Chapter 1 - Introduction to the DIS Gateway

What is the DIS Gateway?

The DIS Gateway (DG) is a software package which provides a generic, portable interface between a DIS simulation network and various application programs. The DG is designed as a client/server architecture, with a single DG Server supporting one or more DG Clients on the same machine. It is designed around DIS 2.0.3.

What is needed to use the DIS Gateway?

The DIS Gateway was developed on a Silicon Graphics Incorporated (SGI) Indigo R4000 running IRIX 5.2. The DG code was compiled using the Verdix Ada Development System (VADS) 6.2.1, X Window System Version X11R5, and Motif 1.2.

What do you need to know to use the DIS Gateway?

If you are using the DG Server, you should be familiar with the DIS 2.0.3 Standard and basic network terminology (such as "IP broadcast address" and "UDP port"). To use the DG Client software, you should again be familiar with the DIS 2.0.3 Standard, and should also have a working knowledge of Ada. You should also be comfortable with the X-Window/Motif environment, since the Graphical User Interfaces (GUIs) for the DG Server and DG Client are written for this environment.

How does the DIS Gateway work?

The DG operates using a client/server relationship. A single DG Server on a machine provides data and network interfacing for one or more DG Clients running on the same machine. The only limitations on the number of Clients connected to a Server are the speed of the processor and the available memory of the machine. Clients can provide the Gateway with information such as ground-truth data for entities, emitter parameters, laser-designated targets, etc. The Server provides clients with up-to-date dead-reckoned positions of simulation entities, their associated emitter/laser/etc. parameters, and other PDU traffic. Information is stored in shared memory areas -- one central area is maintained by the Server (containing global information regarding all entities, emitters, lasers, etc.), and other smaller memory areas, one per Client (containing entity/emitter/laser information specific to that particular Client).

What are the special features of the DIS Gateway?

The primary special feature of the DIS Gateway is its client/server architecture. Every year, computers are becoming more powerful. Platforms with multiple CPUs are becoming more commonplace. These factors make it not only possible, but quite desirable, to run multiple DIS applications on a single computer. Traditional DIS interfaces have had great difficulty implementing this, since they were not designed to share resources (such as the UDP port). Some have developed work-arounds for this problem, but these are inherently inefficient (such as listening to the "real" UDP port of an exercise, and then echoing the data to multiple alternative UDP ports allocated to individual applications. Most interfaces simply do not support multiple DIS applications on a single computer. The DIS Gateway was designed from the beginning to support multiple applications. Since processor-intensive tasks such as PDU filtering and dead reckoning updates are performed in a centralized manner (by the DG Server), any number of Clients can be supported by this data without incurring a proportional increase in system loading. Since multiple Clients can be run on a single machine, it is not even necessary to have more than one machine to design and test complex DIS configurations.

Chapter 2 - How do you use the DIS Gateway?

How do you start the DIS Gateway?

The DG Server is started by running the program "DG_Server". If the default configuration file (DG_Server.Config) is present in the current directory, and if this file specifies a GUI, then the DG_Server will start up the GUI and wait for the user to make configuration changes. Always remember to bring up at least one parameter window and click its *Apply* button if you start the DG_Server with a GUI. Unless you do this, the DG_Server will patiently wait forever in this configuration mode. If you start the DG_Server without a GUI, it will process any configuration parameters in the default configuration file, and will then immediately transition to its runtime state.

DG Client programs can specify a GUI to run via the Initialize_Client procedure or via an entry in the configuration file. As with the DG Server, if the DG Client is started with a GUI, it will stay in configuration mode until an *Apply* button is pressed. Similarly, if no GUI is specified, the client will proceed into its runtime state.

How do you configure the DIS Gateway?

Both the DG Server and DG Clients can be configured at runtime using either the appropriate GUI, a configuration file, or a combination of the two.

The GUIs are standard X Window/Motif applications. The interfaces should be familiar to SGI users. In order for parameter modifications to take effect in the Client or Server, the *Apply* button must be pressed. If a configuration file is saved, it will save only those parameters which have actually been applied -- modifications which have been typed in but not applied will not be reflected in the resulting configuration files.

How do you stop the DIS Gateway?

There are several different ways to stop the DG. The DG Server can be stopped using the XDG_Server GUI by selecting the *Shutdown Server* option of the *File* menu. An alternative is to run the program Shutdown_Server. Stopping the DG Server will result in all DG Clients being stopped as well.

DG Clients can be stopped using the XDG_Client GUI by selecting the *Quit* option of the *File* menu. This will result in the Status code CLI_SYNC_SHUTDOWN being returned the next time Synchronize_With_Server or Client_Connected is called. As mentioned above, stopping the DG Server is equivalent to selecting *Quit* from the XDG_Client GUI.

Note: Since DG Clients are independent programs, they may incorporate other shutdown

methods. For example, one might write a DG Client which stops upon receiving a Stop/Freeze PDU. Such alternative methods of stopping a DG Client are obviously outside of the scope of this manual.

What are the callable routines in the DIS Gateway?

All of the callable routines in the DIS Gateway are related to creating DG Client programs. Table 1 lists routines which have related purposes. Each routine is described in further detail on subsequent pages.

Table 1, Callable Routines

Status Routines	
Success	Checks if a call to a DG routine was successful
Failure	Checks if a call to a DG routine was unsuccessful
Simulation Support Routines	
Initialize_Client	Connects the client application to the DG Server, and optionally loads a configuration file and/or starts the DG Client GUI.
Terminate_Server_Interface	Disconnects the client application from the DG Server.
Synchronize_With_Server	Synchronizes the Client with the DG Server after all dead reckoning updates have been performed.
Client_Connected	Checks to ensure that the Client has not been shut down by either the DG Server or the Client's GUI.
Get_Next_PDU	Retrieves the next PDU (if any) from the DG Server.
Get_Simulation_State	Retrieves the current simulation state.

Get_Entity_Info	Retrieves information for an entity based on Site/Application/Entity IDs.
Get_Entity_Info_By_Hash_Index	Retrieves information for an entity based on its hash table index.
Get_First_Simulation_Entity	Retrieves information for the first entity in the hash table.
Get_Next_Simulation_Entry	Retrieves information for the next entity in the hash table.
Get_Entity_Transmitter	Retrieves information regarding the transmitter (if any) associated with an entity.
Get_Entity_Emission	Retrieves information regarding the emission (if any) associated with an entity.
Send_PDU	Queues a PDU for transmission by the DG Server.
Set_Entity_Info	Change the entity state information for a given entity.
Remove_Entity	Removes the specified entity based on its Site/Application/Entity ID
Remove_Entity_By_Hash_Index	Removes the specified entity based on its hash table index.
Get_Entity_List	Produces a list of active entities in a form that can be used directly with the DIS Library filtering and sorting routines.
Generic PDU Manipulation	
Free_Generic_PDU	Deallocates memory used to store a generic PDU.
Null_Generic_PDU_Ptr	Checks if a generic PDU is empty.

