Release Notes for MMS-EASE *Lite*

MMS-LITE-801-001

MMS-LITE-802-001

Revision 16

Systems Integration Specialists Company, Inc.
6605 19½ Mile Road, Sterling Heights, MI 48314-1408, U.S.A.
Tel: +586-254-0020, Fax: +586-254-0053
E-mail: support@sisconet.com, URL: http://www.sisconet.com

Printed in U.S.A.

02/2007

DWG #000409/2.5

New and Changed Software Features 1

Software Corrections 6

General Application Upgrade Notes for MMS-EASE *Lite* 8

IEC 61850 GSE Management Data and Functions 8

GSE Offset Request Data Structure 8

GSE Reference Request Data Structure 8

Global Error, Offset, and Reference response structures 8

GSE Global Error Response Structure 8

OFFSET Request Results Data Structure 9

GSE Offset Response Data Structure 9

Reference Request Results Data Structure 9

GSE Reference Response Data Structure 9

GSE Management Message Data Structure 10

GSE Management Message Encode Functions 11

GSE Management Message Decode Functions 20

IEC 61850 SampledValue Data and Functions 21

Sampled Value ASDU Data Structure (message contains multiple ASDU) 21

Sampled Value Message Data Structure 21

Functions for sending Sampled Value Messages 22

Functions for Receiving SMPVAL Messages 25

New Utility Function 26

New Reporting Functions 27

Changes to Support more than one VMD 31

Changes to eliminate global variable (`_mvl_curr_net_info`) 32

Enhanced Logging Features 36

IPC Logging 37

IPC Logging in an Application 37

New Data Structures 39

Ethertype Info Data Structure 39

SN_UNITDATA Data Structure 39

MVL_MAX_DYN Data Structure 39

New Functions 41

Logging Functions 48

IEC 61850 Functions 51

Modified Functions 59

SCL Server Sample Application 61

AA-Specific Variables in the Foundry ODF Input File 61

Known Software Anomalies 62

NamedVariableLists created by Foundry may not be used in Reports 62

Notes for Phar Lap TNT ETS Operating System Only 62

heap checking 62

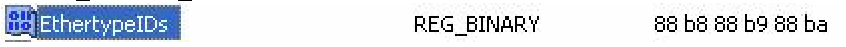
New and Changed Software Features

This release of MMS-EASE *Lite* (V5.06) contains the following changes:

For information on changes and corrections prior to V5.06, please examine Revision 15 of the Release Notes.

1. For Windows – MMS-Lite now uses Microsoft Visual Studio .NET version 2005. All MMS-Lite Samples have been converted from Visual C++ V6.0 workspaces (dsw) to Visual Studio .NET 2005 solutions (sln). **Note that libraries from this product release cannot be linked with applications using Visual C++ V6.0 or Visual Studio .NET 2003 compiler.**

2. Added support for IEC 61850 Sampled Values (see later on in the document for details). Test code may be enabled in **client.c** and **scl_srvr.c** by defining **`SMPVAL_SUPPORT`** in the makefiles. In order to receive Sampled Values as a network client, the PC must have the OSI drivers installed. Furthermore, the OSILLC parameters in the windows registry must be extended. Specifically, the registry key needs to be changed to have the following value inside the following registry parameter:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\OSILLC\Parameters\EthertypeIDs



```
EthertypeIDs                    REG_BINARY          88 b8 88 b9 88 ba
```

Without 0x88BA added to the OSI drivers EthertypeIDs, the Sampled Value Client will not receive the publisher's frames. SISCO support has a sample OSILLC registry file to import which will modify the EthertypeIDs to allow receiving Sampled Value messages. Please contact SISCO support for further details.

3. Added support for IEC 61850 GSE Management (see the section *IEC 61850 GSE Management Data and Functions* for details).

4. Deleted the unimplemented **`gs_wait_mult_event_sem`** function on UNIX, so that problems are detected at compile time, not at runtime.

5. Improved Foundry so that the initialization code generated for NamedVariableLists completely resolves all Variables. This improves efficiency because there is no need to resolve the Variables each time the NamedVariableList is accessed.

