Release Note for BiiN™ Operating System Manuals

This relase note consists of three parts:

- General operating system information
- The *BiiN™/OS Guide* part, and
- The *BiiN™/OS Reference Manual* part (including new chapters to be added to the manual).

General Operating System Information

This release note provides release information for OS V1.01.00 of the BiiN™ operating system. This OS release is compatible with:

Table 1. System Software Set FCS1 R1.2

Release	Description
SM V2.00.01	System Monitor—Firmware on EEPROM.
OSBASE V1.01.00	Operating System and CLEX.
OS UTIL V1.01.00	Utilities 1.
UX V1.01.00	UNIX-Based Software and Utilities.
UTILS V1.01.08	Utilities 2.
SPOOL V1.01.07	Printer/Spooling Software.
MDS V1.00.03	Maintenance and Diagnostics Software.
ADA V1.00.06	Ada Compiler.
C V1.00.06	C Compiler.
FORTRAN V1.00.07	FORTRAN Compiler.
COBOL V1.00.03	COBOL Compiler.
PASCAL V1.00.07	Pascal Compiler.
LINK V1.01.03	Linker.
LIBRARIAN V1.01.01	Librarian.
DEBUGGER V2.00.00	Debugger.
EMACS V2.00.06	Emacs Text Editor.
FORMS V1.02.06	Form Service and Utilities.
SMS V2.00.01	Software Management System.
FTS V1.02.11	ISO File Transfer Software.

Compiling and Linking Information

This section provides some additional information about compiling and linking programs that use most OS packages.

When compiling an Ada program that uses the OS, you must specify the location of the OS package *interface files*. You do this using a pathname of the form:

`/lib/`*`module_name/os_library`*

You can compile Ada programs that make use of OS service calls by attaching OS libraries to your local `ada_library`. The following two libraries are required:

```
/lib/kernel/ada_library

/lib/gcos/ada_library
```

All the other OS libraries also reside in `/lib` modules. The modules contained in `/lib` are:

```
ada_rts/        ctm/              gs0_hw             nulldev/
bdisk/          dist_supp/        gs0hdlcline/       pipe/
bstreamer/      dod_echo_daemon/  gs0landriver/      print_inpmgt/
carrier_mgt/    dod_tftp_daemon/  has/               psmgt/
chouse/         dod_transp_mgt/   hdlc/              pss/
cluster_svr/    envrec/           hsappl/            sct/
cmsup/          ethernet/         ieee8023/          sms/
cmtest/         fe/               iso_echo_daemon/   sort_merge/
comm_trace/     field_access/     iso_transp_mgt/    ssm/
cp_async/       forms/            kernel/            stream_io/
cp_mgt/         fts/              linkcall/          term_inpmgt/
cp_scsi/        gcos/             nodeconfig/        terminfo/
                gdp_diag/                            ux/
```

Each of these modules contains some or all of the following OS libraries:

`views` - A directory containing the views for the service, named *view*.

`image` - The binary object image of the package.

`vtables` - A directory containing one vtable file per view per domain. A vtable file defines the procedure entry points to be included in a view.

`viewdirs` - A directory containing one view directive file for each view supplied by the service.

`ada_library` - An optimized Ada library that provides the Ada interfaces to the service.

`include` - A linker library containing the C header files needed to compile and link a C program that uses C system bindings to call OS function calls.

`src` - package specification source files

`lib` - alias for `ada_library/lib`

`kernel` has one additional library, `clib`, a C library.

General Caveats

This section lists major OS features that are not yet implemented:

1. OS services are not distributed.
2. Subtransactions are not supported. Attempting to start a subtransaction raises `System_Exceptions.operation_not_supported`.

3. The clustered and hashed file organizations are not supported. Hashed indexes are not supported.
4. The OS does not support fault tolerant hardware, that is, hardware configured for fault checking or continuous operation.
5. The report service is not implemented.
6. IPI disks are not supported.
7. Basic tape devices are not supported (although basic streamer devices are supported).

BiiN™/OS Guide Release Information

This manual is about 65% complete. It does not describe:

The clearinghouse

Volume set management

Basic disks, streamers, or tapes

Guidelines for writing utilities

Using time or timed requests

Resource control or accounting

Several types of advanced type managers

Adding device drivers.

Throughout — Several reviewers remarked that excerpts from examples don't always show enough context. If an excerpt is confusing, check the complete example listing in Appendix A. We will be expanding our excerpts in many cases in subsequent releases.

Throughout — Some of the examples are tested. All of them have been compiled, though with an earlier version of the Ada compiler than the version in this system release.

II-3 — If an operation begins to store a master AD for an object, but fails, then no master AD can ever be stored for the object. One way this can happen is to store the master within a transaction and then abort the transaction. Aborting the transaction does undo the storing of the AD, but still no master AD can ever again be stored for the object. A particular case to avoid is:

```
begin
  Directory_Mgt.Store(some_name, x);
exception
  when Directory_Mgt.entry_exists =>
    Directory_Mgt.Delete(some_name);
    Directory_Mgt.Store(some_name, x);
end;
Passive_Store_Mgt.Update(x);
```

This code will not work! The first `Store` tries to store the master AD and fails; the subsequent store works but stores only an alias. The `Update` call will raise `Passive_Store_Mgt.no_master_AD`. Such a code fragment can be rewritten as follows:

```
begin
  Directory_Mgt.Delete(some_name);
exception
  when Directory_Mgt.no_access =>
    null;
      -- There was nothing to delete.
end;
Directory_Mgt.Store(some_name, x);
Passive_Store_Mgt.Update(x);
```

IV-4 — Several of the illustrations show window shapes that are only possible on graphic terminals. Windows on character terminals are always as wide as the screen and are tiled, not overlapping.

IV-4-10	Figure IV-4-5 is incorrect.
IV-4-15	Character display I/O can be used via an opened window even if the window was opened with another access method.
IV-6K	This `Printing` chapter now has updated examples from a new version of `Print_cmd_ex` (included in this release note beginning on the next page). Specific changes are: • An application must specify `Device_Defs.nothing` for the `allow` parameter on an `Open` call. • Ensure that the requested sheet size is within the printer's capability.
IV-7-6	A slot in a relative file is not removed if the record it contains is deleted.
VI-1-16	Table VI-1-2 should also list the `resource_exhausted` local event. By default it is enabled. The default handler kills the process.
VI-3-3	Figure VI-3-1 is incorrect. There are no processes in queues greater than priority 15.

Print_cmd_ex Example Procedure

```
 1  with
 2    Byte_Stream_AM,
 3    CL_Defs,
 4    Command_Handler,
 5    Device_Defs,
 6    Directory_Mgt,
 7    Incident_Defs,
 8    Message_Services,
 9    Process_Mgt,
10    Process_Mgt_Types,
11    Spool_Defs,
12    Spool_Device_Mgt,
13    String_List_Mgt,
14    System,
15    System_Defs,
16    Text_Mgt;
17
18  procedure Print_cmd_ex is
19    --
20    -- Function:
21    --   Defines a command to print from a file or other
22    --   byte stream source
23    --
24    -- History:
25    --   12-??-87, E. Sassone:  Initial Version
26    --   06-30-88, E. Sassone:  Working Version
27    --
28    -- Command Definition:
29    --   The command has the form:
30    --
31    --     print
32    --         [source=<pathname>]
33    --         [on=<pathname>]
34    --
35    --   The on argument can either be a spool queue or a
36    --   printer (for direct printing).  The default is a
37    --   system standard spooling device.  The source
38    --   argument will default to standard input.
39    --
40    --*D*   manage.commands
41    --*D*     create.invocation_command
42    --*D*
43    --*D*       define.argument source :type = string
44    --*D*           set.lexical_class symbolic_name
45    --*D*           set.maximum_length 80
46    --*D*           set.value_default ""
47    --*D*       end
48    --*D*
49    --*D*       define.argument on :type = string
50    --*D*         set.lexical_class symbolic_name
51    --*D*         set.maximum_length 80
52    --*D*         set.value_default ""
53    --*D*       end
54    --*D*     end
55    --*D*  exit
56
57
58    use System;
59
60    msg_obj:  constant System.untyped_word := System.null_word;
61
62    no_print_device_code:
63        constant Incident_Defs.incident_code := (
64            module         => 0,
65            number         => 1,
66            severity       => Incident_Defs.error,
67            message_object => msg_obj);
68
```

```
69    units_not_supported_code:
70        constant Incident_Defs.incident_code := (
71            module         => 0,
72            number         => 2,
73            severity       => Incident_Defs.error,
74            message_object => msg_obj);
75
76    --
77    --*D* manage.messages
78    --
79    no_print_device:  exception;
80    --*D* store :module=0 :number=1 \
81    --*D*      :msg_name=name_space_created_code \
82    --*D*      :short = \
83    --*D*      "Print Device $p1<on> does not exist."
84
85    --
86    --*D* store :module=0 :number=2 \
87    --*D*      :msg_name=units_not_supported_code \
88    --*D*      :short = \
89    --*D*      "Unit $p1<on> not supported."
90
91    opened_cmd:     Device_Defs.opened_device;
92      -- Opened command input device.
93
94    source:         System_Defs.text(80) := (80, 0, (others => ' '));
95      -- Pathname of file or device to print from
96
97    open_source:    Device_Defs.opened_device;
98      -- opened source file or input device
99
100   on_device:      System_Defs.text(Incident_Defs.txt_length) :=
101       (Incident_Defs.txt_length, 0, (others => ' '));
102     -- Pathname of spool queue or printer
103
104   spool_queue:    Device_Defs.device;
105
106   print_device:   Device_Defs.device;
107
108   sheet_size:     constant Spool_Defs.size_t := (80,60);
109     -- NOTE: Make sure this is within the capabilities
110     -- of your printer, otherwise the program will appear to
111     -- execute successfully but there will be no output.
112
113   open_print:     Device_Defs.opened_device;
114     -- opened print_device
115
116   -- buffer variables
117   buffer_size:    constant System.ordinal := 4_096;
118   buffer:         array(1 .. buffer_size) of
119       System.byte_ordinal;
120   bytes_read:     System.ordinal;
121
122 begin
123
124   -- Get command arguments:
125   --
126   opened_cmd := Command_Handler.Open_invocation_command_processing;
127   Command_Handler.Get_string(
128       cmd_odo    => opened_cmd,
129       arg_number => 1,
130       arg_value  => source);
131   Command_Handler.Get_string(
132       cmd_odo    => opened_cmd,
133       arg_number => 2,
134       arg_value  => on_device);
135   Command_Handler.Close(opened_cmd);
136
137   -- uses terminal input if no file specified
138   if source.length = 0 then
```

```
139       open_source :=
140           Process_Mgt.Get_process_globals_entry(
141           Process_Mgt_Types.standard_input);
142          -- standard input from terminal
143     else
144       open_source := Byte_Stream_AM.Open_by_name(
145           name         => source,
146           input_output => Device_Defs.input);
147     end if;
148
149     -- use default queue if not specified
150     if on_device.length = 0 then
151       Text_Mgt.Set(on_device,"/sys/spool_q");
152       -- Current name of default system spool queue
153     end if;
154
155     -- check the "on_device" for spooled or direct
156     -- printing, else error
157     spool_queue := Directory_Mgt.Retrieve(on_device);
158
159     if Spool_Defs.Is_spool_queue(spool_queue) then
160       -- spool file
161       print_device :=
162           Spool_Device_Mgt.Create_print_device(
163               spool_queue => spool_queue,
164               pixel_units => false,
165               print_area  => sheet_size);
166
167     elsif Spool_Defs.Is_print_device(spool_queue) then
168       -- direct printing
169       print_device :=
170           Spool_Device_Mgt.Create_print_device(
171               spool_queue => spool_queue,
172               pixel_units => false,
173               print_area  => sheet_size,
174               print_mode  => Spool_Defs.page_wise);
175
176     else
177       RAISE no_print_device;
178     end if;
179
180     open_print :=
181         Byte_Stream_AM.Ops.Open(
182             dev          => print_device,
183             input_output => Device_Defs.output,
184             allow        => Device_Defs.nothing);
185
186      -- read file in 4K chunks
187      while not Byte_Stream_AM.Ops.At_end_of_file(open_source)
188          loop
189        bytes_read := Byte_Stream_AM.Ops.Read(
190            opened_dev => open_source,
191            buffer_VA  => buffer'address,
192            length     => buffer_size);
193
194        Byte_Stream_AM.Ops.Write(
195            opened_dev => open_print,
196            buffer_VA  => buffer'address,
197            length     => bytes_read);
198      end loop;
199
200     Byte_Stream_AM.Ops.Close(open_source);
201     Byte_Stream_AM.Ops.Close(open_print);
202
203   exception
204
205     when no_print_device =>
206       Message_Services.Write_msg(
207           msg_id => no_print_device_code,
208           param1 => Incident_Defs.message_parameter'(
```

```
209             typ => Incident_Defs.txt,
210             len => on_device.max_length,
211             txt_val => on_device));
212
213   when Spool_Device_Mgt.units_not_supported =>
214      Message_Services.Write_msg(
215          msg_id => units_not_supported_code,
216          param1 => Incident_Defs.message_parameter'(
217               typ => Incident_Defs.txt,
218               len => on_device.max_length,
219               txt_val => on_device));
220
221   when Device_Defs.end_of_file =>
222       Byte_Stream_AM.Ops.Close(open_source);
223       Byte_Stream_AM.Ops.Close(open_print);
224
225 end Print_cmd_ex;
226
```

BiiN™/OS Reference Manual Release Information

This section describes detailed problems or limitations within this OS release. Workarounds are provided for some problems.

This section is organized by service area, service and Ada package. See Chapter 2 for a description of services and service areas. Only those service areas, services and packages that have caveats are listed.

Support Services

Message Service

See the Release Note for *BiiN™ Command and Message Guide.*

Object Service

1. `Passive_Store_Mgt.Copy` is not supported for directories. One effect of this limitation is that if you copy or move an executable program that is connected to command definitions, the connection is lost and must be reestablished. This is because the program's Outside Environment Object (OEO) is a standalone directory.
2. `Passive_Store_Mgt.Set_home_job` is not supported. Therefore, local single-activation type managers are not supported.
3. `Passive_Store_Mgt.Copy` started within a transaction doesn't return claimed disk space if `Abort` was sent from another job.

   This is unlikely to occur in most programming applications since the transaction and the copy occur in the same job. In some applications, a user may surround a copy with the CLEX commands `start.transaction` and `abort.transaction`. In this case, the transaction is started and aborted in the CLEX job while any file copies are performed by some other job. (Both the BiiN™/UX `cp` command and the CLEX `copy.object` utility use `Passive_Store_Mgt.Copy`.

   `Passive_Store_Mgt.Update_with_alternate_rep` does not work if invoked with a page alternate rep that contains one or more data-only pages. If this situation occurs, an appropriate message is pushed on the caller's message stack and "System_Exceptions.system_internal_error" is raised. No damage has been done and the system will continue to function normally.

Directory Services

Naming Service

1. A pathname, when expanded to a full pathname, cannot exceed 256 bytes.
2. Directories do not support logging. The `perform_logging` parameters to `Directory_Mgt.Create_directory` and `Standalone_Directory_Mgt.Create_directory` are ignored.
3. `Directory_Mgt.Get_name` does not read-lock any part of the resulting full pathname.
4. Directories (normal, active, or standalone) do not support the `Passive_Store_Mgt.Copy` call. Programs or scripts which have GCOS command files cannot be copied since the command file is stored in the Outside Environment Object which is a standalone directory. The program or script itself is copied, but `manage.commands` must be rerun on the program in the new location.

5. `Customized_Name_Mgt.Ops` and `Directory_Mgt` `Rename` and `Delete` operations on master entries may fail due to a timestamp conflict if the object in question has another alias on the same volume set and this alias is locked by a more recent transaction. In this case, the transaction enclosing the rename or delete needs to be aborted and the operation should be retried.
6. ID protection sets are currently limited to ten entries. This limits the number of users in a single group.

I/O Services

Basic I/O Service

1. Byte stream I/O to record-structured files is not supported.
2. Record I/O to stream files is not supported.

Character Terminal Service

1. A character window cannot be opened by more than 64 jobs.
2. Character terminal windows do not support record I/O.
3. The `Character_Display_AM.Ops.Ring_bell` call only supports audible alarms, regardless of the `audible` parameter's value. If the underlying device cannot produce an audible alarm, then `Ring_bell` does nothing.
4. The `Terminal_Defs.window_attr.track_cursor` window output control field is not supported. Even if this field is set to true, the view will not track the cursor.
5. The `Terminal_Info.Process_param_string` call does not support `%code if-then-else.`

Print Service and Spool Service

1. The `Printinfo` package, used to describe new printers, is not yet supported.
2. The print service does not provide any information about a printer's physical status (for example, offline, not ready, no paper). The user must check the printer for such problems.
3. Spooled data can be lost if a printer is switched off or disconnected during printing.
4. Removing a spool queue, other than with `remove.spool_queue` or `Spool_Device_Mgt.Delete_device`, will crash the spooling daemon requiring reinstallation of the spool service.
5. Invoke the `stop.pss` utility to safely shutdown spooling before rebooting the system. If the system is locked up so that this cannot be done, then rebooting crashes spooling, requiring reinstallation of the spool service. PSS may be crashed by a system *cold start* without previous shutdown. (The shutdown script includes stopping PSS.)
6. If a small amount of data (less than 1K bytes) is spooled into an empty spool queue, then the spool file immediately disappears from the queue, even though it is not printed or may not be printed at all (for example, due to an offline printer).
7. Print and Spool Services can service several spool queues each of which can only be connected to a single printer.

Print Priority Evaluation of Several Spool Queues:

If you have created several spool queues equipped with the same print priority be aware of this Spool Service behavior:

Current documentation states that in such a case, spool queues of the same priority are spooled out in the following manner:

first file from first queue,
first file from second queue, ...,
first file from n-th queue,
second file from first queue,
second file from second queue, ...,
second file from n-th queue,
third file from first queue, etc.

But Spool Service now spools out in the following manner:

first file from first queue,
second file from first queue, ...,
n-th file from first queue,
first file from second queue,
second file from second queue, ...,
m-th file from second queue, etc.

Spool Queue Print Delay:

Spool Service internally defines an interval of four hours during which spooling out requests are directed against spool files ready for printing.

Accordingly, a spool queue being equipped with a print delay of class *time* behaves as follows:

Beginning with the time of the day specified for the print time, Spool Service spools out all spool files ready for printing during four hours. After print time plus four hours, no new spooled in data are printed.

This also holds for a spool queue with a print delay of class *size*: If data are spooled in before the print time specified is reached, only those spool files smaller than the size limit are immediately printed. Four hours after the beginning of the print time, all spool files ready for printing are spooled out. After print time plus four hours, only those spool files smaller than the size limit are submitted to a printer.

Volume Space Exhaustion:

If during spooling the exception *volume_space_exhausted* is raised, the spool queue affected cannot be removed. The OS raises this exception even if the AD to the spool queue affected is simply retrieved. System Administrators should ensure that the volume set on which a spool queue is installed has enough blocks free for the spool files to be created.

Printer Error Handling:

In case of errors on the PT89 printer such as power failure, low paper, out of ink, or disconnecting of the cable, switch the printer off-line and on-line again. This causes the device driver to receive an XON character.

Printer Configuration Support:

Currently Printer Management does not support the `Detach` and `Stop` operations of the configuration attribute. So to deconfigure one or more printers, modify the System/User SCO and then perform a warmstart.

Native Mode Printing:

Applications doing native mode printing (spooled or directly) are constrained by the following limitations:

- Although on an "Open", Spool Service checks the print area size and position against the size of the currently mounted paper (exception `device_inoperative`), it is still possible to write on the cylinder in native mode.
- Linewrap and scroll and all other page output attributes are not evaluated in native mode.
- If a native mode printing application sets top of form (via an escape sequence), the printer device manager has no chance of resetting the correct top of form.

If you print characters in native mode, they get printed on the paper only if a `FF` or a `LF` or a `CR` is issued. This is a common printer property (although often not documented).

Authority List Protection of Spool Service Objects:

Spool Service supports authority list protection of the different Spool Service objects. The material presented in this section are more *usage suggestions* than *caveats*.

The application (utility) creating a spool queue should take into account that Spool Service protects the spool queue by the authority list found in the process globals of the application (utility) invoking the `Install` function of the `Spool_Queue_Admin` package. Accordingly, the spool queue creator should have registered all users wanting to make use of that spool queue with modify rights (for direct or spooled writing) and use rights (for inquiries) in the authority list of his process globals:

- Create an authority list via the `manage.authority` utility granting use and modify rights to all potential users of the spool queue.
- Register the authority list created in your process globals by the command `set.variable pglob.authority_list <new_authority_list> :global.`
- Alternatively, modify the spool queue's protecting authority list after you have created the spool queue. On spool queue installation, Spool Service creates a standalone directory within which you'll find an entry named `.<spool_queue_basename>_AL`. If you modify this authority list (e.g. by granting "world" for "um" access), the user(s) to register in this list with use and/or modify rights are allowed to deal with that spool queue. But beware that you do not remove "system" (or any of system's rights) from that authority list.
- Spool Service registers the process doing the install as "Spool Queue Administrator". The Spool Queue Administrator itself should make sure to have modify rights and use rights on the printer(s) to be connected with the spool queue.
- When a user having use rights on the spool queue calls `Spool_Queue_Admin.Get_rank_list`, he gets ADs with use rights of all spool files ranked. If Spool Service determines the caller to be either the Spool Queue Administrator or the owner of the spool file, that is, the application having created the spool file on an `Open` of a spooled print device, the corresponding spool file AD(s) additionally will get modify rights. According to the type rights necessary to delete a spool file the owner of the spool file or the Spool Queue Administrator can remove a spool file from the spool queue rank.

Interrelation of BiiN™/UX and Spool Service:

During the installation of Spool Service, `install.pss` activates a (revised) User SCO. Within the appropriate SCO description file there is a line that leads to the start of BiiN™/UX. Whether or not Spool Service successfully is booted in case that line is omitted is undetermined.

Spool Service Installation and User Access:

The user (ID and authority list) installing Spool Service by invoking `install.pss` represents "Spool Service". The user installing Spool Service must be granted access to create entries in the `/msg`, `/sys[/lib]`, and `/tdo` directories. Hence, it is strongly recommend that Spool Service be installed as `system`.

The `system` user should never remove the Spool Service configuration object stored under `/sys/spool`, otherwise Spool Service won't be usable. The same applies if the Spool Service ID (`system`) and/or all access rights are removed from public objects such as spool queue, printer, message file).

Print and Spool Service Interfaces: Following is a list of the Print and Spool Services external interfaces and AM support with comments on the current support.

Package `Spool_Defs`:

Due to missing support for printinfo and printer emulation, `Is_emulation` and `Is_printinfo` always return false.

Package `Spool_Device_Mgt`:

`Get_spool_device_attr_ID` returns a retyped instance of `Extra_Attributes.attribute_1`.

Package `Spool_Queue_Admin`:

Due to the lack printinfo support, `Install` does not evaluate a printinfo reference and `Get_printinfo` returns a `System.null_word`.

Package `Printer_Admin`:

`Set_printer_type` accepts only 'G' for a GENICOM printer or a 'P' for a PT89 printer.

Access Method Support for Spooled and Direct Printing:

Previous documentation stated that during spooling data into a spool file, Print and Spool Services destroy the spool file and close the print device when the exception `File_Defs.volume_space_exhausted` is raised. This functionality is not fully supported, that is, only the exception is propagated but the spool file is not destroyed nor is the print device closed.

The Character Display AM is not supported for native mode print devices in any print mode.

`Character_Display_AM.Ops.Set_enhancement` and `Character_Display_AM.Ops.Set_region_enhancement` are no-ops in both print modes supported.

Filing Service

1. For structured files, the *unordered* file organization is Recommended.
2. Disk space allocated to open temporary files is lost if the system crashes. Disk space allocated to open stream files may also be lost if the system crashes.
3. An unnamed file created within a transaction cannot be removed if the transaction times out.
4. Files cannot contain records longer than 4,000 bytes for unordered or relative files, nor records longer than 60,000 bytes for sequential files.
5. File buckets must be 4K bytes. The `bytes_per_bucket` field in the logical file descriptor, supplied when creating a file, is ignored.
6. Long-term file logging is not supported. The `File_Admin.logical_file_descr.long_term_logging` field is ignored.
7. File audit trails are not supported. The `audit_trail_file` parameter to `File_Admin.Create_file` is ignored.
8. If an opened structured file is destroyed in a transaction and the system crashes before the transaction is resolved, then broken file structures can result. To avoid this problem, don't use `Directory_Mgt.Delete` or `Passive_Store_Mgt.Destroy` to destroy structured files within transactions. Instead, use a `Destroy_file` call.
9. Index keys cannot contain long real fields. For string fields, the `t_block` type or the `t_string` type with or without the `pi_varying` property can be used. However, the `t_string` type with the `pi_header` property cannot be used. These types and properties are defined in the `Data_Definition_Mgt` package.
10. If a `Passive_Store_Mgt.Copy` operation on a file fails, then disk space allocated for the target file may not be reclaimed. Two cases in which disk space is not reclaimed are:
    - The destination volume set becomes full during the copy operation.
    - The copy operation is aborted because an enclosing transaction is aborted, and the job that aborts the transaction is not the same job that started the transaction. Specifically, avoid starting a transaction from the command line with `start.transaction` and then doing a copy operation within that transaction.
11. Records cannot contain multivalued fields.
12. There are three ways to insert records into a relative file: last (insert at EOF), first (use first available slot on the free list), and by a specific record number (by first using `Set_position` to select a record slot). The "first" and "specific record number" techniques cannot both be used with the same file. Mixing these two techniques will have undefined results.
13. The `Field_Access` package does not support conversion between base types.
14. `Field_Access` does not support initializing fields with default values.
15. `Field_Access` does not check constraints.
16. These `File_Admin` calls are not supported:

```
Assign_new_audit_trail_file
Deactivate_index
Get_file_status
Reorganize_file
Reorganize_index
```

17. The `File_Admin.Build_index` call requires its `file` parameter to be an *empty* and *nonopen* file. In other words, an application should build all indexes immediately after creating a file and before inserting any records into it.

18. The `File_Admin.Copy_file` call write-locks (exclusively locks) both the source and target files. `Copy_file` also ignores its `shrink` and `contiguous` boolean parameters, behaving as if both are false.

19. `File_Admin.Empty_file` cannot be called within a transaction. A workaround is to pop the transaction stack just before calling `Empty_file` and then push the popped transactions back onto the stack after the call.

20. `File_Admin.Get_index_status` does not report `num_free_buckets`.

21. Positioning of blocks in the reverse direction has not been fully tested for the `Join_Interface` package.

22. `Record_AM.Ops.Insert_control_record` does not raise `Device_Defs.length_error` when the record length is less than the minimum length or greater than the maximum length.

23. `Record_AM.Ops.Unlock` does not raise an exception if it is called for a non-existent record.

24. A second `Record_AM.Ops.Read` call to read the current record after a successful `Set_position` call raises `Record_AM.invalid_record_address`, if the file was created with `xm_locking` true.

25. After a `Record_AM.Ops.Set_position` call with an invalid record ID, a `Record_AM.Ops.Read` call to read the current record fails with an unspecified exception.

26. The `Sort_Merge_Interface.Special_collation_sort_merge` call only supports the `t_block` type defined by `Data_Definition_Mgt`.

27. `Record_AM.Truncate` only operates in the default mode: EOF is the beginning of the file and all records are removed. (`File_Admin.Empty_file` performs the same function.)

Data Definition Service

1. `Data_Definition_Mgt` does not support binding of message names to message identifiers.

2. `Data_Definition_Mgt` does not support binding of subprogram names to subprogram references.

Volume Set Service

1. Dismounting a volume set can cause a delayed system crash in some cases. Typically, the crash is caused b ecause Memory Management tries to page in a page of a partially-activated object that resides on the dismounted volume set. Since the page-in fails, Memory Management crashes the system. No data is corrupted and the system will operate normally when it is rebooted.

2. The system needs to be rebooted after one or more volume sets have been restored from backup tapes. If this is not done, some information on the newly restored volume sets may not be accessible.

3. **Don't use the system volume set for application files and objects.** Filling the system volume set and then crashing causes rebooting to fail, requiring a complete system rebuild. Another good reason to do all work on other volume sets is that it is impossible to backup the system volume set and later restore it.

4. Volume space allocated to temporary files is not reclaimed after a system crash and restart. Because of this, even the system administrator should avoid routinely logging in as `system`, because temporary files associated with CLEX will then be created on the system volume set, causing space to be lost if the system crashes and is restarted.
5. A volume cannot contain more than 128M bytes.
6. These `Volume_Set_Admin` calls are not supported:

```
Copy_volume_set
Empty_volume_set
Expand_volume_set
Move_volume_set
```

7. The `VSM_Disk_Admin.Rename_disk` call is not available.

Human Interface Services

Command Service

See the *Release Note for Gemini Command Language Executive Guide* and the *Release Note for Gemini Command and Message Guide*.

Form Service

See the *Release Note for Form Services*.

Program Services

Concurrent Programming Service

1. Since the system administrator cannot limit the number of concurrent jobs, a large number of jobs may consume all of virtual memory and the system may lock up. The OS code that is invoked to kill a job may be on disk and unable to be swapped in. The system must be rebooted.
2. The interactive attribute for pipes that are used in communications between EMACS and programs executing in shell windows is now supported. An additional boolean parameter, `is_interactive` has been added to `Pipe_Mgt.Create_pipe`. When set to true, the new pipe is interactive.

```
function Create_pipe(
    max_size:        System.ordinal := 0;
    DDef:            Data_Definition_Mgt.node_reference :=
                       Data_Definition_Mgt.null_node_reference;
    is_interactive:  boolean := false)
  return pipe_AD;
```

Program Building Service

1. It is possible to delete a program while it is executing but doing so may result in unpredictable behavior.

Type Manager Services

Configuration Service

1. OS type managers that support the configuration attribute often have incomplete implementations of that attribute. Many type managers don't fully support these `Configuration.Ops` calls:

```
Detach
Get_creation_parameters
Modify
Scan
Stop
```

Transport Service

1. The `Virtual_Circuit_AM` package does not support expedited data or negotiated connection establishment.

Device Services

Device Driver Service

1. The OS does not support IPI disk drives.
2. The OS driver for basic streamers does not retension the streamer tape. See the `manage.tape` utility description for information on how to retension the tape.

Hardware Interface Services

Hardware Service

1. Error reports logged in the SCT error log are not time-stamped.
2. `PS_Mgt.Retrieve_raw_PS` can be used to retrieve an uninterpreted image of the data in the EEPROM. All other calls in `PS_Mgt` are not supported.
3. `SCT_Access.Get_hw_info` returns a 0 or incorrect value for the starting address of noninterleaved memory.
4. `Test_Support.Test_BXU` only supports these subtests: `error_report`, FRC and parity.
5. `Test_Support.Test_CP` is not supported.
6. `Test_Support.Test_memory_controller` only uses `Test_BXU` to test the BXUs on BXU-based memory boards.

