Network Working Group
INTERNET-DRAFT
Expire in six months

T.G.Smith
Gecko Software
20 November 1996

Orbit Shadow TM - Data Transport Protocol
for
Java Thin Client Applications
to access
Network Management Platforms

<draft-smith-java-appl-00.txt>

_1. _S_t_a_t_u_s _o_f _t_h_i_s _M_e_m_o

_2. _A_b_s_t_r_a_c_t

_2._1. _I_n_t_e_r_e_s_t

This document is of interest to vendors of network management platforms and network management applications, for the management of intranets, private internetworks and the public Internet.

_2._2. _S_t_a_t_u_s _R_e_p_o_r_t

This is the first submission of this RFC.

_2._3. _P_r_o_t_o_c_o_l

This protocol is an application layer protocol for the interchange of data between a network management server and a network management client application. The protocol is intended to be used in operating system independant implementations of "thin" client applications.
This protocol is currently implemented by one manufacturer - Gecko

Software, but is being placed in the public domain as it is of interest to other manufacturers dealing with network management issues.

_3. _I_n_t_r_o_d_u_c_t_i_o_n _t_o _O_R_B_I_T

A GeckoWare product from Gecko Software

ORBIT is an architecture that integrates information technology management platforms, the World Wide Web, HTML browser applications and Java to provide operating system independant access to mission critical management information. Designed to be used with enterprise systems and network management platforms such as

SPECTRUM	from Cabletron Systems
HP OpenView	from Hewlett Packard
SunNET Manager	from Sun Microsystems

ORBIT Planet provides a public domain Java API to access information on a remote management platform, using the ORBIT Star server. The ORBIT Star and Satellite are Java powered, and communicate using ORBIT Shadow an application layer protocol that is implemented using TCP/IP sockets. Any Java compatabile application, specifically HTML browsers, can implement Java applets that access management application information through ORBIT Planet, Shadow and Star.

_3._1. _O_R_B_I_T _S_t_a_r

A server application (daemon) that co-exists with the management application, or remote from the management application where supported by the API or CLI. ORBIT Star accepts API or CLI calls from one or more ORBIT Satellites vi a ORBIT Shadow. These calls are passed to the management application by ORBIT Star and the results passed back to the relevant ORBIT Planet.

_3._2. _O_R_B_I_T _P_l_a_n_e_t

A public domain Java package, consisting of class, method and interface definitions that implement the Application Program Interfaces or Command Line Interfaces of a management application. Communication with a management application is through ORBIT Shadow to an ORBIT Star.

_3._3. _O_R_B_I_T _S_h_a_d_o_w

An application layer protocol, implemented uding TCP/IP sockets, that

controls data requests and responses between ORBIT Stars and ORBIT Planets.

_4. _U_s_e _C_a_s_e _f_o_r _O_r_b_i_t _S_h_a_d_o_w

An example of the application of Orbit Shadow is given as a use case. In this use case, the Orbit implementation is Orbit for SPECTRUM8r9, Cabletron Systems, Inc, enterprise management system. Fred, a network Manager, wishes to be able to view the state of all Cisco routers in his company's network using the Netscape browser that he has installed on his Apple Macintosh. Fred asks Cathy, an application programmer, to build him an application to perform this function.

Cathy designs a series of Java applets that Fred will access from an page on their corporate WWW server. One of the applets that she writes needs to be able to query the state of a specific router (a managed object).Cathy uses Orbit Planet as the interface to interrogate a SPECTRUM system, installed on their network, as to the state of the required router.

Orbit Planet (a public domain Java package) provides Cathy with object classes, methods and interfaces to interrogate and update her SPECTRUM server, using an Orbit Star application on the SpectroSERVER host[1]. Cathy includes the Orbit Planet Java packages in her applet that needs to be able to query the state of a specific router. When the relevant method is invoked[2] Orbit Shadow packages the request into an Orbit Shadow protocol frame and communicates the request to the relevant Orbit Star. From the above, you can see that Orbit Shadow is embedded in Orbit Planet. There is also a corresponding Orbit Shadow embedded in Orbit Star. When this Orbit Shadow receives the request from Cathy's applet, it executes the request and returns the data to the applet. To execute the request, Orbit Star calls the relevant SPECTRUM interface[3], parses and formats the resulting output before handing the data to Orbit Shadow. Orbit Shadow packages the data into an Orbit Shadow protocol frame and communicates the response to the relevant Orbit Planet.