6. Added more **`mvlu_get_leaf_*`** utility functions (see **mvl_uca.c**). These are used by the MVL library and the **scl_srvr** sample application.

7. Added code to parse and process the "SampledValueControl" and "Communication" elements of an SCL file.

8. The **`gse_iec_control_create`** function was modified to allocate and copy the caller's strings, so the caller does not need to make sure the strings persist for the life of the GOOSE message.

9. The functions **`clnp_snet_write`** and **`clnp_etype_write`** were replaced with a simpler, more flexible function, **`clnp_snet_write_raw`**. The function was ported to Windows and LINUX, but users will need to port it to other platforms. Users should look at **clnp_linux.c** to see the changes for LINUX.

10. Added the option to use the "Expat" XML parser by simply defining **`USE_EXPAT`** when compiling **sx_dec.c** and linking to an appropriate "Expat" library. The "Expat" parser seems to be much better at detecting errors in XML files and dealing with unexpected whitespace characters. The "Expat" library is not included with the product, but it should be easy to download the "expat" package from http://expat.sourceforge.net/. On Windows, it should be easy to integrate it with the "Visual Studio 2005" solution as follows:

    a. Add the Expat include directory to "Additional Include Directories" for the **util** library.

b. Convert the Expat library project file **expat_static.dsp** to **expat_static.vcproj** and add it to the MMS Lite VS2005 solution.

c. Change the "Debug" library setting in **expat_static.vcproj** to "Multi-threaded (/Mt)" to agree with other libraries.

d. Add a dependency to each executable project to depend on the **expat_static** library.

e. Build the "Debug Expat" or "Release Expat" solution configuration.

11. Changed the SCL processing to create multiple RCBs for each ReportControl (Buffered or Unbuffered) in the SCL file. The number of RCBs depends on the value of the "max" attribute of the "RptEnabled" element. This change makes it possible to allow each client to change the DatSet element as needed.

12. Changed **`ObjectReference`** attributes of some of the **`DatSet`** and **`LogRef`** to Vstring129.

13. Added new ASN.1 encoding functions: **`ms_local_to_asn1_2`**, **`ms_adl_to_asn1_2`**, and **`ms_aa_to_asn1_2`**. These are very similar to the existing functions **`ms_local_to_asn1`**, **`ms_adl_to_asn1`**, and **`ms_aa_to_asn1`**, but they are easier to use in most cases.

14. IEC 61850 Reporting changes:
For IEC 61850 Reporting, the new functions **`mvl61850_create_rpt_ctrl`**, **`mvl61850_free_rpt_ctrl`**, and **`mvl61850_rpt_service`** ***must be used***. The older functions **`mvlu_create_rpt_ctrl`**, **`mvlu_free_rpt_ctrl`**, and **`mvlu_rpt_service`** now work ***only for UCA Reporting***.

a. Buffered reports are now segmented only when sending, so that changing PDU size does not cause a problem.

b. The **SqNum** attribute is reset to 0 when a BRCB is enabled. It is only incremented when a report is sent, so that if the BRCB is disabled, **SqNum** will NOT increment, even though reports are still being buffered.

c. The Report buffer is purged when **DatSet**, **TrgOps**, **IntgPd**, **BufTm**, or **RptID** is written, but only if the value is changed (for Tissue #322).

d. A BRCB is now reserved only when **RptEna** is written.

e. Attribute types needed for reporting are now automatically created if not found.

f. If the Data or Quality Changes are being saved waiting for the **BufTim** to expire, they will be published before the Integrity Report is sent. All saved changes are reported prior to the Integrity report. This prevents Data and Quality changes from being received out of order. This applies to Buffered as well as unbuffered Reports.

g. If the **RptID** in the RCB is NULL, a **RptID** is automatically generated to send in Reports.

h. Writing the GI attribute is allowed only if the RCB is enabled. A GI Report is generated only if the general-interrogation bit in **TrgOps** is 1.

i. Writing the **DatSet** attribute is now allowed and it causes the new data set to be used when generating subsequent reports.

j. Changed the Report Buffering scheme to save "raw" data in the buffer and encode reports only when sending, so that **SqNum**, etc., can be set/changed each time a report is sent.