BiiN™/OS Reference Manual Caveats

This section lists documentation caveats for the *BiiN™/OS Reference Manual.* This manual is about 85% complete. Packages are listed in alphabetical order.

`Character_Display_AM`

Additional information about menus for character terminal windows includes:

- Up to sixteen menu groups can be associated with a window.
- A menu group can contain up to sixteen menus. However, the sums of the lengths of the menu titles plus five characters for each menu cannot exceed 80 characters. This ensures that the title bar can fit on one line.
- A menu group title bar is displayed in the first row of the terminal screen if the active window has an enabled menu group. The menu group title bar contains the title of each menu plus a letter that can be used to select the menu.
- The maximum number of menu items per menu is either 21 or the number of screen rows minus three, whichever is smaller.

- Menu item text length cannot exceed 65 characters.

`Command_Handler`

The `Trigger_reclamation` call requires a local AD with read and write rep rights for the countable object.

`Command_Handler.Get_line` raises `CL_Defs.illegal_syntax` (symbol not complete) rather than `Device_Defs.end_of_file` if a ^D is entered from the terminal.

`File_Admin`

A minor violation of Level 3 consistency is possible: if a process attempts to read a record that does not exist, and if the record is then inserted by a concurrent process, and then the first process attempts its read again, the inserted record is visible. This feature of Level 3 consistency occurs because the "slot" or "ID" of a nonexistent record is not locked if such a read is attempted.

`File_Admin.Save_unnamed_file` can raise `Passive_Store_Mgt.no_master_AD` exception in the following kind of sequence:

```
File_Admin.Create_unnamed_file
Transaction_Mgt.Start_transaction
File_Admin.Save_unnamed_file
Transaction_Mgt.Abort_transaction
Transaction_Mgt.Start_transaction
File_Admin.Save_unnamed_file
```

This happens because if an operation begins to store a master AD for an object but fails, then no master AD can ever be stored for the object. This happens if the master AD is first stored within a transaction that is aborted.

`Record_AM`

Many calls in this package can raise `ODO_using_different_transaction`, but this exception is not yet listed. There are some other errors in the lists of what exceptions can be raised.

`Lock_all` cannot raise `Device_Defs.device_in_use`.

The system-defined lock escalation counters are 124 locks for `Insert` and `Read`, and 248 for `Update` and `Delete`.

`Record_AM.Ops.Truncate` cannot be used to truncate a sequential file beginning at a particular record ID.

`TM_Transaction_Mgt.Transaction_Resolution`

The *Notes* section for `Commit_transaction` was truncated. This section should read:

A type manager should not release any locks or resources associated with a transaction until its `Commit_transaction` procedure is called.

During post-crash recovery, a type manager may be called to commit a transaction that it has no knowledge of. In such a case, `Commit_transaction` should return normally.

The type manager *must* be able to commit any prepared transaction.

This call should not raise *any* exceptions.

Appendix A

The last two steps in Section A.4 should be corrected to:

1. Make the appropriate directory one of the directories to be searched for include files by the `cg` command:
2.
```
clex-> set.variable cg.incldir ($cg.incldir /lib/kernel/include) \
   CONTINUE CMD: :global
```
3. Place this line in each C source file that uses the OS package:
```
#include <Access_mgt.h>
```

Additional BiiN™/OS Reference Manual Documentation

The following Hardware Interface Service and FTS packages are contained in V1.01.00 BiiN™ Operating System but are not in the 7/88 *BiiN™/OS Reference Manual*:

```
FT_Support
FT_Testing
KMDS_Defs
SCT_Access
SSM_Access
SSM_Defs
Test_Support
FTS_Admin
FTS_Config_Defs
FTS_Transfer
```

Not all of the procedures and functions in the Hardware Interface Service packages function as intended. A table accompanies each of the packages affected.

The *BiiN™/OS Reference Manual* chapters for these packages are attached to this release note. Please add them to your *BiiN™/OS Reference Manual*.

NOTE

`SSM_Defs` and `KMDS_Defs` are only used by privileged (trusted) users.

FT_Support

Procedure/Function	Implemented	Comments
Set_MC_toggle	yes	none.
Set_FRC_split	yes	none.
Set_transient_waiting_period	no	none.
Attach_bus	no	none.
Detach_bus	no	none.
Marry_processor_module	no	none.
Divorce_processor_module	no	none.

FT_Testing

Procedure/Function	Implemented	Comments
Enable_FRC_testing	yes	none.
Test_parity_and_BERL	yes	Due to BXU bug, this may cause a system crash.
Test_error_report	yes	Due to BXU bug, this may cause a system crash.

SCT_Access

Procedure/Function	Implemented	Comments
Retrieve_software_entry	yes	none.
Set_system_monitor_parameters	yes	none.
Retrieve_cardcage_entries	yes	none.
Retrieve_device_entry	yes	none.
Get_hardware_info	yes	Value of non-interleaved memory is wrong.
Get_error_log	yes	Errors are not time stamped.
Reserve_hw_entries	yes	none.
Release_hw_entries	yes	none.

SSM_Access

Procedure/Function	Implemented	Comments
Echo	yes	Single SSM only.
Read_revision	yes	Single SSM only.
Read_UID	yes	Single SSM only.
Read_TOD	yes	Single SSM only.
Read_SSM_Config	yes	Single SSM only.
Write_LED	yes	Single SSM only.
DC_Control	yes	Single SSM only.
Blower_control	yes	Single SSM only.
Read_error_log	yes	Single SSM only.
Read_SSM_inputs	yes	Single SSM only.
Send_to_MD	yes	Single SSM only.

Test_Support

Procedure/Function	Implemented	Comments
Test_GDP	yes	none.
Test_CP	no	Null procedure.
Test_BXU	yes	Only BCL tests from FT_Testing supported.
Test_private_memory	yes	Only with system in diagnostic mode.
Test_memory_controller	yes	BXU-based memory boards only.
Test_memory	yes	BXU-based memory boards only.
Set_board_LED	yes	none.
Set_diagnostic_mode	yes	none.
Set_normal_mode	yes	none.
Map_processor_ID_to_CP	yes	none.

The following Hardware Interface Service and FTS packages are attached. Please add them to your *BiiN™/OS Reference Manual.*

```
FT_Support
FT_Testing
KMDS_Defs
SCT_Access
SSM_Access
SSM_Defs
Test_Support
FTS_Admin
FTS_Config_Defs
FTS_Transfer
```

FT_Support

Provides support for managing Fault Tolerant (FT) hardware functions.

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Calls

`Attach_bus`
: Sends an *attach* bus command to an AP-bus agent.

`Detach_bus`
: Sends a *detach* bus command to the AP-bus agent.

`Divorce_processor_module`
: Divorces a married processor and updates the SCT.

`Marry_processor_module`
: Marries a shadow processor module to a primary processor module.

`Set_FRC_split`
: Sets the FRC SPLITTING ENABLE bit in the FRC Splitting Control (FSC) register of the AP Bus agents.

`Set_MC_toggle`
: Enables the TOGGLE MASTER CHECKER bit in the FRC register of the module's AP Bus agents.

`Set_transient_waiting_period`
: Sets the MAXTIME register of every AP Bus agent in a cardcage.

Summary

Each procedure in this package performs a hardware FT function on one or more modules in a cardcage. `FT_Support` provides the first level of abstraction away from the hardware level for FT operations. These routines manipulate fault tolerant hardware.

All of the procedures in this package automatically update the System Configuration Table (SCT) when necessary.

Exceptions

`not_FRCed`
: The system cannot perform an operation because one of the target modules is not running as an FRC module.

`module_is_QMRed`
: The system cannot perform an operation because one of the target modules is running as a QMR (*married*) modules.

`FT.Support.operation_failed`
: The system cannot complete an operation. This condition usually indicates that one of the components in a target module will not respond to an Inter-agent Command (IAC).

`cannot_be_married`
: The system cannot *marry* two modules. For example, a primary processor module that is running as the core module cannot *marry* a shadow processor module running as a noncore module.

`one_bus_system`
: Failed an attempt to perform a *detach* or *attach* bus operation in a single-bus system. A single-bus system only has one AP Bus per backplane.

Attach_bus

```
procedure Attach_bus(
    bus:       KMDS_Defs.one_bit_field;
    backplane: KMDS_Defs.cardcage_ID_rep := KMDS_Defs.sys);
pragma outerface(VALUE, Attach_bus);
```

Parameters

`bus`	Bus to be attached.
`backplane`	Cardcage location of the bus.

Operation

Sends an *attach* bus command to an AP-bus agent.

This procedure updates the System Configuration Table (SCT) and initiates an error report. The agent is specified by `bus` and `backplane`.

Notes

You can only use this procedure in 2-bus systems.

Exceptions

`one_bus_system`

`FT.Support.operation_failed`

`SCT_Access.not_in_SCT`

`SCT_Access.reserved_by_others`

Detach_bus

```
procedure Detach_bus(
    bus:        KMDS_Defs.one_bit_field;
    backplane:  KMDS_Defs.cardcage_ID_rep := KMDS_Defs.sys);
pragma outerface(VALUE, Detach_bus);
```

Parameters

`bus`	Name of Bus to detach.
`backplane`	Location of bus (system or extension cardcage).

Operation

Sends a *detach* bus command to the AP-bus agent.

The command deactivates the bus specified by `bus` and `backplane`. It updates the SCT and initiates an error report.

Notes

You can only use this procedure in a 2-bus system.

Exceptions

`one_bus_system`

`SCT_Access.not_in_SCT`

`SCT_Access.reserved_by_others`

Divorce_processor_module

```
procedure Divorce_processor_module(
    target:  KMDS_Defs.logical_ID_rep);
pragma outerface(VALUE, Divorce_processor_module);
```

Parameters

`target` ID of module to *divorce.*

Operation

Divorces a married processor and updates the SCT.

Divorcing implies the separation of one AP-bus system from another or the splitting of a two-bus system into a one-bus system.

Exceptions

`SCT_Access.not_in_SCT`

`SCT_Access.reserved_by_others`

`module_not_married`

Marry_processor_module

```
procedure Marry_processor_module(
    primary_module:  KMDS_Defs.logical_ID_rep;
    shadow_module:   KMDS_Defs.logical_ID_rep);
pragma outerface(VALUE, Marry_processor_module);
```

Parameters

`primary_module`
: ID of designated primary module.

`shadow_module`
: ID of designated shadow module.

Operation

Marries a shadow processor module to a primary processor module.

Marriage implies the union of one AP-bus system to another or the creation of a two-bus system from two one-bus systems. After the marriage, this command enables the TOGGLE PRIMARY SHADOW bit and updates the SCT.

Exceptions

`SCT_Access.not_in_SCT`

`SCT_Access.reserved_by_others`

`cannot_be_married`

Set_FRC_split

```
procedure Set_FRC_split(
    target:     KMDS_Defs.logical_ID_rep;
    backplane:  KMDS_Defs.cardcage_ID_rep := KMDS_Defs.sys;
    enable:     boolean := true);
pragma outerface(VALUE, Set_FRC_split);
```

Parameters

`target`	Logical ID of module to set master checker to toggle.
`backplane`	AP-Bus backplane target resident.
`enable`	If true, enable FRC splitting. If false, disable FRC splitting.

Operation

Sets the FRC SPLITTING ENABLE bit in the FRC Splitting Control (FSC) register of the AP Bus agents.

`target` designates the module.

Exceptions

`not_FRCed`

`module_is_QMRed`

`SCT_Access.not_in_SCT`

`SCT_Access.reserved_by_others`

Set_MC_toggle

```
procedure Set_MC_toggle(
    target:     KMDS_Defs.logical_ID_rep;
    backplane:  KMDS_Defs.cardcage_ID_rep := KMDS_Defs.sys;
    enable:     boolean := true);
pragma outerface(VALUE, Set_MC_toggle);
```

Parameters

`target`	Logical ID of module to set master checker to toggle.
`backplane`	AP-Bus backplane target resident.
`enable`	If true, enable MC toggle. If false, disable MC toggle.

Operation

Enables the TOGGLE MASTER CHECKER bit in the FRC register of the module's AP Bus agents.

`target` designates the module.

Exceptions

`module_is_QMRed`

`not_FRCed`

`SCT_Access.not_in_SCT`

`SCT_Access.reserved_by_others`

Set_transient_waiting_period

```
procedure Set_transient_waiting_period(
    max_time:   KMDS_Defs.four_bit_field;
    backplane:  KMDS_Defs.cardcage_ID_rep := KMDS_Defs.sys);
pragma outerface(VALUE, Set_transient_waiting_period);
```

Parameters

`max_time`	Timing value for MAXTIME register.
`backplane`	Cardcage in which to change MAXTIME registers.

Operation

Sets the MAXTIME register of every AP Bus agent in a cardcage.

The register is set to the value in `max_time`. This procedure does not write the MAXTIME TEST bit. As a result, this procedure does not allow you to test the MAXTIME counter. See the *Biin Hardware Reference Manual* for information about the MAXTIME counter.

Exceptions

`FT.Support.operation_failed`

`SCT_Access.not_in_SCT`

FT_Testing

Provides operations used for latent fault testing.

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Calls

`Enable_FRC_Testing`
: Sets the TESTING ENABLE bit in the test detection register.

`Test_error_report`
: Tests the priority circuits in the fault handling logic.

`Test_parity_and_BERL`
: Tests the parity checking logic of the AP bus.

Summary

This package provides operations used for latent fault testing. Fault tolerant operations must be aware of which CPUs are operating as a shadow/primary or master/checker CPU. They must also know on which AP bus the boards are operating and must satisfy certain test conditions.

Exceptions

`cannot_run_test`
: Cannot set up to run test.

Declarations

agent_locations

```
type agent_locations is(
    shdw_checker_bus0,
    shdw_checker_bus1,
    shdw_master_bus0,
    shdw_master_bus1,
    prim_checker_bus0,
    prim_checker_bus1,
    prim_master_bus0,
    prim_master_bus1);

  for agent_locations use (
    shdw_checker_bus0    => 2#000#,
    shdw_checker_bus1    => 2#001#,
    shdw_master_bus0     => 2#010#,
    shdw_master_bus1     => 2#011#,
    prim_checker_bus0    => 2#100#,
    prim_checker_bus1    => 2#101#,
    prim_master_bus0     => 2#110#,
    prim_master_bus1     => 2#111#);
```

Lists all of the possible logical locations of an MCU or a BXU AP-Bus Agent in a logical module. The following list interprets the bit positions for this representation:

bit 0	Indicates the AP-bus component.
bit 1	Indicates master or checker, 0 = checker, 1 = master.
bit 2	Indicates primary or shadow, 0 = shadow, 1 = primary.

Enumeration Literals:

`shdw_checker_bus0`
: Shadow Checker on AP-Bus 0.

`shdw_checker_bus1`
: Shadow Checker on AP-Bus 1.

`shdw_master_bus0`
: Shadow Master on AP-Bus 0.

`shdw_master_bus1`
: Shadow Master on AP-Bus 1.

`prim_checker_bus0`
: Primary Checker on AP-Bus 0.

`prim_checker_bus1`
: Primary Checker on AP-Bus 1.

`prim_master_bus0`
: Primary Master on AP-Bus 0.

`prim_master_bus1`
: Primary Master on AP-Bus 1.

agent_test_list

```
type agent_test_list is array (agent_locations) of boolean;
```

Physical components within a logical module to test.

test_all

```
test_all:  constant agent_test_list := (others => true);
```

Available AP-Bus agents to test.

agent_test_results

```
type agent_test_results is(
    not_run,
    passed,
    no_response,
    failed);

  for agent_test_results'size use System.storage_unit;
```

Possible test results for each physical component.

Enumeration Literals:

`not_run`	Test was not run on component.
`passed`	Component passed test.
`no_response`	Component did not respond to IAC which initiated test.
`failed`	Component failed the test.

test_results

```
type test_results is array(agent_locations) of agent_test_results;

  pragma pack(test_results);
```

Enable_FRC_Testing

```
procedure Enable_FRC_Testing(
    module_ID:  KMDS_Defs.logical_ID_rep);
pragma outerface(VALUE, Enable_FRC_Testing);
```

Parameters

`module_ID` Logical ID of module on which to enable testing.

Operation

Sets the TESTING ENABLE bit in the test detection register.

The FRC circuits are self-checking whenever this bit is set. There is no need for any special test sequences to check their operation. Once enabled, the FRC circuits continue to check themselves until an error report turns off the TESTING ENABLE bit. Any error report turns off the TESTING ENABLE bit and disables FRC testing. See the *BiiN Hardware Reference Manual* for information about the test detection register.

Notes

This procedure modifies the contents of the COM Register. The FRC circuits use the value in this register when they check themselves. See the system monitor information in the *BiiN System Administrator's Guide* for COM register information.

Exceptions

`KMDS_Defs.unresponsive_target`

`FT_Support.operation_failed`

Test_error_report

```
procedure Test_error_report(
    module_ID:          KMDS_Defs.logical_ID_rep;
    passed:  out        boolean;
    agent_results: out test_results;
    agents_to_test:     agent_test_list := test_all);
pragma outerface(VALUE, Test_error_report);
```

Parameters

`module_ID` Address of target AP-Bus agent(s).

`passed` If true, all agents passed test.

`agent_results`
Test results for each physical component.

`agents_to_test`
List of agents to be tested.

Operation

Tests the priority circuits in the fault handling logic.

In addition, it corrupts the parity comparison in the long form error report receiver, which results in an Error Reporting error. It corrupts the comparison of the two messages sent within an error report, which also results in an Error Reporting error. See the *BiiN Diagnostic User's Guide* for information about error reporting. The returned parameter is true if the test succeeds, and false if the test fails. You can only use this test for testing BXUs and MCUs.

Exceptions

`KMDS_Defs.unresponsive_target`

`FT_Support.operation_failed`

Test_parity_and_BERL

```
procedure Test_parity_and_BERL(
    module_ID:          KMDS_Defs.logical_ID_rep;
    passed:  out        boolean;
    agent_results: out test_results;
    agents_to_test:     agent_test_list := test_all);
pragma outerface(VALUE, Test_parity_and_BERL);
```

Parameters

`module_ID` Address of target AP-Bus agent(s).

`passed` If true, all agents passed test.

`agent_results`
Test results for each physical component.

`agents_to_test`
List of agents to be tested.

Operation

Tests the parity checking logic of the AP bus.

The package tests both parity trees of the parity checking logic of the AP bus. This operation also tests BXUs and MCUs. Since this test causes error reports, it also checks the functionality of the `BERL`. If the returned parameter is true, the test was successful; false if the test fails.

Exceptions

`KMDS_Defs.unresponsive_target`

`FT_Support.operation_failed`

KMDS_Defs

Contains basic definitions that the System Monitor, the Secondary Bootstrap Loader, and the Operating System need to operate.

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Summary

This package contains basic definitions used by the System Monitor, the Secondary Bootstrap Loader, and the Operating System. Changes to the types defined in this package may affect any or all of the systems listed above.

Declarations

string_range

```
subtype string_range is System.ordinal range 2 .. 255;
```

In general, using Ada strings causes the compiler to allocate the data from heap (= object allocation), requiring system software support. Hence, the System Monitor has to define its own `string` type in order to make sure all allocations can be done on the stack. Note, the discriminant in fix_length_string is constrained, hence the compiler does not need to worry about worst case allocation (<>).

fix_length_string

```
type fix_length_string(
    len: string_range) is
  record
    val:  SCT_Types.char_array(1 .. len);
  end record;
```

fix_string_VA

```
type fix_string_VA is access fix_length_string;

  pragma access_kind(fix_string_VA, virtual);

  for fix_length_string use
    record
      len             at 0 range 0 .. 31;
    end record;
```

fix_4_string

```
subtype fix_4_string   is fix_length_string(4);
```

fix_8_string

```
subtype fix_8_string   is fix_length_string(8);
```

fix_16_string

```
subtype fix_16_string  is fix_length_string(16);
```

fix_32_string

```
subtype fix_32_string  is fix_length_string(32);
```

fix_64_string

```
subtype fix_64_string  is fix_length_string(64);
```

one_bit_field

```
subtype one_bit_field is System.ordinal range 0 .. 16#1#;
```

1 bit field.

two_bit_field

```
subtype two_bit_field is System.ordinal range 0 .. 16#3#;
```

2 bit field.

three_bit_field

```
subtype three_bit_field is System.ordinal range 0 .. 16#7#;
```

3 bit field.

four_bit_field

```
subtype four_bit_field is System.ordinal range 0 .. 16#F#;
```

4 bit field.

five_bit_field

```
subtype five_bit_field is System.ordinal range 0 .. 16#1F#;
```

5 bit field.

six_bit_field

```
subtype six_bit_field is System.ordinal range 0 .. 16#3F#;
```

6 bit field.

seven_bit_field

```
subtype seven_bit_field is System.ordinal range 0 .. 16#7F#;
```

7 bit field.

nine_bit_field

```
subtype nine_bit_field is System.ordinal range 0 .. 16#1FF#;
```

9 bit field.

ten_bit_field

```
subtype ten_bit_field is System.ordinal range 0 .. 16#3FF#;
```

10 bit field.

eleven_bit_field

```
subtype eleven_bit_field is System.ordinal range 0 .. 16#7FF#;
```

11 bit field.

twelve_bit_field

```
subtype twelve_bit_field is System.ordinal range 0 .. 16#FFF#;
```

12 bit field.

thirteen_bit_field

```
subtype thirteen_bit_field is System.ordinal range 0 .. 16#1FFF#;
```

13 bit field.

fourteen_bit_field

```
subtype fourteen_bit_field is System.ordinal range 0 .. 16#3FFF#;
```

14 bit field.

fifteen_bit_field

```
subtype fifteen_bit_field is System.ordinal range 0 .. 16#7FFF#;
```

15 bit field.

seventeen_bit_field

```
subtype seventeen_bit_field is System.ordinal range 0 .. 16#1FFFF#;
```

17 bit field.

eighteen_bit_field

```
subtype eighteen_bit_field is System.ordinal range 0 .. 16#3FFFF#;
```

18 bit field.

nineteen_bit_field

```
subtype nineteen_bit_field is System.ordinal range 0 .. 16#7FFFF#;
```

19 bit field.

twenty_bit_field

```
subtype twenty_bit_field is System.ordinal range 0 .. 16#FFFFF#;
```

20 bit field.

twenty_one_bit_field

```
subtype twenty_one_bit_field is System.ordinal range 0 .. 16#1FFFFF#;
```

21 bit field.

twenty_two_bit_field

```
subtype twenty_two_bit_field is System.ordinal range 0 .. 16#3FFFFF#;
```

22 bit field.

twenty_four_bit_field

```
subtype twenty_four_bit_field is System.ordinal range 0 .. 16#FFFFFF#;
```

24 bit field.

raw_4_bytes

```
type raw_4_bytes is array(
    System.ordinal range 0 .. 3) of System.byte_ordinal;

pragma pack(raw_4_bytes);
```

raw_6_bytes

```
type raw_6_bytes is array(
    System.ordinal range 0 .. 5) of System.byte_ordinal;

pragma pack(raw_6_bytes);
```

raw_8_bytes

```
type raw_8_bytes is array(
    System.ordinal range 0 .. 7) of System.byte_ordinal;

pragma pack(raw_8_bytes);
```

raw_16_bytes

```
type raw_16_bytes is array(
    System.ordinal range 0 .. 15) of System.byte_ordinal;

pragma pack(raw_16_bytes);
```

raw_32_bytes

```
type raw_32_bytes is array(
    System.ordinal range 0 .. 31) of System.byte_ordinal;

pragma pack(raw_32_bytes);
```

raw_64_bytes

```
type raw_64_bytes is array(
    System.ordinal range 0 .. 63) of System.byte_ordinal;

pragma pack(raw_64_bytes);
```

zero_to_6

```
subtype zero_to_6  is System.ordinal range 0 .. 6;
```

zero_to_23

```
subtype zero_to_23 is System.ordinal range 0 .. 23;
```

zero_to_59

```
subtype zero_to_59 is System.ordinal range 0 .. 59;
```

zero_to_99

```
subtype zero_to_99 is System.ordinal range 0 .. 99;
```

one_to_31

```
subtype one_to_31  is System.ordinal range 1 .. 31;
```

one_to_12

```
subtype one_to_12  is System.ordinal range 1 .. 12;
```

TOD

```
type TOD is
  record
    hundredths_sec:           zero_to_99;
    seconds:                  zero_to_59;
    minutes:                  zero_to_59;
    hours:                    zero_to_23;
    day_of_week:              zero_to_6;
    date:                     one_to_31;
    month:                    one_to_12;
    year:                     zero_to_99;
  end record;

  for TOD use
    record
      hundredths_sec  at 0 range 0 .. 6;
      seconds         at 1 range 0 .. 6;
      minutes         at 2 range 0 .. 5;
      hours           at 3 range 0 .. 4;
      day_of_week     at 4 range 0 .. 2;
      date            at 5 range 0 .. 4;
      month           at 6 range 0 .. 3;
      year            at 7 range 0 .. 6;
    end record;
```

Defines the system's TOD counter (in hundredths of a second).

max_error_index

```
max_error_index:  constant := 27;
```

Maximum error record supported (range 0 .. max_error_index).

max_ext_box

```
max_ext_box:      constant System.ordinal := 7;
```

Maximum I/O extension box number (range 0 .. max_ext_box).

max_device_index

```
max_device_index: constant System.ordinal := 7;
```

Maximum device number supported by the System Monitor. (range 0 .. max_device_index).

max_mem_rec

```
max_mem_rec:      constant System.ordinal := 7;
```

Maximum record number of a boot or dump image.

max_slot_number

```
max_slot_number:  constant System.ordinal := 13;
```

Maximum possible number of slots in a cardcage.

hw_header_size

```
hw_header_size:   constant := 8;
```

Size of header portion of `hw_entry_rep`.

hw_body_size

```
hw_body_size:     constant := SCT_Types.hw_entry_size - hw_header_size;
```

Size of body portion of `hw_entry_rep`.

cardcage_ID_rep

```
type cardcage_ID_rep is (sys, ext);
```

slot_number

```
subtype slot_number is System.ordinal range 0 .. max_slot_number;
```

Although slot 0 represents an invalid number, it is a convenient number to define as a default value.

module_ID_rep

```
type module_ID_rep is
  record
    slot:        slot_number;
    don_t_care:  two_bit_field;
    cardcage:    cardcage_ID_rep;
  end record;

  for module_ID_rep use
    record
      slot           at 0 range 0 .. 3;
      don_t_care     at 0 range 4 .. 5;
      cardcage       at 0 range 6 .. 7;
    end record;
```

Fields:

slot

don_t_care Must be initialized to zero.

cardcage

null_module_ID

```
null_module_ID:  constant module_ID_rep := (cardcage   => sys,
                                            don_t_care => 0,
                                            slot       => 0);
```

AP_bus_reg

```
subtype AP_bus_reg is System.ordinal;
```

Defines an arbitrary AP-Bus register.

This type normally represents all AP-Bus registers (in the System Monitor and SCT). This type will be retyped to the desired register type only if you need to examine the contents of a particular register (e.g. physical ID).

position

```
subtype position is System.ordinal;
```

position (sector #) on device

physical_addr

```
subtype physical_addr is System.ordinal;
```

Physical memory address.

memory_descr

```
type memory_descr is
  record
    addr:    physical_addr;
    length:  System.ordinal;
  end record;

  for memory_descr use
    record
      addr      at  0 range 0 .. 31;
      length    at  4 range 0 .. 31;
    end record;
```

Fields:

addr	Start address in memory.
length	Length of record in bytes.

mem_rec

```
type mem_rec is array(0 .. max_mem_rec) of memory_descr;
```

Global types related to I/O devices.

IO_device_ID

```
type IO_device_ID is
  record
    io_addr:             System.short_ordinal;
    application_index:   four_bit_field;
    don_t_care:          three_bit_field;
    processor:           one_bit_field;
    module_ID:           module_ID_rep;
  end record;

  for IO_device_ID use
    record
      io_addr             at 0 range  0 .. 15;
      application_index   at 0 range 16 .. 19;
      don_t_care          at 0 range 20 .. 22;
      processor           at 0 range 23 .. 23;
      module_ID           at 0 range 24 .. 31;
    end record;
```

Fields:

`io_addr` Controller address.

`application_index`
Index for identifying a CP application.

`don_t_care` Must be initialized to zero.

`processor` Local processor ID.

`module_ID` System bus number and slot number.

device_number

```
subtype device_number is System.ordinal range 0 .. max_device_index;
```

device_list

```
type device_list is array(device_number) of IO_device_ID;
```

null_device

```
null_device:  constant IO_device_ID := (module_id         => null_module_ID,
                                        processor         => 0,
                                        don_t_care        => 0,
                                        application_index => 0,
                                        io_addr           => 16#0000#);
```

hw_entry_type

```
type hw_entry_type is (
             free,
             cardcage,
             module,
             device);

  for hw_entry_type'size use System.storage_unit;
```

Types of SCT elements.

Enumeration Literals:

`free` Free element, currently not used.

`cardcage` Cardcage entry.

`module` Module (board) entry.

`device` Device (boot/console) entry.

functional_status

```
type functional_status is (
              not_used,
              online,
              offline,
              faulty,
              offline_or_faulty);

  for functional_status'size use System.storage_unit;

  for functional_status use(
              not_used                => 0,
              online                  => 1,
              offline                 => 2,
              faulty                  => 3,
              offline_or_faulty       => 4);
```

Possible status of hardware configuration parts.