Valid_Generic_PDU_Ptr	Checks if a generic PDU contains data.
Generic_Ptr_To_PDU_Header_Ptr	Returns a pointer to the header section of a generic PDU.
Generic_Ptr_To_ Entity_State_PDU_Ptr	Converts a generic PDU pointer to an entity state PDU pointer.
Generic_Ptr_To_ Fire_PDU_Ptr	Converts a generic PDU pointer to a fire PDU pointer.
Generic_Ptr_To_ Detonation_PDU_Ptr	Converts a generic PDU pointer to a detonation PDU pointer.
Generic_Ptr_To_ Service_Request_PDU_Ptr	Converts a generic PDU pointer to a service request PDU pointer.
Generic_Ptr_To_ Resupply_Offer_PDU_Ptr	Converts a generic PDU pointer to a resupply offer PDU pointer.
Generic_Ptr_To_ Resupply_Recieved_PDU_Ptr	Converts a generic PDU pointer to a resupply received PDU pointer.
Generic_Ptr_To_ Repair_Complete_PDU_Ptr	Converts a generic PDU pointer to a repair complete PDU pointer.
Generic_Ptr_To_ Repair_Response_PDU_Ptr	Converts a generic PDU pointer to a repair response PDU pointer.
Generic_Ptr_To_ Collision_PDU_Ptr	Converts a generic PDU pointer to a collision PDU pointer.
Generic_Ptr_To_ Create_Entity_PDU_Ptr	Converts a generic PDU pointer to a create entity PDU pointer.

Generic_Ptr_To_ Remove_Entity_PDU_Ptr	Converts a generic PDU pointer to a remove entity PDU pointer.
Generic_Ptr_To_ Start_Resume_PDU_Ptr	Converts a generic PDU pointer to a start/resume PDU pointer.
Generic_Ptr_To_ Stop_Freeze_PDU_Ptr	Converts a generic PDU pointer to a stop/freeze PDU pointer.
Generic_Ptr_To_ Acknowledge_PDU_Ptr	Converts a generic PDU pointer to an acknowledge PDU pointer.
Generic_Ptr_To_ Emission_PDU_Ptr	Converts a generic PDU pointer to an emission PDU pointer.
Generic_Ptr_To_ Laser_PDU_Ptr	Converts a generic PDU pointer to a laser PDU pointer.
Generic_Ptr_To_ Transmitter_PDU_Ptr	Converts a generic PDU pointer to a transmitter PDU pointer.
Generic_Ptr_To_ Receiver_PDU_Ptr	Converts a generic PDU pointer to a receiver PDU pointer.
Generic_Ptr_To_ Action_Request_PDU_Ptr	Converts a generic PDU pointer to an action request PDU pointer.
Generic_Ptr_To_ Data_Qeury_PDU_Ptr	Converts a generic PDU pointer to a data query PDU pointer.
Generic_Ptr_To_ Set_Data_PDU_Ptr	Converts a generic PDU pointer to a set data PDU pointer.
Generic_Ptr_To_	Converts a generic PDU pointer to a data PDU

Data_PDU_Ptr	pointer.
Generic_Ptr_To_ Event_Report_PDU_Ptr	Converts a generic PDU pointer to a event report PDU pointer.
Generic_Ptr_To_ Message_PDU_Ptr	Converts a generic PDU pointer to a message PDU pointer.
Generic_Ptr_To_ Signal_PDU_Ptr	Converts a generic PDU pointer to a signal PDU pointer.

DG_Status.Success

Purpose — Checks if a call to a DG routine was successful.

Syntax

```
function Success(
 Status : in DG_Status.STATUS_TYPE)
return BOOLEAN;
```

Package — DG_Status_.ada

Remarks — All of the callable procedures in the DG return a Status parameter, indicating the success or failure of the particular call. Rather than have the programmer use a construct like

if DG_Status.”=“(Status, DG_Status.SUCCESS) then ...

in their code, the Success and Failure functions are supplied instead. These may help improve code readability.

Return Value — Success returns TRUE if *Status* was set to DG_Status.SUCCESS, and returns FALSE for all other values.

Example

```
...
DG_Client.Initialize_Client(
   Load_GUI => TRUE,
   Status   => Local_Status);
--
-- Print initialization success/failure message
--
if (DG_Status.Success(Local_Status)) then
  Text_IO.Put_Line(
    "Successfully initialized the DG Client software");
else
  Text_IO.Put_Line(
    "DG Client initialization failed!");
  raise DG_CLIENT_FAILURE;
end if;
...
```

DG_Status.Failure

Purpose — Checks if a call to a DG routine was unsuccessful.

Syntax

```
function Failure(
 Status : in DG_Status.STATUS_TYPE)
return BOOLEAN;
```

Package — DG_Status_.ada

Remarks — All of the callable procedures in the DG return a Status parameter, indicating the success or failure of the particular call. Rather than have the programmer use a construct like

if DG_Status.”=“(Status, DG_Status.SUCCESS) then ...

in their code, the Success and Failure functions are supplied instead. These may help improve code readability.

Return Value — Failure returns FALSE if *Status* was set to DG_Status.SUCCESS, and returns TRUE for all other values.

Example

```
...
Local_Status : DG_Status.STATUS_TYPE;
...
DG_Client.Initialize_Client(
   Load_GUI => TRUE,
   Status   => Local_Status);
--
-- Exit program if client initialization fails
--
if (DG_Status.Failure(Local_Status)) then
  Text_IO.Put_Line(
    "DG Client initialization failed!");
  raise DG_CLIENT_FAILURE;
end if;
...
```

DG_Client.Initialize_Client

Purpose — Connects the client application to the DG Server, and optionally loads a configuration file and/or starts the DG Client GUI.