END-NOTES
----

[1]   Orbit Star - SPECTRUM does not necessarily have to co-reside with a SpectroSERVER. V1.x of Orbit Star - SPECTRUM uses the SPECTRUM Command Line Interface and can reside on a separate host to the SpectroSERVER, provided that the SPECTRUM CLI is installed on the same host as Orbit Star - SPECTRUM. Compatabile versions of SpectroSERVER and

SPECTRUM CLI can be on different Operating System platforms, which implies that Orbit Star - SPECTRUM can be deployed on a different host operating system to that of the SpectroSRVER.

2 The method that corresponds to the SPECTRUM Command Line Interface (CLI) show attributes [mh=<model handle>] attr=<attribute id>

3 This would be the same interface as invoked by the Orbit Planet method in footnote 2.

_5. _P_r_o_t_o_c_o_l _D_e_f_i_n_i_t_i_o_n

_5._1. _F_r_a_m_e _S_t_r_u_c_t_u_r_e

An Orbit Shadow protocol frame contains the fields set out in "Table 1 - Orbit Shadow Protocol Frame".

Field number	Field descriptor
0	**version**
1	sequence
2	command
3	final
4	platform
5	interface
6	database
7	object
8	attribute
9	attribute type
10	value

Table 1 - Orbit Shadow Protocol Frame

Each of the fields in the protocol frame are described below.

_5._1._1. _V_e_r_s_i_o_n

The field version identifies the revision of Orbit Shadow that is in use. The field is an signed integer value with a length of eight (8) bits.

_5._1._2. _S_e_q_u_e_n_c_e

The field sequence is used ensure that information sent between the Orbit Star and the Orbit Planet remains in sequence. The field is a signed integer value with a length of thiry-two (32) bits.

_5._1._3. _C_o_m_m_a_n_d

The command field identifies the protocol action to be carried out by the receipient of the protocol frame. The defined commands are

8

Command	Abbreviation	Integer Value
authorise	**AUTH**	**0**
request	REQ	1
response	RES	2
positive acknowledge	ACK	3
negative acknowledge	NACK	4
wait	WAIT	5
terminate	TERM	6

Table 2 - Defined values for field: command

These commands are implemented in the protocol frame as a signed integer value with a length of eight (8) bits. Detailed examples of the use of these commands is given in section "Protocol State Model".

_5._1._4. _F_i_n_a_l

The final field is a boolean flag, implemented as a single bit. This flag is set to true (1) if the protocol frame is the last frame for the specified command and is set to false (0) if the frame is not the last frame in an exchange for a specified command. This flag is used when sending bulk data in a request or a response.

_5._1._5. _P_l_a_t_f_o_r_m

The platform field defines the network or systems management platform that Orbit Star interfaces with and is implemented as an signed integer of length eight (8) bits. Currently defined values for platform are

8

Value	Application	Platform	Abbreviation
-128	Reserved	-	-
..			
-1	Reserved	-	-
0	Cabletron SPECTRUM	Command Line Interface	csiCLI
1	Cabletron SPECTRUM	SSAPI ver 4.0	csiSSAPI

8

Value	Application	Platform	Abbreviation
2	**SunNET Manager**	**-**	**sunNET**
3	Sun Solstice Enterprise Manager	-	sunENT
4	HP Network Node Manager ver 3.x	-	hpNNM3
5	HP Network Node Manager ver 4.x	-	hpNNM4

Table 3 - Defined values for field: platform

_5._1._6. _I_n_t_e_r_f_a_c_e

The field interface defines the specific interface on the specified platform that this protocol frame refers to and is implemented as a signed integer eight (8) bits in length. Interface values are unique when used in conjunction with a specific platform value. Currently defined values for the field "interface" are set out in Table 4. The "interface" values for platform "csiCLI" use the typographical conventions from the SPECTRUM "Command Line Interface User Guide".

