUG | 10 | Turns debug on/off. Calls the SetEnableApps and SetEnableSeverity methods. |
| NEPTUNE\_TEST | 11 | Used to call various neptune public tests explicitly. These tests are only in the neptune\_tst directory and are not built into the standard code. |
| UNSOLICITED | 12 | When set, the server will be able to send unsolicited messages. Can be used for sending debug messages to the client program. |

## The Status Types

The Status Types from 0 to 0xFF are reserved for the Neptune STS layer.  Currently, the following values are defined:

|  |  |  |
| --- | --- | --- |
| Status | Value | Description |
| OK | 0 | Neptune handled and processed the command successfully. |
| BAD\_CRC | 1 | This message had a bad CRC and could not be handled. |
| BAD\_LENGTH | 2 | This message had a length greater than 1024 bytes, and could not be handled. |
| BAD\_IP\_MSG | 3 | The IP Address message can not be handled for some reason. Possibly, a bad IP address was entered. |
| NO\_MATCH\_UA | 4 | The IP Address message can not be handled because the Unit Address does not match the Unit Address of the box. |
| BAD\_TID | 5 | The MSP-type message has a bad or unknown table ID (i.e., MSP Message Type). |
| NA | 6 | Not Applicable. Used for Unsolicited Messages. |

## The Message Body

The message body will contain data either sent to or received from the client. Most messages sent to the client will contain data in the body to command the IRD to do something. Diagnostics requests may not have anything in the body of the request message, but will always have text in the body of the reply message. The following are descriptions of what is in the message body of the various message types.

### RMT

For the RMT message, the client sends a message with a body of one byte. This byte will be the command to send a keypress to the unit, and can be any of the following:

|  |  |  |
| --- | --- | --- |
| Name | Value | Notes |
| SPARE | 0 |  |
| ARROW\_RIGHT | 1 |  |
| MUTE | 2 |  |
| RED | 3 | ARROW\_UP\_LT |
| VOL\_DOWN | 4 |  |
| EXIT | 5 |  |
| POWER | 6 |  |
| VOL\_UP | 7 |  |
| GREEN | 8 | ARROW\_DN\_LT |
| DIGIT8 | 9 |  |
| DIGIT7 | 10 |  |
| CHAN\_UP | 11 |  |
| YELLOW | 12 | ARROW\_UP\_RT |
| PPV | 13 |  |
| DIGIT9 | 14 |  |
| DIGIT1 | 15 |  |
| CHAN\_DOWN | 16 |  |
| DIGIT2 | 17 |  |
| DIGIT3 | 18 |  |
| BLUE | 19 | ARROW\_DN\_RT |
| OPTIONS | 20 |  |
| LIST | 21 |  |
| GO\_BACK | 22 |  |
| DIGIT4 | 23 |  |
| LAST\_CHAN | 24 |  |
| DIGIT5 | 25 |  |
| INTERESTS | 26 |  |
| DIGIT6 | 27 |  |
| ENTER | 28 |  |
| ARROW\_LEFT | 29 |  |
| GUIDE | 30 |  |
| HELP | 31 |  |
| ARROW\_DOWN | 32 |  |
| BROWSE | 33 |  |
| FAVOURITES | 34 |  |
| ARROW\_UP | 35 |  |
| DIGIT0 | 36 |  |
| SOURCE | 37 |  |
| SELECT | 38 |  |
| INFO | 39 |  |
| FAST\_FWD | 40 |  |
| REWIND | 41 |  |
| RECORD | 42 |  |
| LOCKS | 43 |  |
| PAUSE | 44 |  |
| STOP | 45 |  |
| PLAY | 46 |  |
| SKIP\_AHEAD | 47 |  |
| SKIP\_BACK | 48 |  |
| ASPECT | 49 |  |
| SPARE | 50 |  |
| SPARE | 51 |  |
| INTERACTIVE | 52 |  |

The response message from the server will not contain anything in the body. It will indicate success or otherwise in the status field.

### DIAG

The client sends a DIAG message to the server with a body of either 0, 1, 2, or 3 bytes. If the body contains no bytes, then the version of the diagnostics is returned. This is because occasionally the diagnostics data may be updated to include more fields, and this is one way to tell what fields the DRIP diagnostics can report back. If the body contains 1 byte, then all the data associated with that screen number will be returned. Screen numbers are defined as:

|  |  |
| --- | --- |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |
| D2 | 5 |
| E1 | 6 |
| E2 | 7 |
| F | 8 |
| R | 9 |
| A2 | 10 |
| Channel Scan | 11 |
| D3 | 12 |

If there are two bytes in the body, the first byte will be the screen number, the second will be the line number, and this indicates that all data corresponding to that screen and line number will be returned. Similarly, if there are three bytes, the first byte is the screen, second byte is line, and third byte is item number. Only the corresponding item will be returned.