Syntax

```
procedure Initialize_Client(
Load_Configuration_File : in     BOOLEAN := FALSE;
Configuration_File      : in     STRING  := "";
Load_GUI                : in     BOOLEAN := FALSE;
GUI_Program             : in     STRING  := "";
GUI_Display             : in     STRING  := "0";
Client_Name             : in     STRING  := "";
Status                  :    out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Remarks — This procedure establishes a connection between the DG Client and the DG Server. If Load_Configuration_File is set TRUE and a file name included in Configuration_File, then the specified configuration file will be loaded. If Load_GUI is TRUE and a program name given for GUI_Program, then the specified GUI will be started as well. The Client_Name parameter can be used to individualize the displays of the XDG_Client GUI. Whatever string is given for this parameter will appear in the title bars of the XDG_Client GUI windows.

Return Value — Initialize_Client can return the following values in Status:

SUCCESS	No errors were encountered in initialization.
CLI_INI_LOGIN_DENIED_FAILURE	The DG Server did not permit this client to log in.
CLI_INI_FAILURE	An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Client.Terminate_Server_Interface

Purpose — Disconnects the client application from the DG Server.

Syntax

```
procedure Terminate_Server_Interface(
    Status : out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Remarks — This routine informs the DG Server that the client is shutting down its connection with the Server. All resources in use by the Client will then be deallocated.

Return Value — Terminate_Server_Interface can return the following values in Status:

SUCCESS	No errors were encountered in disconnecting the client.	
	CLI_TSI_FAILURE	An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Client.Synchronize_With_Server

Purpose — Synchronizes the Client with the DG Server after all dead reckoning updates have been performed.

Syntax

```
procedure Synchronize_With_Server(
Overrun : out BOOLEAN;
Status  : out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Remarks — Either Synchronize_With_Server or Client_Connected should be called each processing cycle to ensure that the client or the DG Server have not been shut down.

Return Value — Synchronize_With_Server sets Overrun as follows

TRUE — Synchronize_With_Server was called after the DG Server had updated all the dead reckoned positions.

FALSE

Synchronize_With_Server can return the following values in Status:

SUCCESS — No errors were encountered.

CLI_SYNC_SHUTDOWN — The DG Server or the Client GUI has signaled that this client should shut itself down.

CLI_SYNC_FAILURE — An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Client.Client_Connected

Purpose — Checks to ensure that the Client has not been shut down by either the DG Server or the Client's GUI.

Syntax

```
procedure Client_Connected(
    Status : out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Remarks — Either Client_Connected or Synchronize_With_Server should be called every processing cycle, to ensure that the client or the DG Server have not been shut down.

Return Value — Client_Connected can return the following values in Status:

SUCCESS — No errors were encountered.

CLI_SYNC_SHUTDOWN — The DG Server or the Client GUI has signaled that this client should shut itself down.

CLI_CONNECT_FAILURE — An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Client.Get_Next_PDU

Purpose — Retrieves the next PDU (if any) from the DG Server.

Syntax

```
        procedure Get_Next_PDU(
PDU_Pointer : out DG_Generic_PDU.
                GENERIC_PDU_POINTER_TYPE;
Status      : out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Remarks — This routine is normally called in a loop until either an error or a NULL pointer is returned. There are routines in the DG_Generic_PDU package to convert between generic and specific pointers. These are documented below.

Return Value — Get_Next_PDU will set PDU_Pointer to NULL if there are no PDUs to process.

Get_Next_PDU can return the following values in Status:

SUCCESS	No errors were encountered.
CLI_GNP_FAILURE	An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Client.Get_Simulation_State

Purpose — Retrieves the current simulation state.

Syntax

```
procedure Get_Simulation_State(
Simulation_State   : out SIMULATION_STATE_TYPE;
Stop_Freeze_Reason : out DIS_Types.A_REASON_TO_STOP;
Status             : out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Remarks — This data is maintained by the DG Server, and is updated based upon Simulation Management PDUs received by the Server. If simulation management PDUs are not used in a particular exercise, or if the Server is configured to filter out simulation management PDUs, then this routine will not provide an accurate assessment of the simulation state.

Return Value — Get_Simulation_State can return the following values in Status:

SUCCESS — No errors were encountered.

DG_Client.Get_Entity_Info

Purpose — Retrieves information for an entity based on Site/Application/Entity IDs.

Syntax

```
procedure Get_Entity_Info(
Entity_ID   : in     DIS_Types.AN_ENTITY_IDENTIFIER;
Entity_Info :    out DIS_PDU_Pointer_Types.
                     ENTITY_STATE_PDU_PTR;
Status      :    out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Return Value — Get_Entity_Info can return the following values in Status:

SUCCESS	No errors were encountered.
CLI_GEI_ENTITY_NOT_FOUND_FAILURE	The entity identifier provided does not match any known entity in the hash table.
CLI_GEI_FAILURE	An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Client.Get_Entity_Info_By_Hash_Index

Purpose — Retrieves information for an entity based on its hash table index.

Syntax

```
procedure Get_Entity_Info_By_Hash_Index(
Entity_Index : in     INTEGER;
Entity_Info  :    out DIS_PDU_Pointer_Types.
                      ENTITY_STATE_PDU_PTR;
Status       :    out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Return Value — Get_Entity_Info_By_Hash_Index can return the following values in Status:

SUCCESS — No errors were encountered.

DG_Client.Get_First_Simulation_Entity

Purpose Retrieves information for the first entity in the hash table.

Syntax

```
procedure Get_First_Simulation_Entity(
Entity_Info :  out DIS_PDU_Pointer_Types.
                   ENTITY_STATE_PDU_PTR;
Status      :  out DG_Status.STATUS_TYPE);
```

Package DG_Client_.ada

Return Value Get_First_Simulation_Entity can return the following values in Status:

SUCCESS No errors were encountered.

DG_Client.Get_Next_Simulation_Entity

Purpose Retrieves information for the next entity in the hash table.

Syntax

```
procedure Get_Next_Simulation_Entity(
Entity_Info : out DIS_PDU_Pointer_Types.
                 ENTITY_STATE_PDU_PTR;
Status      : out DG_Status.STATUS_TYPE);
```

Package DG_Client_.ada

Return Value Get_Next_Simulation_Entity can return the following values in Status:

SUCCESS No errors were encountered.

DG_Client.Get_Entity_Transmitter

Purpose — Retrieves information regarding the transmitter (if any) associated with an entity.