Enumeration Literals:

`not_used`	Slot is currently not used.
`online`	Module is available for use.
`offline`	Module is not available because it is reserved for use by maintenance software.
`faulty`	Module is not available because it is faulty.
`offline_or_faulty`	Module is not available and system cannot determine why.

component_class

```
type component_class is (BXU, MCU);
```

physical_ID_rep

```
type physical_ID_rep is
  record
    class:      component_class;
    component:  five_bit_field;
  end record;

  for physical_ID_rep use
    record
      class                   at 0 range 5 .. 5;
      component               at 0 range 0 .. 4;
    end record;
```

logical_ID_rep

```
subtype logical_ID_rep is six_bit_field;
```

AP-bus logical ID.

psor_select

```
type psor_select is (psor_0, psor_1);

  for psor_select use(
      psor_0 => 0,
      psor_1 => 1);
```

processor_ID

```
type processor_ID is
  record
    unit:  logical_ID_rep;
    psor:  psor_select;
    zero:  one_bit_field;
  end record;

  for processor_ID use
    record
      zero    at 0 range 0 .. 0;
      psor    at 0 range 1 .. 1;
      unit    at 0 range 2 .. 7;
    end record;
```

null_psor_ID

```
null_psor_ID:  constant processor_ID := (psor => psor_0,
                                         unit => 0,
                                         zero => 0);
```

arb_ID_rep

```
type arb_ID_rep is
  record
    cycle:    four_bit_field;
    drive:    two_bit_field;
  end record;

  for arb_ID_rep use
    record
      cycle   at 0    range 0 .. 3;
      drive   at 0    range 4 .. 5;
    end record;
```

AP-bus arbitration ID.

invalid_arb_ID

```
invalid_arb_ID:  constant arb_ID_rep := (cycle => 16#F#,
                                         drive => 16#3#);
```

Invalid arbitration ID value, usually given to passive modules.

error_record

```
type error_record is
  record
    error_type:     System.short_ordinal;
    time_stamp:     raw_6_bytes;
    unit_name:      fix_16_string;
    ord_param1:     System.ordinal;
    ord_param2:     System.ordinal;
  end record;

  for error_record use
    record
      error_type    at  0 range 0 .. 15;
      time_stamp    at  2 range 0 .. (6 * 8) - 1;
      ord_param1    at  8 range 0 .. 31;
      ord_param2    at 12 range 0 .. 31;
      unit_name     at 16 range 0 .. (16 + 4) * 8 - 1;
    end record;
```

Fields:

`error_type`	Type of error being recorded.
`time_stamp`	Time when error was logged.
`unit_name`	Name of module which is associated with the error.
`ord_param1`	Parameter #1, meaning defined by error_type.
`ord_param2`	Parameter #2, meaning defined by error_type.

error_record_size

```
error_record_size:  constant := 36;
```

Size of an error record in bytes.

error_log_index

```
subtype error_log_index is System.ordinal range 0 .. max_error_index;
```

Index for entries in SCT's error log.

error_log_rep

```
type error_log_rep is array(
  SCT Error log represention.

  This error log is a circular buffer.  The
  most recent entry in this buffer must be indicated by an index of
  type error_log_index.
ror_log_index) of error_record;

  pragma pack(error_log_rep);
```

error_log_size

```
error_log_size:  constant := error_record_size * (max_error_index + 1);
```

Size of SCT error log in bytes.

error_log_record

```
type error_log_record is
  record
    error_count:  System.ordinal;
    count_time:  raw_6_bytes;
    last_error:  error_log_index;
    error_log:   error_log_rep;
  end record;

  for error_log_record use
    record
      error_count       at  0 range 0 .. 31;
      count_time        at  4 range 0 .. (8 * 6) - 1;
      last_error        at 12 range 0 .. 31;
      error_log         at 16 range 0 .. error_log_size * 8 - 1;
    end record;

  for error_log_record'size use SCT_Types.error_log_size * 8;
```

Error Log Record. Used to overlay area in SCT object reserved for error log.

Fields:

`error_count`	Total number of errors reported, since `count_time`.
`count_time`	Time when `error_count` was last reset to 0.
`last_error`	Pointer to last entry put into the log.
`error_log`	Circular buffer used to record error reports.

error_log_VA

```
type error_log_VA is access error_log_record;

  pragma access_kind(error_log_VA, virtual);
```

ext_box_rep

```
type ext_box_rep is array(System.ordinal range 0 .. max_ext_box) of boolean;

  pragma pack(ext_box_rep);
```

Array of flags. If true, extension box is switched on. The list defines one entry for each I/O extension box.

addr_recognizer

```
type addr_recognizer is
  record
    mask:   AP_bus_reg;
    match:  AP_bus_reg;
  end record;

  for addr_recognizer use
    record
      mask      at  0 range 0 .. 31;
      match     at  4 range 0 .. 31;
    end record;
```

Represents address recognizers as defined for the different AP bus agents.

Every BXU has four address recognizers on its local bus interface and one on its AP bus interface. The MCU supports one address recognizer. Space for the other address recognizers is not used for MCUs.

Fields:

`mask`	Memory window's size.
`match`	Base address of memory window.

addr_recognizer_set

```
type addr_recognizer_set is array(0 .. 4) of addr_recognizer;
```

null_addr_rec

```
null_addr_rec:  constant addr_recognizer_set := (
                    others          => (0, 0));
```

Null values for `addr_recognizers`.

VLSI_locations

```
type VLSI_locations is(
      bus0,
      bus1,
      psor0,
      psor1);
```

Enumeration Literals:

bus0	Component is connected to AP-Bus 0.
bus1	Component is connected to AP_Bus 1.
psor0	Component is processor 0 on the local bus.
psor1	Component is processor 1 on the local bus.

VLSI_status

```
type VLSI_status is array(VLSI_locations) of functional_status;
```

VLSI_desc

```
type VLSI_desc is
  record
    physical_ID:    physical_ID_rep;
    logical_ID:     logical_ID_rep;
    arb_ID:         arb_ID_rep;
    status:         VLSI_status;
  end record;

  for VLSI_desc use
    record
      physical_ID   at  0 range 0 .. 7;
      logical_ID    at  1 range 0 .. 7;
      arb_ID        at  2 range 0 .. 7;
      status        at  4 range 0 .. 31;
    end record;

  for VLSI_desc'size use 8*8;
```

null_VLSI_desc

```
null_VLSI_desc:  constant VLSI_desc := (
                    physical_ID     => (BXU, 0),
                    logical_ID      => 0,
                    arb_ID          => invalid_arb_ID,
                    status          => (others => not_used));
```

component_flags

```
type component_flags is array (System.ordinal range 0 .. 3) of boolean;

  pragma pack(component_flags);
```

Component flag array. Each position in this array corresponds to a component position on a board.

memory_types

```
type memory_types is(
    not_available,
    DRAM,
    SRAM,
    PROM);

  for memory_types use(
      not_available => 0,
      DRAM          => 1,
      SRAM          => 2,
      PROM          => 3);
```

Types of memory modules.

Enumeration Literals:

`not_available`
: Memory module is not available.

DRAM
: Memory module contains dynamic RAM.

SRAM
: Memory module contains static RAM.

PROM
: Memory module contains ROM.

memory_desc

```
type memory_desc is
  record
    mem_type:  memory_types;
    index:     four_bit_field;
  end record;

  for memory_desc use
    record
      index           at 0    range 0 .. 3;
      mem_type        at 0    range 4 .. 5;
    end record;
```

Breakdown of memory module fields in COM Word 1.

Fields:

mem_type	Type of memory in this module.
index	Index in system memory module sizes table.

base_boards

```
type base_boards is(
    GS2_psor,
    gen_IO,
    BXU_mem,
    MCU_mem,
    free_4,
    free_5,
    free_6,
    free_7,
    GS1_FRC,
    GS1_SBC,
    free_10,
    free_11,
    free_12,
    free_13,
    free_14,
    GS2_probe);

  for base_boards use(
    GS2_psor          => 2#0000#,
    gen_IO            => 2#0001#,
    BXU_mem           => 2#0010#,
    MCU_mem           => 2#0011#,
    free_4            => 2#0100#,
    free_5            => 2#0101#,
    free_6            => 2#0110#,
    free_7            => 2#0111#,
    GS1_FRC           => 2#1000#,
    GS1_SBC           => 2#1001#,
    free_10           => 2#1010#,
    free_11           => 2#1011#,
    free_12           => 2#1100#,
    free_13           => 2#1101#,
    free_14           => 2#1110#,
    GS2_probe         => 2#1111#);
```

Known AP-bus base board types.

IOPMs

```
type IOPMs is(
    no_PM,
    low_speed_on_board,
    E_bus,
    LAN_PM,
    HDLC,
    SCSI,
    IPI_master,
    IPI_slave,
    multibus_II_adapter,
    high_speed_on_board);

  for IOPMs use(
```

```
    no_PM                   => 0,
    low_speed_on_board      => 1,
    E_bus                   => 2,
    LAN_PM                  => 3,
    HDLC                    => 4,
    SCSI                    => 5,
    IPI_master              => 6,
    IPI_slave               => 7,
    multibus_II_adapter     => 8,
    high_speed_on_board     => 9);
```

Known types of IO Personality Modules (IOPMs).

Enumeration Literals:

`no_PM` IOPM not available.

`low_speed_on_board`

`E_bus`

`LAN_PM`

`HDLC`

`SCSI`

`IPI_master`

`IPI_slave`

`multibus_II_adapter`

`high_speed_on_board`

IODMs

```
type IODMs is(
    no_DM,
    digital_modem,
    analog_modem);

  for IODMs use(
    no_DM             => 0,
    digital_modem     => 1,
    analog_modem      => 2);
```

Known types of IO Distribution Modules (IODMs).

Enumeration Literals:

`no_DM` IODM not installed.

`digital_modem`

`analog_modem`

com_word

```
type com_word (number: two_bit_field := 0) is
  record
    parity:                    boolean;
    case number is
      when 0 =>
        core:                  boolean;
        core_enabled:          boolean;
        two_buses:             boolean;
        CP_fail:               component_flags;
        GDP_fail:              component_flags;
        init_count:            two_bit_field;
        AP_agents:             component_flags;
        CPs:                   component_flags;
        GDPs:                  component_flags;
        board_type:            base_boards;
      when 1 =>
        artwork_rev:           three_bit_field;
        assembly_rev:          four_bit_field;
        buffered:              boolean;
        memory_interleave:     two_bit_field;
        memory_initialized:    boolean;
        on_board_mem:          memory_desc;
        mem_mod_B:             memory_desc;
        mem_mod_A:             memory_desc;
      when 2 =>
        reserved_2:            nine_bit_field;
        IOPM_3:                IOPMs;
        IOPM_2:                IOPMs;
        IOPM_1:                IOPMs;
        IOPM_0:                IOPMs;
        IODM:                  IODMs;
      when 3 =>
        uC_firmware_rev:       System.byte_ordinal;
        company:               one_bit_field;
        overflow_rev:          System.byte_ordinal;
        reserved_3:            twelve_bit_field;
    end case;
  end record;

  pragma suppress(discriminant_check,com_word);

  for com_word use
    record
      number                   at 0    range 30 .. 31;
      parity                   at 0    range  0 .. 0;
      core                     at 0    range  1 .. 1;
      core_enabled             at 0    range  2 .. 2;
      two_buses                at 0    range  3 .. 3;
      CP_fail                  at 0    range  4 .. 7;
      GDP_fail                 at 0    range  8 .. 11;
      init_count               at 0    range 12 .. 13;
      AP_agents                at 0    range 14 .. 17;
      CPs                      at 0    range 18 .. 21;
      GDPs                     at 0    range 22 .. 25;
      board_type               at 0    range 26 .. 29;
      artwork_rev              at 0    range  1 .. 3;
      assembly_rev             at 0    range  4 .. 7;
      buffered                 at 0    range  8 .. 8;
      memory_interleave        at 0    range  9 .. 10;
      memory_initialized       at 0    range 11 .. 11;
      on_board_mem             at 0    range 12 .. 17;
      mem_mod_B                at 0    range 18 .. 23;
      mem_mod_A                at 0    range 24 .. 29;
      reserved_2               at 0    range  1 .. 9;
      IOPM_3                   at 0    range 10 .. 13;
      IOPM_2                   at 0    range 14 .. 17;
```

```
    IOPM_1                  at 0    range 18 .. 21;
    IOPM_0                  at 0    range 22 .. 25;
    IODM                    at 0    range 26 .. 29;
    uC_firmware_rev         at 0    range  1 .. 8;
    company                 at 0    range  9 .. 9;
    overflow_rev            at 0    range 10 .. 17;
    reserved_3              at 0    range 18 .. 29;
  end record;
```

Fields:

number	Format of COM Words.
parity	Odd parity bit for all COM words.
core	
core_enabled	
two_buses	
CP_fail	
GDP_fail	
init_count	
AP_agents	
CPs	
GDPs	
board_type	
artwork_rev	
assembly_rev	
buffered	
memory_interleave	
memory_initialized	
on_board_mem	
mem_mod_B	
mem_mod_A	
reserved_2	
IOPM_3	
IOPM_2	
IOPM_1	
IOPM_0	
IODM	
uC_firmware_rev	
company	
overflow_rev	
reserved_3	

invalid_com_word

```
invalid_com_word:  constant com_word(3) := (
  Invalid value for COM word.
                     number          => 3,
                     reserved_3      => 0,
                     uC_firmware_rev => 0,
                     company         => 0,
                     overflow_rev    => 0,
                     parity          => false);
```

env_status_bits

```
type env_status_bits is (
    PSa0_installed,
    PSa1_installed,
    PSb0_installed,
    PSb1_installed,
    PSa0_5v_failed,
    PSa0_12v_failed,
    PSa1_5v_failed,
    PSa1_12v_failed,
    PSb0_failed,
    PSb1_failed,
    buf_5v_failed,
    other_AC_down,
    UPS_on,
    sys_batt_fault,
    extrn_batt_fault,
    PS_temp_fault,
    cage_temp_fault,
    periph_temp_fault,
    air_intake_fault,
    blower_fault,
    box0_fault,
    box1_fault,
    box2_fault,
    box3_fault,
    reserved_1,
    reserved_2,
    reserved_3,
    reserved_4,
    reserved_5,
    reserved_6,
    reserved_7,
    reserved_8);
```

Enumeration Literals:

`PSa0_installed`
: Power Supply A:0 installed.

`PSa1_installed`
: Power Supply A:1 installed.

`PSb0_installed`
: Power Supply B:0 installed.

`PSb1_installed`
: Power Supply B:1 installed.

`PSa0_5v_failed`
: Power Supply A:0 5 volt failed.

`PSa0_12v_failed`
: Power Supply A:0 12 volt failed.

`PSa1_5v_failed`
: Power Supply A:1 5 volt failed.

`PSa1_12v_failed`
: Power Supply A:1 12 volt failed.

`PSb0_failed` Power Supply B:0 failed.

`PSb1_failed` Power Supply B:1 failed.

`buf_5v_failed`
: Buffered 5 volt supply failed.

`other_AC_down`
: AC feed to other Power Supply C down.

`UPS_on` Uninterruptable Power Supply on.

`sys_batt_fault`
: Internal system battery failed.

`extrn_batt_fault`
: External battery failed.

`PS_temp_fault`
: Temperature fault in power supply.

`cage_temp_fault`
: Temperature fault in cardcage.

`periph_temp_fault`
: Temperature fault in peripheral area.

`air_intake_fault`
: Temperature fault in air intake.

`blower_fault` Blower fault.

`box0_fault` Fault signalled by Extension Box 0.

`box1_fault` Fault signalled by Extension Box 1.

`box2_fault` Fault signalled by Extension Box 2.

`box3_fault` Fault signalled by Extension Box 3.

`reserved_1` reserved for future expansion.

`reserved_2` reserved for future expansion.

`reserved_3` reserved for future expansion.

`reserved_4` reserved for future expansion.

`reserved_5` reserved for future expansion.

`reserved_6` reserved for future expansion.

`reserved_7` reserved for future expansion.

`reserved_8` reserved for future expansion.

env_status_array

```
type env_status_array is array (one_bit_field, env_status_bits) of boolean;

  pragma pack(env_status_array);
```

This two dimensional array is used to completely describe the environmental status of the system and extension containers, in the largest possible system. The first index selects the SSM which is provided the information, and the second index selects an environmental status bit.

env_status_array_size

```
env_status_array_size:  constant :=  2 * 32;

  for env_status_array'size use env_status_array_size;
```

Maximum size of environmental status array (2 SSMs and 32 status booleans per SSM);

cardcage_rep

```
type cardcage_rep is
  record
    num_slots:        System.byte_ordinal;
    two_bus_system:   boolean;
    status_bus_0:     boolean;
    status_bus_1:     boolean;
    status_SSB_0:     boolean;
    status_SSB_1:     boolean;
    SSM0_link:        slot_number;
    SSM1_link:        slot_number;
    io_ext_box:       ext_box_rep;
    box_type:         System.byte_ordinal;
    env_status:       env_status_array;
  end record;

  for cardcage_rep use
    record
      num_slots         at  0  range 0 .. 7;
      two_bus_system    at  2  range 0 .. 7;
      status_bus_0      at  3  range 0 .. 7;
      status_bus_1      at  4  range 0 .. 7;
      status_SSB_0      at  5  range 0 .. 7;
      status_SSB_1      at  6  range 0 .. 7;
      SSM0_link         at  7  range 0 .. 7;
      SSM1_link         at  8  range 0 .. 7;
      io_ext_box        at  11 range 0 .. 7;
      box_type          at  12 range 0 .. 7;
      env_status        at  13 range 0 .. env_status_array_size - 1;
    end record;

    for cardcage_rep'size use hw_body_size*8;
```

bus_info:

Fields:

num_slots	Number of slots on backplane.
two_bus_system	If true, two bus system.
status_bus_0	If true, GO AP-Bus 0.
status_bus_1	If true, GO AP-Bus 1.
status_SSB_0	If true, GO SSB 0.
status_SSB_1	If true, GO SSB 1 (Serial System Bus).
SSM0_link	Slot number on which the SSM1 connection is established.
SSM1_link	Slot number on which the SSM2 connection is established.
io_ext_box	Status of io_extension_boxes.
box_type	Enclosure or box type. This is the type of physical package in which the cardcage is housed. This byte is heavily encoded, to decode it, retype it to be of type SSM_Defs.ssm_loc_rec.
env_status	Environmental status for the System and Extension containers.

module_rep

```
type module_rep is
  record
    diag_lock:              System.ordinal;
    slot:                   slot_number;
    spouse:                 slot_number;
    buffered:               boolean;
    FRC_err_count:          System.byte_ordinal;
    FRC_d:                  boolean;
    QMR_d:                  boolean;
    primary:                boolean;
    VLSI_master:            VLSI_desc;
    VLSI_checker:           VLSI_desc;
    com_0:                  com_word;
    com_1:                  com_word;
    com_2:                  com_word;
    com_3:                  com_word;
    res_0:                  System.ordinal;
    res_1:                  System.ordinal;
    config_request_param:   System.ordinal;
    memory_size:            System.ordinal;
    bus_interleaved:        boolean;
    cache_status:           functional_status;
    addr_recognizer:        addr_recognizer_set;
     res_2:                 System.ordinal;
  end record;

  for module_rep use
    record
      diag_lock             at   0 range  0 .. 31;
      slot                  at   4 range  0 .. 7;
      spouse                at   5 range  0 .. 7;
      buffered              at   6 range  7 .. 7;
      FRC_err_count         at   8 range  0 .. 7;
      FRC_d                 at   9 range  0 .. 0;
      QMR_d                 at   9 range  1 .. 1;
      primary               at   9 range  2 .. 2;
      VLSI_master           at  16 range  0 .. 8*8-1;
      VLSI_checker          at  24 range  0 .. 8*8-1;
      com_0                 at  32 range  0 .. 31;
      com_1                 at  36 range  0 .. 31;
      com_2                 at  40 range  0 .. 31;
      com_3                 at  44 range  0 .. 31;
```

```
    res_0                 at  48 range  0 .. 31;
    res_1                 at  52 range  0 .. 31;
    config_request_param  at  56 range  0 .. 31;
    memory_size           at  64 range  0 .. 31;
    bus_interleaved       at  68 range  0 .. 7;
    cache_status          at  69 range  0 .. 7;
    addr_recognizer       at  72 range  0 .. 5*8*8-1;
    res_2                 at 112 range  0 .. 31;
  end record;

  for module_rep'size use hw_body_size*8;
```

Fields:

`diag_lock` If not equal to zero then, this module has been reserved by a diagnostic or maintenance process and other processes should not attempt to access the AP-Bus agents on it. The non-zero value used to reserve a module is the reserving process' ID, which is the binary form of the process' AD.

`slot` Slot number of module.

`spouse` Slot number of spouse module.

`buffered` The memory on this module was battery backed-up when INIT occurred (corresponds to the (AP_Bus agents WARM START bit)

`FRC_err_count` Number of FRC errors occurred (used to identify transient errors).

`FRC_d` If true, the module is FRC'd.

`QMR_d` If true, this module is QMR'd.

`primary` If true, this module is the hardware defined PRIMARY.

`VLSI_master` VLSI IDs and status MASTER bus.

`VLSI_checker` VLSI IDs and status CHECKER bus.

`com_0` COM words contain board configuration information, such as board type and VLSI.

`com_1` Configuration, IO_distribution module type, INIT_counter, CTRL-bits, IO_personality.

`com_2` Module types, layout type, and revision level.

`com_3` Extended revision level.

`res_0` Reserved for additional board-level information.

`res_1` Reserved for additional board-level information.

`config_request_param` Configuration parameters copied from Parameter Store.

`memory_size` Memory available on module (size in bytes).

`bus_interleaved` If true, this module has its address recognizer set for bus interleaving.

`cache_status` Status of the on-board cache.

`addr_recognizer` Address recognizer values for up to five recognizers (i.e., for one BXU of a module); note, the second BXU has the same set, or can be derived out of this set (e.g., in case of interleaving).

res_2 — Handle to allow using the additional space using external type definitions as long as this definition is not yet updated.

null_module_rep

```
null_module_rep:  constant module_rep := (
                    diag_lock            => 0,
                    slot                 => 0,
                    spouse               => 0,
                    buffered             => false,
                    FRC_err_count        => 0,
                    FRC_d                => false,
                    QMR_d                => false,
                    primary              => false,
                    VLSI_master          => null_VLSI_desc,
                    VLSI_checker         => null_VLSI_desc,
                    com_0                => invalid_com_word,
                    com_1                => invalid_com_word,
                    com_2                => invalid_com_word,
                    com_3                => invalid_com_word,
                    res_0                => 0,
                    res_1                => 0,
                    config_request_param => 0,
                    memory_size          => 0,
                    bus_interleaved      => false,
                    cache_status         => offline,
                    addr_recognizer      => null_addr_rec,
                    res_2                => 0);
```

IO_application

```
type IO_application is (
    unused, ASYNC, HDLC_LS, HDLC_HS, LAN, SCSI, IPI, SSB, EDS);

  for IO_application'size use 8;
```

device_param_rep

```
type device_param_rep is array (0 .. 1) of raw_64_bytes;
```

An SCT device entry can hold two Parameter Store device parameter entries.

device_rep

```
type device_rep is
  record
    level:           System.byte_ordinal;
    device_flavor:   IO_application;
    device_ID:       IO_device_ID;
    parameter:       device_param_rep;
  end record;

  for device_rep use
    record
      level          at  0  range 0 .. 7;
      device_flavor  at  3  range 0 .. 7;
```

```
    device_ID       at  4  range 0 .. 31;
    parameter       at 16  range 0 .. 2*64*8-1;
  end record;

  for device_rep'size use hw_body_size*8;
```

Fields:

`level` Identifies auxiliary entries used to record additional I/O parameters for a device. If this entry is equal to 0 then is this the master entry for the device. If it is not equal to 0 this entry contains extra params. which did not fit into the master entry.

`device_flavor` Type of application required to handle the device (used to identify CP application and Device Driver)

`device_ID` Device ID of the device to which this entry belongs.

`parameter` Array of device-specific parameters to be used in Device Object.

hardware_entry_rep

```
type hardware_entry_rep (entry_type: hw_entry_type := cardcage) is
  record
    cardcage_ID:      cardcage_ID_rep;
    status:           functional_status;
    case entry_type is
      when cardcage =>
        cardcage:             cardcage_rep;
      when module =>
        module:               module_rep;
      when device =>
        device:               device_rep;
      when free =>
        null;
    end case;
  end record;

  pragma suppress(discriminant_check,hardware_entry_rep);

  for hardware_entry_rep use
    record
      entry_type   at  0  range 0 .. 7;
      cardcage_ID  at  1  range 0 .. 7;
      status       at  3  range 0 .. 7;
      cardcage     at  hw_header_size  range 0 .. hw_body_size*8-1;
      module       at  hw_header_size  range 0 .. hw_body_size*8-1;
      device       at  hw_header_size  range 0 .. hw_body_size*8-1;
    end record;

    for hardware_entry_rep'size use SCT_Types.hw_entry_size * 8;
```

Fields:

`entry_type`

`cardcage_ID` Indicates which cardcage this module is associated with.

`status` Current status of the corresponding configuration part.

`cardcage` This entry describes a cardcage.

`module`	This entry describes a module (i.e. board).
`device`	This entry describes a boot or console device.

hardware_entry_VA

```
type hardware_entry_VA is access hardware_entry_rep;

  pragma access_kind(hardware_entry_VA, virtual);
```

hardware_entry_header

```
type hardware_entry_header is
  record
    entry_type:   hw_entry_type;
    cardcage_ID:  cardcage_ID_rep;
    status:       functional_status;
  end record;

  for hardware_entry_header use
    record
      entry_type      at 0  range 0 .. 7;
      cardcage_ID     at 1  range 0 .. 7;
      status          at 3  range 0 .. 7;
    end record;
```

Used by the System Monitor to do partial initialization of hardware entries. The representation of this type must match the representation for the header portion of `hardware_entry_rep`.

system_type_rep

```
type system_type_rep is (
   GS_0,
   GS_1,
   GS_2,
   SIM);

  for system_type_rep use(
      GS_0 => 0,
      GS_1 => 1,
      GS_2 => 2,
      SIM  => 3);
```

Enumeration Literals:

`GS_0`	Identifies a GS 0 system.
`GS_1`	Identifies a GS 1 system.
`GS_2`	Identifies a GS 2 system.
`SIM`	For debugging only.

system_mode_rep

```
type system_mode_rep is (normal,
                         diagnostic_required,
                         diagnostic_ready,
                         diagnostic,
                         diagnostic_done);
```

Enumeration Literals:

`normal` System is configured for and running in normal system operation mode.

`diagnostic_required`
System is running, but requests to get transferred to diagnostic mode.

`diagnostic_ready`
System has been reconfigured and is ready to run in diagnostic mode.

`diagnostic` System is configured for and running in diagnostic mode.

`diagnostic_done`
System is ready to get reconfigured for normal mode of operation.

sm_ctrl_param

```
type sm_ctrl_param is
  record
    gdp_test:         boolean;
    cp_test:          boolean;
    bxu_test:         boolean;
    mem_ctrl_test:    boolean;
    mem_array_test:   boolean;
    spare_console:    boolean;
    spare_boot_dev:   boolean;
    spare_image:      boolean;
    auto_dump:        boolean;
    auto_continue:    boolean;
    auto_mask:        boolean;
    load:             boolean;
    start:            boolean;
  end record;

  for sm_ctrl_param use
    record
      gdp_test          at  0  range 0 .. 0;
      cp_test           at  0  range 1 .. 1;
      bxu_test          at  0  range 2 .. 2;
      mem_ctrl_test     at  0  range 3 .. 3;
      mem_array_test    at  0  range 4 .. 4;
      spare_console     at  2  range 0 .. 0;
      spare_boot_dev    at  2  range 1 .. 1;
      spare_image       at  2  range 2 .. 2;
      auto_dump         at  3  range 0 .. 0;
      auto_continue     at  3  range 1 .. 1;
      auto_mask         at  3  range 2 .. 2;
      load              at  3  range 6 .. 6;
      start             at  3  range 7 .. 7;
    end record;
```

System Monitor's control parameters. System software may use those entries to control the System Monitor's behavior during the next initialization sequence.

Fields:

`gdp_test`	Enables GDP confidence test.
`cp_test`	Enables CP confidence test.
`bxu_test`	Enables BXU confidence test.
`mem_ctrl_test`	Enables memory controller test, either (MCU or BXU).
`mem_array_test`	Enables memory array testing.
`spare_console`	Enables spare system console.
`spare_boot_dev`	Enables spare boot device.
`spare_image`	Enables spare boot image.
`auto_dump`	Enables auto dump.
`auto_continue`	If true, forces the System Monitor to continue operation in the auto mode after failures (e.g. error during dumping).
`auto_mask`	Used to force entering the System Monitor's manual mode in cases the system normally would perform an AUTO_START (e.g. restart after power failure) when the System Monitor had been active before the PF occurred.
`load`	If true, forces the System Monitor to load an image.
`start`	If true, enables the System Monitor to activate booted image System Configuration Table.

number_of_ranges

```
number_of_ranges:  constant := 8;
```

Maximum number of special memory ranges which can be allocated.

range_description

```
type range_description is
  record
    valid:            boolean;
    start_pa:         physical_addr;
    size_in_pages:    System.short_ordinal;
    cacheable:        boolean;
    AD:               System.untyped_word;
  end record;

  for range_description use
    record
      valid           at  0 range 0 .. 7;
      cacheable       at  1 range 0 .. 7;
      size_in_pages   at  2 range 0 .. 15;
      start_pa        at  4 range 0 .. 31;
      AD              at  8 range 0 .. 31;
    end record;
```

Fields:

`valid`	If true, This range description is valid.
`start_pa`	Starting physical address of range, must be on a page boundary.
`size_in_pages`	Number of contiguous pages reserved for object.
`cacheable`	If true, Object is cacheable.
`AD`	Access Descriptor, created and filled in by SBL, which points to memory described by this record.

memory_ranges

```
type memory_ranges is array(1 .. number_of_ranges) of range_description;

  pragma pack(memory_ranges);
```

software_entry

```
type software_entry is
  record
    system_type:       system_type_rep;
    system_subtype:    System.short_ordinal;
    system_mode:       system_mode_rep;
    conf_complete:     boolean;
    start_event:       system_start_event;
    sm_ctrl:           sm_ctrl_param;
    FT_config:         raw_8_bytes;
    init_count:        System.ordinal;
    max_count:         System.ordinal;
    dev:               device_list;
    SM_OT:             physical_addr;
    SM_PRCB:           physical_addr;
    SM_res_1:          System.untyped_word;
    SM_res_2:          System.untyped_word;
    SM_res_3:          System.untyped_word;
    self_IAC:          System.untyped_word;
    reserved_memory:   memory_ranges;
    image_version:     raw_32_bytes;
    image_addr:        mem_rec;
    image_OT:          physical_addr;
    image_PRCB:        physical_addr;
    dump_dev:          IO_device_ID;
    dump_position:     position;
    dump_rec:          mem_rec;
    spare_boot_pos:    position;
    MM_IO:             System.address;
    control_reg:       System.byte_ordinal;
  end record;

  for software_entry use
    record
      system_type          at   0 range 0 .. 7;
      system_subtype       at   1 range 0 .. 15;
      system_mode          at   4 range 0 .. 7;
      start_event          at   5 range 0 .. 7;
      conf_complete        at   6 range 0 .. 7;
      sm_ctrl              at   8 range 0 .. 31;
      init_count           at  12 range 0 .. 31;
      max_count            at  16 range 0 .. 31;
      FT_config            at  20 range 0 .. 63;
```

```
    dev                 at  32 range 0 .. 32*8-1;
    SM_OT               at  64 range 0 .. 31;
    SM_PRCB             at  68 range 0 .. 31;
    SM_res_1            at  72 range 0 .. 31;
    SM_res_2            at  76 range 0 .. 31;
    SM_res_3            at  80 range 0 .. 31;
    self_IAC            at  84 range 0 .. 31;
    reserved_memory     at  96 range 0 .. 12*number_of_ranges*8-1;
    image_version       at 256 range 0 .. 32*8-1;
    image_addr          at 288 range 0 .. 64*8-1;
    image_OT            at 352 range 0 .. 31;
    image_PRCB          at 356 range 0 .. 31;
    dump_dev            at 360 range 0 .. 31;
    dump_position       at 364 range 0 .. 31;
    dump_rec            at 368 range 0 .. 64*8-1;
    spare_boot_pos      at 432 range 0 .. 31;
    MM_IO               at 484 range 0 .. 63;
    control_reg         at 492 range 0 ..  7;
  end record;

  for software_entry'size use SCT_Types.sys_sw_size * 8;
```

Contains user visible information from the software entry.