k. The BRCB **EntryID** is automatically initialized with the current time in the first 4 bytes and 0 in the last 4 bytes, so that after a reboot, the initial EntryID is almost always unique. If you want a different initial **EntryID**, you can call **`mvl61850_brcb_entryid_init`** after **`mvl61850_rpt_ctrl_create`** (see usage example in **uca_srvr.c**). The value in the last 4 bytes of **EntryID** is incremented as each report is generated.

l. When a new GI report is buffered, any older GI report in the buffer is discarded.

m. The **Resv** attribute of a URCB is now used to reserve the URCB for one client. If a client writes a value of 1 to **Resv**, the server will not allow other clients to write members of the URCB until the first client writes a value of 0. If **Resv**=0, and a client enables the URCB, the server automatically sets **Resv**=1.

15. A pointer to **`MVL_TYPE_CTRL`** is saved in the new **`type_ctrl`** member of **`MVL_VAR_ASSOC`**, so that type information is easier to access.

16. Deleted the functions **`u_mvl_get_nvl`** and **`u_mvl_free_nvl`** from **mvl_uca.c**. Instead, **`mvl_vmd_find_nvl`** is used.
NOTE: these functions are still used in some code dependent on the **`USE_MANUFACTURED_OBJS`** define, but this is never defined in IEC 61850 or UCA applications.

17. Improved the setting of error codes in MMS Write responses to be more accurate.

18. Changed the IEC 61850 GOOSE sample application to be configured using SCL.

19. Applications using SCL must now include a callback function **`u_mvl_scl_set_initial_value`** (see **scl_srvr.c**). This function is only called if the SCL parser cannot convert the initial value according to the standard. The user may implement a special conversion in this function, but usually it should just return an error so that parsing stops.

20. Simplified the ODF file for the **scl_srvr** sample application, so that all MMS objects created from SCL.

21. Added the **`mvl61850_datset_wr_ind`** function to **leafmap.xml** so that it is included in the leaf function pointer table. This is ***important*** because the SCL parsing code must automatically map this function.

22. **IMPORTANT**: Data changes for IEC 61850 Reports are no longer automatically detected by the MVL library. The change detection was flawed because it treated Quality changes as Data changes, and detected changes in irrelevant attributes. Instead, it is now required that the “leaf” functions detect changes, and set the appropriate reason by setting the **`rpt_reason`** member of **`MVL_VAR_ASSOC`** (see example in **`set_rpt_reason`** in **userleaf.c**).

23. Changed **`mvlu_find_comp_type`** to find ‘non-dynamic’ types as well as ‘dynamic’ types. Previously, the function would only find ‘dynamic’ types.

24. Added the **`reserved_1`** member to the **`RUNTIME_CTRL`** structure. It allows the flexibility to store additional type information, and it is very useful during SCL processing.

25. Added several new functions (**`ms_rt_bld_*`**) to construct a type definition without TDL.

26. Added functions to create and maintain multiple VMDs to make client applications easier to implement (the old functions assumed there was only one global VMD). Several functions were changed to take an additional VMD argument and some were replaced with new functions (see *Changes to Support more than one VMD* in a later section details).

27. Deleted the **`MVL_DYN_ASN1_TYPES`** define and all code that depended on it. This define enabled code to dynamically generate the ASN.1 encoding of type definitions for GetVariableAccessAttributes responses, instead of using static encoding generated by Foundry. From now on, only the dynamic code is provided.

28. Added the functions **`mvlu_rpt_ctrl_destroy_all`** and **`mvl61850_rpt_ctrl_destroy_all`**.

29. Changed IEC 61850 control handling so that when an **Oper** is being written, the write is only allowed if the entire **Oper** structure is written in a single MMS Write request. You cannot write a higher level structure that contains an **Oper** structure and you cannot write individual elements of an **Oper** structure.

30. Deleted the global variables **`_mvl_curr_net_info`** and **`_mvl_curr_usr_ind_ctrl`**.

31. Completely changed arguments for **`scl2_datatype_create_all`** and **`scl2_ld_create_all`**, so that multiple SCL files may be processed and multiple VMDs created.