Syntax

```
procedure Get_Entity_Transmitter(
    Entity_ID       : in  DIS_Types.AN_ENTITY_IDENTIFIER;
    Transmitter_Info : out DIS_PDU_Pointer_Types.
                           TRANSMITTER_PDU_PTR;
    Status          : out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Return Value — Get_Entity_Transmitter can return the following values in Status:

SUCCESS	No errors were encountered.

DG_Client.Get_Entity_Emission

Purpose Retrieves information regarding the emission (if any) associated with an entity.

Syntax

```
procedure Get_Entity_Emission(
    Entity_ID      : in  DIS_Types.AN_ENTITY_IDENTIFIER;
    Emission_Info  : out DIS_PDU_Pointer_Types.
                         EMISSION_PDU_PTR;
    Status         : out DG_Status.STATUS_TYPE);
```

Package DG_Client_.ada

Return Value Get_Entity_Emission can return the following values in Status:

SUCCESS No errors were encountered.

DG_Client.Send_PDU

Purpose — Qeueus a PDU for transmission by the DG Server.

Syntax

```
procedure Send_PDU(
PDU_Address : in     System.ADDRESS;
Status      :    out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Return Value — Send_PDU can return the following values in Status:

SUCCESS	No errors were encountered.
CLI_SEND_FAILURE	An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Client.Get_Entity_List

Purpose — Produces a list of active entities in a form that can be used directly with the DIS Library filtering and sorting routines.

Syntax

```
procedure Get_Entity_List(
Entity_List : out DL_Linked_List_Types.
        Entity_State_List.PTR;
Status      : out DG_Status.STATUS_TYPE);
```

Package — DG_Client_.ada

Remarks — This routine was designed to be used in conjunction with the various DIS Library list manipulation routines, such as filtering and sorting. Please see the DL SRM form more information about these routines.

Return Value — Get_Entity_List can return the following values in Status:

SUCCESS	No errors were encountered.
CLI_GEL_FAILURE	An unknown error occurred which was trapped by the "when OTHERS" exception handler.

DG_Generic_PDU.Free_Generic_PDU

Purpose — Deallocates memory used to store a generic PDU.

Syntax

```
procedure Free_Generic_PDU(
    S: in out GENERIC_PDU_POINTER_TYPE);
```

Package — DG_Generic_PDU_.ada

Remarks — This should be used to free any PDUs provided by the DG_Client.Get_Next_PDU routine, once all processing on the PDU is complete.

Return Value — None.

DG_Generic_PDU.Null_Generic_PDU_Ptr

Purpose — Checks if a generic PDU is empty.

Syntax

```
function Null_Generic_PDU_Ptr(
 Ptr : in GENERIC_PDU_POINTER_TYPE)
return BOOLEAN;
```

Package — DG_Generic_PDU_.ada

Remarks — This function is provided simply to improve code readability. It is actually nothing more than a check of the pointer against NULL.

Return Value — Returns TRUE if *Ptr* is NULL, FALSE otherwise.

DG_Generic_PDU.Valid_Generic_PDU_Ptr

Purpose — Checks if a generic PDU contains data.

Syntax

```
function Valid_Generic_PDU_Ptr(
 Ptr : in GENERIC_PDU_POINTER_TYPE)
return BOOLEAN;
```

Package — DG_Generic_PDU_.ada

Remarks — This function is provided simply to improve code readability. It is actually nothing more than a check of the pointer against NULL.

Return Value — Returns TRUE if *Ptr* is not NULL, and FALSE otherwise.

DG_Generic_PDU.Generic_Ptr_To_PDU_Header_Ptr

Purpose — Returns a pointer to the header section of a generic PDU.

Syntax

```
function Generic_Ptr_To_PDU_Header_Ptr(
 X : GENERIC_PDU_POINTER_TYPE)
return PDU_HEADER_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This function is designed to be used in conjunction with the various Generic_Ptr_To_xxx_PDU_Ptr routines described below. Generic_Ptr_To_PDU_Header_Ptr permits the header region of the PDU to be examined. Based on the contents of the PDU Type field, the correct conversion routine to obtain the specific PDU can be determined.

Return Value — Pointer to the PDU header region.

DG_Generic_PDU.Generic_Ptr_To_Entity_State_PDU_Ptr

Purpose — Converts a generic PDU pointer to an entity state PDU pointer.

Syntax

```
function Generic_Ptr_To_Entity_State_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.ENTITY_STATE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This function converts a generic PDU pointer into an Entity State PDU pointer. This is one of an entire suite of routines to convert a generic PDU pointer into any of the DIS PDUs. These should be used in conjunction with the Generic_Ptr_To_PDU_Header_Ptr routine -- this routine will enable the user to examine the header fields, including the PDU_Type field. Based on the contents of the PDU_Type field, the appropriate Generic_Ptr_To_xxx_PDU_Ptr routine can be invoked to change the generic PDU into a specific PDU.

Return Value — A pointer to an entity state PDU.

Example

```
DG_Client.Get_Next_PDU(
  PDU_Pointer => Generic_PDU_Ptr,
  Status      => Status);
if (DG_Status.Success(Status)) then
   case (DG_Generic_PDU.Generic_Ptr_To_PDU_Header_Ptr).PDU_Type is
     when DIS_Types.FIRE_PDU =>
       My_Fire_PDU_Routine(
         DG_Generic_PDU.Generic_Ptr_To_Fire_PDU_Ptr(
           Generic_PDU_Ptr));
     when DIS_Types.DETONATION_PDU =>
       My_Detonation_Routine(
         DG_Generic_PDU.Generic_Ptr_To_Detonation_PDU_Ptr(
            Generic_PDU_Ptr));
     when ...
   end case;
end if;
```

DG_Generic_PDU.Generic_Ptr_To_Fire_PDU_Ptr

Purpose — Converts a generic PDU pointer to a fire PDU pointer.

Syntax

```
function Generic_Ptr_To_Fire_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.FIRE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Detonation_PDU_Ptr

Purpose — Converts a generic PDU pointer to a detonation PDU pointer.

Syntax

```
function Generic_Ptr_To_Detonation_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.DETONATION_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PD.Generic_Ptr_To_Service_Request_PDU_Ptr

Purpose — Converts a generic PDU pointer to a service request PDU pointer.

Syntax

```
function Generic_Ptr_To_Service_Request_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.SERVICE_REQUEST_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Resupply_Offer_PDU_Ptr

Purpose — Converts a generic PDU pointer to a resupply offer PDU pointer.

Syntax

```
function Generic_Ptr_To_Resupply_Offer_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.RESUPPLY_OFFER_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Resupply_Received_PDU_Ptr

Purpose — Converts a generic PDU pointer to a resupply received PDU pointer.