8

Platform	Interface Value	Interface Description
csiCLI	**000**	**reserved**
csiCLI	001	connect [hostname] [lh=landscape_handle]
csiCLI	002	disconnect
csiCLI	003	ack alarm aid=alarm_id [lh=landscape_handle]
csiCLI	004	create alarm cond=alarm_condition cause=alarm_cause mh=model_handle
csiCLI	005	create association rel=relation lmh=left_model_handle rmh=right_model_handle.
csiCLI	006	create event type=event_type text=event_text [mh=model_handle | lh=landscape_handle]
csiCLI	007	create model mth=model_type_handle [attr=attribute_id,

```
                                             val=value ...]
```

8

Platform	Interface	Interface
	Value	Description
		[lh=landscape_handle]
csiCLI	008	current [mh=model_handle | lh=landscape_handle]
csiCLI	009	destroy alarm [-n] aid=alarm_id [lh=landscape_handle]
csiCLI	010	destroy association [-n] rel=relation lmh=left_model_handle rmh=right_model_handle.
csiCLI	011	destroy model [-n] mh=model_handle
csiCLI	012	jump [text_string]
csiCLI	013	seek attr=attribute_id, val=value [lh=landscape_handle]
csiCLI	014	setjump [-n] text_string
csiCLI	015	show alarms [-x] [mh=model_handle | lh=landscape_handle]
csiCLI	016	show associations [mh=model_handle]
csiCLI	017	show attributes [attr=attribute_id [,iid=instance_id][,next]] [attr=attribute_id [,iid=instance_id][,next]...] [mh=model_handle]
csiCLI	018	show attributes mth=model_type_handle [lh=landscape_handle]
csiCLI	019	show children [rel=relation] [mh=model_handle]
csiCLI	020	show events [-x] [mh=model_handle | lh=landscape_handle]
csiCLI	021	show inheritance mth=model_type_handle [lh=landscape_handle]
csiCLI	022	show models [lh=landscape_handle]
csiCLI	023	show parents [rel=relation] [mh=model_handle]
csiCLI	024	show relations [lh=landscape_handle]
csiCLI	025	show rules rel=relation
csiCLI	026	show types [lh=landscape_handle]

```
        csiCLI         027      show landscapes
```

8

Platform	Interface Value	Interface Description
csiCLI	**028**	**update [mh=model_handle]** attr=attribute_id [,iid=instance_id],val=value [attr=attribute_id [,iid=instance_id],val=value...]
csiCLI	029	update [-n] mth=model_type_handle attr=attribute_id,val=value [attr=attribute_id,val=value...] [lh=landscape_handle]

Table 4 - Defined values for field: interface

_5._1._7. _D_a_t_a_b_a_s_e

The field database defines the target database on the platform for which the interface is to be invoked, and is implemented as a signed integer thirty two (32) bits in length. This field is used when the target platform supports distributed databases or has multiple data sources. The usage of this field is dependant on the value of platform. The semantic definitions for database that are currently defined are

8

Platform	Semantic value of database
csiCLI	**SPECTRUM VNM Landscape Handle**
csiSSAPI	SPECTRUM VNM Landscape Handle

Table 5 'Defined Values for field: database'

_5._1._8. _O_b_j_e_c_t

The field object defines an object in the "database" on the "platform", and is implemented as an signed integer thirty two (32) bits in length. The semantic value of object is specific for each platform. The semantic definitions for object that are currently defined are

Platform	Semantic value of object

8

csiCLI	**SPECTRUM VNM model handle**
csiSSAPI	SPECTRUM VNM model handle

Table 6 'Defined values for field: object'

_5._1._9. _A_t_t_r_i_b_u_t_e

The field attribute defines an attribute of an object, and is implemented as a signed integer thirty two (32) bits in length. The semantic value of attribute is specific for each platform. The semantic definitions for object that are currently defined are

8

Platform	Semantic value of attribute
csiCLI	**SPECTRUM VNM attribute handle**
csiSSAPI	SPECTRUM VNM attribute handle

Table 7 'Defined values for field: attribute'

_5._1._1_0. _A_t_t_r_i_b_u_t_e _T_y_p_e

The field "attribute type" describes the data type implemented by the "attribute". This field is implemented as a signed integer eight (8) bits in length. The Orbit Shadow protocol uses primitive data types as defined by Sun Microsystem's Java TM programming language. The definition of these primitive data types can be found at

http://www.javasoft.com

8

Field Value	Data type
0	**Boolean**
1	Char
2	Integer
3	Long
4	Float
5	Double
6	String

Table 8 'Defined values for field: attribute type'

_5._1._1_1. _V_a_l_u_e

The field value contains the data which is the value of the attribute specified in the Orbit Shadow protocol frame, and is implemented as a variable length sequence of bytes.