32. Improved the code that generates type definitions from SCL so that it now properly saves initial values specified in <Val> elements inside <DA> or <BDA> elements. The initial values are written to any variable referencing the type definition. To implement this change, the code was mostly rewritten to use the new **`ms_rt_bld_*`** functions.

33. Added test code in **client.c** to configure using SCL. The code is triggered by a new command line argument.

34. Reduced the number of different versions of each sample application, built with different versions of the "stack" library.

35. **`FILE_LOG_EN`** is no longer a default option in **`sLogCtrl`**. Applications using SISCO logging need to turn this flag **ON** if logging to a log file is desired:

```
sLogCtrl->logCtrl |= LOG_FILE_EN;
```

36. MMS_LITE samples were revised to use IPC Logging. The source file slogipc.c was added to the vcproj and makefiles for all sample applications. Please refer to **logcfgx.c**, **client.c**, **server.c**, **uca_srvr.c**, and **scl_srvr.c** for additions/changes to the sample source code.

37. Removed **logcfg.xsd** and **logcfg.dtd** from this product distribution.

38. Changed the IEC 61850 server code to allow the **`stVal`** attribute of the **`Mod`** attribute (INC class) or **`Beh`** attribute (INS class) to be configured in SCL as "Enum" or "INT32". Because the Tissues regarding **`CtxInt`** (120, 146, 171, and 234) are not entirely clear at this date, this change should allow users to configure whatever type is finally clarified in the standard.

39. The define **`ALLOW_MULTIPLE_REQUESTS_OUT`**, documented in the Reference Manual, is no longer used. It was deleted from the source code sometime before V4.2951, but the change may not have been documented in earlier releases.

40. Changed the functions in **gensock2.c**, **tp0_socks.c**, and **slogipc.c** to use only one mutex semaphore to avoid potential deadlocks. The semaphore is controlled by the macros **`S_LOCK_UTIL_RESOURCES`** and **`S_UNLOCK_UTIL_RESOURCES`** (these macros replace **`S_LOCK_RESOURCES`** and **`S_UNLOCK_RESOURCES`**).

41. Changed the "gensock2" interface (in **gensock2.c**) as follows:

    a. Added the capability to manage multiple "gensock" contexts by passing a pointer to a **`GEN_SOCK_CTXT`** structure to most functions.

    b. Replaced the **`uSockConnect`** function pointer with 2 separate pointers **`uSockConnectInd`** (called only when a connect "Indication" is received) and **`uSockConnectConf`** (called only when a connect "Confirm" is received).

c. Added the optional functions **`sockEventPut`** and **`sockEventGet`**. The **`sockEventPut`** function puts a socket event on a list. It may be called from a callback function to save an event for later processing. The **`sockEventGet`** function gets a socket event from the list when the application is ready to process it. These 2 functions are thread-safe.

d. Removed the **`gs_poll_mode`** flag that controlled an undocumented and confusing gensock option.

42. Changed the function **`u_ml_get_rt_type`** in the **mmsl** library to just log an error and return **`SD_FAILURE`**. The library contains a "default" function which may be replaced by a user-customized function. In the past, the "default" function called "mvl" functions, so it required linking to the **mvl** library, even though most applications never executed this function. Any user that depends on the old behavior must create their own "customized" function which will override the library function (see example in **uca_srvr.c**).