Syntax

```
function Generic_Ptr_To_Resupply_Received_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.RESUPPLY_RECEIVED_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Repair_Complete_PDU_Ptr

Purpose — Converts a generic PDU pointer to a repair complete PDU pointer.

Syntax

```
function Generic_Ptr_To_Repair_Complete_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.REPAIR_COMPLETE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Repair_Response_PDU_Ptr

Purpose — Converts a generic PDU pointer to a repair response PDU pointer.

Syntax

```
function Generic_Ptr_To_Repair_Response_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.REPAIR_RESPONSE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Collision_PDU_Ptr

Purpose — Converts a generic PDU pointer to a collision PDU pointer.

Syntax

```
function Generic_Ptr_To_Collision_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.COLLISION_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Create_Entity_PDU_Ptr

Purpose — Converts a generic PDU pointer to a create entity PDU pointer.

Syntax

```
function Generic_Ptr_To_Create_Entity_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.CREATE_ENTITY_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Remove_Entity_PDU_Ptr

Purpose — Converts a generic PDU pointer to a remove entity PDU pointer.

Syntax

```
function Generic_Ptr_To_Remove_Entity_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.REMOVE_ENTITY_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Start_Resume_PDU_Ptr

Purpose — Converts a generic PDU pointer to a start/resume PDU pointer.

Syntax

```
function Generic_Ptr_To_Start_Resume_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.START_RESUME_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Stop_Freeze_PDU_Ptr

Purpose — Converts a generic PDU pointer to a stop/freeze PDU pointer.

Syntax

```
function Generic_Ptr_To_Stop_Freeze_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.STOP_FREEZE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Acknowledge_PDU_Ptr

Purpose — Converts a generic PDU pointer to an acknowledge PDU pointer.

Syntax

```
function Generic_Ptr_To_Acknowledge_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.ACKNOWLEDGE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Emission_PDU_Ptr

Purpose — Converts a generic PDU pointer to an emission PDU pointer.

Syntax

```
function Generic_Ptr_To_Emission_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.EMISSION_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Laser_PDU_Ptr

Purpose — Converts a generic PDU pointer to a laser PDU pointer.

Syntax

```
function Generic_Ptr_To_Laser_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.LASER_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Transmitter_PDU_Ptr

Purpose — Converts a generic PDU pointer to a transmitter PDU pointer.

Syntax

```
function Generic_Ptr_To_Transmitter_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.TRANSMITTER_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Receiver_PDU_Ptr

Purpose — Converts a generic PDU pointer to a receiver PDU pointer.

Syntax

```
function Generic_Ptr_To_Receiver_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.RECIEVER_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_to_Action_Request_PDU_Ptr

Purpose — Converts a generic PDU pointer to an action request PDU pointer.

Syntax

```
function Generic_Ptr_To_Action_Request_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.ACTION_REQUEST_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Action_Response_PDU_Ptr

Purpose — Converts a generic PDU pointer to an action response PDU pointer.

Syntax

```
function Generic_Ptr_To_Action_Response_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.ACTION_RESPONSE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Data_Query_PDU_Ptr

Purpose — Converts a generic PDU pointer to a data query PDU pointer.

Syntax

```
function Generic_Ptr_To_Data_Query_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.DATA_QUERY_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Set_Data_PDU_Ptr

Purpose — Converts a generic PDU pointer to a set data PDU pointer.

Syntax

```
function Generic_Ptr_To_Set_Data_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.SET_DATA_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Data_PDU_Ptr

Purpose — Converts a generic PDU pointer to a data PDU pointer.

Syntax

```
function Generic_Ptr_To_Data_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.DATA_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Event_Report_PDU_Ptr

Purpose — Converts a generic PDU pointer to an event report PDU pointer.

Syntax

```
function Generic_Ptr_To_Event_Report_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.EVENT_REPORT_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Message_PDU_Ptr

Purpose — Converts a generic PDU pointer to a message PDU pointer.

Syntax

```
function Generic_Ptr_To_Message_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.MESSAGE_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

DG_Generic_PDU.Generic_Ptr_To_Signal_PDU_Ptr

Purpose — Converts a generic PDU pointer to a signal PDU pointer.

Syntax

```
function Generic_Ptr_To_Signal_PDU_Ptr(
 X : in GENERIC_PDU_PTR)
return DIS_PDU_Pointer_Types.SIGNAL_PDU_PTR;
```

Package — DG_Generic_PDU_.ada

Remarks — This routine is essentially identical to the Generic_Ptr_To_Entity_State_PDU_Ptr routine above. Please see this for return value information and example code.

Chapter 3 - How do you modify the DIS Gateway?

What would you do to port the DG to a different architecture, such as the VAX?

The DG Server contains two routines to help account for differences between the network representation of a PDU and its final representation on a given host system. In the package DG_Host_Specific are two routines, Translate_Host_To_Net and Translate_Net_To_Host. Any manipulation required to get between host and net data formats should be added to these routines. One example of this would be the little-endian to big-endian conversions and floating point representation conversions required for a VAX system.

What is the basic "outline" of a DG Client?

Oftentimes it is helpful to have a code example to use as a sort of outline or template for your own coding attempts. The following code illustrates a simple implementation of a DG Client, highlighting some of the most frequent functions, and illustrating their proper placement.

```
procedure
```

Chapter 4 - How do you troubleshoot the DIS Gateway?

What should I do if the DG Server or my Client crashes?

If the DG Server or a DG Client program crashes, it will almost certainly leave certain system resources marked as being "in use". Although the operating system will clean up most process resources following a program crash, it does not free any of the inter-process communications (IPC) resources. These must be deallocated by hand. There are two commands that assist with this -- ipcs and ipcrm. The ipcs command returns status information about IPC resources. The ipcrm command can be used to remove IPC resources. The ipcrm command takes two parameters. The first parameter indicates the type of IPC resource to remove: "-q" indicates a message queue, "-m" indicates a shared memory area, and "-s" indicates a semaphore. The second parameter is the ID of the resource, which can be determined from the output of the ipcs command.