Fields:

`system_type`	Type of system (e.g. GS2).
`system_subtype`	Differentiates system flavors (initialized with value from Parameter Store).
`system_mode`	Mode in which the system is currently operating.
`conf_complete`	If true, the system has been completely configured.
`start_event`	Event which caused the system start.
`sm_ctrl`	Control parameter set used by the System Monitor to control the next init sequence. The parameters are set to default if the SCT has to be built before activating the loaded image. They can be set by the current image if a particular init sequence is desired with the next system restart.
`FT_config`	FT configuration parameters from Parameter Store.
`init_count`	Counter will be initialized with 0; every time the start event is a h/w-driven watchdog timer reset, INIT_COUNT will be incremented. Re-init stops if count exceeds MAX_COUNT; the counter has to be reset by the booted image.
`max_count`	Threshold value for INIT_COUNT.
`dev`	List of eightr device aliases. 0: default System Console 1: default Boot Device 2: spare System Console 3: spare Boot Device 4-7: not predefined
`SM_OT`	Physical addresses of Object
`SM_PRCB`	Table (OT) and Processor Control Block (PRCB).
`SM_res_1`	Reserved for System Monitor.
`SM_res_2`	Reserved for System Monitor.
`SM_res_3`	Reserved for System Monitor.
`self_IAC`	AD to an object that allows to send an IAC to itself.

`reserved_memory`
List of special reseved memory ranges for which SBL must create objects.

`image_version`
ID, version number, and date of the image loaded. During auto dumping used as ID for the dump data, hence, system software should set this field appropriately. Also note, version number gets incremented by one with every dump.

`image_addr` Start addresses and lengths of the image's records

`image_OT` physical addresses of Object

`image_PRCB` Table (OT) and Processor Control Block (PRCB).

`dump_dev` Information to perform auto dumping.

`dump_position`
Information to perform auto dumping.

`dump_rec` Information to perform auto dumping.

`spare_boot_pos`
Position of spare image (i.e. spare `VSM_disk_header`) on primary boot device (0: disk does not provide spare image).

`MM_IO` Virtual address allowing DDs to access I/O registers.

`control_reg` Actual control register value.

software_entry_VA

```
type software_entry_VA is access software_entry;

  pragma access_kind(software_entry_VA, virtual);
```

hw_info_rep

```
type hw_info_rep is
  record
    num_system_buses:      System.byte_ordinal;
    num_slots:             System.byte_ordinal;
    num_GDP:               System.byte_ordinal;
    num_CP:                System.byte_ordinal;
    stable_mem_addr:       physical_addr;
    stable_mem_length:     System.ordinal;
    interl_mem_addr:       physical_addr;
    interl_mem_length:     System.ordinal;
    non_interl_mem_addr:   physical_addr;
    non_interl_mem_length: System.ordinal;
  end record;

  for hw_info_rep use
    record
      num_system_buses          at  0  range 0 .. 7;
      num_slots                 at  1  range 0 .. 7;
      num_GDP                   at  2  range 0 .. 7;
      num_CP                    at  3  range 0 .. 7;
      stable_mem_addr           at  4  range 0 .. 31;
      stable_mem_length         at  8  range 0 .. 31;
      interl_mem_addr           at 12  range 0 .. 31;
      interl_mem_length         at 16  range 0 .. 31;
      non_interl_mem_addr       at 20  range 0 .. 31;
      non_interl_mem_length     at 24  range 0 .. 31;
```

```
    end record;

pragma external;
```

Hardware summary information record.

Fields:

`num_system_buses`
: Number of system buses.

`num_slots`
: Number of slots in cardcage.

`num_GDP`
: Number physical GDPs.

`num_CP`
: Number physical CPs.

`stable_mem_addr`
: Physical start address (byte).

`stable_mem_length`
: Size in bytes.

`interl_mem_addr`
: Physical start address (byte).

`interl_mem_length`
: Size in bytes.

`non_interl_mem_addr`
: Physical start address (byte).

`non_interl_mem_length`
: Size in bytes.

SCT_Access

Provides access to the System Configuration Table (SCT).

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Calls

`Get_error_log`
: Retrieves the error log record from the SCT.

`Get_hardware_info`
: Gets hardware information about a cardcage.

`Release_hw_entries`
: Reserves modules which have already been reserved by the calling process.

`Reserve_hw_entries`
: Reserves the specified modules entries in SCT and returns the list of modules in the cardcage.

`Retrieve_cardcage_entries`
: Returns a record which contains an entry describing the selected modules in the designated cardcage.

`Retrieve_device_entry`
: Retrieves information stored in the SCT about a device.

`Retrieve_software_entry`
: Extracts user visible information from the SCT's software entry and returns this information in a record.

`Set_system_monitor_parameters`
: Sets the System Monitor parameters in the SCT's software entry.

Summary

This package provides access to the SCT using four types:

- `Retrieve` functions return records which contain images of various entries in the SCT (card-module, device, or cardcage entries).

- The `Set` procedure allows the caller to set the System Monitor's control parameters.
- `Reserve/Release` functions allow the caller to reserve and release various card-modules in a cardcage.
- The `Get` procedure returns either the summary for the SCT hardware information (that is, data on modules in a particular cardcage) or it's error logs.

This package provides you with read access only to the System Configuration Table except for the System Monitor control parameters in the software entry.

The SCT consists of three parts. The first part includes the system software entry which contains information used by the system software to configure the system.

The second part contains an error log. The error log is a circular buffer that records the most recent hardware-related errors. It also contains a total error count field which can be used to detect an error log overflow.

The third part of the SCT contains the hardware entries. These entries contain a detailed description of the current hardware configuration.

Exceptions

`reserved_by_others`
: An attempt was made to reserve modules already reserved by another process or release a module already reserved by another process, using a `Reserve` or `Release` function.

`not_in_SCT`
: Indicates a non-existant HW entry in the SCT.

`inconsistent_data`
: Indicates an attempt to update the SCT with corrupt data.

Declarations

max_parameter_number

```
max_parameter_number:  constant := 8;
```

Maximum number of parameters for each IO device in the SCT's device list.

free_hw_entry

```
free_hw_entry:   constant   KMDS_Defs.hardware_entry_rep :=
                   (entry_type     => KMDS_Defs.free,
                    cardcage_ID    => KMDS_Defs.sys,
                    status         => KMDS_Defs.not_used);
```

HW entry of type free.

module_array

```
type module_array is array (
      System.ordinal range
      0 .. KMDS_Defs.max_slot_number) of KMDS_Defs.hardware_entry_rep;
```

List of module data, in a cardcage.

entry_list

```
type entry_list is
  record
    cardcage:  KMDS_Defs.hardware_entry_rep;
    modules:   module_array;
  end record;
```

List of entries.

parameter_array

```
type parameter_array is array (
   1 .. max_parameter_number) of KMDS_Defs.device_param_rep;
```

Define the data structures that hold device entry information retrieved either from the SCT, or information used to update the SCT.

device_entry

```
type device_entry is
  record
    cardcage_ID:   KMDS_Defs.cardcage_ID_rep;
    status:        KMDS_Defs.functional_status;
    module:        KMDS_Defs.slot_number;
    device_ID:     KMDS_Defs.IO_device_ID;
    param_length:  System.ordinal;
    parameters:    parameter_array;
  end record;
```

The following fields define pertinent elements of the device.

Fields:

`cardcage_ID`	ID of the cardcage device belongs to.
`status`	Functional status of the device.
`module`	Slot number of the device.
`device_ID`	Device ID information.
`param_length`	The number of entries placed in the parameter field.
`parameters`	Device parameters.

reserve_by_option

```
type reserve_by_option is (
  physical_ID, logical_ID, slot_number, ignore);

  for reserve_by_option'size use System.storage_unit;
```

Module identification options to reserve modules in a cardcage.

module_reserve_options

```
type module_reserve_options (
  option:  reserve_by_option := physical_ID) is
  record
    case option is
      when physical_ID =>
        phys_ID:  KMDS_Defs.physical_ID_rep;
      when logical_ID  =>
        logic_ID:  KMDS_Defs.logical_ID_rep;
      when slot_number =>
        slot_num:  KMDS_Defs.slot_number;
      when ignore      =>
        null;
    end case;
  end record;

pragma suppress(discriminant_check,module_reserve_options);

for module_reserve_options use
  record
    option       at  0  range 0 .. 7;
    phys_ID      at  1  range 0 .. 7;
    logic_ID     at  1  range 0 .. 7;
```

```
    slot_num      at  1  range 0 .. 7;
  end record;
```

Reserve the module with this physical ID.

Fields:

`option`

`phys_ID`

`logic_ID` Reserve the module with this logical_ID.

`slot_num` Reserve the module with this slot_number.

module_entries_array

```
type module_entries_array is array (
      1 .. KMDS_Defs.max_slot_number) of module_reserve_options;
```

Array type specifies a set of modules in a cardcage with it's slot number, VLSI `physical_ID`, or VLSI `logical_ID`. Reserve and Release functions then use it to reserve or release the modules in a cardcage. To specify the ID of each module, you need to set the option field of each array entry as defined by `module_reserve_options` record declaration.

dont_care_list

```
dont_care_list:  constant module_entries_array := module_entries_array'(
      1 .. KMDS_Defs.max_slot_number => (option => ignore));
```

Denotes an empty list of modules passed to `Reserve_hw_entries` when the list of modules to reserve is not needed (reserving all modules in the cardcage), and passed to `Release_hw_entries` when the list of modules to release is not needed. This releases all modules in the cardcage reserved by the current process.

requested_modules

```
type requested_modules is (
  all_modules, cardcage, offline, bad);
```

Used by the retrieve functions to indicate what set of modules in the cardcage to retrieve information from.

- `all_modules`: Retrieve info from all modules in cardcage.
- `cardcage`: Retrieve info from the cardcage entry, only.
- `offline`: Retrieve info from modules with functional status of `offline` in cardcage.
- `bad`: Retrieve info from modules with functional status of `offline`, `faulty`, or, `offline_or_faulty` in cardcage.

Get_error_log

```
function Get_error_log(
    zero_error_count: boolean := false)
  return KMDS_Defs.error_log_record;
pragma outerface(value, Get_error_log);
```

Parameters

`zero_error_count`
: A directive to zero (clear) out the error count field (total number of errors logged) in the System Configuration Table.

Return Type and Value

`KMDS_Defs.error_log_record`
: The retrieved fault record.

Operation

Retrieves the error log record from the SCT.

If `zero_error_count` is true, the SCT error count is set to zero.

Exceptions

`not_in_SCT`

Get_hardware_info

```
function Get_hardware_info(
    cardcage_ID:  KMDS_Defs.cardcage_ID_rep)
  return KMDS_Defs.hw_info_rep;
pragma outerface(value, Get_hardware_info);
```

Parameters

`cardcage_ID` ID of cardcage to access.

Return Type and Value

`KMDS_Defs.hw_info_rep`
Summary of cardcage hardware information.

Operation

Gets hardware information about a cardcage.

Scans through the entire System Configuration Table, gathers all the hardware information belonging to the indicated cardcage, and returns a summary of this information.

Exceptions

`not_in_SCT`

Release_hw_entries

```
function Release_hw_entries(
    cardcage_ID:    KMDS_Defs.cardcage_ID_rep;
    release_list:   module_entries_array := dont_care_list;
    force:          boolean := false;
    every_module:   boolean := true)
  return module_entries_array;
pragma outerface(value, Release_hw_entries);
```

Parameters

`cardcage_ID`	ID of the cardcage to be accessed.
`release_list`	List of modules to be released.
`force`	Release the specified module entries regardless of who had reserved them originally, when set to true.
`every_module`	If true, release every module reserved by the caller.

Return Type and Value

`module_entries_array`
List of modules that were released.

Operation

Reserves modules which have already been reserved by the calling process.

The list of modules to be released is specified by `release_list`. Modules in this list must be identified through their `physical_ID`, `logical_ID`, or `slot_number`. Therefore, it is possible to specify a list of modules within a cardcage identified through different types of ID's.

If `every_module` is true, the function releases every module in the cardcage already reserved by the caller. The list of returned modules is returned for the caller's verification and must be identical to `release_list`.

Warning

Improper use of `force` may result in unpredictable diagnostics behavior and results.

Notes

Modules are released by the caller's process ID. If any of the modules specified in release_list are reserved by another process ID, then none of the modules in this list are released, and an exception is raised.

`force` forces the release of module entries which are reserved with a process number different than that of the caller. It should only be used to release those module entries that are remained reserved by processes that no longer exist (i.e. abnormally terminated).

Exceptions

```
not_in_SCT
reserved_by_others
```

Reserve_hw_entries

```
function Reserve_hw_entries(
    cardcage_ID:    KMDS_Defs.cardcage_ID_rep;
    reserve_list:   module_entries_array := dont_care_list;
    force:          boolean := false;
    every_module:   boolean := true)
  return module_entries_array;
pragma outerface(value, Reserve_hw_entries);
```

Parameters

`cardcage_ID`	ID of the cardcage to be accessed.
`reserve_list`	List of modules to be reserved.
`force`	Reserve the specified module entries regardless of who had reserved them originally, when set to true.
`every_module`	Reserve every module, if true.

Return Type and Value

`module_entries_array`
List of modules that were reserved.

Operation

Reserves the specified modules entries in SCT and returns the list of modules in the cardcage.

The returned list of modules include the given ID (`cardcage_ID`) reserved for the caller.

`reserve_list` specifies the list of modules to reserve. The caller should specify their identity through the module's `physical_ID`, `logical_ID`, or `slot_number`. Any entry set to `ignore` is ignored. This allows you to specify a list of modules within a cardcage reserved through different ID options. The list of reserved modules is returned for the caller's verification and must be identical to `reserve_list`.

If `every_module` is true, every module in the cardcage is reserved.

Warning

Improper use of this parameter may result in unpredictable diagnostics behavior and results.

Notes

Modules are reserved by the caller's process ID. If any of the modules specified in `reserve_list` are reserved by another process ID, then none of the modules in this list are reserved, and an exception is raised.

`force` forces reservation of module entries which have a different process number than that of the caller. It should only be used to reserve those module entries that remain reserved with processes that no longer exist (i.e. abnormally terminated).

Exceptions

```
not_in_SCT
reserved_by_others
```

Retrieve_cardcage_entries

```
function Retrieve_cardcage_entries(
    cardcage:  KMDS_Defs.cardcage_ID_rep;
    modules:   requested_modules := all_modules)
  return entry_list;
pragma outerface(value, Retrieve_cardcage_entries);
```

Parameters

`cardcage`	ID of cardcage entry from which to retrieve information.
`modules`	Set of module entries in cardcage from which to retrieve information.

Return Type and Value

`entry_list`	Record of all modules in the cardcage.

Operation

Returns a record which contains an entry describing the selected modules in the designated cardcage.

The contents of `modules` defines the set of modules to retrieve, as follows:

```
"all_modules": Retrieve information about every module.

"offline":     Retrieve information about modules with
               functional status of "offline".

"bad":          Retrieve information about modules with
               functional status of "offline",
               "faulty", or "offline_or_faulty".
```

Notes

Entries in the returned array are indexed by their slot number. For example, if a module entry resides on slot 5, the procedure places it in the fifth entry of the array. It then sets the unused entries in the array to `free_hw_entry`.

Exceptions

`not_in_SCT`

Retrieve_device_entry

```
function Retrieve_device_entry(
    device_num:  KMDS_Defs.device_number)
  return device_entry;
pragma outerface(value, Retrieve_device_entry);
```

Parameters

`device_num` Number of device to for which to retrieve information.

Return Type and Value

`device_entry` Information about device.

Operation

Retrieves information stored in the SCT about a device.

`device_num` is the displacement of the device's ID in the software entries device list. For example, if the caller wanted to retrieve information about the actual System Console `device`, the procedure sets the device to 0.

Exceptions

`not_in_SCT`

Retrieve_software_entry

```
function Retrieve_software_entry
  return KMDS_Defs.software_entry;
pragma outerface(value, Retrieve_software_entry);
```

Return Type and Value

`KMDS_Defs.software_entry`
User visible software information.

Operation

Extracts user visible information from the SCT's software entry and returns this information in a record.

Set_system_monitor_parameters

```
procedure Set_system_monitor_parameters(
    parameters:  KMDS_Defs.sm_ctrl_param);
pragma outerface(value, Set_system_monitor_parameters);
```

Parameters

`parameters` New System Monitor Control Parameters.

Operation

Sets the System Monitor parameters in the SCT's software entry.

These parameters control System Monitor behavior during the next system warm start.

SSM_Access

Allows the caller to access System Support Module (SSM) functions.

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Calls

`Blower_control`
: Sends a `BlwrCtl` request to the default SSM.

`DC_control`
: Sends a `DCCtl` request to the default SSM. DC power in the box (i.e. system enclosure) selected by `box` will be either turned on or off depending on the value of `turn_on`.

`Echo`
: Sends an `EchoChar` request to the master microcontroller, which returns the character.

`Read_error_log`
: Sends a `RdErrLog` request to the selected SSM. The error log from the specified SSM is returned.

`Read_NID`
: Sends a `RdNID` request to the SSM microcontroller, which then returns a Unique ID.

`Read_revision`
: Sends a `RdRev` request to the master microcontroller, which returns the revision level of its firmware.

`Read_SSM_config`
: Sends a `RdSSMCfg` request to the SSM. The SSM replies with a description of its configuration.

`Read_SSM_inputs`
: Sends a `RdInp` request to the selected SSM. The raw input signals to the selected SSM are returned.

`Read_TOD`
: Sends a `RdTOD` request to the SSM. The SSM replies with the value of its back-up Time Of Day (TOD) timer.

`Write_LED`
: Sends a `WrLED` request to the default SSM. The SSM will set the mode of the system error LED to `mode`.

`Write_TOD`
: Sends a `WrTOD` request to the SSM.

Summary

This package contains procedures which allow the caller to access some of the System Support Module's (SSM) functions. The functions accessible by this package allow System Administrators, (or Diagnostic Users with System Administrator level access) to retrieve information from the SSM, and to test the communication path to the SSM.

In some systems, there will be a primary and a secondary SSM. When appropriate, the procedures and functions in this package will have an input parameter, which allows the caller to send a request to a specific SSM.

The default handling for a request is: to send it to all SSMs in the system, and return the reply from the primary SSM. If the primary SSM is not available, or does not respond, the secondary SSM's reply will be returned to the caller.

The communication path to the SSM consists of a microcontroller (sometimes called the Master microcontroller), which is attached to one of the computational system's buses and the Serial System Bus (SSB). The microcontroller receives requests for SSM services from system software (e.g. this package), and transmits them across the SSB to the SSM, which acts on the request, and returns a reply. For more information about how the SSM hardware works see the hardware reference manuals.

Warning

Misuse of these SSM functions could result in a system crash or physical damage to the system (e.g. overheating because the blowers were turned off). This interface should only be used by software and users who are aware of this possibility.

To fully understand the consequences of calls to the procedures in this package the user must be familiar with the SSM hardware.

Exceptions

`request_nacked`
: Raised when a request is not accepted by the SSM.

Declarations

SSM_select

```
type SSM_select is(
    primary,
    secondary,
    default);

  for SSM_select use(
    primary    => 0,
    secondary  => 1,
    default    => 2);
```

Blower_control

```
procedure Blower_control(
    turn_on:          boolean;
    full_speed:       boolean;
    select_SSM:       SSM_select := default);
```

Parameters

`turn_on`	True => Turn blower on, False => Turn blower off.
`full_speed`	True => Blower on 100%, False => Blower on 50%.
`select_SSM`	Selects which SSM the request will be sent to.

Operation

Sends a `BlwrCtl` request to the default SSM.

The air blower in the system continer will be either turned on or off, and if it is turned on it will be set to full or half speed depending on the value of `full_speed`.

Exceptions

`request_nacked`

DC_control

```
procedure DC_control(
    turn_on:          boolean;
    buffered:         boolean := true;
    psIIa:            boolean := true;
    psIIb:            boolean := true;
    select_SSM:       SSM_select := default;
    box:              SSM_Defs.enclosure_select := SSM_Defs.main_system);
```

Parameters

`turn_on`	True => Turn power on, False => Turn power off.
`buffered`	True => Control buffered power supply.
`psIIa`	True => Control power to PS-IIa's.
`psIIb`	True => Control power to PS-IIb's.
`select_SSM`	Selects which SSM the request will be sent to.
`box`	Selects enclosure whose power will be affected.

Operation

Sends a `DCCtl` request to the default SSM. DC power in the box (i.e. system enclosure) selected by `box` will be either turned on or off depending on the value of `turn_on`.

Also, if `psIIb_only` is true then only the PS-IIb power supplies in the box will be affected.

If the caller specifies SSM_Defs.reserved_ext for the `box` a `System_Exceptions.bad_parameter` exception will be raised.

Exceptions

`request_nacked`

Echo

```
function Echo(
    character:  KMDS_Defs.seven_bit_field;
    select_SSM:  SSM_select := default)
  return KMDS_Defs.seven_bit_field;
```

Parameters

`character`	Character to be echoed.
`select_SSM`	Selects which SSM the request will be sent to.

Return Type and Value

`KMDS_Defs.seven_bit_field`
Character echoed by master microcontroller.

Operation

Sends an `EchoChar` request to the master microcontroller, which returns the character.

This function's returned value is the character echoed by the master microcontroller.

Exceptions

`request_nacked`

Read_error_log

```
function Read_error_log(
    select_SSM:       SSM_select := default)
  return SSM_Defs.SSM_error_log;
```

Parameters

`select_SSM` Selects which SSM the request will be sent to.

Return Type and Value

`SSM_Defs.SSM_error_log`
Error log from selected SSM.

Operation

Sends a `RdErrLog` request to the selected SSM. The error log from the specified SSM is returned.

Exceptions

`request_nacked`

Read_NID

```
function Read_NID(
    select_SSM:  SSM_select := default)
  return System.ordinal;
```

Parameters

`select_SSM` Selects which SSM the request will be sent to.

Return Type and Value

`System.ordinal`
SSM Unique ID.

Operation

Sends a `RdNID` request to the SSM microcontroller, which then returns a Unique ID.

Exceptions

`request_nacked`

Read_revision

```
function Read_revision(
    select_SSM:  SSM_select := default)
  return System.byte_ordinal;
```

Parameters

`select_SSM` Selects which SSM the request will be sent to.

Return Type and Value

`System.byte_ordinal`
Master micro firmware revision level.

Operation

Sends a `RdRev` request to the master microcontroller, which returns the revision level of its firmware.

Exceptions

`request_nacked`

Read_SSM_config

```
function Read_SSM_config(
    select_SSM:        SSM_select := default)
  return SSM_Defs.SSM_config;
```

Parameters

`select_SSM` Selects which SSM the request will be sent to.

Return Type and Value

`SSM_Defs.SSM_config`
SSM's configuration information.

Operation

Sends a `RdSSMCfg` request to the SSM. The SSM replies with a description of its configuration.

Exceptions

`request_nacked`

Read_SSM_inputs

```
function Read_SSM_inputs(
    select_SSM:        SSM_select := default)
  return SSM_Defs.SSM_inputs;
```

Parameters

`select_SSM` Selects which SSM the request will be sent to.

Return Type and Value

`SSM_Defs.SSM_inputs`
Raw input signals to selected SSM.

Operation

Sends a `RdInp` request to the selected SSM. The raw input signals to the selected SSM are returned.

Exceptions

`request_nacked`

Read_TOD

```
function Read_TOD(
    select_SSM:  SSM_select := default)
  return KMDS_Defs.TOD;
```

Parameters

`select_SSM` Selects which SSM the request will be sent to.

Return Type and Value

`KMDS_Defs.TOD`
Back-up TOD time from SSM.

Operation

Sends a `RdTOD` request to the SSM. The SSM replies with the value of its back-up Time Of Day (TOD) timer.

Exceptions

`request_nacked`

Write_LED

```
procedure Write_LED(
    LED_ID:             KMDS_Defs.two_bit_field;
    mode:               SSM_Defs.LED_modes);
```

Parameters

LED_ID	Selects the LED whose mode will be altered.
mode	Selected LED's new mode.

Operation

Sends a WrLED request to the default SSM. The SSM will set the mode of the system error LED to mode.

Either the System Error LED or the Online Replacement LED can be selected.

Exceptions

request_nacked

Write_TOD

```
procedure Write_TOD(
    time:  KMDS_Defs.TOD);
```

Parameters

`time`	Time of day to send to backup TOD timers.

Operation

Sends a `WrTOD` request to the SSM.

`time` is the data portion of this request.

Exceptions

`request_nacked`

SSM_Defs

Defines types and constants used to interface to the SSM.

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Summary

The *System Support Module* provides the following functions:

- Watchdog timer
- Time-of-day backup timer
- Power supply control and monitoring
- Control panel interface
- Unique system identification number (UID)
- Link to the serial system bus
- Generalization of intialization events
- Environmental monitoring.

Declarations

SSM_requests

```
type SSM_requests is (
    echo_char,
    read_rev,
    gen_OR_int,
    read_status,
    read_config,
    sys_init,
    DC_cntrl,
    blower_cntrl,
    read_err_log,
    read_inputs,
    load_env_timer,
    read_TOD,
    write_TOD,
    write_LED,
    write_watchdog,
    config_SSM,
    read_NID,
    send_to_MD,
    config_OR_int,
    single_init0,
    single_init1,
    double_init,
    load_COM_word,
    basic,
    read_TOD_RAM,
    write_TOD_RAM,
    load_mem_init_bit);
```

Standard messages which can be sent to or received from an SSM.

SSM_replies

```
type SSM_replies is (
    acknowledge,
    neg_acknowledge,
    echo_char_reply,
    read_rev_reply,
    read_status_reply,
    read_config_reply,
    read_err_log_reply,
    read_inputs_reply,
    read_TOD_reply,
    read_NID_reply,
    read_TOD_RAM_reply,
    no_reply);
```

SSM replies to requests.

unsol_msgs

```
type unsol_msgs is (
    switch_request,
    power_fail,
    environment_fail,
    from_MD,
    external_interrupt,
    undefined,
    request_timeout);
```

Types of unsolicited messages which can be generated by SSM.

enclosure_select

```
type enclosure_select is (
    tower_0,
    tower_1,
    tower_2,
    tower_3,
    main_system,
    reserved_ext);

  for enclosure_select use(
    tower_0       => 2#000#,
    tower_1       => 2#001#,
    tower_2       => 2#010#,
    tower_3       => 2#011#,
    main_system   => 2#100#,
    reserved_ext  => 2#101#);
```

Used to select the system enclosure upon which the SSM Command will act.

watchdog_interval

```
type watchdog_interval is
  record
    time_base:      KMDS_Defs.two_bit_field;
    time_interval:  KMDS_Defs.seven_bit_field;
  end record;
```

Watchdog timer interval.

Fields:

`time_base` Values of time base are: 0 => timer is off. 1 => 0.1 second per count. 2 => 1.0 second per count. 3 => undefined.

`time_interval` Number of counts in interval.

disable_timer

```
disable_timer:      constant watchdog_interval := (time_base     => 0,
                                                   time_interval => 0);
```

Turn off the watchdog timer.

min_timeout

```
min_timeout:        constant watchdog_interval := (time_base     => 1,
                                                   time_interval => 1);
```

Shortest possible watchdog timeout interval (100 milliseconds).

max_timeout

```
max_timeout:        constant watchdog_interval := (time_base     => 2,
                                                   time_interval => 16#7F#);
```

Maximum possible watchdog timeout internal (127 seconds or about 2 minutes).

standard_timeout

```
standard_timeout:   constant watchdog_interval := (time_base     => 1,
                                                   time_interval => 100);
```

Standard short timeout interval. Current value of 10 seconds is arbitrary.

ssm_loc_rec

```
type ssm_loc_rec is (
     ssm_container,
     ssm_tower,
     ssm_mini);

  for ssm_loc_rec use(
     ssm_container                    =>  2#000#,
     ssm_tower                        =>  2#001#,
     ssm_mini                         =>  2#010#);
```

SSM location.

Enumeration Literals:

`ssm_container`
: SSM installed in container.

`ssm_tower`
: SSM installed in tower.

`ssm_mini`
: SSM installed in minibox.

SSM_config

```
type SSM_config is
  record
    psIIa_0:         boolean;
    psIIa_1:         boolean;
    psIIb_0:         boolean;
    psIIb_1:         boolean;
    psIIc_1:         boolean;
    ssm_location:    ssm_loc_rec;
    firmware_rev:    System.byte_ordinal;
    board_rev:       KMDS_Defs.four_bit_field;
    board_layout:    KMDS_Defs.three_bit_field;
    company:         KMDS_Defs.one_bit_field;
    rev_overflow1:   KMDS_Defs.four_bit_field;
    rev_overflow2:   KMDS_Defs.four_bit_field;
  end record;

  for SSM_config use
    record
      psIIa_0          at 0    range 0 .. 0;
      psIIa_1          at 0    range 1 .. 1;
      psIIb_0          at 0    range 2 .. 2;
      psIIb_1          at 0    range 3 .. 3;
      psIIc_1          at 0    range 4 .. 4;
      ssm_location     at 1    range 0 .. 7;
      firmware_rev     at 2    range 0 .. 7;
      board_rev        at 3    range 0 .. 3;
      board_layout     at 3    range 4 .. 6;
      rev_overflow1    at 4    range 0 .. 3;
      company          at 4    range 4 .. 4;
      rev_overflow2    at 5    range 0 .. 3;
    end record;
```

SSM configuration information.

Fields:

`psIIa_0`	PS-IIa-0 installed.
`psIIa_1`	PS-IIa-1 installed.
`psIIb_0`	PS-IIb-0 installed.
`psIIb_1`	PS-IIb-1 installed.
`psIIc_1`	PS-IIc-1 installed.
`ssm_location`	Where SSM is installed.
`firmware_rev`	SSM firmware rev. level.
`board_rev`	SSM board rev. level.
`board_layout`	SSM board layout level.
`company`	Company which produced SSM board.
`rev_overflow1`	SSM Board revision overflow area.
`rev_overflow2`	SSM Board revision overflow area.