43. New files:

cmd\gnu\gse_mgmt_test.mak	Builds IEC 61850 GSE Mgmt test executable for LINUX, etc.
cmd\gnu\gse_mgmt.mak	Builds IEC 61850 GSE Mgmt library for LINUX, etc.
cmd\gnu\scl_tpxs0.mak	Builds IEC 61850 Sampled Value test executable for LINUX, etc.
cmd\gnu\smpval.mak	Builds IEC 61850 Sampled Value library for LINUX, etc.
cmd\gnu\ositpxs.mak	Builds new stack library for LINUX, etc.
cmd\gnu\cositpxs0.mak	Builds new sample client including IEC 61850 Sampled Value test code.
cmd\gnu\sositpxs0.mak	Builds sample server with ositpxs stack library.
cmd\gnu\uositpxs0.mak	Builds sample UCA server with ositpxs stack library.
cmd\gnu\scl_tpxs0.mak	Builds new sample SCL server including IEC 61850 Sampled Value test code.
cmd\win32\gse_mgmt_test.vcproj	Builds IEC 61850 GSE Mgmt test executable for Windows.
cmd\win32\gse_mgmt.vcproj	Builds IEC 61850 GSE Mgmt library for Windows.
cmd\win32\scl_tpxs0.vcproj	Builds IEC 61850 Sampled Value test executable for Windows.
cmd\win32\smpval.vcproj	Builds IEC 61850 Sampled Value library for Windows.
inc\gse_mgmt.h	Prototypes, etc for IEC 61850 GSE Management support.
mvl\src\mvl61850_rpt.c	Most IEC 61850 reporting code moved from **mvlu_rpt.c** to this file.
mvl\usr\gse_mgmt\*.*	IEC 61850 GSE Management sample application.
src\smpval\smpval_dec.c	Functions to decode IEC 61850 Sampled Value messages.
src\smpval\smpval_enc.c	Functions to encode IEC 61850 Sampled Value messages.
uca\goose\gse_mgmt_dec.c	Functions to decode IEC 61850 GSE Management messages.
uca\goose\gse_mgmt_enc.c	Functions to decode IEC 61850 GSE Management messages.

Deleted file:

clnp_dos.c	Old subnet interface code – no longer used.

Software Corrections

This release of MMS-EASE *Lite* (V5.06) contains the following corrections:

- An assertion was removed from the transaction queuing algorithm in **ethsub.c**.
- The spelling of the IEC 61850 “BufTm” attribute was corrected from “BufTim” when processing SCL.
- Fixed the handling of the “Specification With Result” parameter when processing a MMS Read indication (please refer to **`spec_in_result`** handling in the **`mvlas_read_resp`** function in **s_read.c**).
- The code generated by Foundry for Domain-specific Journals was corrected.
- Fixed buffer overflow problems in the SCL parser when reading attribute values longer than 64 characters. Also eliminated unnecessary copying of SCL data.
- The initial value of DatSet generated from SCL for GOOSE and LOG control blocks was corrected.
- Fixed a memory leak in the **`clnp_read`** function.
- The **`LastApplError`** variable is no longer created at startup but created temporarily only when needed. This prevents clients from seeing the variable in a GetNameList response.
- In **iec_rx.c**, a call to **`clnp_snet_free`** was added to fix a memory leak. Any user application based on this code might need a similar change.
- The Foundry does NOT create **`mvl_rt_tables`** if it would be empty (empty array is illegal).
- Corrected the DeleteNamedVariableList indication processing so that if the “Scope of Delete” is “SPECIFIC”, the deletable flag is checked for each NVL before allowing the deletion.
- Changed the type of the “T” attribute of controls from "EntryTime" to "Timestamp", to resolve Tissue #35. Changed the sample SCL file and **iec_cdc.odf**.
- Changed the CMV attribute from “mcval” (incorrect) to “cVal” (correct) in the sample SCL file.
- Fixed the error class and code sent in Error responses for GetVariableAccessAttributes and GetNamedVariableListAttributes indications.
- The length of USERST was corrected in **gentypes.odf**.
- Corrected memory leaks on error conditions in **mvlu_rt.c** and **mvl_typ2.c**.
- The IEC 61850 Controls code was fixed to send a LastApplError report if a write to **`Oper`**, **`SBOw`**, or **`Cancel`** fails. This code depends on the user “leaf” functions to set the report data by filling in the LastApplError member of the **`MVLAS_WR_VA_CTRL`** structure. To see how this should be done, look for LastApplError in **userleaf.c**.
- Corrected a memory leak in **mvl_obj.c** module for the **`last_data`** field.

- Corrected a memory leak in **`u_a_associate_ind`** (**mvl_acse.c**) when the **`u_mvl_connect_ind_ex`** function returned a failure.

- Corrected an ASN.1 decoding problem with ASN.1 Generalized Time values near “1970-01-01 00:00:00” in timezones east of GMT. The problem often occurred when receiving a MMS File Directory response from a server that had uninitialized file timestamps.