For example, assume that (for whatever reason) a simulation has just crashed. Here is what the cleanup might look like:

First, see which resources are still allocated:

```
# ipcs
IPC status from /dev/kmem as of Fri Sep 16 11:47:10 1994
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
q   3500 0x00000001 --rw-rw-rw-    brett      mfs
Shared Memory:
m      0 0x000009a4 --rw-rw-rw-     root      sys
m   8201 0x00000002 --rw-rw-rw-    brett      mfs
m   7002 0x00000001 --rw-rw-rw-    brett      mfs
m   6903 0x00007003 --rw-rw-rw-    brett      mfs
m   6404 0x00007002 --rw-rw-rw-    brett      mfs
Semaphores:
s    640 0x00003801 --ra-ra-ra-    brett      mfs
```

In this example, the user "brett" was running the simulation. Only resources belonging to brett need to be deallocated. If multiple people were running simulation components (for example, multiple clients), then each person must eliminate their own resources. In the example above, you can see that user "root" has a shared memory area allocated. Since root was not involved in the simulation, there is no need to worry about its resources.

```
# ipcrm -q 3500
# ipcrm -m 8201
# ipcrm -m 7002
# ipcrm -m 6903
```

```
# ipcrm -m 6404
# ipcrm -s 640
```

Appendix - Additional information about the DIS Gateway

Unit/Filename Cross-Reference

The following lists show files for routines common to both the Server and Clients, code specific to the Server, and code specific to the Clients.

Independent units

DG_Load_Configuration_File	common/DG_CFM_Load_Configuration_File.ada
DG_Client_Interface	common/DG_Client_Interface(_).ada
DG_Configuration_File_Management	common/DG_Configuration_File_Management(_).ada
DG_Generic_Error_Processing	common/DG_Generic_Error_Processing(_).ada
DG_Generic_PDU	common/DG_Generic_PDU(_).ada
DG_Hash_Table_Support	common/DG_Hash_Table_Support(_).ada
Entity_Hash_Index	common/DG_HTS_Entity_Hash_Index.ada
DG_IPC_Keys	common/DG_IPC_Keys(_).ada
DG_Login_Queue	common/DG_Login_Queue(_).ada
Client_Login	common/DG_LQ_Client_Login.ada
Client_Logout	common/DG_LQ_Client_Logout.ada
Create_Login_Queue	common/DG_LQ_Create_Login_Queue.ada
Get_Client_Login	common/DG_LQ_Get_Client_Login.ada
Remove_Login_Queue	common/DG_LQ_Remove_Login_Queue.ada
Send_Server_Info	common/DG_LQ_Send_Server_Info.ada
DG_Math	common/DG_Math.ada
DG_PDU_Buffer	common/DG_PDU_Buffer.ada
PB_Add	common/DG_PB_Add.ada
PB_Read	common/DG_PB_Read.ada
DG_Shared_Memory	common/DG_Shared_Memory(_).ada
Map_Memory	common/DG_SM_Map_Memory.ada
Remove_Memory	common/DG_SM_Remove_Memory.ada
Unmap_Memory	common/DG_SM_Unmap_Memory.ada

DG_Server_Interface common/DG_Server_Interface.ada

DG_Simulation_Management common/DG_Simulation_Management(_).ada
- **Store_Emitter_Data** common/DG_SimMgmt_Store_Emitter_Data.ada
- **Store_Entity_Data** common/DG_SimMgmt_Store_Entity_Data.ada
- **Store_Laser_Data** common/DG_SimMgmt_Store_Laser_Data.ada
- **Store_Receiver_Data** common/DG_SimMgmt_Store_Receiver_Data.ada
- **Store_Simulation_Data** common/DG_SimMgmt_Store_Simulation_Data.ada
- **Store_Transmitter_Data** common/DG_SimMgmt_Store_Transmitter_Data.ada

DG_Status common/DG_Status.ada

DG_Synchronization common/DG_Synchronization(_).ada
- **Initialize_Client_Synchronization** common/DG_Sync_Initialize_Client_Synchronization.ada
- **Initialize_Server_Synchronization** common/DG_Sync_Initialize_Server_Synchronization.ada
- **Synchronize_Client** common/DG_Sync_Synchronize_Client.ada
- **Synchronize_With_Server** common/DG_Sync_Synchronize_With_Server.ada
- **Termination_Client_Synchronization** common/DG_Sync_Terminate_Client_Synchronization.ada

Generic_Linked_List common/Generic_Linked_List.ada

Independent units
- **DG_Server** (Control Unit) server/DG_Server.ada
- **DG_Filter_PDU** server/DG_Filter_PDU.ada
- **DG_Remove_Expired_Entities** server/DG_Remove_Expired_Entities.ada
- **DG_Start_Server_GUI** server/DG_Start_Server_GUI.ada

DG_Client_Tracking server/DG_Client_Tracking(_).ada
- **Add_Client** server/DG_CT_Add_Client.ada
- **Process_Client_Interfaces** server/DG_CT_Process_Client_Interfaces.ada
- **Process_Login_Queue** server/DG_CT_Process_Login_Queue.ada
- **Remove_Client** server/DG_CT_Remove_Client.ada
- **Shutdown_Client** server/DG_CT_Shutdown_Clients.ada
- **Synchronize_Clients** server/DG_CT_Synchronize_Clients.ada

DG_Dead_Reckoning_Support server/DG_Dead_Reckoning_Support(_).ada

DG_Host_Specific server/DG_Host_Specific(_).ada
- **Translate_Host_To_Net** server/DG_HS_Translate_Host_To_Net.ada
- **Translate_Net_To_Host** server/DG_HS_Translate_Net_To_Host.ada

DG_Network_Interface_Support server/DG_Network_Interface_Support(_).ada
- **Establish_Network_Interface** server/DG_NIS_Establish_Network_Interface.ada
- **Receive_PDU** server/DG_NIS_Receive_PDU.ada
- **Terminate_Network_Interface** server/DG_NIS_Terminate_Network_Interface.ada
- **Transmit_PDU** server/DG_NIS_Transmit_PDU.ada

DG_Server_Configuration_File_Management
server/DG_Server_Configuration_File_Management(_).ada
- **Process_Server_Configuration_Data**
server/DG_Server_CFM_Process_Server_Configuration_Data.ada
- **Save_Configuration_File**
server/DG_Server_CFM_Save_Configuration_File.ada

DG_Server_Error_Processing server/DG_Server_Error_Processing(_).ada

DG_Server_GUI server/DG_Server_GUI(_).ada

DG_Timer server/DG_Timer(_).ada
- **Change_Timer** server/DG_Timer_Change_Timer.ada
- **Initialize_Timer** server/DG_Timer_Initialize_Timer.ada
- **SIGALRM_Handler** server/DG_Timer_SIGALRM_Handler.ada
- **Synchronize** server/DG_Timer_Synchronize.ada
- **Terminate_Timer** server/DG_Timer_Terminate_Timer.ada

DG_Start_Client client/DG_Start_Client_GUI.ada

DG_Client client/DG_Client(_).ada
- **Client_Connected** client/DG_Client_Client_Connected.ada
- **Initialize_Client** client/DG_Client_Initialize_Client.ada
- **Send_PDU** client/DG_Client_Send_PDU.ada
- **Synchronize_With_Server** client/DG_Client_Synchronize_With_Server.ada
- **Terminate_Server_Interface** client/DG_Client_Terminate_Server_Interface.ada

DG_Client_Configuration_File_Management
client/DG_Client_Configuration_File_Management(_).ada
- **Process_Client_Configuration_Data**
client/DG_Client_CFM_Process_Client_Configuration_Data.ada
- **Save_Configuration_File**
client/DG_Client_CFM_Save_Configuration_File.ada

DG_Client_Error_Processing client/DG_Client_Error_Processing(_).ada

DG_Client_GUI client/DG_Client_GUI.ada

Error Message/Error Code Cross-Reference

The following is a list of each error code and the unit in which it can be set.