LED_modes

```
type LED_modes is (
      off,
      on,
      slow,
      fast);

  for LED_modes use(
      off     => 2#00#,
      on      => 2#01#,
      slow    => 2#10#,
      fast    => 2#11#);
```

LED modes.

Enumeration Literals:

`off`	Turn the LED off.
`on`	Turn the LED on.
`slow`	Set LED to slow blink (1Hz).
`fast`	Set LED to fast blink (5Hz).

SSM_error_log

```
type SSM_error_log is
  record
    parity_errors:    System.byte_ordinal;
    message_retries:  System.byte_ordinal;
  end record;

  for SSM_error_log use
    record
      parity_errors           at 0  range 0 .. 7;
      message_retries         at 4  range 0 .. 7;
    end record;
```

SSM error log.

Fields:

`parity_errors`
: Number of parity errors.

`message_retries`
: Number of retried messages.

SSM_inputs

```
type SSM_inputs is
  record
    not_5F_A0:     boolean;
    not_12F_A0:    boolean;
    not_5F_A1:     boolean;
    not_12F_A1:    boolean;
    not_DCF_B0:    boolean;
    not_BDCF:      boolean;
```

```
    not_BA_F:       boolean;
    not_MAN_BUT:    boolean;
    not_OLR_BUT:    boolean;
    not_RST_BUT:    boolean;
    not_TEST:       boolean;
    not_AIR_F:      boolean;
    not_ERR:        boolean;
    not_DCF_B1:     boolean;
    FLT_EX0:        boolean;
    FLT_EX1:        boolean;
    FLT_EX2:        boolean;
    FLT_EX3:        boolean;
    not_TF_I:       boolean;
    not_TF_CC:      boolean;
    not_TF_P:       boolean;
    not_CPS_A0:     boolean;
    not_CPS_A1:     boolean;
    not_CPS_B0:     boolean;
    not_CPS_B1:     boolean;
    not_5BF_C:      boolean;
    not_CDCF:       boolean;
    TOWER:          boolean;
    not_UPS:        boolean;
    not_BAX_F:      boolean;
    not_TF_A:       boolean;
    MINIBOX:        boolean;
    DCON_S:         boolean;
    not_ACF:        boolean;
    ACFX:           boolean;
    not_EXT_INT0:   boolean;
    not_EXT_INT1:   boolean;
    not_EXT_INT2:   boolean;
    not_EXT_INT3:   boolean;
    not_EXT_INT4:   boolean;
    not_EXT_INT5:   boolean;
    not_EXT_INT6:   boolean;
    not_EXT_INT7:   boolean;
  end record;

  for SSM_inputs use
    record
      not_5F_A0       at 0    range 0 .. 0;
      not_12F_A0      at 0    range 1 .. 1;
      not_5F_A1       at 0    range 2 .. 2;
      not_12F_A1      at 0    range 3 .. 3;
      not_DCF_B0      at 0    range 4 .. 4;
      not_BDCF        at 0    range 5 .. 5;
      not_BA_F        at 0    range 6 .. 6;
      not_MAN_BUT     at 1    range 0 .. 0;
      not_OLR_BUT     at 1    range 1 .. 1;
      not_RST_BUT     at 1    range 2 .. 2;
      not_TEST        at 1    range 3 .. 3;
      not_AIR_F       at 1    range 4 .. 4;
      not_ERR         at 1    range 5 .. 5;
      not_DCF_B1      at 1    range 6 .. 6;
      FLT_EX0         at 2    range 0 .. 0;
      FLT_EX1         at 2    range 1 .. 1;
      FLT_EX2         at 2    range 2 .. 2;
      FLT_EX3         at 2    range 3 .. 3;
      not_TF_I        at 2    range 4 .. 4;
      not_TF_CC       at 2    range 5 .. 5;
      not_TF_P        at 2    range 6 .. 6;
      not_CPS_A0      at 3    range 0 .. 0;
      not_CPS_A1      at 3    range 1 .. 1;
      not_CPS_B0      at 3    range 2 .. 2;
      not_CPS_B1      at 3    range 3 .. 3;
      not_5BF_C       at 3    range 4 .. 4;
      not_CDCF        at 3    range 5 .. 5;
      TOWER           at 3    range 6 .. 6;
      not_UPS         at 4    range 0 .. 0;
```

```
    not_BAX_F       at 4    range 1 .. 1;
    not_TF_A        at 4    range 2 .. 2;
    MINIBOX         at 4    range 3 .. 3;
    DCON_S          at 4    range 4 .. 4;
    not_ACF         at 4    range 5 .. 5;
    ACFX            at 4    range 6 .. 6;
    not_EXT_INT0    at 5    range 0 .. 0;
    not_EXT_INT1    at 5    range 1 .. 1;
    not_EXT_INT2    at 5    range 2 .. 2;
    not_EXT_INT3    at 5    range 3 .. 3;
    not_EXT_INT4    at 6    range 0 .. 0;
    not_EXT_INT5    at 6    range 1 .. 1;
    not_EXT_INT6    at 6    range 2 .. 2;
    not_EXT_INT7    at 6    range 3 .. 3;
end record;
```

Names of the SSM input signals. A `not` prefix indicates that the signal uses negative true logic.

Fields:

`not_5F_A0`

`not_12F_A0`

`not_5F_A1`

`not_12F_A1`

`not_DCF_B0`

`not_BDCF`

`not_BA_F`

`not_MAN_BUT`

`not_OLR_BUT`

`not_RST_BUT`

`not_TEST`

`not_AIR_F` In minibox : not_BL_F

`not_ERR`

`not_DCF_B1`

`FLT_EX0`

`FLT_EX1`

`FLT_EX2`

`FLT_EX3`

`not_TF_I`

`not_TF_CC`

`not_TF_P`

`not_CPS_A0`

`not_CPS_A1`

`not_CPS_B0`

`not_CPS_B1`

`not_5BF_C`

not_CDCF

TOWER

not_UPS

not_BAX_F

not_TF_A

MINIBOX

DCON_S

not_ACF In minibox: not_RESET

ACFX In tower: CPS_C1

not_EXT_INT0

not_EXT_INT1

not_EXT_INT2

not_EXT_INT3

not_EXT_INT4

not_EXT_INT5

not_EXT_INT6

not_EXT_INT7

OR_int_type

```
type OR_int_type is (
      disable,
      on_line_insert,
      turn_on_stable_store,
      local_reset,
      turn_on_board);

  for OR_int_type use(
      disable                 =>  2#000#,
      on_line_insert          =>  2#001#,
      turn_on_stable_store    =>  2#010#,
      local_reset             =>  2#011#,
      turn_on_board           =>  2#100#);
```

Possible OR interrupts.

Enumeration Literals:

`disable` Disable the OR interrupt.

`on_line_insert`
Use OR interrupt for on_line insertion on AP bus.

`turn_on_stable_store`
Use OR interrupt for stable store turn on.

`local_reset` Use OR interrupt for local reset of board.

`turn_on_board`
Use OR interrupt for board turn on after insertion.

board_mode

```
type board_mode is (
      normal,
      lb0_active,
      lb1_active,
      default);

  for board_mode use(
      normal                  =>  2#00#,
      lb0_active              =>  2#01#,
      lb1_active              =>  2#10#,
      default                 =>  2#11#);
```

Possible mode of boards after init.

Enumeration Literals:

`normal`	Follow the predicted BXU/MCU mode.
`lb0_active`	Use LB0's resource, for GDP board FRC split mode.
`lb1_active`	Use LB1's resource, for GDP board, only LB1 is functional.
`default`	Use default mode, for GDP board FRC mode for others lb0 mode.

OR_param_rec

```
type OR_param_rec (
    select_int: OR_int_type := local_reset) is
  record
    case select_int is
      when local_reset        =>
        core:       boolean;
        non_core:   boolean;
        init_mode:  board_mode;
      when others             =>
        null;
    end case;
  end record;

  pragma suppress(discriminant_check,OR_param_rec);

  for OR_param_rec use
    record
      select_int      at 0    range 0 .. 2;
      core            at 0    range 3 .. 3;
      non_core        at 0    range 4 .. 4;
      init_mode       at 0    range 5 .. 6;
    end record;
```

Fields:

`select_int`	Possible OR interrupt configurations.
`core`	True => all core boards pay attention.
`non_core`	True => all non_core boards pay attention.
`init_mode`	Indicates which mode the board should come up in after init.

SSM_param_rec

```
type SSM_param_rec is
  record
    box:         enclosure_select;
    test:        boolean;
    battery:     boolean;
    extension:   boolean;
  end record;

  for SSM_param_rec use
    record
      box              at 0    range 0 .. 2;
      test             at 0    range 4 .. 4;
      battery          at 0    range 5 .. 5;
      extension        at 0    range 6 .. 6;
    end record;
```

Parameter for configure SSM command.

Fields:

box	Selects box which will be affected.
test	If true, test mode on.
battery	If true, external battery connected.
extension	If true, container connected.

DC_cntrl_rec

```
type DC_cntrl_rec is
  record
    box:         enclosure_select;
    turn_on:     boolean;
    buffered:    boolean;
    PSIIa:       boolean;
    PSIIb:       boolean;
  end record;

  for DC_cntrl_rec use
    record
      box              at 0    range 0 .. 2;
      buffered         at 0    range 3 .. 3;
      PSIIa            at 0    range 4 .. 4;
      PSIIb            at 0    range 5 .. 5;
      turn_on          at 0    range 6 .. 6;
    end record;
```

Format of a DC power control record.

Fields:

box	Selects box which will be affected.
turn_on	If true, turn DC power on. If false, turn DC power off.
buffered	If true, affect source of buffered memory power.
PSIIa	If true, affect PS-IIa power supplies.

PSIIb — If true, affect PS-IIb power supplies.

blower_cntrl_rec

```
type blower_cntrl_rec is
  record
    box:          enclosure_select;
    full_speed:   boolean;
    turn_on:      boolean;
  end record;

  for blower_cntrl_rec use
    record
      box           at 0    range 0 .. 2;
      full_speed    at 0    range 5 .. 5;
      turn_on       at 0    range 6 .. 6;
    end record;
```

Format of an air blower control record.

Fields:

box — Selects box which will be affected.

full_speed — If true, set blower for full speed. If false, set blower for half speed.

turn_on — If true, turn blower on. If false, turn blower off.

SSM_status

```
type SSM_status is
  record
    watchdog_overflow:  boolean;
    DC_on:              boolean;
    reset:              boolean;
    AC_power_return:    boolean;
    sys_init_request:   boolean;
  end record;

  for SSM_status use
    record
      watchdog_overflow       at 0    range 0 .. 0;
      DC_on                   at 0    range 1 .. 1;
      reset                   at 0    range 2 .. 2;
      AC_power_return         at 0    range 3 .. 3;
      sys_init_request        at 0    range 4 .. 4;
    end record;
```

SSM status data.

Fields:

watchdog_overflow — INIT assertion due to watchdog timer.

DC_on — INIT assertion due to DC-ON button.

reset — INIT assertion due to RESET button.

AC_power_return — INIT assertion due to return of AC power.

`sys_init_request`
INIT assertion due to SSM Init. request.

NID_byte

```
type NID_byte is
  record
    NID_nibble:  KMDS_Defs.four_bit_field;
    zero:        KMDS_Defs.four_bit_field;
  end record;

  for NID_byte use
    record
      NID_nibble      at 0    range 0 .. 3;
      zero            at 0    range 4 .. 7;
    end record;
```

Format of raw NID data byte.

Fields:

`NID_nibble`	Low four bits of byte contain NID data.
`zero`	High four bits are zero.

raw_NID

```
type raw_NID is array (System.ordinal range 1 .. 8) of NID_byte;

  pragma pack(raw_NID);
```

Raw NID data returned by SSM.

raw_data_buffer

```
type raw_data_buffer is array (System.ordinal range 1 .. 8)
    of System.byte_ordinal;

  pragma pack(raw_data_buffer);
```

Raw data buffer.

request_format

```
type request_format (
     request: SSM_requests := write_TOD) is
  record
    case request is
      when echo_char |
           send_to_MD =>
        char:  KMDS_Defs.seven_bit_field;
      when read_rev |
           gen_OR_int |
           read_NID |
           read_TOD |
           single_init0 |
```

```
            single_init1 |
            double_init |
            basic |
            read_TOD_RAM =>
          null;
        when read_status |
            sys_init |
            read_err_log |
            read_inputs |
            read_config =>
          box:          enclosure_select;
        when DC_cntrl =>
          DC:           DC_cntrl_rec;
        when blower_cntrl =>
          blower:       blower_cntrl_rec;
        when load_env_timer =>
          counter:      enclosure_select;
          count:        KMDS_Defs.seven_bit_field;
        when write_TOD |
            write_TOD_RAM =>
          time:         KMDS_Defs.TOD;
        when write_LED =>
          LED_select:   KMDS_Defs.two_bit_field;
          mode:         LED_modes;
        when write_watchdog =>
          interval:     watchdog_interval;
        when config_SSM =>
          SSM_param:    SSM_param_rec;
        when config_OR_int =>
          OR_param:     OR_param_rec;
        when load_COM_word =>
          index:        System.ordinal range 0 .. 2;
        when load_mem_init_bit =>
          set_bit:      boolean;
      end case;
    end record;

    pragma suppress(discriminant_check,request_format);

    for request_format use
      record
        request         at 0    range 0 .. 7;
        char            at 1    range 0 .. 7;
        box             at 1    range 0 .. 2;
        DC              at 1    range 0 .. 7;
        blower          at 1    range 0 .. 7;
        counter         at 1    range 0 .. 2;
        count           at 2    range 0 .. 7;
        time            at 1    range 0 .. 8 * 8 - 1;
        LED_select      at 1    range 0 .. 1;
        mode            at 1    range 5 .. 6;
        interval        at 1    range 0 .. 2 * 8 - 1;
        SSM_param       at 1    range 0 .. 7;
        OR_param        at 1    range 0 .. 7;
        index           at 1    range 5 .. 6;
        set_bit         at 1    range 0 .. 0;
    end record;
```

Fields:

request	Format of SSM requests with data bytes.
char	Character argument. High-order bit will be forced to 0.
box	Selects enclosure whose SSM will perform the request.
DC	DC power supply control data.

blower	Blower control data.
counter	Selects SSM whose environment timer will be loaded.
count	Count.
time	Time of Day.
LED_select	Selects the LED (System Error or Online Rep OK) whose mode will be changed.
mode	Display mode for LED (i.e. on, off, fast blink, or slow blink).
interval	Time before watchdog timer goes off.
SSM_param	Informs SSM about configuration.
OR_param	Configures the OR interrupt.
index	Indicates which COM word to load (i.e. 0, 1, or 2).
set_bit	If true, set Memory Initialized bit in COM word 1. If false, clear Memory Initialized bit.

reply_format

```
type reply_format (
    reply:  SSM_replies := acknowledge) is
  record
    case reply is
      when echo_char_reply |
           read_rev_reply =>
        char:          KMDS_Defs.seven_bit_field;
      when read_status_reply =>
        status:        SSM_status;
      when read_config_reply =>
        config:        SSM_config;
      when read_err_log_reply =>
        error_log:     SSM_error_log;
      when read_inputs_reply =>
        inputs:        SSM_inputs;
      when read_TOD_reply |
           read_TOD_RAM_reply =>
        TOD:           KMDS_Defs.TOD;
      when read_NID_reply =>
        NID:           raw_NID;
      when others =>
        raw_data:      raw_data_buffer;
    end case;
  end record;

  pragma suppress(discriminant_check,reply_format);

  for reply_format use
    record
      reply            at 0    range 0 .. 7;
      char             at 1    range 0 .. 7;
      status           at 1    range 0 .. 7;
      config           at 1    range 0 .. 6 * 8 - 1;
      error_log        at 1    range 0 .. 6 * 8 - 1;
      inputs           at 1    range 0 .. 7 * 8 - 1;
      TOD              at 1    range 0 .. 8 * 8 - 1;
      NID              at 1    range 0 .. 8 * 8 - 1;
      raw_data         at 1    range 0 .. 8 * 8 - 1;
  end record;
```

unsol_msg_format

```
type unsol_msg_format (
    msg:  unsol_msgs := power_fail) is
  record
    primary:          boolean;
    source:           enclosure_select;
    case msg is
      when switch_request =>
        manual_request:       boolean;
        online_replacement:   boolean;
        DC_off:               boolean;
        test:                 boolean;
      when power_fail =>
        AC_fail_other_PSIIc:  boolean;
        AC_ret_other_PSIIc:   boolean;
        UPS_on:               boolean;
        UPS_off:              boolean;
        sys_battery_fail:     boolean;
        ext_battery_fail:     boolean;
        buf_5v_bus_fail:      boolean;
        PSIIa0_5v_fail:       boolean;
        PSIIa0_12v_fail:      boolean;
        PSIIa1_5v_fail:       boolean;
        PSIIa1_12v_fail:      boolean;
        PSIIb0_fail:          boolean;
        PSIIb1_fail:          boolean;
        buf_5v_supply_fail:   boolean;
        AC_fail:              boolean;
        AC_returns:           boolean;
      when environment_fail =>
        PSIIc_temp:           boolean;
        cage_temp_air:        boolean;
        periph_temp_air:      boolean;
        air_intake_temp:      boolean;
        blower:               boolean;
        mini_0:               boolean;
        mini_1:               boolean;
        mini_2:               boolean;
        mini_3:               boolean;
      when from_MD =>
        char:                 System.byte_ordinal;
      when external_interrupt =>
        ext_int0:             boolean;
        ext_int1:             boolean;
        ext_int2:             boolean;
        ext_int3:             boolean;
        ext_int4:             boolean;
        ext_int5:             boolean;
        ext_int6:             boolean;
        ext_int7:             boolean;
      when others =>
        null;
    end case;
  end record;

  pragma suppress(discriminant_check,unsol_msg_format);

  for unsol_msg_format use
    record
      msg              at 0    range 0 .. 7;
      primary          at 4    range 0 .. 7;
      source           at 1    range 0 .. 7;
      manual_request           at 2    range 0 .. 0;
      online_replacement       at 2    range 1 .. 1;
      DC_off                   at 2    range 2 .. 2;
      test                     at 2    range 3 .. 3;
      AC_fail_other_PSIIc      at 2    range 0 .. 0;
```

```
    AC_ret_other_PSIIc      at 2    range 1 .. 1;
    UPS_on                  at 2    range 2 .. 2;
    UPS_off                 at 2    range 3 .. 3;
    sys_battery_fail        at 2    range 4 .. 4;
    ext_battery_fail        at 2    range 5 .. 5;
    buf_5v_bus_fail         at 2    range 6 .. 6;
    PSIIa0_5v_fail          at 3    range 0 .. 0;
    PSIIa0_12v_fail         at 3    range 1 .. 1;
    PSIIa1_5v_fail          at 3    range 2 .. 2;
    PSIIa1_12v_fail         at 3    range 3 .. 3;
    PSIIb0_fail             at 3    range 4 .. 4;
    PSIIb1_fail             at 3    range 5 .. 5;
    buf_5v_supply_fail      at 3    range 6 .. 6;
    AC_fail                 at 4    range 0 .. 0;
    AC_returns              at 4    range 1 .. 1;
    PSIIc_temp              at 2    range 0 .. 0;
    cage_temp_air           at 2    range 1 .. 1;
    periph_temp_air         at 2    range 2 .. 2;
    air_intake_temp         at 2    range 3 .. 3;
    blower                  at 2    range 4 .. 4;
    mini_0                  at 3    range 0 .. 0;
    mini_1                  at 3    range 1 .. 1;
    mini_2                  at 3    range 2 .. 2;
    mini_3                  at 3    range 3 .. 3;
    char                    at 2    range 0 .. 7;
    ext_int0                at 2    range 0 .. 0;
    ext_int1                at 2    range 1 .. 1;
    ext_int2                at 2    range 2 .. 2;
    ext_int3                at 2    range 3 .. 3;
    ext_int4                at 3    range 0 .. 0;
    ext_int5                at 3    range 1 .. 1;
    ext_int6                at 3    range 2 .. 2;
    ext_int7                at 3    range 3 .. 3;
end record;
```

Fields:

`msg` Format of data in an unsolicited message.

`primary` If true, this message came from primary SSM. If false, this message came from secondary SSM.

`source` System box (i.e. enclosure) which caused this message.

`manual_request`
Manual mode requested.

`online_replacement`
Online replacement requested.

`DC_off` Turn off DC power request.

`test` Test request.

`AC_fail_other_PSIIc`
AC power to the other PS-IIc failed.

`AC_ret_other_PSIIc`
AC power returned on the other PS-IIc.

`UPS_on` Container's UPS is on.

`UPS_off` Container's UPS is off.

`sys_battery_fail`
Battery failure in a container.

`ext_battery_fail`
External battery failure.

`buf_5v_bus_fail`
Buffered +5VDC bus failure in a container.

`PSIIa0_5v_fail`
PS-IIa-0 +5VDC failure in a container.

`PSIIa0_12v_fail`
PS-IIa-0 12VDC failure in a container.

`PSIIa1_5v_fail`
PS-IIa-1 +5VDC failure in a container.

`PSIIa1_12v_fail`
PS-IIa-1 12VDC failure in a container.

`PSIIb0_fail` PS-IIb-0 failure in a container.

`PSIIb1_fail` PS-IIb-1 failure in a container.

`buf_5v_supply_fail`
Buffered +5VDC supply failure in a container.

`AC_fail` AC input power fail.

`AC_returns` AC input power returns after failure.

`PSIIc_temp` PS-IIc internal temperature fault in a container.

`cage_temp_air`
Temp/Airflow fault in a container's cardcage.

`periph_temp_air`
Temp/Airflow fault in a container's peripherals.

`air_intake_temp`
Ambient air intake temp. fault in a container.

`blower` Blower fault.

`mini_0` Fault in extension minibox 0.

`mini_1` Fault in extension minibox 1.

`mini_2` Fault in extension minibox 2.

`mini_3` Fault in extension minibox 3.

`char`

`ext_int0` External interrupt 0 is active.

`ext_int1` External interrupt 1 is active.

`ext_int2` External interrupt 2 is active.

`ext_int3` External interrupt 3 is active.

`ext_int4` External interrupt 4 is active.

`ext_int5` External interrupt 5 is active.

`ext_int6` External interrupt 6 is active.

`ext_int7` External interrupt 7 is active.

unsol_msg_VA

```
type unsol_msg_VA is access unsol_msg_format;
  pragma ACCESS_KIND(unsol_msg_VA, VIRTUAL);
```

invalid_slot_number

```
invalid_slot_number:    constant KMDS_Defs.slot_number := 0;
```

SCT_Mgt indicates an empty slot by setting the slot number to zero.

SCT_SSM_warning

```
SCT_SSM_warning:        constant := 2;
```

SCT fault type (f_type) code for SSM Daemon warnings.

no_warning

```
no_warning:             constant := 0;
```

timeout

```
timeout:                constant := 1;
```

send_not_started

```
send_not_started:       constant := 2;
```

overlapped_requests

```
overlapped_requests:    constant := 3;
```

illegal_control_char

```
illegal_control_char:  constant := 4;
```

unexpected_reply

```
unexpected_reply:      constant := 5;
```

bad_parity

```
bad_parity:            constant := 6;
```

extraneous_etx

```
extraneous_etx:        constant := 7;
```

unexpected_exception

```
unexpected_exception:  constant := 8;
```

extraneous_nack

```
extraneous_nack:       constant := 9;
```

system_error_LED

```
system_error_LED:  constant KMDS_Defs.two_bit_field := 2#01#;
```

online_rep_LED

```
online_rep_LED:    constant KMDS_Defs.two_bit_field := 2#10#;
```

Constants defined for quick check for standard replies.

ack_reply

```
ack_reply:  constant reply_format(reply => acknowledge) :=(
                              reply    => acknowledge,
                              raw_data => (others => 0));
```

nack_reply

```
nack_reply:  constant reply_format(reply => neg_acknowledge) :=(
                              reply    => neg_acknowledge,
                              raw_data => (others => 0));
```

Encodings of SSM control characters. Note: The control characters used to comunicate with the SSM are NOT ASCII. They have a special unique encoding.

stx

```
stx:                    constant System.byte_ordinal := 2#10000000#;
```

etx

```
etx:                    constant System.byte_ordinal := 2#11111000#;
```

ack

```
ack:                    constant System.byte_ordinal := 2#11111001#;
```

nack

```
nack:                   constant System.byte_ordinal := 2#11111010#;
```

rdy

```
rdy:                    constant System.byte_ordinal := 2#11111011#;
```

sync

```
sync:                    constant System.byte_ordinal := 2#11111100#;
```

control_char_format

```
type control_char_format is
  record
    control:  boolean;
    c_bits:   KMDS_Defs.four_bit_field;
    m_bits:   KMDS_Defs.three_bit_field;
  end record;

  for control_char_format use
    record
      control at 0    range 7 .. 7;
      c_bits  at 0    range 3 .. 6;
      m_bits  at 0    range 0 .. 2;
    end record;
```

Used to break down control bytes into meaningful bit fields.

not_stx

```
not_stx:    constant KMDS_Defs.four_bit_field := 2#1111#;
```

The `c_bits` have this value for all control characters except `stx`.

etx_m_bits

```
etx_m_bits:  constant KMDS_Defs.three_bit_field := 2#000#;
```

ack_m_bits

```
ack_m_bits:  constant KMDS_Defs.three_bit_field := 2#001#;
```

nack_m_bits

```
nack_m_bits: constant KMDS_Defs.three_bit_field := 2#010#;
```

rdy_m_bits

```
rdy_m_bits:  constant KMDS_Defs.three_bit_field := 2#011#;
```

Constants used to build Designate Master request.

des_master_code

```
des_master_code:        constant System.byte_ordinal := 16#08#;
```

des_master_des_db1

```
des_master_des_db1:     constant System.byte_ordinal := 2#01000000#;
```

des_master_undes_db1

```
des_master_undes_db1:  constant System.byte_ordinal := 2#00000000#;
```

Encodings stx byte of unsolicited messages.

switch_request_code

```
switch_request_code:    constant System.byte_ordinal := 16#48#;
```

power_fail_code

```
power_fail_code:        constant System.byte_ordinal := 16#49#;
```

environment_fail_code

```
environment_fail_code: constant System.byte_ordinal := 16#4A#;
```

from_MD_code

```
from_MD_code:           constant System.byte_ordinal := 16#4B#;
```

ext_interrupt_code

```
ext_interrupt_code:     constant System.byte_ordinal := 16#4C#;
```

Memory Addresses for accessing SSM Master Micros

primary_address

```
primary_address:        constant System.ordinal := 16#2000_0000#;
```

secondary_address

```
secondary_address:      constant System.ordinal := 16#2004_0000#;
```

primary_BXU_match

```
primary_BXU_match:      constant KMDS_Defs.fourteen_bit_field :=
                          primary_address / (2**18);
```

secondary_BXU_match

```
secondary_BXU_match:    constant KMDS_Defs.fourteen_bit_field :=
                          secondary_address / (2**18);
```

BXU_mask

```
BXU_mask:               constant KMDS_Defs.fourteen_bit_field := 16#3FFF#;
```

core_status_address

```
core_status_address:    constant System.ordinal := 16#00200000#;
```

core_data_address

```
core_data_address:      constant System.ordinal := 16#00400000#;
```

data_disp

```
data_disp:              constant := 0;
```

Word displacement

status_disp

```
status_disp:            constant := 4;
```

Word displacement

echo_char_req_code

```
echo_char_req_code:             constant System.byte_ordinal := 16#09#;
```

read_rev_req_code

```
read_rev_req_code:              constant System.byte_ordinal := 16#0A#;
```

gen_OR_int_code

```
gen_OR_int_code:                constant System.byte_ordinal := 16#0B#;
```

read_status_req_code

```
read_status_req_code:           constant System.byte_ordinal := 16#18#;
```

read_config_req_code

```
read_config_req_code:           constant System.byte_ordinal := 16#19#;
```

sys_init_req_code

```
sys_init_req_code:              constant System.byte_ordinal := 16#1A#;
```

DC_cntrl_req_code

```
DC_cntrl_req_code:          constant System.byte_ordinal := 16#1B#;
```

blower_cntrl_req_code

```
blower_cntrl_req_code:      constant System.byte_ordinal := 16#1C#;
```

read_err_log_req_code

```
read_err_log_req_code:      constant System.byte_ordinal := 16#1D#;
```

read_inputs_req_code

```
read_inputs_req_code:       constant System.byte_ordinal := 16#1E#;
```

load_env_timer_req_code

```
load_env_timer_req_code:    constant System.byte_ordinal := 16#1F#;
```

read_TOD_req_code

```
read_TOD_req_code:          constant System.byte_ordinal := 16#20#;
```

write_TOD_req_code

```
write_TOD_req_code:         constant System.byte_ordinal := 16#21#;
```

write_LED_req_code

```
write_LED_req_code:         constant System.byte_ordinal := 16#22#;
```

write_watchdog_req_code

```
write_watchdog_req_code:        constant System.byte_ordinal := 16#23#;
```

config_SSM_req_code

```
config_SSM_req_code:            constant System.byte_ordinal := 16#24#;
```

read_NID_req_code

```
read_NID_req_code:              constant System.byte_ordinal := 16#25#;
```

read_TOD_RAM_req_code

```
read_TOD_RAM_req_code:          constant System.byte_ordinal := 16#26#;
```

write_TOD_RAM_req_code

```
write_TOD_RAM_req_code:         constant System.byte_ordinal := 16#27#;
```

send_to_MD_req_code

```
send_to_MD_req_code:            constant System.byte_ordinal := 16#30#;
```

config_OR_int_code

```
config_OR_int_code:             constant System.byte_ordinal := 16#41#;
```

single_init0_req_code

```
single_init0_req_code:          constant System.byte_ordinal := 16#42#;
```

single_init1_req_code

```
single_init1_req_code:        constant System.byte_ordinal := 16#43#;
```

double_init_req_code

```
double_init_req_code:         constant System.byte_ordinal := 16#44#;
```

load_COM_word_req_code

```
load_COM_word_req_code:       constant System.byte_ordinal := 16#45#;
```

basic_req_code

```
basic_req_code:               constant System.byte_ordinal := 16#46#;
```

load_mem_init_bit_code

```
load_mem_init_bit_code:       constant System.byte_ordinal := 16#47#;
```

xmit_index

```
subtype xmit_index is System.byte_ordinal range 1 .. 11;
```

Index into transmission buffers.