- Fixed the handling of **`sboTimeout`** for IEC 61850 controls to use milliseconds (not seconds).

- Corrected the initial value of **`LogRef`** for IEC 61850 Logs to be an ObjectReference.

General Application Upgrade Notes for MMS-EASE *Lite*

IEC 61850 GSE Management Data and Functions

The following data structures and functions may be used to send and receive IEC 61850 GSE Management messages.

GSE Offset Request Data Structure

The offset request functions pass a pointer to this structure.

```
typedef struct
  {
  ST_UINT32 stateID;                 /* Reference ID assigned by client.  */
  ST_CHAR ident [MAX_VSTRING_BUF+1]; /* Description string.               */
  ST_INT numVStrings;                /* Number of strings in references   */
                                     /* array.                            */
  ST_CHAR **references;              /* References array.                 */
  } GSE_OFFSET_REQ;
```

GSE Reference Request Data Structure

Reference request functions pass a pointer to this structure.

```
typedef struct {
  ST_UINT32 stateID;                 /* Reference ID assigned by client.    */
  ST_CHAR ident [MAX_VSTRING_BUF+1]; /* Description string.                 */
  ST_INT numIntegers;                /* Number of integers in offset array. */
  ST_UINT32 *offset;                 /* Array of offsets.                   */
  } GSE_REF_REQ;
```

Global Error, Offset, and Reference response structures

GSE Global Error Response Structure

Global Error response functions pass a pointer to this structure.

```
typedef struct
  {
  ST_UINT32 stateID;                   /* Reference ID assigned by client.  */
  ST_CHAR   ident [MAX_VSTRING_BUF+1]; /* Description string.               */
  ST_UINT32 confRev;                   /* Configuration revision.           */
  ST_UINT32 glbError;                  /* This variable can be GLB_ERR_OTHER*/
                        /* GLB_ERR_UNKNOWN_CTRL_BLK, GLB_ERR_RSP_TOO_LARGE, */
                        /* or GLB_ERR_CTRL_BLK_CONFIG_ERR.                  */
  } GSE_GLB_ERR_RSP;
```

OFFSET Request Results Data Structure

This structure is used in the **GSE_OFFSET_RSP** structure.

```
typedef struct
  {
  ST_INT    rsp_type;   /* SD_SUCCESS if offset, or SD_FAILURE if error.*/
  ST_UINT32 offset;     /* offset is set if rsp_type = SD_SUCCESS.      */
  ST_UINT32 error;      /* error is set if rsp_type = SD_FAILURE.       */
  } OFFSET_REQ_RESULTS;
```

GSE Offset Response Data Structure

This structure is used in the **GSE_MGMT_RSP** structure. Offset response functions pass a pointer to this structure.

```
typedef struct
  {
  ST_UINT32 stateID;           /* Reference ID assigned by client.   */
  ST_CHAR   ident [MAX_VSTRING_BUF+1];    /* Description string.     */
  ST_UINT32 confRev;                      /* Configuration revision. */
  ST_CHAR   datSet [MAX_VSTRING_BUF+1];   /* Data set description.   */
  ST_INT    numResults;       /* Number of results in result array.  */
  OFFSET_REQ_RESULTS *result; /* Request result array.               */
  } GSE_OFFSET_RSP;
```

Reference Request Results Data Structure

Used in the **GSE_REF_RSP** structure.

```
typedef struct
  {
  ST_INT    rsp_type; /* SD_SUCCESS if reference, or SD_FAILURE if error.*/
  ST_CHAR   reference [MAX_VSTRING_BUF+1];      /* Set for ref response. */
  ST_UINT32 error;      /* error is set if rsp_type = SD_FAILURE.        */
  } REF_REQ_RESULTS;
```

GSE Reference Response Data Structure

Used in the **GSE_MGMT_RSP** structure. Reference response functions pass a pointer to this structure.

```
typedef struct
  {
  ST_UINT32 stateID;          /* Reference ID assigned by client.   */
  ST_CHAR   ident [MAX_VSTRING_BUF+1];    /* Description string.    */
  ST_UINT32 confRev;                      /* Configuration revision.*/
  ST_CHAR   datSet [MAX_VSTRING_BUF+1];   /* Data set description.  */
  ST_INT    numResults;       /* Number of results in result array. */
  REF_REQ_RESULTS *result;    /* Request result array.              */
  } GSE_REF_RSP;
```