DG_Hash Table Support
ENTIDX_LOOP_FAILURE,
ENTIDX_FAILURE,

DG_Login_Queue

Client_Login
LQ_CLILOGIN_FAILURE
OTHERS exception
LQ_CLILOGIN_MSGGET_FAILURE
MsgGet call failed
LQ_CLILOGIN_MSGSND_FAILURE
MsgSnd call failed
LQ_CLILOGIN_MSGRCV_FAILURE
MsgRcv call failed

Client_Logout
LQ_CLILOGOUT_FAILURE
OTHERS exception
LQ_CLILOGOUT_MSGSND_FAILURE
MsgSnd call failed
LQ_CLILOGOUT_MSGRCV_FAILURE
MsgRcv call failed

Create_Login_Queue
LQ_CLQ_FAILURE
OTHERS exception
LQ_CLQ_MSGGET_FAILURE
MsgGet call failed

Get_Client_Login
LQ_GCL_FAILURE
OTHERS exception

Remove_Login_Queue
LQ_RLQ_FAILURE
OTHERS exception
LQ_RLQ_MSGCTL_FAILURE
MsgCtl call failed

Send_Server_Info
LQ_SSI_FAILURE
OTHERS exception
LQ_SSI_MSGSND_FAILURE
MsgSnd call failed

DG_Shared_Memory

Map_Memory
SM_MAPMEM_FAILURE
OTHERS exception

SM_MAPMEM_SHMAT_FAILURE
ShMAt call failed
SM_MAPMEM_SHMGET_FAILURE
ShMGet call failed

Remove_Memory
SM_REMMEM_FAILURE
OTHERS exception
SM_REMMEM_SHMCTL_FAILURE
ShMCtl call failed
SM_REMMEM_SHMGET_FAILURE
ShMGet call failed

Unmap_Memory
SM_UNMAPMEM_FAILURE
OTHERS exception
SM_UNMAPMEM_SHMDT_FAILURE
ShMDt call failed

DG_Synchronization

Initialize_Client_Synchronization
SYNC_INITCLI_FAILURE
OTHERS exception
SYNC_INITCLI_SEMCTL_FAILURE
SemCtl call failed
SYNC_INITCLI_SEMGET_FAILURE
SemGet call failed

Initialize_Server_Synchronization
SYNC_INISRV_FAILURE
OTHERS exception
SYNC_INISRV_SEMGET_FAILURE
SemGet call failed

Synchronize_Client
SYNC_CLI_FAILURE
OTHERS exception
SYNC_CLI_SEMCTL_FAILURE
SemCtl call failed

Synchronize_With_Server
SYNC_SRV_FAILURE
OTHERS exception
SYNC_SRV_SEMCTL_FAILURE
SemCtl call failed
SYNC_SRV_SEMOP_FAILURE
SemOp call failed

Terminate_Client_Synchronization
SYNC_TERMCLI_FAILURE
OTHERS exception
SYNC_TERMCLI_SEMCTL_FAILURE
SemCtl call failed

DG_Start_Server_GUI
- **DG_Start_Server_GUI**
  - DG_SERVER_GUI_FAILURE
    - OTHERS exception
  - DG_SERVER_GUI_EXECVE_FAILURE
    - ExecVE call failed

DG_Start_Client_GUI
- **DG_Start_Client_GUI**
  - DG_CLIENT_GUI_FAILURE
    - OTHERS exception
  - DG_CLIENT_GUI_EXECVE_FAILURE
    - ExecVE call failed

DG_Timer
- **Change_Timer**
  - TIMER_CHANGE_FAILURE
    - OTHERS exception
  - TIMER_CHANGE_SETITIMER_FAILURE
    - SetITimer call failed
- **Initialize_Timer**
  - TIMER_INIT_FAILURE
    - OTHERS exception
  - TIMER_INIT_SETITIMER_FAILURE
    - SetITimer call failed
- **Synchronize**
  - TIMER_SYNC_FAILURE
    - OTHERS exception
- **Terminate_Timer**
  - TIMER_TERM_FAILURE
    - OTHERS exception
  - TIMER_TERM_SETITIMER_FAILURE
    - SetITimer call failed
- **SIGALRM_Handler**
  - TIMER_SIGALRM_FAILURE
    - OTHERS exception
  - TIMER_SIGALRM_SETITIMER_FAILURE
    - SetITimer call failed

DG_Client_Tracking
- **Process_Login_Queue**
  - TRACK_PLQ_FAILURE
    - OTHERS exception
  - TRACK_PLQ_UNKNOWN_CLIENT_FAILURE
    - Logout by unknown client
- **Add_Client**
  - TRACK_ADD_FAILURE
    - OTHERS exception
- **Remove_Client**

TRACK_REM_FAILURE
OTHERS exception
TRACK_REM_UNKNOWN_CLIENT
Client not found in list
Synchronize_Clients
TRACK_SYNC_FAILURE
OTHERS exception
Shutdown_Clients
TRACK_SHUTDOWN_FAILURE
OTHERS exception
Process_Client_Interfaces
TRACK_PCI_FAILURE
OTHERS exception

DG_PDU_Buffer
Read
PB_READ_FAILURE
OTHERS exception
Add
PB_ADD_FAILURE
OTHERS exception
PB_ADD_PDU_TOO_BIG_FAILURE
PDU larger than buffer

DG_Server_Interface
Map_Interface
SVRIF_MAP_FAILURE
OTHERS exception
Unmap_Interface
SVRIF_UNMAP_FAILURE
OTHERS exception