Maximum size of a SSM message is 11 bytes (1 stx byte + 8 data bytes + 1 parity byte + 1 etx byte).

xmit_buffer

```
type xmit_buffer is array (xmit_index) of System.byte_ordinal;
  pragma pack(xmit_buffer);
```

Transmit buffer array.

xmit_buffer_VA

```
type xmit_buffer_VA is access xmit_buffer;

  pragma ACCESS_KIND(xmit_buffer_VA, VIRTUAL);
```

reply_stx_codes

```
reply_stx_codes:  constant array(SSM_replies) of System.byte_ordinal := (
      acknowledge           => 16#00#,
      neg_acknowledge       => 16#01#,
      echo_char_reply       => 16#10#,
      read_rev_reply        => 16#11#,
      read_status_reply     => 16#28#,
      read_config_reply     => 16#29#,
      read_err_log_reply    => 16#2B#,
      read_inputs_reply     => 16#2C#,
      read_TOD_reply        => 16#2A#,
      read_NID_reply        => 16#2D#,
      read_TOD_RAM_reply    => 16#2E#);
```

request_bit_mask

```
type request_bit_mask is array(
  xmit_index range 1 .. 8) of System.byte_ordinal;

  pragma pack(request_bit_mask);
```

request_data_record

```
type request_data_record is
  record
    data_length:      System.byte_ordinal;
    reply:            SSM_replies;
    stx_code:         System.byte_ordinal;
    mask:             request_bit_mask;
  end record;

  for request_data_record use
    record
      data_length     at 0    range 0 .. 7;
      reply           at 1    range 0 .. 7;
      stx_code        at 2    range 0 .. 7;
      mask            at 3    range 0 .. 8 * 8 - 1;
    end record;
```

Request data record.

Fields:

`data_length`	Number of data bytes in request.
`reply`	Expected reply to this request.
`stx_code`	Start of TeXt byte for this request.

mask — Mask used to force reserved data bits to zero.

request_data_table

```
request_data_table:  constant array(SSM_requests) of request_data_record :=
      (echo_char       => (data_length   => 1,
                           reply         => echo_char_reply,
                           stx_code      => stx + echo_char_req_code,
                           mask          => (16#7F#, others => 0)),

       read_rev        => (data_length   => 0,
                           reply         => read_rev_reply,
                           stx_code      => stx + read_rev_req_code,
                           mask          => (others => 0)),

       gen_OR_int      => (data_length   => 0,
                           reply         => acknowledge,
                           stx_code      => stx + gen_OR_int_code,
                           mask          => (others => 0)),

       read_status     => (data_length   => 1,
                           reply         => read_status_reply,
                           stx_code      => stx + read_status_req_code,
                           mask          => (16#07#, others => 0)),

       read_config     => (data_length   => 1,
                           reply         => read_config_reply,
                           stx_code      => stx + read_config_req_code,
                           mask          => (16#07#, others => 0)),

       sys_init        => (data_length   => 1,
                           reply         => no_reply,
                           stx_code      => stx + sys_init_req_code,
                           mask          => (16#07#, others => 0)),

       DC_cntrl        => (data_length   => 1,
                           reply         => acknowledge,
                           stx_code      => stx + DC_cntrl_req_code,
                           mask          => (16#7F#, others => 0)),

       blower_cntrl    => (data_length   => 1,
                           reply         => acknowledge,
                           stx_code      => stx + blower_cntrl_req_code,
                           mask          => (16#67#, others => 0)),

       read_err_log    => (data_length   => 1,
                           reply         => read_err_log_reply,
                           stx_code      => stx + read_err_log_req_code,
                           mask          => (16#07#, others => 0)),

       read_inputs     => (data_length   => 1,
                           reply         => read_inputs_reply,
                           stx_code      => stx + read_inputs_req_code,
                           mask          => (16#07#, others => 0)),

       load_env_timer  => (data_length   => 2,
                           reply         => acknowledge,
                           stx_code      => stx + load_env_timer_req_code,
                           mask          => (16#07#, 16#7F#, others => 0)),

       read_TOD        => (data_length   => 0,
                           reply         => read_TOD_reply,
                           stx_code      => stx + read_TOD_req_code,
                           mask          => (others => 0)),

       write_TOD       => (data_length   => 8,
                           reply         => acknowledge,
```

```
                    stx_code      => stx + write_TOD_req_code,
                    mask          => (16#7F#,
                                      16#3F#,
                                      16#3F#,
                                      16#1F#,
                                      16#07#,
                                      16#1F#,
                                      16#0F#,
                                      16#7F#)),

write_LED       => (data_length   => 1,
                    reply         => acknowledge,
                    stx_code      => stx + write_LED_req_code,
                    mask          => (16#63#, others => 0)),

write_watchdog  => (data_length   => 2,
                    reply         => acknowledge,
                    stx_code      => stx + write_watchdog_req_code,
                    mask          => (16#03#, 16#7F#, others => 0)),

config_SSM      => (data_length   => 1,
                    reply         => acknowledge,
                    stx_code      => stx + config_SSM_req_code,
                    mask          => (16#77#, others => 0)),

read_NID        => (data_length   => 0,
                    reply         => read_NID_reply,
                    stx_code      => stx + read_NID_req_code,
                    mask          => (others => 0)),

send_to_MD      => (data_length   => 1,
                    reply         => acknowledge,
                    stx_code      => stx + send_to_MD_req_code,
                    mask          => (16#7F#, others => 0)),

config_OR_int   => (data_length   => 1,
                    reply         => acknowledge,
                    stx_code      => stx + config_OR_int_code,
                    mask          => (16#7F#, others => 0)),

single_init0    => (data_length   => 0,
                    reply         => acknowledge,
                    stx_code      => stx + single_init0_req_code,
                    mask          => (others => 0)),

single_init1    => (data_length   => 0,
                    reply         => acknowledge,
                    stx_code      => stx + single_init1_req_code,
                    mask          => (others => 0)),

double_init     => (data_length   => 0,
                    reply         => acknowledge,
                    stx_code      => stx + double_init_req_code,
                    mask          => (others => 0)),

load_COM_word   => (data_length   => 1,
                    reply         => acknowledge,
                    stx_code      => stx + load_COM_word_req_code,
                    mask          => (16#60#, others => 0)),

basic           => (data_length   => 0,
                    reply         => acknowledge,
                    stx_code      => stx + basic_req_code,
                    mask          => (others => 0)),


read_TOD_RAM    => (data_length   => 0,
                    reply         => read_TOD_RAM_reply,
                    stx_code      => stx + read_TOD_RAM_req_code,
                    mask          => (others => 0)),
```

```
        write_TOD_RAM    => (data_length    => 8,
                             reply          => acknowledge,
                             stx_code       => stx + write_TOD_RAM_req_code,
                             mask           => (16#7F#,
                                                16#3F#,
                                                16#3F#,
                                                16#1F#,
                                                16#07#,
                                                16#1F#,
                                                16#0F#,
                                                16#7F#)),

        load_mem_init_bit => (data_length    => 1,
                             reply          => acknowledge,
                             stx_code       => stx + load_mem_init_bit_code,
                             mask           => (16#01#, others => 0)));

pragma external;
```

Test_Support

This package allows the user to access KMDS's built-in diagnostic functions. In general, differences in implementation are masked by the test procedures provided here. For example: Test_memory_controller can be used to test any type of memory controller in any type of system, and the caller does not have to be aware of what type of memory controller is being tested. The `subtest_result` record, which is returned by all of the test procedures, has been generalized to simplify things. Some of the parameters do not make sense in all of the tests. It is up to the caller to determine what to do with the results and isolate the failure to a specific part of the module. The test procedures will run different sets of subtests depending on the type of module under test and the type of system.

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Calls

`GDP_diag_driver`
: This is a template for a GDP Diagnostics Driver procedure.

`Map_processor_ID_to_CP`
: This function maps a processor ID to a CP object.

`Set_board_LED`
: Attempts to turn the LED on the board in the given slot number on or off as indicated by LED_on.

`Set_diagnostic_mode`
: This procedure sets the system to be brought up in diagnostics mode at its next boot. It also brings all of the online memory board entries offline with in the System Configuration Table. The functional status of the stable store is not altered.

`Set_normal_mode`
: This procedure sets the system to be brought up in normal mode at its next boot. It also brings all of the offline memory board entries online with in the System Configuration Table. The functional status of the stable store and the memory modules with status of faulty, or offline_or_faulty is not altered.

`Test_BXU`
: This test makes extensive use testability features built into the VLSI component in order to test AP bus register set, timer, FRC circuits, parity checking logic, error reporting mechanisms, and the cache. These functions will be tested in the logical BXU identified by `logical ID` and `bus_ID`.

`Test_CP` Test_CP initiates execution of the CP's built-in tests. These tests are the Internal and External Self-Tests and the IOS Test. `passed` indicates whether the CP passed the tests. The `subtests` array indicates whether the individual subtests passed. If `passed` is true than all of the subtests passed.

`Test_GDP`

Test_GDP runs the given diagnostics routines on the specified GDP. In general these subroutines should address those areas of the GDP that are not covered by its self-test. `test_procs` is the virtual address to the list of test procedure to run on the selected GDP. The GDP is identified through the given `logical_ID` and `processor_select`. If `test_procs` is null then a standard test will be run. `timeout` specifies the maximum time (in milliseconds) for the test procedures to run on the GDP.

`Test_memory`

Built-in test features are used to test the memory in the range between `starting_displacement` and `ending_displacement`. The "ending_ displacement " is the address of the memory location which was tested most recently. If Test_memory failed, it is the address of the failed word. The output parameter passed is TRUE, if the test succeeded, and FALSE, if the test failed. All displacements are byte displacements, however the two least significant bits are always masked to zero, so testing must start and end on word boundaries.

`Test_memory_controller`

This procedure tests the ECC generation of the memory controllers in the memory module. This is done by loading data with `bad` ECC into memory (using special memory controller hooks), and then reading that data back via a normal memory access. Normal, memory error reporting will be disabled during this test. Since some memory modules can have more than one memory controller (one for each system bus), and BXU based memory controllers are actually on a local bus, the out parameter `controller` will indicate which memory controller failed.

`Test_private_memory`

The procedure tests the external RAM used by BXU caches. This test may only be run when the system is in the diagnostic mode of operation. `module` is the ID of the module (GDP or IO) which contains the cache RAM to be tested. If this test finds a failure `faulty_disp` will indicate which cache RAM is faulty. If no failures are found `passed` will be True.

`Virtual_ADR`

Exceptions

`unresponsive_target`

Raised when an attempt to deactivate or activate a hardware component (i.e. a processor) has failed.

`diag_target_unknown`

Raised when the necessary information pertaining to the hardware component under test cannot be retrieved.

`wrong_target_type`

Raised when a test is run on an incorrect component. For example running GDP diagnostics on memory.

`invalid_test_range`
: Raised when caller specifies a test range other than the test allows to be specified (e.g., Test_Memory_controller allows only 'bcl' or 'all_tests'

`invalid_memory_range`
: Raised when caller specifies a starting displacement bigger than the ending displacement, or when either exceeds the size of the memory module

`target_not_offline`
: Raised when a (memory) module does not have offline status before testing.

`CP_not_in_pool`
: Raised by Map_processor_ID_to_CP when CP_Mgt's pool of CPs does not contain a CP with the requested processor ID.

`interleaved_partner_not_found`
: Raised when testing a memory module requires an inter- leaved partner memory module that cannot be found.

Declarations

possible_sub_results

```
type possible_sub_results is (passed,
                              failed,
                              not_run,
                              running);
```

Enumeration Literals:

`passed`	The subtest was executed correctly.
`failed`	The subtest was executed incorrectly.
`not_run`	The subtest was not run (or does not exist).
`running`	Subtest is currently running.

test_type

```
type test_type is (memory,
                   GDP,
                   CP,
                   MCU,
                   BXU);
```

a_ok

```
a_ok:                constant integer := 0;
```

Reserved for use when there is no error.

unknown_error

```
unknown_error:       constant integer := 1;
```

Signals an undefined error.

bad_access

```
bad_access:          constant integer := -1;
```

Used for errors in accessing the registers or locations.

could_not_run_test

```
could_not_run_test:  constant integer := 2;
```

Indicates that the test code tried to run the subtest but could not complete the test for some reason. This is probably due to a lack of resources due to a conflict with another test procedure.

memory_record

```
type  memory_record is
  record
    data_word:    System.ordinal;
    tag:          KMDS_Defs.one_bit_field;
    ecc_check:    KMDS_Defs.seven_bit_field;
    spare:        KMDS_Defs.one_bit_field;
    reserved:     KMDS_Defs.seven_bit_field;
  end record;

  for memory_record use
    record
      data_word  at 0   range 0..31;
      tag        at 0   range 32..32;
      ecc_check  at 0   range 33..39;
      spare      at 0   range 40..40;
      reserved   at 0   range 41..47;
    end record;
```

Fields:

data_word	The actual 32-bit data word.
tag	The GDP-defined tag bit.
ecc_check	The check bits generated by ECC
spare	The spare bit provided by memory.
reserved	Only used to make the container size an even multiple of bytes.

memory_controller_desc

```
type memory_controller_desc is (none,
                                local_bus,
                                sys_bus0,
                                sys_bus1);
```

Enumeration Literals:

none	None of the memory controllers failed the test.
local_bus	The memory controller on local bus failed (private memory)
sys_bus0	The MCU connected to system bus 0 failed.
sys_bus1	The MCU connected to system bus 1 failed.

memory_test_types

```
type memory_test_types is (data_bus_ripple,
                           address_bus_ripple,
                           march,
                           refresh);
```

Enumeration Literals:

`data_bus_ripple`
: Ripple test of data bus.

`address_bus_ripple`
: Ripple test of address bus.

`march`
: Marching test both up and down.

`refresh`
: Array refresh test.

memory_controller_subtests

```
type memory_controller_subtests is (all_tests,
                                    bcl,
                                    memory_array_test,
                                    partial_word_update,
                                    ECC_correct_detect);
```

Enumeration Literals:

`all_tests`
: Performs all Memory_controller_subtests

`bcl`
: Test MCU/BXU AP bus interface

`memory_array_test`

`partial_word_update`

`ECC_correct_detect`
: This type describes the subtests that can be done on private memory (GDP/GENIO cache/local memory)

private_memory_subtests

```
type private_memory_subtests is (nibble,
                                 stuck_at);
```

Enumeration Literals:

`nibble`
: Checks 4 bit wide static RAMs.

`stuck_at`
: Checks for bits stuck a 0 or 1.

test_step

```
type test_step is (all_bits,
                   data,
                   ecc,
                   tag);
```

Enumeration Literals:

`all_bits`	Perform test on data, ecc, tag bits
`data`	Perform test on data portion of memory array only
`ecc`	Perform test on ecc portion of memory array only
`tag`	Perform test on tag bits only

pattern_miscompare

```
pattern_miscompare:        constant integer := 2;
```

Used for errors detected by comparing expected and actual values.

pattern_fault

```
pattern_fault:             constant integer := 3;
```

Used for the errors detected by the HW.

parity_fault

```
parity_fault:              constant integer := 4;
```

Used for tag/spare errors detected by BXU_mem hardware.

ECC_detect_fault

```
ECC_detect_fault:          constant integer := 5;
```

Used for incorrect detecting of pattern errors (i.e., should have been detected by hardware, but wasn't).

ECC_parity_detect_fault

```
ECC_parity_detect_fault:    constant integer := 6;
```

Used for incorrect detecting of tag/spare errors (i.e., should have been detected by hardware, but wasn't).

array_not_usable

```
array_not_usable:           constant integer := -2;
```

Used by controller tests when the memory array won't work for the controller test.

bi_word

```
type bi_word is
  record
    word0: System.ordinal;
    word1: System.ordinal;
  end record;

for bi_word use
  record
    word0 at 0 range 0 .. 31;
    word1 at 4 range 0 .. 31;
  end record;
```

data_types

```
type data_types is (one_byte,
                    two_bytes,
                    four_bytes,
                    eight_bytes,
                    sixteen_bytes,
                    mem_record);
```

Enumeration Literals:

one_byte	One byte data.
two_bytes	Two byte data.
four_bytes	Four byte data.
eight_bytes	Eight byte data.
sixteen_bytes	Sixteen byte data.
mem_record	Define a record which hold up to the largest (sixteen_bytes, AKA quad_word) of the memory data types.

pattern_rep

```
type pattern_rep  is
  record
    any_data_type:    System_Defs.quad_word;
    memory_record:    Test_Support.memory_record;
  end record;

  for pattern_rep use
    record
      any_data_type     at  0  range  0 .. 127;
      memory_record     at  16 range  0 ..  47;
    end record;
```

Fields:

any_data_type
: Type has space for the largest data (quad_word). It can also have smaller types. Used by memory controller test.

memory_record
: The 41-bit memory record. This is most commonly used by memory tests.

mem_subtest_result

```
type mem_subtest_result (bcl_test: boolean := false) is
  record
    sub_result:         possible_sub_results := not_run;
    failure_code:       integer := 0;
    case bcl_test is
      when false   =>
        last_displacement: System.ordinal;
        type_of_data:      data_types := mem_record;
        expected:          pattern_rep := (any_data_type => (word0 => 0,
                                                             word1 => 0,
                                                             word2 => 0,
                                                             word3 => 0),
                                           memory_record => (data_word  => 0,
                                                             tag        => 0,
                                                             ecc_check  => 0,
                                                             spare      => 0,
                                                             reserved   => 0)
        actual:            pattern_rep := (any_data_type => (word0 => 0,
                                                             word1 => 0,
                                                             word2 => 0,
                                                             word3 => 0),
                                           memory_record => (data_word  => 0,
                                                             tag        => 0,
                                                             ecc_check  => 0,
                                                             spare      => 0,
                                                             reserved   => 0)
        expected_syndrome: KMDS_Defs.seven_bit_field := 0;
        actual_syndrome:   KMDS_Defs.seven_bit_field := 0;
        controller:        memory_controller_desc;
      when true    =>
        component_status:   FT_Testing.test_results;
    end case;
  end record;

  pragma suppress(discriminant_check,mem_subtest_result);

  for mem_subtest_result use
    record
```

```
    bcl_test           at  0 range 0 ..  0;
    sub_result         at  0 range 1 ..  7;
    failure_code       at  1 range 0 .. 31;
    last_displacement  at  5 range 0 ..  31;
    type_of_data       at  9 range 0 ..   7;
    expected           at 10 range 0 .. 175;
    actual             at 32 range 0 .. 175;
    expected_syndrome  at 54 range 0 ..   7;
    actual_syndrome    at 55 range 0 ..   7;
    controller         at 56 range 0 ..   7;
    component_status   at  5 range 0 .. 63;
  end record;
```

Fields:

`bcl_test`

`sub_result` Indicates results of this subtest.

`failure_code` Indicates why test failed. Values for this field are assigned by the subtests so the same value may have different meanings for different subtests. True if all memory controllers passed.

`last_displacement`
Address of last word tested

`type_of_data` This identifies which type of data.

`expected` Data expected to be read from array.

`actual` Actual data from array that was read read.

`expected_syndrome`
The ECC syndrome bits expected.

`actual_syndrome`
The ECC syndrome bits received.

`controller` When testing controller, contains ID of controller. When testing BXU/MCU, contains ID of the bus agents.

`component_status`

mem_array_results

```
type mem_array_results   is array (memory_test_types) of mem_subtest_result;

  pragma pack(mem_array_results);
```

mem_ctlr_results

```
type mem_ctlr_results    is array (memory_controller_subtests) of mem_subtest

  pragma pack(mem_ctlr_results);
```

private_mem_results

```
type private_mem_results is array (private_memory_subtests) of mem_subtest_re
  pragma pack(private_mem_results);
```

max_number_subtests

```
max_number_subtests:     constant := 12;
```

Each GDP diagnostics routine may write data pertaining to the results of it's particular test to a diagnosis buffer. The following is the size of such an array.

module_diag_buffer_size

```
module_diag_buffer_size: constant System.ordinal := 8;
```

Maximum number of processes that are allowed to be spawned by a diagnostic process, it is an arbitrary value.

max_number_of_processes

```
max_number_of_processes: constant System.ordinal := 50;
```

The default maximum time (in milliseconds) to wait for a set of given GDP tests to complete. Tests that do not return in this time frame indicate a faulty (or dead) GDP. This value is used by procedure Test_GDP. Users of this routine may wish to specify a different timeout period as necessary.

GDP_test_timeout_period

```
GDP_test_timeout_period: constant System.ordinal := 1000;
```

The structure of the buffer used to disclose farther diagnosis of the GDP under test by the test routine. Developers of the GDP diagnostics routine may wish to use such a structure to report more meaningful messages by their routine.

diagnosis_buffer

```
type diagnosis_buffer is array (1 .. module_diag_buffer_size) of System.byte_
```

GDP_subtest_result

```
type GDP_subtest_result (bcl_test: boolean := false) is
  record
    sub_result:        possible_sub_results := not_run;
    failure_code:      integer := 0;
    completed:         boolean := false;
    test_report:       diagnosis_buffer;
  end record;

  for GDP_subtest_result use
    record
      sub_result        at  0 range 0 .. 7;
      failure_code      at  1 range 0 .. 31;
      completed         at  5 range 0 .. 31;
      test_report       at  9 range 0 .. 63;
    end record;
```

Fields:

`bcl_test`

`sub_result`	Indicates results of this subtest.
`failure_code`	Indicates why test failed. Values for this field are assigned by the subtests so the same value may have different meanings for different subtests. True if all memory controllers passed.
`completed`	Signals that the current GDP diagnostics routine is completed, when it is set to true.
`test_report`	Optional value. May be used by user's GDP test routines to write diagnostic messages (i.e. It may be used to write certain error messages).

subtest_index

```
subtype subtest_index is System.ordinal range 1 .. max_number_subtests;
```

GDP_subtest_results

```
type GDP_subtest_results is array (subtest_index) of GDP_subtest_result;

  pragma pack(GDP_subtest_results);
```

GDP_diag_routine_list

```
type GDP_diag_routine_list is array (subtest_index) of System.subprogram_type

  pragma pack(GDP_diag_routine_list);
```

diag_proc_list

```
type diag_proc_list is
  record
    num_tests:     subtest_index;
    diag_modules:  GDP_diag_routine_list;
  end record;
```

Fields:

num_tests	Number of test routine pointers in diag_modules.
diag_modules	List of pointers to GDP diagnostics routines.

diag_proc_list_VA

```
type diag_proc_list_VA is access diag_proc_list;

  pragma access_kind(diag_proc_list_VA, virtual);
```

diag_status_buffer

```
type diag_status_buffer is
  record
    done:    boolean := false;
    results: GDP_subtest_results;
  end record;
```

Fields:

done	Status of the entire GDP diagnostics routine. When set to true, it indicates that all of the given test routines have completed running on the designated GDP (normally or abnormally).
results	List of GDP diagnostics test results.

process_AD_list

```
type process_AD_list is array (1 .. max_number_of_processes) of Process_Mgt_T
```

The following data structure is used by the callers to Test_GDP to define the necessary data for this module. Using this information Test_GDP will locate the processor under test, sets it up, and runs the given diagnostics routines on it.

processor_under_test_data

```
type processor_under_test_data is
  record
    cardcage_ID: KMDS_Defs.cardcage_ID_rep := KMDS_Defs.sys;
    slot:        KMDS_Defs.slot_number := 0;
    psor:        KMDS_Defs.one_bit_field := 0;
    master:      boolean := true;
  end record;
```

Fields:

`cardcage_ID`	ID of the cardcage containing the processor board, that holds the GDP to be diagnosed.
`slot`	Slot number of the processor board in the given cardcage. Note that value of zero indicates an invalid slot number.
`psor`	The GDP to be tested.
`master`	Test the master GDP when set to true.

BXU_subtests

```
type BXU_subtests is (
    FRC,
    parity,
    error_report,
    timeout,
    LERL,
    cache_directory,
    IO_prefetch,
    extend);
```

This first group of subtests only require access to the BXU from the AP-Bus side. These tests make up the BXU BCL test.

Enumeration Literals:

`FRC`	Start self checking FRC circuits.
`parity`	Check parity detection circuits.
`error_report`	Check error reporting circuits.
`timeout`	Bus bad access timeout.
`LERL`	Check Local Error Report Line (LERL).
`cache_directory`	Check internal cache directory.
`IO_prefetch`	Check IO prefetcher (IO boards only).
`extend`	Check additional BXU logical units. I.e. IAC message support, Memory support, and etc.

BXU_subtest_result

```
type BXU_subtest_result (logical_failure: boolean := false) is
  record
    sub_result:         possible_sub_results := not_run;
    failure_code:       integer := 0;
    case logical_failure is
      when true     =>
        logical_detail:    System.ordinal;
      when false    =>
        component_status:    FT_Testing.test_results;
    end case;
  end record;

  pragma suppress(discriminant_check,BXU_subtest_result);

  for BXU_subtest_result use
    record
      logical_failure    at  0 range 0 .. 0;
      sub_result         at  0 range 1 .. 7;
      failure_code       at  1 range 0 .. 31;
      logical_detail     at  5 range 0 .. 31;
      component_status   at  5 range 0 .. 63;
    end record;
```

Fields:

`logical_failure`

`sub_result` Indicates results of this subtest.

`failure_code` Indicates why test failed. Values for this field are assigned by the subtests so the same value may have different meanings for different subtests. True if all memory controllers passed.

`logical_detail`

`component_status`

BXU_subtest_results

```
type BXU_subtest_results is array (BXU_subtests) of BXU_subtest_result;

  pragma pack(BXU_subtest_results);
```

CP_subtests

```
type CP_subtests is (
    self_test,
    IOS);
```

Enumeration Literals:

`self_test` CP's built-in internal/external self-tests.

`IOS` CP's built-in IOS Test.

other_subtest_result

```
type other_subtest_result is
  record
    sub_result:         possible_sub_results := not_run;
    failure_code:       integer := 0;
  end record;

  for other_subtest_result use
    record
      sub_result         at  0 range 1 .. 7;
      failure_code       at  1 range 0 .. 31;
    end record;
```

Fields:

`sub_result`	Indicates results of this subtest.
`failure_code`	Indicates why test failed. Values for this field are assigned by the subtests so the same value may have different meanings for different subtests. True if all memory controllers passed.

CP_subtest_results

```
type CP_subtest_results is array (CP_subtests) of other_subtest_result;

  pragma pack(CP_subtest_results);
```

GDP_diag_driver_type

```
subtype GDP_diag_driver_type is System.subprogram_type;
```

GDP_diag_driver

```
procedure GDP_diag_driver(
    subprogram:     GDP_diag_driver_type;
    result:     out GDP_subtest_result);
```

Parameters

`subprogram`

`result` Result of the GDP diagnostics.

Operation

This is a template for a GDP Diagnostics Driver procedure.

procedure <Name> (result: out GDP_subtest_result);

pragma subprogram_value(GDP_diag_driver, <Name>);

where <Name> is the user's GDP diagnostic routine procedure. `result` is the result of the test just run. The actual diagnostics routine is reponsible for updating it's fields.

The pragma indicates that <Name> provides an alternate body to the GDP_diag_driver template defined above. `result` specifies the results of the test that was just run on the GDP under test.

Map_processor_ID_to_CP

```
function Map_processor_ID_to_CP(
    psor_id:    KMDS_Defs.processor_ID)
  return CP_Mgt.CP_AD;
```

Parameters

psor_id

Return Type and Value

CP_Mgt.CP_AD

Operation

This function maps a processor ID to a CP object.

Exceptions

System_Exceptions.bad_parameter
This is raised if processor ID is null.

Set_board_LED

```
procedure Set_board_LED(
    slot:       KMDS_Defs.slot_number;
    LED_on:     boolean := false);
```

Parameters

`slot`	The slot number in the given cardcage that the board resides on.
`LED_on`	Signal to turn the LED on the subject board on (when true), or off (when false).

Operation

Attempts to turn the LED on the board in the given slot number on or off as indicated by LED_on.

Set_diagnostic_mode

```
procedure Set_diagnostic_mode;
```

Operation

This procedure sets the system to be brought up in diagnostics mode at its next boot. It also brings all of the online memory board entries offline with in the System Configuration Table. The functional status of the stable store is not altered.

Set_normal_mode

```
procedure Set_normal_mode;
```

Operation

This procedure sets the system to be brought up in normal mode at its next boot. It also brings all of the offline memory board entries online with in the System Configuration Table. The functional status of the stable store and the memory modules with status of faulty, or offline_ or_faulty is not altered.

Test_BXU

```
procedure Test_BXU(
    logical_ID:         KMDS_Defs.logical_ID_rep;
    bus_ID:             KMDS_Defs.VLSI_locations;
    passed:      out    boolean;
    subtests:    out    BXU_subtest_results;
    BCL_only:           boolean := false);
```

Parameters

`logical_ID`	Logical ID of target logical BXU.
`bus_ID`	System Bus BXU is attached to.
`passed`	If true, than all subtests were passed.
`subtests`	Array with results of the subtests.
`BCL_only`	If true, only run the BXU BCL subtests.

Operation

This test makes extensive use testability features built into the VLSI component in order to test AP bus register set, timer, FRC circuits, parity checking logic, error reporting mechanisms, and the cache. These functions will be tested in the logical BXU identified by `logical ID` and `bus_ID`.

If the parameter `BCL_only` is set to true, then only subtests which do not need to access the BXU from the local bus side will be executed.

If the system is not in Diagnostic mode then only the BCL tests can be executed. If the system is in Diagnostic mode then all of the tests can be run.

If `passed` is true then all of the subtests run on the logical BXU passed. Otherwise, (i.e. `passed` is false) one or more of the subtests failed.

Exceptions

`FT_Testing.cannot_run_test`

`SCT_Access.reserved_by_others`

`SCT_Access.not_in_SCT`

Test_CP

```
procedure Test_CP(
    logical_ID:         KMDS_Defs.logical_ID_rep;
    passed:      out    boolean;
    subtests:    out    CP_subtest_results;
    psor:               KMDS_Defs.psor_select := KMDS_Defs.psor_0);
```

Parameters

`logical_ID`	Logical ID of module containing target CP.
`passed`	If true, than all subtests were passed.
`subtests`	Array with results of the subtests.
`psor`	Processor in module to be tested.

Operation

Test_CP initiates execution of the CP's built-in tests. These tests are the Internal and External Self-Tests and the IOS Test. `passed` indicates whether the CP passed the tests. The `subtests` array indicates whether the individual subtests passed. If `passed` is true than all of the subtests passed.

Exceptions

`diag_target_unknown`

`unresponsive_target`

`wrong_target_type`

`SCT_Access.not_in_SCT`

Test_GDP

```
procedure Test_GDP(
    GDP_data:       processor_under_test_data;
    passed:     out boolean;
    status:     out diag_status_buffer;
    test_procs:     diag_proc_list_VA := null;
    timeout:        System.ordinal := GDP_test_timeout_period);
```

Parameters

GDP_data	Data used to identify and locate the processor to be tested.
passed	If true, then the GDP has passed all of the subtests.
status	Status buffer with results of the subtests.
test_procs	List of pointers to diagnostics procedures to run on the selected processor. It is to be prepared by the users of Test_GDP.
timeout	The maximum time period needed by a set of GDP diagnostics routines (given through test_procs) to execute on the GDP under test.