GSE Management Message Data Structure

This structure contains all information decoded from a received GSE Management message.

```
typedef struct {
  ST_UINT32 stateID;          /* Reference ID assigned by client.         */
  ST_INT    msgType;          /* Set to the following values when message */
                              /* is decoded.  Used to free memory         */
                              /* allocated for various message structures.*/
                              /* Valid message types are:                 */
                              /* GSE_MSG_TYPE_GOOSE_ELE_REQ,              */
                              /* GSE_MSG_TYPE_GS_REF_REQ,                 */
                              /* GSE_MSG_TYPE_GSSE_DATA_OFF_REQ,          */
                              /* GSE_MSG_TYPE_GO_REF_RSP,                 */
                              /* GSE_MSG_TYPE_GOOSE_ELE_RSP,              */
                              /* GSE_MSG_TYPE_GS_REF_RSP,                 */
                              /* GSE_MSG_TYPE_GSSE_DATA_OFF_RSP, or       */
                              /* GSE_MSG_TYPE_GLOBAL_ERROR_RSP.           */

  union GSE_MSG {             /* Union of all GSE messages.               */
    GSE_REF_REQ     refReq;   /* msgType = GSE_MSG_TYPE_GO_REF_REQ or     */
                              /* GSE_MSG_TYPE_GS_REF_REQ.                 */
    GSE_OFFSET_REQ  offReq;   /* msgType = GSE_MSG_TYPE_GOOSE_ELE_REQ or  */
                              /* GSE_MSG_TYPE_GSSE_DATA_OFF_REQ.          */
    GSE_OFFSET_RSP  offRsp;   /* msgType = GSE_MSG_TYPE_GOOSE_ELE_RSP or  */
                              /* GSE_MSG_TYPE_GSSE_DATA_OFF_RSP.          */
    GSE_REF_RSP     refRsp;   /* msgType = GSE_MSG_TYPE_GO_REF_RSP or     */
                              /* GSE_MSG_TYPE_GS_REF_RSP.                 */
    GSE_GLB_ERR_RSP glbErrRsp;/* msgType = GSE_MSG_TYPE_GLOBAL_ERROR_RSP. */
    } msg;
} GSE_MGMT_MSG;
```

GSE Management Message Encode Functions

getGlbErrorRspEncode

Usage: This function encodes the GSE Management Global Error response, including the 14 byte Ethertype header. The parameters in the **(GSE_GLB_ERR_RSP *)** and **(ETYPE_INFO *)** structures must be filled in before this function is called. The (**ST_UCHAR ***) argument passed to this function points to the encode buffer and **asn1DataBufLen** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **asn1DataLenOut** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:
```
ST_UCHAR * getGlbErrorRspEncode (GSE_GLB_ERR_RSP *ctrl,
                                 ST_UCHAR *asn1DataBuf,
                                 ST_INT asn1DataBufLen,
                                 ST_INT *asn1DataLenOut,
                                 ETYPE_INFO *etype_info,
                                 ST_UINT8 *dstMac,
                                 ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to a **GSE_GLB_ERR_RSP** data structure.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **ETYPE_INFO** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a **(ST_UCHAR *)** pointer to the ASN1 encoded message buffer.

getGoRefReqEncode

Usage: This function encodes the GSE Management Go Reference request, including the 14 byte Ethertype header. The parameters in the **(GSE_REF_REQ *)** and **(ETYPE_INFO *)** structures must be filled in before this function is called. The (**ST_UCHAR ***) argument passed to this function points to the encode buffer and **asn1DataBufLen** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **asn1DataLenOut** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:

```
ST_UCHAR *getGoRefReqEncode (GSE_REF_REQ *ctrl,
                             ST_UCHAR *asn1DataBuf,
                             ST_INT asn1DataBufLen,
                             ST_INT *asn1DataLenOut,
                             ETYPE_INFO *etype_info,
                             ST_UINT8 *dstMac,
                             ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to a **GSE_REF_REQ** structure. The structure contains a pointer to an array of reference offsets.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **ETYPE_INFO** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**ST_UCHAR ***) pointer to the ASN1 encoded message buffer.