The return message’s body will contain text and new line characters to make a readable diagnostics return message. It will be in the format of

Data\_description = value.

For the Channel Scan diagnostics, this information is not on a diagnostics screen, but is on the hidden Channel Scan screen. The first byte in the body is 11, the second is 1, and the third byte will be 1-7 depending on the item of interest according to the arrangement in the Channel Scan screen.

For the Diag D3 screen, each call to return the entire screen’s worth of data will return the next eight lines of data. For the DSRs, up to 32 section filters may be implemented requiring up to four calls to get Diag D3 screen data. This data may also be obtained on a line-by-line basis.

### IP\_ADDR

The IP\_ADDR message is sent in order to assign an IP address to a box with a certain UA address. For this reason, the body of the IP address message contains both the Unit Address and the IP address.  Both values are null-terminated character strings.  The UA’s located at the start of the body, and the IP address is at an offset of 20 bytes from the start of the body. There is no checksum in the UA, and all leading 0’s are included. For the IP address, leading zeros are not included. For example:

A Unit Address will be in the format of xxx-xxxxx-xxxxx. For example: 000-03135-45176.

An IP address will be in the format of xxx.xxx.xxx.xxx. For example 168.84.55.40.

The main use of this message is to assign an address on a box that comes from the factory, and hence, has no IP address currently set. The message is sent broadcast-style to all receivers (i.e., with an IP address of 255.255.255.255). All boxes receive the message, whether they have an IP address set or not. Then, the box whose UA matches the UA of the sent message, will set its IP address to that contained in the message body.

All boxes that receive this message will send a reply message. The reply message will not contain anything in the body. The reply Status word can be any of the following:

OK – The message was received and the IP address has been set.

BAD\_IP\_MSG – The box tried to set its IP address but was unable to do so. Possibly the IP address was wrong.

NO\_MATCH\_UA – The box did not try to set the IP address because the UA did not match that of the box.

### MSP\_MSG

The MSP\_MSG is a message that the client can send to the server to input an MSP-type message to section filter. The MSP message will be contained in the body. An example of an MSP-type message would be the following:

C5 00 0B 00 50 32 FB 5C 36 0A FD 5E EE 89

This is a system time message. It contains a header, the message itself, and a CRC in accordance with the MSP spec. The first byte is the Table ID, which specifies the message type. C5 is a system time message. As an illustration to all the fields of this message and what they mean, the message can be broken out as follows:

C5 – message\_type

00 0B – MPEG\_table\_format (1 bit), multicast16\_addressed (1 bit), reserved (2 bits), message\_length (12 bits)

00 – message\_type\_version\_field

50 – quality (4 bits), reserverd (4 bits)

32 FB 5C 36 – system\_time

0A – GPS\_UTC\_offset = 10 seconds

FD 5E EE 89 - CRC

The reply message will contain nothing in the body. The status field will indicate success or not, and possible status reasons for this message are:

OK – The message was received and the IP address has been set.

BAD\_TID – The table ID, i.e., message\_type, does not match any known table ID in the box.

### FRONT\_PANEL

The Front Panel message simulates pressing the front panel buttons. The body contains two bytes defined as the following:

|  |  |
| --- | --- |
| Power | 0x01 |
| Go Back | 0x02 |
| Enter | 0x80 |
| Arrow Left | 0x08 |
| Arrow Down | 0x10 |
| Arrow Up | 0x20 |
| Arrow Right | 0x40 |
| Options | 0x04 |
| Little Red Button | 0x100 |

The response message from the server will not contain anything in the body. It will indicate success or otherwise in the status field.

### REPTBACK\_STATUS

The Reportback Status message requests information relating to the reportback. The format of the body will be

Reportback\_Status = status time date

The time and date fields are the time and date for the last successful reportback.

The status field represents the status of the last attempted reportback, not necessarily the last successful reportback. It will tell whether the last attempted reportback was a success or a failure, and if it was a failure, the specific reason of the reportback failure. The status can be any of the following:

"Successful ",

"PHONE\_CALL\_FAILURE",

"VIRTUAL\_CALL\_FAILURE",

"MODEM\_FAILURE",

"SMART\_CARD\_FAILURE",

"EMMG\_FAILURE",

"MISSING\_PHONE\_NUMBER",

"NACK\_RECEIVED",

"NO\_ACK\_RECEIVED",

"MODEM\_ERROR\_NODIALTONE",

"MODEM\_ERROR\_BUSYSIGNAL",

"MODEM\_ERROR\_NOCARRIER",

"MODEM\_ERROR\_INCOMINGCALL",

"MODEM\_ERROR\_NOANSWER",

"OFFHOOK\_DURING\_CALL",

"SECOND\_ATTEMPT\_FAILED",

"INCOMING\_ERROR",

"USER\_ABORT

For example, in a normal case, indicating a successful reportback, the message body will contain a text string that will look something like this:

Reportback\_Status = Successful 01:41 11/06/2007

### STATS

The Stats Request Message is used to call the various “stats” functions in the Neptune Code. The Stats functions are as follows:

|  |  |
| --- | --- |
| TunerStats | 0x01 |
| DmStats | 0x02 |
| CcStats | 0x03 |
| SfStats | 0x04 |
| SvcStats | 0x05 |
| VctStats | 0x06 |
| TdtStats | 0x07 |
| MmtStats | 0x08 |
| CdtStats | 0x09 |
| ChanStats | 0x0a |
| SiStats | 0x0b |
| SystemStats | 0x0c |
| VcoStats | 0x0d |
| AudioStats | 0x0e |
| VideoStats | 0x0f |
| HdmiStats | 0x10 |
| DlStats | 0x11 |
| SysTimeStats | 0x12 |

Depending on which Stats request is made, there may be either one or four additional bytes. This can be used to handle the “flag” or “chanNum” parameter that some of the Stats commands use.

The response message from the server will not contain anything in the body. It will indicate success or otherwise in the status field.

### DEBUG

The Debug Request Message consists of either the SetEnableApps or SetEnableSeverity command followed by either 1 byte or 4 bytes. The first byte of the message body will be set to 1 for SetEnableApps and 2 for SetEnableSeverity. The next four bytes will be the parameter value for SetEnableApps, and a single byte for the SetEnableSeverity where a byte value of 1 to 4 corresponds to severity levels 1000 to 4000.

For instance, the message body for the SetEnableApps command will look like

0x01 0xff 0xff 0xff 0xff

For the command “SetEnableApps -1”.

And the SetEnableSeverity message body will look like

0x02 0x03

For “SetEnableSeverity 3000”.

The response message from the server will not contain anything in the body. It will indicate success or otherwise in the status field.

### NEPTUNE\_TEST

The Request Message for Neptune\_Test is used to call specific tests in the neptune\_tst directory that were built as part of the Neptune development process. As such, the code to handle this will be in the test directory and will not be in the final release given to the customer. The first byte of the message body will contain the test type, and the following bytes will be used for any parameters that are needed.

It is intended that the test type be a value indicating a Neptune Public API test to run. For instance, a 1 might be Player Initialization test, and a 2 might be a Player Tune test corresponding to a channel number contained in the message body. These are still being organized as to which tests to include and how best to represent them in the message body.

### UNSOLICITED

The request message enables/disables unsolicited messages. The request message body contains a single byte, 0=off, 1=on. It shall be enabled by default.

Reply messages originate from the server and contain text in the message body that can be displayed in the client program. The text message shall be null-terminated.

Unsolicited messages from the server are currently defined as level 3 debug messages (notice) and higher.

1. **Future designs**

The protocol could be used to inject guide data, purge guide data, inject EPMs, inject TAMS, to start recording, stop recording, play a recording, delete a recording, clear the call log, delete all recordings, fetching the box date, time, getting timer counts for DVR, Auto-Tune, Reminder, getting a count of the recordings, playback position, view mode (standby, active), front panel state (On/Off) and other things. This protocol should allow for easy addition of more features, and allow for use on future products.

## Player (Drip Client)

This interface is written in Visual C++/C# and provides the user with a simple way to send DRIP commands to the DSR. It also provides a mechanism to “play” simple scripts to build test suites. We have used the Drip player in the past to run simple “channel up” tests and tests like this for hours. The Drip player can also be used to send simple MSP commands (PIM, STM, VCO, UIM, etc) that have been captured or prepared by hand to the DSR as well.

## DRIP DROP (Drip Client)

This interface is written in Visual C++/C# and provides the user with a windows application and several “canned” test suites including VCO’s, UIMs, TAMs, DVR tests, Guide/menu navigation and Channel up/down.

## DRIP Logger (Drip Client)

This interface is written in Visual C++/C# and provides the user with a simple way to send DRIP commands to the DSR and log the results that come back in a text file. In the past this has been useful when testing NIT changes and map changes.

## Browser (Drip Client)

This interface uses your PC’s browser to send/receive commands/status and provides the user with a simple way to get diagnostics from the DSR. Shaw has taken over the diagnostics since the Jenner release uses the following command structure for Browser mode.

<http://10.77.4.66:8082/diagnostics?a>

Simply by typing the DSR’s IP address:port / diagnostics?a (where ?a can be any diagnostic ?b, ?c ?d, etc) The DSR will return the text associated with the diagnostic screen.