Operation

Test_GDP runs the given diagnostics routines on the specified GDP. In general these sub-routines should address those areas of the GDP that are not covered by its self-test. `test_procs` is the virtual address to the list of test procedure to run on the selected GDP. The GDP is identified through the given `logical_ID` and `processor_select`. If `test_procs` is null then a standard test will be run. `timeout` specifies the maximum time (in milliseconds) for the test procedures to run on the GDP.

If the output parameter `passed` is TRUE, the test succeeded, if it is FALSE the test failed. In either case, the array `subtests` will contain details about which subtests passed or failed. If `passed` is true than all of the subtests passed.

Warning

Some diagnostics routines may crash the system or inactivate the GDP under test. Thus it should be extremely important to the users of this procedure to make sure the system is prepared for such a possibility (specially in a single processor system).

Exceptions

`diag_target_unknown`

`unresponsive_target`

`SCT_Access.not_in_SCT`

`SCT_Access.reserved_by_others`

Test_memory

```
procedure Test_memory(
    slot:                          KMDS_Defs.slot_number;
    selected_test:                 memory_test_types;
    step:                          test_step;
    starting_displacement:         System.ordinal;
    ending_displacement:           System.ordinal;
    num_errors:              in out System.ordinal;
    test_result:                out mem_subtest_result;
    passed:                     out boolean);
```

Parameters

`slot` Slot number of memory module to be tested.

`selected_test`
Run this memory test.

`step` selects memory region to be tested

`starting_displacement`
Beginning displacement to start test. Default of zero means absolute low address.

`ending_displacement`
Final displacement of test. Default of zero means absolute high address.

`num_errors` (in) The maximum number of errors to detect before returning. (out) The number of errors actuall detected.

`test_result` Record with results of the test.

`passed` True if memory module passes.

Operation

Built-in test features are used to test the memory in the range between `starting_displacement` and `ending_displacement`. The "ending_displacement " is the address of the memory location which was tested most recently. If Test_memory failed, it is the address of the failed word. The output parameter passed is TRUE, if the test succeeded, and FALSE, if the test failed. All displacements are byte displacements, however the two least significant bits are always masked to zero, so testing must start and end on word boundaries.

Exceptions

`unresponsive_target`

`wrong_target_type`

`SCT_Access.not_in_SCT`

Test_memory_controller

```
procedure Test_memory_controller(
    slot:              KMDS_Defs.slot_number;
    selected_test:     Memory_controller_subtests;
    test_results:  out mem_ctlr_results;
    passed:        out boolean);
```

Parameters

`slot`	Slot number of memory module to be tested.
`selected_test`	Run this controller test.
`test_results`	Array with results of subtests.
`passed`	True if all memory controllers passed.

Operation

This procedure tests the ECC generation of the memory controllers in the memory module. This is done by loading data with `bad` ECC into memory (using special memory controller hooks), and then reading that data back via a normal memory access. Normal, memory error reporting will be disabled during this test. Since some memory modules can have more than one memory controller (one for each system bus), and BXU based memory controllers are actually on a local bus, the out parameter `controller` will indicate which memory controller failed.

Note: If a failure is detected in the first memory controller tested, the module's second memory controller will not be tested.

Note: Before this procedure calls the code to actually test a BXU- based memory controller, it first calls some BXU tests to test the memory module's BXU and AP-Bus interface.

Exceptions

`wrong_target_type`

`unresponsive_target`

`SCT_Access.not_in_SCT`

Test_private_memory

```
procedure Test_private_memory(
    logical_ID:       KMDS_Defs.logical_ID_rep;
    passed:     out   boolean;
    subtests:   out   private_mem_results);
```

Parameters

`logical_ID`	ID of module containing memory to be tested.
`passed`	If True, then memory passed all tests.
`subtests`	Array with results of the subtests.

Operation

The procedure tests the external RAM used by BXU caches. This test may only be run when the system is in the diagnostic mode of operation. `module` is the ID of the module (GDP or IO) which contains the cache RAM to be tested. If this test finds a failure `faulty_disp` will indicate which cache RAM is faulty. If no failures are found `passed` will be True.

Exceptions

`FT_Testing.cannot_run_test`

`SCT_Access.reserved_by_others`

`SCT_Access.not_in_SCT`

Virtual_ADR

```
function Virtual_ADR(
     s:  System.address)
   return diag_proc_list_VA;
```

Parameters

`s` Source value, of type `System.address`.

Return Type and Value

`diag_proc_list_VA`
Result type.

Operation

Changes an expression's Ada type.

Notes

No runtime code is generated.

FTS_Admin

Contains administrative operations for a *File Transfer Service* (FTS), including defining remote entities, changing service parameters, and retrieving service information.

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Calls

`Change_service_params`
: Changes some parameters of the active service.

`Define_account`
: Creates an account, which is allowed to be used by the specified users (name, location).

`Disable_local_entity`
: Disables one or all local entities; subsequent connection requests are rejected.

`Enable_local_entity`
: Enables one or all local entities; new connection requests can be processed through the entities.

`Force_activity_state`
: Sets the desired state for the activity identified by the request identifier.

`Get_account_info`
: Gets information about a given account.

`Get_account_names`
: Gets the names of all accounts.

`Get_activity_descriptions`
: Gets descriptions of the current activities (transfers) within the service.

`Get_local_entity_info`
: Get local entity's parameters, status, and statistics.

`Get_local_entity_titles`
: Gets the titles of all local entities.

`Get_remote_entity_info`
Gets the system definition record for a remote entity.

`Get_remote_entity_names`
Gets the local names of all remote entities.

`Get_service_info`
Gets service parameters and statistics.

`Get_transfer_info`
Gets descriptions of the current activities within the service and information about all current transfer requests.

`Register_remote_entity`
Registers a remote entity within the service.

`Switch_remote_entity`
Enables or disables a remote entity for local users.

Summary

FTS allows the local system to exchange files with registered remote systems. An FTS consists of one or more local entities, some registered remote entities, and some remote user *accounts*. One or more transfers (*activities*) may be in progress, or enqueued.

Administering the FTS

Some parameters of the service may be changed dynamically by calling `Change_service_params`. Other parameters may be changed after stopping the service (see the `Configure` utility).

`Get_service_info`
Gets information about the FTS itself, including both parameters and status.

Local Entities

Files are transferred through local entities. Local entities are identified by *title*. Currently, only one local entity is supported by FTS. There are several calls to manage local entities:

`Enable_local_entity` and `Disable_local_entity`
Enables or disables one or all local entities. Either the initiating or the responding parts or both may enabled or disabled.

`Get_local_entity_titles`
Gets a list of all local entities' titles (names).

`Get_local_entity_info`
Gets information about a local entity, given the entity's title. There are three items gotten: the local entity's parameters, and both its initiating and responding parts' statuses and statistics.

Remote Entities

Remote entities are the local system's view of remote systems. A remote system may have several remote entities in it; each entity is treated separately.

`Register_remote_entity`
: Registers a remote entity with the local FTS. The remote entity's name, address, title, type (BiiN or non-BiiN), and other parameters may be set. The parameters are read by calling `Get_remote_entity_info`.

`Switch_remote_entity`
: Enables or disables a remote entity for local users.

`Get_remote_entity_names`
: Gets a list of the registered remote entities' names.

`Get_remote_entity_info`
: Given a remote entity's name, gets its status and parameters.

File Transfers

File transfers (*activities*) occur as a result of transfer requests from the `manage.transfer` utility or the `FTS_Transfer` package's procedural interface.

`Get_activity_descriptions`
: Gets a list of activity (transfer) description records, including the associated local and remote entities, the transfer state, and the activity and transfer request IDs.

`Get_transfer_info`
: Gets the same information as `Get_activity_descriptions`, but also includes specific transfer information (last operation, local and remote filenames, transfer parameters, log file if any, and so forth).

Accounts

An *account* is used for access control for remote users. An account has a name and contains a list of allowed remote users' names and locations; these remote users, at the allowed locations, can transfer files to and from the local system. An account can also include a local login name for the remote user, a home (base) directory, and passwords for the local or public login.

`Define_account`
: Defines one account's passwords and parameters, including a list of allowed user names and locations.

`Get_account_names`
: Gets a list of the current accounts' names.

`Get_account_info`
: Given an account name, gets a list of its users' names and locations, as well as the account's local login name, local directory, and its passwords.

Warning

If a remote user uses an unknown (undefined) account, and the FTS parameter `FTS_Config_Defs.service_params_t.reject_unknown_users` is false, *there is no further protection.* The unknown remote user can access any destination account where the destination user's protection set includes the `system` ID.

Declarations

desired_activity_state_t

```
type desired_activity_state_t is (
    suspended,
    resume,
    aborted,
    process);
```

Parameter to `Force_activity_state`.

Enumeration Literals:

`suspended`	Activity is suspended until a `resume`.
`resume`	Resume a suspended activity.
`aborted`	Abort an activity being processed.
`process`	Process the activity as soon as possible.

Change_service_params

```
procedure Change_service_params(
    service:  FTS_Config_Defs.service_AD_t;
    params:   FTS_Config_Defs.service_params_t);
```

Parameters

`service`	FTS service, with control rights.
`params`	New service parameters.

Operation

Changes some parameters of the active service.

For a description of the parameters see `FTS_Config_Defs.service_params_t`.

Only those parameters for which the service need not be stopped can be changed:

- `finished_requests_deadline`
- `low_cost_time`
- `reject_anonymous_users`
- `reject_unauthorized_users`
- `reject_unknown_users`
- `suspended_deadline`
- `timed_queuing_enabled`
- `timed_requests_offset`
- `timeout_deadline`
- `trace_level`

Define_account

```
procedure Define_account(
    service:            FTS_Config_Defs.service_AD_t;
    account_name:       System_Defs.text;
    users:              FTS_Config_Defs.identifications_t;
    local_login:        System_Defs.text := System_Defs.null_text;
    public_password:    System_Defs.text := System_Defs.null_text;
    implicit_password:  System_Defs.text := System_Defs.null_text;
    home_dir:           System_Defs.text := System_Defs.null_text);
```

Parameters

`service` FTS service, with control rights.

`account_name` Account name.

If an account with the same name exists already, the existing account definition is replaced by the new definition.

`users` User names and locations allowed to use this account. If `users.name` is null, all names will match. If `user.location` is null, all locations will match.

`local_login` Optional. Local login name. If `System_Defs.null_text` (default), the local login name is the account's name.

`public_password`
Optional. Public password to be matched by remote users, rather than the account's password. If `System_Defs.null_text` (default), only the local account's password must be matched.

`implicit_password`
Optional. Password that is supplied for the local login.

`home_dir` Optional. Directory subtree to be accessible when using this account.

Operation

Creates an account, which is allowed to be used by the specified users (name, location).

The remote users allowed to use this account are specified by name and location. The locations associated with the user names must be registered as remote entities.

Exceptions

`FTS_Config_Defs.string_too_long`

Disable_local_entity

```
procedure Disable_local_entity(
    service:              FTS_Config_Defs.service_AD_t;
    title:                System_Defs.text := System_Defs.null_text;
    initiating_part:      boolean := true;
    responding_part:      boolean := true;
    wait_until_idle:      boolean := true;
    max_seconds_to_wait:  System.ordinal := 3600;
    abort_connections:    boolean := false);
```

Parameters

`service` FTS service, with control rights.

`title` Optional. Local entity's title. If `System_Defs.null_text` (default), all local entities are disabled.

`initiating_part`
Optional. If true (default), the initiating part of the local entity is disabled.

`responding_part`
Optional. If true (default), the responding part of the local entity is disabled.

`wait_until_idle`
Optional. If true (default), the local entity will not be disabled until it is idle. If false, the entity will be disabled immediately.

`max_seconds_to_wait`
Optional. Number of seconds to wait before disabling the entity.

`abort_connections`
Optional. If true, active connections through this entity are aborted. If false (default), the currently active connections will be completed normally.

Operation

Disables one or all local entities; subsequent connection requests are rejected.

Either the initiating (sending) part, or the responding (receiving) part, or both parts of the entity(s) may be disabled.

Exceptions

`FTS_Config_Defs.unknown_entity`
`title` is unknown.

`FTS_Config_Defs.string_too_long`
`title` is longer than the maximum name size.

Enable_local_entity

```
procedure Enable_local_entity(
    service:          FTS_Config_Defs.service_AD_t;
    title:            System_Defs.text := System_Defs.null_text;
    initiating_part:  boolean := true;
    responding_part:  boolean := true);
```

Parameters

`service` FTS service, with control rights.

`title` Optional. Local entity's title. If `System_Defs.null_text` (default), all local entities are enabled.

`initiating_part`
Optional. If true (default), the initiating part of the local entity is enabled.

`responding_part`
Optional. If true (default), the responding part of the local entity is enabled.

Operation

Enables one or all local entities; new connection requests can be processed through the entities.

Either the initiating (sending) part, or the responding (receiving) part, or both parts of the entity(s) may be enabled.

Exceptions

`FTS_Config_Defs.unknown_entity`

`FTS_Config_Defs.string_too_long`

Force_activity_state

```
procedure Force_activity_state(
    service:        FTS_Config_Defs.service_AD_t;
    request_id:     System.ordinal;
    desired_state:  desired_activity_state_t);
```

Parameters

`service` FTS service, with control rights.

`request_id` Request identification, from a `Get_activity_descriptions` call.

`desired_state`

Desired activity state:

`suspended`	A batched or timed batch activity is suspended until a `resume`.
`resume`	Activity processing is resumed if it was `suspended`.
`aborted`	If the activity is still queued it is marked as aborted and will never be processed. If the activity is already processed, the connection will be aborted.
`process`	The activity is forced to process as soon as possible.

Operation

Sets the desired state for the activity identified by the request identifier.

Exceptions

`System_Defs.bad_parameter`
: `request_id` is unknown.

`FTS_Config_Defs.state_invalid`
: The activity is already aborted or terminated, or `resume` was requested without a previous `suspend`, or `suspend` was requested for an active transfer.

Get_account_info

```
procedure Get_account_info(
    service:                FTS_Config_Defs.service_AD_t;
    account_name:           System_Defs.text;
    users:              out FTS_Config_Defs.identifications_t;
    local_login:        out System_Defs.text;
    public_password:    out System_Defs.text;
    implicit_password:  out System_Defs.text;
    home_dir:           out System_Defs.text);
```

Parameters

`service` FTS service, with control rights.

`account_name` Local account's name.

`users` Variable that receives a list of the users allowed to use the account.

`local_login` Variable that receives the account's local login name, if any.

`public_password`
Variable that receives the account's public password, if any.

`implicit_password`
Variable that receives the account's implicit password, if any.

`home_dir` Variable that receives the account's home directory, if any.

Operation

Gets information about a given account.

The returned `users.length` field indicates the total number of user identifications.

The caller should check if the returned `users.length` field is greater than `users.max_length`. If so, only those identifications which fit in the provided array are returned; re-allocate an array with `max_length >= users.length` and repeat the call.

Similarly, for the four returned text records, the returned `text.length` field indicates the length of each text. The caller should check if any of the returned `text.length` fields are greater than `text.max_length`. If so, only the characters which fit in the text are returned; re-allocate a text record with `max_length >= text.length` and repeat the call.

Exceptions

`FTS_Config_Defs.unknown_account`
The account does not exist.

`FTS_Config_Defs.string_too_long`

Get_account_names

```
procedure Get_account_names(
    service:      FTS_Config_Defs.service_AD_t;
    names:    out System_Defs.string_list);
```

Parameters

`service`	FTS service, with control rights.
`names`	Variable that receives a list of account names.

Operation

Gets the names of all accounts.

The returned `names.count` field indicates the total number of accounts.

The caller should check if the returned `names.length` field is greater than `names.max_length`. If so, only those names which fit in the string list are returned; re-allocate a string list record with `max_length >= names.length` and repeat the call.

Get_activity_descriptions

```
procedure Get_activity_descriptions(
    service:         FTS_Config_Defs.service_AD_t;
    activities:  out FTS_Config_Defs.activities_t);
```

Parameters

`service`	FTS service, with control rights.
`activities`	Variable that receives a list of activities (transfer requests).

Operation

Gets descriptions of the current activities (transfers) within the service.

The returned `activities.length` field indicates the total number of activity descriptions.

The caller should check if the returned `activities.length` field is greater than `activities.max_length`. If so, only those activity descriptions which fit in the provided array are returned; re-allocate an array with `max_length >= activities.length` and repeat the call.

Get_local_entity_info

```
procedure Get_local_entity_info(
    service:              FTS_Config_Defs.service_AD_t;
    title:                System_Defs.text;
    info:              out FTS_Config_Defs.entity_params_t;
    istatus:           out FTS_Config_Defs.status_t;
    rstatus:           out FTS_Config_Defs.status_t;
    reset_statistics:     boolean := false);
```

Parameters

`service`	FTS service, with control rights.
`title`	Entity's title.
`info`	Variable that receives the entity's definition.
`istatus`	Variable that receives the status and statistics of the entity's initiating part.
`rstatus`	Variable that receives the status and statistics of the entity's responding part.
`reset_statistics`	If true, statistics are reset.

Operation

Get local entity's parameters, status, and statistics.

Exceptions

`FTS_Config_Defs.unknown_entity`
`title` is not a defined local entity.

`FTS_Config_Defs.string_too_long`

Get_local_entity_titles

```
procedure Get_local_entity_titles(
    service:      FTS_Config_Defs.service_AD_t;
    titles:   out System_Defs.string_list);
```

Parameters

`service`	FTS service, with control rights.
`titles`	Variable that receives a list of local entity titles.

Operation

Gets the titles of all local entities.

The returned `titles.count` field indicates the total number of entities.

The caller should check if the returned `titles.length` field is greater than `titles.max_length`. If so, only those titles which fit in the string list are returned; re-allocate a string list record with `max_length >= titles.length` and repeat the call.

Get_remote_entity_info

```
procedure Get_remote_entity_info(
    service:      FTS_Config_Defs.service_AD_t;
    name:         System_Defs.text;
    info:     out FTS_Config_Defs.system_rec_t);
```

Parameters

`service`	FTS service, with control rights.
`name`	Remote entity's local name.
`info`	Variable that receives the entity's definition.

Operation

Gets the system definition record for a remote entity.

Exceptions

`FTS_Config_Defs.unknown_entity`
: name is not a registered remote entity.

`FTS_Config_Defs.string_too_long`
: name is too long.

Get_remote_entity_names

```
procedure Get_remote_entity_names(
    service:      FTS_Config_Defs.service_AD_t;
    names:    out System_Defs.string_list);
```

Parameters

`service`	FTS service, with control rights.
`names`	Variable that receives a list of remote entities' names.

Operation

Gets the local names of all remote entities.

The returned `names.count` field indicates the total number of remote entities.

The caller should check if the returned `names.length` field is greater than `names.max_length`. If so, only those names which fit in the string list are returned; re-allocate a string list record with `max_length >= names.length` and repeat the call.

Get_service_info

```
procedure Get_service_info(
    service:              FTS_Config_Defs.service_AD_t;
    info:              out FTS_Config_Defs.service_params_t;
    status:            out FTS_Config_Defs.status_t;
    reset_statistics:     boolean := false);
```

Parameters

`service` FTS service, with control rights.

`info` Variable that receives the service parameters.

`status` Variable that receives the status and statistics of the service.

`reset_statistics`
Optional. If true, statistics are reset for the service and for all local entities.

Operation

Gets service parameters and statistics.

Get_transfer_info

```
procedure Get_transfer_info(
    service:        FTS_Config_Defs.service_AD_t;
    activities:  out FTS_Config_Defs.activities_t;
    infos:       out FTS_Config_Defs.transfer_infos_t);
```

Parameters

`service`	FTS service, with control rights.
`activities`	Variable that receives a list of activities.
`infos`	Variable that receives a list of transfer information records.

Operation

Gets descriptions of the current activities within the service and information about all current transfer requests.

The returned `activities.length` and `infos.length` fields indicate the total number of activity descriptions and transfers.

The caller should check if either of the returned `x.length` fields is greater than `x.max_length`. If so, only those records which fit in the provided array are returned; re-allocate an array with `max_length >= x.length` and repeat the call.

Register_remote_entity

```
procedure Register_remote_entity(
    service:                FTS_Config_Defs.service_AD_t;
    name:                   System_Defs.text;
    object_id:              System_Defs.text;
    transport_address:      System_Defs.text;
    session_selector:       System_Defs.text := System_Defs.null_text;
    presentation_selector:  System_Defs.text := System_Defs.null_text;
    title:                  System_Defs.text := System_Defs.null_text;
    alien:                  boolean := true;
    time_out_deadline:      integer := 2;
    virtual_filename_spec:  boolean := false);
```

Parameters

`service` FTS service, with control rights.

`name` Remote entity's local name.

`object_id` Numeric form of the object identifier for the remote entity.

`transport_address`
Transport service part of the remote entity's presentation address.

`session_selector`
Optional. Session service part of the presentation address.

`presentation_selector`
Optional. Presentation service part of the presentation address.

`title` Optional. Remote entity's title. If `System_Defs.null_text` (default), `name` is used.

`alien` Optional. If true (default), the remote entity is on an alien (non-BiiN) system.

`time_out_deadline`
Optional. Number of minutes to wait after a request before retry or abort.

`virtual_filename_spec`
Optional. If true, remote filenames must be specified in the virtual filestore convention. If false (default), remote filenames must be specified in the convention of the remote system.

Operation

Registers a remote entity within the service.

The remote entity is initially enabled.

The *presentation address* of the remote entity is composed of:

transport address must be provided (and which is in turn composed of network address and TSAP endpoint).

session selector may be provided.

presentation selector
may be provided.

The object ID, address, and selectors are specified as a sequence of numbers (decimal byte values) separated by periods, as `47.1.0.101`

Exceptions

`FTS_Config_Defs.string_too_long`

`System_Exceptions.bad_parameter`
: The syntax of the number string is wrong. Either a component is missing, or too many components are contained, or a component has an invalid value.

Switch_remote_entity

```
procedure Switch_remote_entity(
    service:                FTS_Config_Defs.service_AD_t;
    name:                   System_Defs.text;
    enable:                 boolean;
    previously_enabled:  out boolean);
```

Parameters

`service` FTS service, with control rights.

`name` Registered remote entity's local name.

`enable` If true, the remote entity is enabled; else it is disabled.

`previously_enabled`
Variable that receives true if the remote entity was enabled before this call; false otherwise.

Operation

Enables or disables a remote entity for local users.

If a remote entity is disabled, it cannot be addressed through FTS.

Exceptions

`FTS_Config_Defs.unknown_entity`
`name` does not designate a registered remote entity.

FTS_Config_Defs

Defines types, objects, and exceptions used for the administration of the *File Transfer Service* (FTS).

Security

Access to this package is restricted to callers carrying a privileged ID. See your System Administrator for access.

Summary

This package defines service and entity objects and their parameters and status records. Remote system objects, parameters, and remote user identifications are also defined.

These definitions are used by the `FTS_Admin` package. A status record (`transfer_info_t`) from the `FTS_Transfer` package is used to declare an array of status records for the `FTS_Admin.Get_transfer_info` call.

Terms in *italics* are defined in the ISO *File Transfer, Access, and Management* protocol, `ISO 8571 FTAM`.

Exceptions

`state_invalid`
: The service is in an improper state for a requested operation.

`entity_not_idle`
: An entity to be aborted is not idle.

`already_attached`
: The ISO transport service is already attached.

`no_ts_attached`
: The ISO transport service is not yet attached.

`unknown_entity`
: An entity is not registered within the service.

`unknown_account`
: An account name is unknown within the service.

`no_entity_attached`
: No local entity has been attached to the service.

`string_too_long`
: A text parameter exceeded an implementation-defined constraint (see `subtype identifier_t`).

Declarations

service_obj_t

```
type service_obj_t is limited private;
```

service_AD_t

```
type service_AD_t is access service_obj_t;
   pragma access_kind(service_AD_t, AD);
```

entity_obj_t

```
type entity_obj_t is limited private;
```

entity_AD_t

```
type entity_AD_t is access entity_obj_t;
   pragma access_kind(entity_AD_t, AD);
```

default_syntax_t

```
type default_syntax_t is (
   binary,
   text);
```

Default syntax for data transfer.

Enumeration Literals:

`binary`	Binary data is a sequence of uninterpreted octets.
`text`	Text data is a sequence of characters, with special handling of format effectors such as <CR>.

line_t

```
subtype line_t is System_Defs.text(252);
```

Used for directory pathnames.

short_line_t

```
subtype short_line_t is System_Defs.text(124);
```

Used for transport addresses, session and presentation selectors.

identifier_t

```
subtype identifier_t is System_Defs.text(28);
```

Used for titles, remote and local entities, and remote user's names and locations.

service_params_t

```
type service_params_t is
      record
        max_initiator_connections:        System.ordinal;
        max_direct_connections:           System.ordinal;
        max_responder_connections:        System.ordinal;
        timed_requests_offset:            integer;
        low_cost_time:                    integer;
        checkpoint_strategy:              integer;
        bulk_data_size:                   integer;
        buffer_size:                      integer;
        copy_limit:                       integer;
        public_directory:                 line_t;
        FTS_directory:                    line_t;
        requeue_after_boot:               boolean;
        timed_queuing_enabled:            boolean;
        security_attribute_group:         boolean;
        private_attribute_group:          boolean;
        file_access_service_class:        boolean;
        enhanced_file_mgt_service_class:  boolean;
        unconstrained_service_class:      boolean;
        restart_functional_unit:          boolean;
        recovery_functional_unit:         boolean;
        reject_unknown_users:             boolean;
        reject_anonymous_users:           boolean;
        reject_unauthorized_users:        boolean;
        timeout_deadline:                 integer;
        finished_requests_deadline:       integer;
        suspended_deadline:               integer;
        default_syntax:                   default_syntax_t;
      end record;

for service_params_t use
  record
    max_initiator_connections       at   0 range 0 .. 31;
    max_direct_connections          at   4 range 0 .. 31;
    max_responder_connections       at   8 range 0 .. 31;
    timed_requests_offset           at  12 range 0 .. 31;
    low_cost_time                   at  16 range 0 .. 31;
    checkpoint_strategy             at  20 range 0 .. 31;
    bulk_data_size                  at  24 range 0 .. 31;
    buffer_size                     at  28 range 0 .. 31;
    copy_limit                      at  32 range 0 .. 31;
    public_directory                at  36 range 0 .. 256*8-1;
    FTS_directory                   at 292 range 0 .. 256*8-1;
    requeue_after_boot              at 548 range 0 .. 7;
    timed_queuing_enabled           at 549 range 0 .. 7;
    security_attribute_group        at 550 range 0 .. 7;
```

```
    private_attribute_group          at 551 range 0 .. 7;
    file_access_service_class        at 552 range 0 .. 7;
    enhanced_file_mgt_service_class at 553 range 0 .. 7;
    unconstrained_service_class      at 554 range 0 .. 7;
    restart_functional_unit          at 555 range 0 .. 7;
    recovery_functional_unit         at 556 range 0 .. 7;
    reject_unknown_users             at 557 range 0 .. 7;
    reject_anonymous_users           at 558 range 0 .. 7;
    reject_unauthorized_users        at 559 range 0 .. 7;
    timeout_deadline                 at 560 range 0 .. 31;
    finished_requests_deadline       at 564 range 0 .. 31;
    suspended_deadline               at 568 range 0 .. 31;
    default_syntax                   at 572 range 0 .. 15;
  end record;
```

Defines parameters for an FTS service. Some parameters may be compared against the implemented functionality.

Fields:

`max_initiator_connections`
: Maximum number of simultaneously running initiator processes (including batched and non-batched processing). The batch daemon will not be started if the number of batched connections is zero.

`max_direct_connections`
: Maximum number of connections which can be established directly, without the batcher.

`max_responder_connections`
: Maximum number of simultaneously running responder processes. If zero, the responder daemon will not be started.

`timed_requests_offset`
: Number of minutes by which the start time of a timed request must exceed the current time.

`low_cost_time`
: Hour of the day at which transfer costs are low.

`checkpoint_strategy`
: selects the checkpoint strategy:

0	No checkpointing.
1	Checkpoints between *data elements*.
2	Checkpoints between *data units*.
3	Checkpoint insertion determined by `bulk_data_size`, below.

`bulk_data_size`
: Minimum number of bytes before inserting a checkpoint. Only used if `checkpoint_strategy` is 3.

`buffer_size` Size of the *protocol data unit* (PDU) receive buffer, in bytes.

`copy_limit` Maximum size of copied data elements, in bytes. Data elements which are smaller than `copy_limit` are passed by value (copied) during encoding; other (larger) elements are passed by reference.

`public_directory`
: The directory, if any, for anonymous or unauthorized users.

`FTS_directory`
: Root directory for the FTS directories.

`requeue_after_boot`
: If true, connections interrupted by local system crash or reset should be requeued.

`timed_queuing_enabled`
: If true, timed batch transfer requests may be queued. If false, timed batch transfer requests are rejected.

`security_attribute_group`
: If true, the *security attribute* group should be supported.

`private_attribute_group`
: If true, the *private attribute* group should be supported.

`file_access_service_class`
: If true, the *file access service class* should be supported

`enhanced_file_mgt_service_class`
: If true, the *enhanced file management service class* should be supported.

`unconstrained_service_class`
: If true, the *unconstrained service* class should be supported.

`restart_functional_unit`
: If true, the *restart* functional unit should be supported.

`recovery_functional_unit`
: If true, the *recovery* functional unit should be supported

`reject_unknown_users`
: If true, *unknown* users are not allowed access. If false, unknown users are treated like anonymous users.

`reject_anonymous_users`
: If true, *anon* users are not allowed access. An accounting object must be defined for anonymous users; see `FTS_Admin.Define_account`.

`reject_unauthorized_users`
: If true, remote users who do not provide a filestore password are rejected. If false, they are treated as *unknown* users.

`timeout_deadline`
: Default deadline, in seconds, after which timeout will be raised when the remote systems keeps quiet.

`finished_requests_deadline`
: Deadline, in minutes, when information about finished requests (requests which have been completed or rejected) should be removed.

  Information about transfer requests will also be removed if `FTS_Transfer.Transfer_info` is called after transfer completion.

`suspended_deadline`
: Deadline, in minutes, after which suspended requests (not timed suspended) should be resumed. This parameter affects suspended transfers which may have been forgotten.