_5._2. _P_r_o_t_o_c_o_l _S_t_a_t_e _M_o_d_e_l

_5._2._1. _O_v_e_r_v_i_e_w

The exchange of Orbit Shadow frames between an Orbit Star and an Orbit Planet is described by a series of data flow diagrams. Data exchange between the Orbit Planet and Orbit Star is asynchronous and bi-directional, and is always initiated by an Orbit Planet. For every exchange that is initiated by an Orbit Planet, an Orbit Star will reply with either a positive or negative acknowledgement. A session between an Orbit Planet and an Orbit Star is initiated with a user authentication exchange, followed by a sequence of requests and responses. The session is not necessarily explicitly terminated, but can be terminated by the Orbit Planet, or the Orbit Star.

_5._2._2. _S_e_s_s_i_o_n _I_n_i_t_i_a_t_i_o_n

A session between an Orbit Planet and an Orbit Star requires that user authentication take place at least once, before any other requests are serviced. If no authentication has occured, then the Orbit Star will deny service to the Orbit Planet. Table 9 "Initial state - no authorisation", shows the protocol exchange where an Orbit Star denies service when no authentication has taken place.

8

Frame	Orbit Planet	Orbit Star	Frame detail

1	**REQ**		**object=<..>,**
			attribute=<..>,
			value=..

2		NACK	object=<..>,
			attribute=<..>,
			value=..

Table 9 "Initial state - no authorisation"

An authorisation exchange is shown in Table 10 "Initial state - authorisation exchange". In this exchange, both the user name and the

user password are accepted by the Orbit Star as valid.

8

Frame	Orbit Planet	Orbit Star	Frame detail

1	**AUTH**		**object=null,**
			attribute=<user name>,
			value=user name

2		ACK	object=<user object>,
			attribute=<user name>,
			value=user name

3	AUTH		object=<user object>,
			attribute=<user password>,
			value=password

4		ACK	object=<user object>,
			attribute=<user password>,
			value=password

Table 10 "Initial state - authorisation exchange"

If either the user name or user password is invalid, the Orbit Star will send a negative acknowledgement of the authentication request. An example of this is shown in

8

Frame	Orbit Planet	Orbit Star	Frame detail

1	**AUTH**		**object=null,**
			attribute=<user name>,
			value=user name

2		ACK	object=<user object>,
			attribute=<user name>,
			value=user name

3	AUTH		object=<user object>,
			attribute=<user password>,
			value=password

4		NACK	object=<user object>,
			attribute=<user password>,
			value=password

Table 11 "Initial state - authentication failure"

_5._2._3. _R_e_s_q_u_e_s_t _A_n_d _R_e_s_p_o_n_s_e

After the Orbit Planet has successfully identified the end-user to the Orbit Star, requests for access to the Orbit Star platform

interfaces can be made. On receiving a request, the Orbit Star will acknowledge that the request is to be serviced, or refuse the request through a negative acknowledgement. Table 12 "Request - service refusal" shows the protocol exchange when the Orbit Star refuses to service an Orbit Planet request.

8

Frame	Orbit Planet	Orbit Star	Frame detail

1	**REQ**		**object=<..>,**
			attribute=<..>,
			value=..

2		NACK	object=<..>,
			attribute=<..>,
			value=..

Table 12 "Request - service refusal"

Assuming that the Orbit Star is able to service the request from the Orbit Planet, the exchange might be as set out in Table 13 "Request - simple response".

8

Frame	Orbit Planet	Orbit Star	Frame detail

1	**REQ**		**object=<..>,**
			attribute=<..>,
			value=..

2		ACK	object=<..>,
			attribute=<..>,
			value=..

3	RES	object=<..>,	attribute=<..>,
			value=..

4	ACK		object=<..>,
			attribute=<..>,
			value=..

Table 13 "Request - simple response"

_5._2._4. _S_e_s_s_i_o_n _T_e_r_m_i_n_a_t_i_o_n

To be completed.

_6. _S_e_c_u_r_i_t_y _C_o_n_s_i_d_e_r_a_t_i_o_n_s

There is a possible requirement for encryption of passwords in the user authentication exchange in session initiation.

_7. _R_e_f_e_r_e_n_c_e_s

_8. _A_u_t_h_o_r'_s _A_d_d_r_e_s_s

Tony Gordon Smith
Gecko Software
17 Paragon Place
Blackheath
London
SE3 0SP
United Kingdom

Phone: +44-(0)700 0GECKO
        +44-(0)700 043256
Fax: +44-(0)700 740175
EMail: tony@geckoware.com