DG_Client_Interface
Map_Interface
CLIIF_MAP_FAILURE
OTHERS exception
Unmap_Interface
CLIIF_UNMAP_FAILURE
OTHERS exception

DG_Client
Synchronize_With_Server
CLI_SYNC_FAILURE
OTHERS exception
CLI_SYNC_SHUTDOWN
Server commanding shutdown
Terminate_Server_Interface
CLI_TSI_FAILURE
OTHERS exception
Get_Next_PDU

CLI_GNP_FAILURE
OTHERS exception

Get_Entity_Info
CLI_GEI_FAILURE
OTHERS exception
CLI_GEI_ENTITY_NOT_FOUND_FAILURE
Entity ID not found

Get_Entity_List
CLI_GEL_FAILURE
OTHERS exception

Client_Connected
CLI_CONNECT_FAILURE
OTHERS exception

Initialize_Client
CLI_INI_FAILURE
OTHERS exception
CLI_INI_LOGIN_DENIED_FAILURE
Server did not permit login

Send_PDU
CLI_SEND_FAILURE
OTHERS exception

DG_Server_GUI

Map_Interface
SRVGUI_MI_FAILURE
OTHERS exception

Unmap_Interface
SRVGUI_UI_FAILURE
OTHERS exception

DG_Client_GUI

Map_Interface
CLIGUI_MI_FAILURE
OTHERS exception

Unmap_Interface
CLIGUI_UI_FAILURE
OTHERS exception

DG_Network_Interface_Support

Establish_Network_Interface
NIS_ENI_FAILURE
OTHERS exception
NIS_ENI_SOCKET_FAILURE
Socket call failed
NIS_ENI_SETSOCKOPT_FAILURE
SetSockOpt call failed
NIS_ENI_FCNTL_SETOWN_FAILURE
FCntl(F_SETOWN) call failed
NIS_ENI_FCNTL_SETFL_FAILURE

FCntl(F_SETFL) call failed
NIS_ENI_BIND_FAILURE
Bind call failed
Terminate_Network_Interface
NIS_TNI_FAILURE
OTHERS exception
NIS_TNI_CLOSE_FAILURE
Close call failed
Receive_PDU
NIS_RCVPDU_FAILURE
OTHERS exception
NIS_RCVPDU_RECVFROM_FAILURE
RecvFrom call failed
Transmit_PDU
NIS_TXPDU_FAILURE
OTHERS exception
NIS_TXPDU_SENDTO_FAILURE
SendTo call failed
DG_Dead_Reckoning_Support
Update_Entity_Positions
DRS_EEP_FAILURE
OTHERS exception
DRS_EEP_UPDATE_POSITION_FAILURE
DL Update_Position call failed
DG_Filter_PDU
DG_Filter_PDU
FILTER_FAILURE
OTHERS exception
DG_Simulation_Management
Store_Emitter_Data
SIMMGMT_STREMIT_FAILURE
OTHERS exception
SIMMGMT_STREMIT_NO_ENTITY_FAILURE
Emitter's entity unknown
SIMMGMT_STREMIT_TABLE_FULL
Emitter hash table full
Store_Entity_Data
SIMMGMT_STRENTITY_FAILURE
OTHERS exception
SIMMGMT_STRENTITY_TABLE_FULL
Entity hash table full
Store_Laser_Data
SIMMGMT_STRLAS_FAILURE
OTHERS exception
SIMMGMT_STRLAS_NO_ENTITY_FAILURE
Laser's entity unknown

SIMMGMT_STRLAS_TABLE_FULL
Laser hash table full

Store_Receiver_Data

SIMMGMT_STRREC_FAILURE
OTHERS exception

SIMMGMT_STRREC_NO_ENTITY_FAILURE
Receiver's entity unknown

SIMMGMT_STRREC_TABLE_FULL
Receiver hash table full

Store_Simulation_Data

SIMMGMT_STRSIM_FAILURE
OTHERS exception

SIMMGMT_STRSIM_UNKNOWN_PDU_FAILURE
Unrecognized/unhandled PDU

Store_Transmitter_Data

SIMMGMT_STRTRAN_FAILURE
OTHERS exception

SIMMGMT_STRTRAN_NO_ENTITY_FAILURE
Transmitter's entity unknown

SIMMGMT_STRTRAN_TABLE_FULL
Transmitter hash table full

DG_Remove_Expired_Entities

DG_Remove_Expired_Entities

REE_FAILURE
OTHERS exception

DG_Configuration_File_Management

Load_Configuration_File

CFM_LCF_FAILURE
OTHERS exception

CFM_LCF_VALUE_MISSING_FAILURE
No value in line

CFM_LCF_EQUAL_MISSING_FAILURE
No "=" in line

CFM_LCF_KEYWORD_MISSING_FAILURE
No keyword in line

CFM_LCF_INVALID_FILENAME_FAILURE
Invalid config filename

DG_Server_Configuration_File_Management

Save_Configuration_File

SRVCFM_SCF_FAILURE
OTHERS exception

Process_Server_Configuration_Data

SRVCFM_PSCD_FAILURE
OTHERS exception

SRVCFM_PSCD_KEYWORD_FAILURE
Invalid keyword detected

DG_Client_Configuration_File_Management
- **Save_Configuration_File**
  - CLICFM_SCF_FAILURE
    - OTHERS exception
- **Process_Client_Configuration_Data**
  - CLICFM_PCCD_FAILURE
    - OTHERS exception
  - CLICFM_PCCD_KEYWORD_FAILURE
    - Invalid keyword detected

DG_Generic_Error_Processing
- **Report_Error**
  - GEP_RE_OVERFLOW
    - Error queue overflow

DG_Server
- **DG_Server**
  - SRV_OVERRUN
    - Timeslice exceeded

Placeholder (developmental use ONLY!)
- DG_PLACEHOLDER_ERROR);

JFT-149-DG.SRM, 30-Sept-94

Rev. A

SOFTWARE REFERENCE MANUAL (SRM)

FOR THE

DIS GATEWAY (DG) CSCI 1

OF THE

ADA DISTRIBUTED INTERACTIVE SIMULATION (ADIS) PROJECT

CONTRACT NO. N00421-92-D-0028

CDRL SEQUENCE NO. A009

Prepared for:

Naval Air Warfare Center Aircraft Division
Flight Test and Engineering Group

Prepared by:

J.F. Taylor, Inc.
R. 235 and Maple Rd.
Lexington Park, MD 20653

Authenticated by:	Approved by:
(Contracting Agency)	(Contractor)
(Date)	(Date)

Table of Contents

Tables