`default_syntax`
: Default syntax to be used for data transfer, either `binary` or `text`.

entity_params_t

```
type entity_params_t is
    record
        title:                  identifier_t;
        object_identifier:      line_t;
        max_in_connections:     System.ordinal;
        max_out_connections:    System.ordinal;
        tsap_endpoint:          System.ordinal;
        session_selector:       System.ordinal;
        presentation_selector:  System.ordinal;
      end record;

for entity_params_t use
  record
    title                 at   0 range 0 .. 32*8-1;
    object_identifier     at  32 range 0 .. 256*8-1;
    max_in_connections    at 288 range 0 .. 31;
    max_out_connections   at 292 range 0 .. 31;
    tsap_endpoint         at 296 range 0 .. 31;
    session_selector      at 300 range 0 .. 31;
    presentation_selector at 304 range 0 .. 31;
  end record;
```

Defines the information necessary to create a *local entity*.

Fields:

`title` Local entity's title.

`object_identifier`
Local entity's object identifier.

`max_in_connections`
Maximum number of remotely requested connections allowed through this entity.

`max_out_connections`
Maximum number of locally requested connections allowed through this entity.

`tsap_endpoint`
Desired *transport service access point* (TSAP) endpoint (one of the well-known endpoints). The well-known FTS endpoint is `97`. The value `0` requests dynamic assignment of the TSAP endpoint (no well-known TSAP endpoint available).

`session_selector`
Local entity's *session selector*. The value `0` means no session selector.

`presentation_selector`
Presentation selector. The value `0` means no presentation selector.

system_rec_t

```
type system_rec_t is
    record
        enabled:                boolean;
        defined:                boolean;
        title:                  identifier_t;
        object_identifier:      short_line_t;
        transport_address:      short_line_t;
        session_selector:       short_line_t;
        presentation_selector:  short_line_t;
        alien:                  boolean;
        timeout_deadline:       integer;
        virtual_filename_spec:  boolean;
        private_attr:           boolean;
  end record;

  for system_rec_t use
    record
      enabled               at   0 range 0 .. 7;
      defined               at   1 range 0 .. 7;
      title                 at   4 range 0 .. 32*8-1;
      object_identifier     at  36 range 0 .. 128*8-1;
      transport_address     at 164 range 0 .. 128*8-1;
      session_selector      at 292 range 0 .. 128*8-1;
      presentation_selector at 420 range 0 .. 128*8-1;
      timeout_deadline      at 548 range 0 .. 31;
      alien                 at 552 range 0 .. 7;
      virtual_filename_spec at 553 range 0 .. 7;
      private_attr          at 554 range 0 .. 7;
    end record;
```

Defines a *remote entity.*

Fields:

`enabled` If true, this entity definition can be used in addressing a remote entity.

`defined` If true, this definition has been explicitly defined with `FTS_Admin.Register_remote_entity`. If false, this definition has been created by FTS due to a request from a unknown remote system.

`title` FTAM application entity title of the remote entity.

`object_identifier`
Object identifier of the remote entity.

`transport_address`
Transport service part of the presentation address for the remote entity.

`session_selector`
Session service part of the presentation address for the remote entity.

`presentation_selector`
Presentation service part of the presentation address for the remote entity.

`alien` If true, the remote entity resides on an alien (non-BiiN) system.

`timeout_deadline`
Number of minutes to wait after a request before retry or abort. `0` indicates that the default deadline (`service_params_t.timeout_deadline`) for the service is used.

`virtual_filename_spec`
If true, FTS has to provide the filename in FTAM *virtual filestore*

convention. If false, in the convention of the remote system to the remote entity.

`private_attr` If true, there is a special use of the *private* attribute.

status_t

```
type status_t is
    record
        enabled:                          boolean;
        nr_of_entities:                   System.ordinal;
        nr_of_activities:                 System.ordinal;
        nr_of_connections:                System.ordinal;
        requests_queued:                  System.ordinal;
        total_number_of_connections:      System.ordinal;
        total_number_of_local_requests:   System.ordinal;
        total_number_of_remote_requests:  System.ordinal;
        rejected_remote_requests:         System.ordinal;
        recovered_connections:            System.ordinal;
        aborted_connections:              System.ordinal;
        number_of_timeouts:               System.ordinal;
      end record;
```

Contains status and statistics information for either an entity or the total service. If a record of this type is retrieved for the service the component values are related to the service. If a record of this type is retrieved for a specific entity the component values are related to that entity only.

Fields:

`enabled` If true, either the entity is allowed to accept requests or the service has been started.

`nr_of_entities`
For the service, the number of local entities.

`nr_of_activities`
Current number of activities.

`nr_of_connections`
Number of currently active connections.

`requests_queued`
Number of currently queued requests.

`total_number_of_connections`
Total number of connections established since the service was started or statistics were reset.

`total_number_of_local_requests`
Total number of local connection requests since the service was started or statistics were reset.

`total_number_of_remote_requests`
Total number of remote connection requests since the service was started or statistics were reset.

`rejected_remote_requests`
Number of remote requests which had to be rejected because the maximum number of connections already existed.

`recovered_connections`
Number of connections which have been recovered since the service was started or statistics were reset.

`aborted_connections`
: Number of connections which have been aborted since the service was started or statistics were reset.

`number_of_timeouts`
: Number of timeouts which have occurred since the service was started or statistics were reset.

activity_description_t

```
type activity_description_t is
    record
        initiating_entity:  boolean;
        daemon:             boolean;
        state:              FTS_Transfer.transfer_state_t;
        remote_entity:      identifier_t;
        local_entity:       identifier_t;
        time:               System_Defs.system_time_units;
        activity_id:        System.ordinal;
        request_id:         System.ordinal;
        recovered:          boolean;
      end record;
```

Describes an activity (a transfer) within the service.

Fields:

`initiating_entity`
: If true, the activity is handled by an initiating entity. If false, by the responding entity.

`daemon`
: If true, this activity is handled by a daemon.

`state`
: Activity's current state.

`remote_entity`
: Local name of the involved remote entity.

`local_entity`
: Title of the involved local entity.

`time`
: Meaning depends on the activity's state:

`queued`	Submitting time.
`timed_queued`	Desired start time.
`suspended`	Time when the request becomes active.
`timed_suspended`	Resume time.
`requesting`	Actual start time.
`transferring`	Actual start time.
`transferred`	Actual start time.
`terminated`	Time of completion.
`aborted`	Time of abortion.
`rejected`	Time of rejection.

`activity_id`
: Unique activity identifier for one pair of local and remote entities.

`request_id`
: Internal request identifier, unique within the service.

recovered — If true, this activity has been recovered.

activity_descriptions_t

```
type activity_descriptions_t is array(positive range <>) of
    activity_description_t;
```

Array of activity description records, used in activities_t, below.

activities_t

```
type activities_t(
  max_length:  integer) is
  record
    length:  integer;
    value:   activity_descriptions_t(1..max_length);
  end record;
```

List of activities, gotten by FTS_Admin.Get_activity_descriptions or FTS_Admin.Get_transfer_info.

Fields:

max_length	Maximum number of activities.
length	Actual number of activities.
value	Array (list) of max_length activity records.

transfer_info_list_t

```
type transfer_info_list_t is array(positive range <>) of
    FTS_Transfer.transfer_info_t;
```

Array of transfer information records, used in transfer_infos_t, below.

transfer_infos_t

```
type transfer_infos_t(
  max_length:  integer) is
      record
        length:  integer;
        value:   transfer_info_list_t(1..max_length);
      end record;
```

List of transfer information records, gotten by FTS_Admin.Get_transfer_info.

Fields:

max_length	Maximum number of transfers.
length	Actual number of transfers.

value	Array (list) of max_length transfer information records.

identification_t

```
type identification_t is
      record
        name:      identifier_t;
        location:  identifier_t;
      end record;
```

Remote users are identified by their initiator names and locations.

Remote users' access to the local filestore is controlled through a local account which has a list of valid remote user identifications.

Fields:

name	Initiator name.
location	Application title of the remote filestore.

identification_list_t

```
type identification_list_t is array(natural range <>) of
    identification_t;
```

Array of user identification records, used in identifications_t, below.

identifications_t

```
type identifications_t(
  max_length:  integer) is
      record
        length:  integer;
        value:   identification_list_t(1..max_length);
      end record;

private
```

List of remote user/location pairs. Given to FTS_Admin.Define_account and gotten from FTS_Admin.Get_account_info.

Fields:

max_length	Maximum number of identifications.
length	Actual number of identifications.
value	Array (list) of max_length identification records.

FTS_Transfer

Provides the procedural interface of the *File Transfer Service* (FTS), including transfer operations.

Calls

`Abort_transfer`
: Aborts a transfer.

`Resume_transfer`
: Restarts a suspended transfer.

`Suspend_transfer`
: Suspends a batched data transfer either until a `Resume_transfer` call, or for a given duration.

`Transfer`
: Initiates a file transfer, returning the transfer ID.

`Transfer_info`
: Gets information about a transfer, including the current state of the transfer.

Summary

FTS implements the ISO *File Transfer, Access, and Management* protocol, `ISO 8571 FTAM`. *Italicized* terms are defined in that protocol.

This package provides operations for:

- transferring files between the local system and remote systems,
- suspending and resuming a transfer,
- aborting a transfer,
- retrieving transfer status information.

Transferring a File

The only information needed for a `Transfer` call is the local and remote filenames.

Optionally, you may specify whether the source file is to be removed after the transfer, what to do if the destination file exists, the type of processing (synchronous, asynchronous, batch, or timed batch), passwords for the remote filestore, whether a log file is used, and an event to be signalled upon transfer completion.

A file transfer may result in the loss of some file attributes, if either the local or the remote filestore does not support those attributes. For example, an indexed file may be transferred as a sequential file.

Types of File Transfers

There are four ways of processing a transfer request:

- synchronously within the current job,
- asynchronously within the current job,
- in the batch queue, to be executed as soon as possible,
- in the batch queue, with a given starting time.

Suspending and Resuming a Transfer

A transfer in the (timed) batch queue may be suspended by calling `Suspend_transfer`. The suspension may be indefinite (until a `Resume_transfer` call) or timed. A timed suspended transfer is automatically resumed after the given duration.

A transfer in progress cannot be suspended.

When a batch or timed batch transfer is suspended, it may be delayed past its scheduled start time. If a timed batch transfer is timed suspended, the suspension time is added to the transfer's start time.

Aborting a Transfer

A transfer can be aborted at any time with `Abort_transfer`. When a transfer is aborted, the associated connection, if any, is closed. A batch transfer that has not been started is removed from the queue.

Aborted transfers cannot be resumed.

Transfer Information

The status of a transfer is gotten by calling `Transfer_info`. The status record may be retrieved only once for a completed transfer. The status record contains information about the transfer, including the last transfer operation, the transfer state, reason for abort if any, direction of transfer, local and remote filenames, starting time, and transfer duration.

For synchronous transfers, non-recoverable errors are indicated by exceptions. For other types of transfers, non-recoverable errors are indicated by the `abort_reason` field with an incident code corresponding to the exception.

If logging was enabled for a transfer, detailed information about the transfer is contained in the given log file.

Exceptions

`append_not_supported`
: The append operation on an existing destination file is not supported by the remote filestore.

`access_control_not_supported`
: Attribute access control is not supported by the remote filestore.

`transfer_aborted`
: The connection broke and could not be recovered.

`destination_file_already_exists`
: The destination file already exists and the file should not be extended or overwritten (see `Transfer` parameter `if_file_exists`).

`destination_file_busy`
: The destination file is currently in use and may not be written to concurrently.

`document_type_not_applicable`
: The specified document type cannot be associated with the file; the document type may be too complex.

`document_type_not_supported`
: The required document type is unknown or is not supported by the remote filestore.

`document_type_not_supported_locally`
: The required document type is not supported by the local filestore.

`document_type_unknown`
: A document type number has been specified for which no document type is defined locally.

`illegal_remote_filename`
: A filename was specified in an incorrect format. For example, it could not be divided in filestore and filename components.

`illegal_start_time`
: The specified start time has already passed.

`illegal_transfer_id`
: A given ID is not a transfer ID. This exception may be raised by every operation, and is therefore not mentioned explicitly.

`insufficient_functionality`
: The remote filestore is not able to read or write a file as requested.

`insufficient_permission_on_source`
: The user does not have permission to read the source file. If the source file is a local file, it may not exist.

`insufficient_permission_for_deletion`
: The user does not have permission to delete a file, either the destination file prior to creation, or the source file after transfer.

`insufficient_permission_for_creation`
: The user does not have permission to create the destination file.

`insufficient_rights`
: A given transfer ID does not have the appropriate rights for invoking the operation. This may be raised by every operation, hence not mentioned.

`protocol_error`
: A protocol error by the remote system has been encountered.

`remote_filestore_keeps_quiet`
: The remote filestore did not acknowledge a connection request within a given amount of time.

`remote_filestore_unknown`
: No information is available about the addressed remote filestore.

`source_file_busy`
: The source file is currently in use and may not be read concurrently.

`source_file_not_exist`
: The source file does not exist.

`suspend_rejected`
: A requested suspend operation could not be applied successfully to a transfer.

`transfer_already_suspended`
: A given transfer has already been suspended.

`transfer_completed`
: A given transfer cannot be suspended because it has already completed.

`transfer_not_suspended`
: A transfer to be resumed was not suspended.

`transfer_timed_suspended`
: A transfer cannot be directly resumed because it was suspended for a given duration.

`transfer_rejected_locally`
: A transfer request was rejected by the local system due to resource exhaustion.

`transfer_rejected_remotely`
: A transfer request was rejected by the remote filestore without any specific comment.

`unacceptable_request`
: A transfer was requested for two remote files. Only one of the filenames may be a remote filename.

`unknown_location`
: A location specified in an access control condition is unknown.

`unsupported_parameter_value`
: A parameter value is not supported locally. For example, too many conditions for access control in `Transfer`, or a filename is too long.

`user_locally_unacceptable`
: The caller is not allowed to use the transfer operation.

`user_remotely_unacceptable`
: The caller of the transfer operation (the initiator) is not allowed to access files on the remote filestore.

Declarations

transfer_object_t

```
type transfer_object_t is limited private;
```

Represents a transfer.

transfer_id_t

```
type transfer_id_t is access transfer_object_t;

      pragma access_kind(transfer_id_t,AD);
```

ID of a transfer.

info_rights

```
info_rights:  constant Object_Mgt.rights_mask :=
    Object_Mgt.use_rights;
```

Required to get information about the status of a transfer (`Transfer_info` call).

control_rights

```
control_rights:  constant Object_Mgt.rights_mask :=
    Object_Mgt.control_rights;
```

Required to invoke an operation concerning the transfer (`Suspend_transfer`, `Resume_transfer`, and `Abort_transfer` calls).

access_set_t

```
type access_set_t is (
   read,
   write,
   read_attributes,
   delete,
   create);
```

Enumerates the possible actions on a file during a file transfer.

Enumeration Literals:

`read` Read the file.

`write` Write the file.

`read_attributes` Read the file's attributes.

`delete`	Delete the file.
`create`	Create the file.

access_t

```
type access_t is array(access_set_t) of boolean;
```

Array of access conditions. A true element indicates that the associated (indexing) access operation is allowed. See also `type access_passwords_t`.

identifier_t

```
subtype identifier_t is System_Defs.text(28);
```

Container for short texts, such as passwords, object identifiers, and titles.

null_identifier

```
null_identifier:  constant identifier_t := (
    28, 0, (others => ' '));
```

Null or default identifier.

line_t

```
subtype line_t is System_Defs.text(252);
```

Container for long texts, such as local and remote filenames.

access_passwords_t

```
type access_passwords_t is array(access_set_t) of identifier_t;
```

Array of access passwords. Each element is a password which may be necessary for the corresponding access.

condition_t

```
type condition_t is
  record
      action_list:  access_t := (others => false);
      identity:     identifier_t := null_identifier;
      passwords:    access_passwords_t := (others => null_identifier);
      location:     identifier_t := null_identifier;
  end record;
```

File access condition record. Each of the four components must be satisfied to fulfill the condition.

Fields:

`action_list`	Types of access which are allowed by this access condition.
`identity`	Accessor identity to be matched, if any.
`passwords`	Passwords to be matched, if any, for each allowed type of access.
`location`	Accessor location (filestore name) to be matched, if any.

condition_array_t

```
type condition_array_t is array(natural range <>) of condition_t;
```

Array of access condition records, used in `access_control_t`, below.

access_control_t

```
type access_control_t(
  maxnr:  natural) is
    record
      actlen:  natural;
      value:   condition_array_t(1..maxnr);
  end record;
```

Array of access control conditions under which access to a file is allowed. Access to a file is allowed if at least one of these conditions (which in turn consist of four terms, all of which have to be matched) is satisfied.

Fields:

`maxnr`	Maximum number of access control conditions.
`actlen`	Number of valid access control conditions in `value` array.
`value`	Array of `maxnr` access control conditions.

default_access_control

```
default_access_control: access_control_t(0) :=(
    0, 0, (others => (
              (others=>false),
              null_identifier,
              (others => null_identifier),
              null_identifier)));
```

Default (null) access control value.

MAX_CONDITIONS

```
MAX_CONDITIONS:  constant integer := 4;
```

Maximum number of conditions which can be specified with a `Transfer` call.

document_type_t

```
subtype document_type_t is integer range 0..127;
```

A document type describes the file model and the file contents in a standardized manner, hence allowing the file characteristics to be maintained in the remote filestore.

Some document types are predefined by FTAM, others will be defined successively by standardization organisations. Therefore, the following list may be extended in later versions of this package. Other values may be used when they are implemented in FTS.

FTAM_1

```
FTAM_1:  constant document_type_t := 1;
```

Unstructured text file.

FTAM_2

```
FTAM_2:  constant document_type_t := 2;
```

Sequential text file.

FTAM_3

```
FTAM_3:  constant document_type_t := 3;
```

Unstructured binary file.

FTAM_4

```
FTAM_4:  constant document_type_t := 4;
```

Sequential binary file.

processing_t

```
type processing_t is (
  synchronous,
  asynchronous,
  batch,
  timed_batch);
```

Possible modes of file transfer processing.

Enumeration Literals:

`synchronous`	A `Transfer` call returns only after the transfer is complete or aborted.
`asynchronous`	A `Transfer` call executes concurrently.
`batch`	A transfer request is put into the batch queue.
`timed_batch`	A transfer request is put into the batch queue for execution at a specified time.

if_file_exists_t

```
type if_file_exists_t is (
  error,
  overwrite,
  append);
```

Action to be taken when the destination file already exists.

Enumeration Literals:

`error`	An error will be signaled.
`overwrite`	The existing destination file will be overwritten.
`append`	The new data will be appended to the existing file.

duration_t

```
type duration_t is
  record
    hours:    System.byte_ordinal range 0 .. 255;
    minutes:  System.byte_ordinal range 0 .. 59;
    seconds:  System.byte_ordinal range 0 .. 59;
  end record;
```

An amount of time, used for timed suspensions and for timing transfers.

Fields:

`hours`	Number of hours.
`minutes`	Number of minutes.
`seconds`	Number of seconds.

time_t

```
type time_t is
  record
    month:          System.byte_ordinal range 0..12;
    day:            System.byte_ordinal range 0..31;
    hour:           System.byte_ordinal range 0..23;
    minute:         System.byte_ordinal range 0..59;
    second:         System.byte_ordinal range 0..59;
    relative_time:  boolean;
  end record;
```

Describes a time (such as the start time of batch request), either relative to the current time or absolute.

Fields:

`month`	Number of month (January = 1, ..., December = 12).
`day`	Day of the month.
	Day 0 indicates that no start time has been specified.
`hour`	Hour of the day, in 24-hour format.
`minute`	Minute of the hour.
`second`	Second of the minute.
`relative_time`	If true, the specified time is relative to the time when the request was invoked. If false, the specified time is absolute.

no_start_time

```
no_start_time: constant time_t := (0,0,0,0,0,false);
```

Null or default start time.

operation_t

```
type operation_t is (
  Transfer,
  Suspend_transfer,
  Resume_transfer,
  Abort_transfer,
  Transfer_state);
```

Enumerates the possible transfer operations. Used in the transfer information record (see `transfer_info_t`).

Enumeration Literals:

`Transfer`	The last operation was a `Transfer` call.
`Suspend_transfer`	The last operation was a `Suspend_transfer` call.

`Resume_transfer`
The last operation was a `Resume_transfer` call.

`Abort_transfer`
The last operation was a `Abort_transfer` call.

`Transfer_state`
The last operation was a `Transfer_info` call.

transfer_state_t

```
type transfer_state_t is (
  queued,
  timed_queued,
  suspended,
  timed_suspended,
  requesting,
  transferring,
  transferred,
  terminated,
  rejected,
  aborted);
```

Enumerates the possible states of a transfer. Used in the transfer information record (see `transfer_info_t`).

Enumeration Literals:

`queued` The transfer is in the batch queue.

`timed_queued` The transfer is in the timed batch queue.

`suspended` The transfer has been suspended.

`timed_suspended`
The transfer has been suspended for a given duration.

`requesting` Connection requests are due to be issued.

`transferring` The transfer is in progress.

`transferred` The file has been completely transferred.

`terminated` The connection has been terminated after a successful transfer.

`rejected` The transfer request has been rejected.

`aborted` The transfer and the connection have been aborted.

transfer_info_t

```
type transfer_info_t is
  record
    last_operation:          operation_t;
    state:                   transfer_state_t;
    abort_reason:            Incident_Defs.incident_code;
    diagnostic:              System.short_ordinal;
    processing:              processing_t;
    attributes_altered:      boolean;
    filename_truncated:      boolean;
    remote_to_local:         boolean;
    remote_filestore:        identifier_t;
    remote_file:             line_t;
    local_file:              line_t;
```

```
    access_control:             access_control_t(MAX_CONDITIONS);
    erase_source:               boolean;
    if_file_exists:             if_file_exists_t;
    destination_file_existed:   boolean;
    document_type:              document_type_t;
    document_type_simplified:   boolean;
    access_passwords:           access_passwords_t;
    logging_enabled:            boolean;
    logfile:                    line_t;
    start_time:                 time_t;
    transferred_bytes:          integer;
    transfer_time:              duration_t;
  end record;
```

Transfer information record gotten by `Transfer_info`.

Fields:

`last_operation`
: Last operation which has been performed for this transfer request.

`state`
: Transfer state.

`abort_reason`
: Reason for a transfer being aborted, for asynchronous, batched, and timed batch transfers.

`diagnostic`
: Last diagnostic code according to *FTAM Diagnostic Definitions (Part III)*.

`processing`
: Processing mode for this transfer.

`attributes_altered`
: If true, the attributes of the destination file have been altered due to filestore restrictions.

`filename_truncated`
: If true, the remote filename has been truncated by the remote filestore due to restrictions on filename length.

`remote_to_local`
: If true, the transfer request is from remote to local filestores. If false, the transfer is from local to remote filestores.

`remote_filestore`
: Remote filestore's name.

`remote_file`
: Remote file's name. May have been truncated (when `filename_truncated` is true).

`local_file`
: Local file's pathname.

`access_control`
: Access control conditions for this transfer request. If the `Transfer` call's `dest_access_control` parameter was specified, contains the actually requested access controls.

  Values will be contained in this record only if the transfer ID given to the `Transfer_info` operation has control rights.

`erase_source`
: If true, the source file is to be erased after the transfer (a file move).

`if_file_exists`
: Action to be taken if the destination file exists.

`destination_file_existed`
: If true, the creation of a new file caused the deletion of a previously existing file.

`document_type`
: Actual document type name/number.

`document_type_simplified`
: If true, the original document type was unacceptable to the destination filestore and had to be simplified.

`access_passwords`
: Access control passwords for this transfer request.

  Values will be contained in this record only if the transfer ID given to the `Transfer_info` operation has control rights.

`logging_enabled`
: If true, transfer logging is performed.

`logfile`
: Transfer log file's pathname. Only valid if `logging_enabled` is true.

`start_time`
: Actual transfer start time.

`transferred_bytes`
: Number of bytes successfully transferred.

`transfer_time`
: Total amount of time needed for the file transfer.

transfer_info_VA_t

```
type transfer_info_VA_t is access transfer_info_t;

  pragma access_kind(transfer_info_VA_t,virtual);
```

Virtual address of a transfer information record.

Abort_transfer

```
procedure Abort_transfer(
    transfer_id:  transfer_id_t);
```

Parameters

`transfer_id` ID of transfer to be aborted.

Operation

Aborts a transfer.

A transfer cannot be resumed after it has been aborted. If a transfer is still in the batch queue, the transfer request is removed from the queue. Even a suspended transfer can be aborted.

Exceptions

`transfer_aborted`
: The transfer was already aborted.

`transfer_completed`
: The transfer was already completed.

Resume_transfer

```
procedure Resume_transfer(
    transfer_id:  transfer_id_t;
    restart:      boolean := false);
```

Parameters

`transfer_id`	ID of suspended transfer to be resumed.
`restart`	Optional. If true, the transfer should be restarted at the beginning of the file. If false (default), the transfer is continued at the last acknowledged checkpoint.

Operation

Restarts a suspended transfer.

Exceptions

`insufficient_rights`

`remote_filestore_unknown`

`remote_filestore_keeps_quiet`

`transfer_aborted`

`transfer_not_suspended`

`transfer_timed_suspended`
A `timed_suspended` transfer cannot be resumed; it will be automatically resumed after the given delay.

`transfer_completed`

Suspend_transfer

```
procedure Suspend_transfer(
    transfer_id:      transfer_id_t;
    timed_resume:     boolean := false;
    suspension_time:  duration_t := (0,1,0) );
```

Parameters

`transfer_id` ID of transfer to be suspended.

`timed_resume` Optional. If true, the transfer is to be resumed automatically after `suspension_time` has expired. If false (default) the user has to restart the transfer explicitly with a `Resume_transfer` call.

`suspension_time`
Optional. Duration after which the suspended transfer is automatically continued. The default suspension time is one minute.

Ignored if `timed_resume` is false.

Operation

Suspends a batched data transfer either until a `Resume_transfer` call, or for a given duration.

Suspending a batch transfer affects its start time.

Exceptions

`transfer_aborted`

`transfer_already_suspended`

`transfer_completed`

`insufficient_rights`

`suspend_rejected`
The local FTS has determined that the transfer cannot be suspended.

Transfer

```
function Transfer(
    source_file:          System_Defs.text;
    destination_file:     System_Defs.text;
    document_type:        document_type_t := 0;
    dest_access_control:  access_control_t := default_access_control;
    erase_source:         boolean := false;
    if_file_exists:       if_file_exists_t := error;
    processing:           processing_t := asynchronous;
    start_time:           time_t := no_start_time;
    access_passwords:     access_passwords_t := (others => null_identifier);
    filestore_password:   identifier_t := null_identifier;
    remote_account:       identifier_t := null_identifier;
    logging_enabled:      boolean := false;
    logfile:              System_Defs.text := System_Defs.null_text;
    terminate_action:     Event_Mgt.action_record := Event_Mgt.null_action)
  return transfer_id_t;
```

Parameters

`source_file` Source file's pathname.

`destination_file`
Destination file's pathname.

`document_type`
Optional. Document type to be associated with the file and used for the transfer. A value of `0` (default) indicates the `unknown` document type.

`dest_access_control`
Optional. If the source file is created, specifies access conditions for the newly created file. Default is no restrictions.

`erase_source` Optional. If true, the source file is erased after the transfer (a file move). If false (default), the source file is not erased (a file copy).

`if_file_exists`
Optional. Action to be taken if the destination file already exists. Possible values are `error`, `overwrite`, and `append`.

`processing` Optional. Type of data transfer, either `synchronous`, `asynchronous`, `batch`, or `timed_batch`.

`start_time` Optional. Transfer starting time for timed batch requests. If `no_start_time` (default), FTS uses the service's default `low_cost_transfer_time`.

Ignored if `processing` is not `timed_batch`.

`access_passwords`
Optional. Set of access passwords which may be necessary for the remote filestore to permit operation.

`filestore_password`
Optional. Remote login's password.

`remote_account`
Optional. Remote account name. The default is the caller's login name.

`logging_enabled`
Optional. If true, detailed transfer information is written to a log file.

`logfile` Optional. Log file's pathname. If `System_Defs.null_text` (default), standard output is used.

Ignored if `logging_enabled` is false.

`terminate_action`
Optional. Action to be signalled on transfer termination.

Ignored if `processing` is `synchronous`, `batch`, or `timed_batch`.

Return Type and Value

`transfer_id_t`
Transfer request ID.

Operation

Initiates a file transfer, returning the transfer ID.

The caller defines the transfer direction by specifying the remote filename either as the source or as the destination.

The transfer ID is used for subsequent operations (such as `Suspend_transfer`).

The local filename is specified as usual (for example, `/my_dir/local_file`). The remote filename must be of the form `/FTS/`*remote_system_name*`/`*remote_filename*.

For synchronous transfers (`processing => synchronous`), all errors are reported by exceptions. In the other types of processing, errors concerning parameters are reported by exceptions. For asynchronous and batch transfers, errors occuring during the transfer are only reported in the `transfer_info_t.abort_reason` field.

If logging is enabled (`logging_enabled => true`), error messages and information about the transfer are written to the given log file (default is standard output). The error messages are similar to the exception names and may be more detailed.

The caller may request to be notified by an event (`terminate_action`) about normal or abnormal completion of an non-synchronous transfer. If synchronous processing is requested, any action record is ignored.

FTAM introduced document types to describe classes of files and file contents (see subtype `document_type_t`). Several document types have been defined, and new document types may be defined later. Not all of the defined document types are supported by FTS. If no document type is specified (`document_type => 0`), FTS chooses the actual document type of the file which is to be transferred; otherwise, the given document type is used. The specified document type must be either the same as the actual document type of the file, or a simplification; otherwise the exception `document_type_not_applicable` is raised. The document type actually used can be found in the transfer information record.

Exceptions

`access_control_not_supported`

`append_not_supported`

`destination_file_already_exists`

`destination_file_busy`

`document_type_not_applicable`

document_type_not_supported

document_type_not_supported_locally

document_type_unknown

illegal_remote_filename

illegal_start_time

insufficient_permission_on_source

insufficient_permission_for_deletion

insufficient_permission_for_creation

insufficient_functionality

protocol_error

remote_filestore_keeps_quiet

remote_filestore_unknown

source_file_busy

source_file_not_exist

transfer_aborted

transfer_rejected_locally

An asynchronous transfer cannot be performed due to a restricted number of direct requests.

transfer_rejected_remotely

unknown_location

unsupported_parameter_value

unacceptable_request

Two remote or two local filenames were specified.

user_locally_unacceptable

user_remotely_unacceptable

Transfer_info

```
procedure Transfer_info(
    transfer_info_VA:  transfer_info_VA_t;
    transfer_id:       transfer_id_t);
```

Parameters

`transfer_info_VA`
: Virtual address of a transfer information record which receives the information.

`transfer_id`
: ID of transfer for which information is retrieved.

Operation

Gets information about a transfer, including the current state of the transfer.

If the transfer has been completed (perhaps unsuccessfully), the transfer information can only be retrieved once.

Exceptions

`transfer_completed`
: The transfer information record has already been retrieved.