getGoRefRspEncode

Usage: This function encodes the GSE Management Go Reference response, including the 14 byte Ethertype header. The parameters in the (**`GSE_REF_RSP *`**) and (**`ETYPE_INFO *`**) structures must be filled in before this function is called. The (**`ST_UCHAR *`**) argument passed to this function points to the encode buffer and **`asn1DataBufLen`** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **`asn1DataLenOut`** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:
```
ST_UCHAR *getGoRefRspEncode (GSE_REF_RSP *ctrl,
                             ST_UCHAR *asn1DataBuf,
                             ST_INT asn1DataBufLen,
                             ST_INT *asn1DataLenOut,
                             ETYPE_INFO *etype_info,
                             ST_UINT8 *dstMac,
                             ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to a **`GSE_REF_RSP`** structure.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **`ETYPE_INFO`** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**`ST_UCHAR *`**) pointer to the ASN1 encoded message buffer.

getGOOSEEleNumReqEncode

Usage: This function encodes the GSE Management GOOSE Element Number request, including the 14 byte Ethertype header. The parameters in the (**`GSE_OFFSET_REQ *`**) and (**`ETYPE_INFO *`**) structures must be filled in before this function is called. The (**`ST_UCHAR *`**) argument passed to this function points to the encode buffer and **`asn1DataBufLen`** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **`asn1DataLenOut`** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:

```
ST_UCHAR *getGOOSEEleNumReqEncode (GSE_OFFSET_REQ *ctrl,
                                   ST_UCHAR *asn1DataBuf,
                                   ST_INT asn1DataBufLen,
                                   ST_INT *asn1DataLenOut,
                                   ETYPE_INFO *etype_info,
                                   ST_UINT8 *dstMac,
                                   ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to an **`GSE_OFFSET_REQ`** structure. The structure contains a pointer to an array of visible reference strings. The **`numVStrings`** variable must be set to the number of visible reference strings in the array.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **`ETYPE_INFO`** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**`ST_UCHAR *`**) pointer to the ASN1 encoded message buffer.

getGOOSEEleNumRspEncode

Usage: This function encodes the GSE Management GOOSE Element Number request, including the 14 byte Ethertype header. The parameters in the (**`GSE_OFFSET_RSP *`**) and (**`ETYPE_INFO *`**) structures must be filled in before this function is called. The (**`ST_UCHAR *`**) argument passed to this function points to the encode buffer and **`asn1DataBufLen`** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **`asn1DataLenOut`** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:
```
ST_UCHAR *getGOOSEEleNumRspEncode (GSE_OFFSET_RSP *ctrl,
                                   ST_UCHAR *asn1DataBuf,
                                   ST_INT asn1DataBufLen,
                                   ST_INT *asn1DataLenOut,
                                   ETYPE_INFO *etype_info,
                                   ST_UINT8 *dstMac,
                                   ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to an **`GSE_OFFSET_RSP`** structure.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **`ETYPE_INFO`** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**`ST_UCHAR *`**) pointer to the ASN1 encoded message buffer.

getGsRefReqEncode

Usage: This function encodes the GSE Management Gs Reference request, including the 14 byte Ethertype header. The parameters in the **(GSE_REF_REQ *)** and **(ETYPE_INFO *)** structures must be filled in before this function is called. The (**ST_UCHAR ***) argument passed to this function points to the encode buffer and **asn1DataBufLen** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **asn1DataLenOut** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:

```
ST_UCHAR *getGsRefReqEncode (GSE_REF_REQ *ctrl,
                             ST_UCHAR *asn1DataBuf,
                             ST_INT asn1DataBufLen,
                             ST_INT *asn1DataLenOut,
                             ETYPE_INFO *etype_info,
                             ST_UINT8 *dstMac,
                             ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to a **GSE_REF_REQ** structure. The structure contains a pointer to an array of reference offsets. The **numIntegers** variable must be set to the number of offsets in the array.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **ETYPE_INFO** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**ST_UCHAR ***) pointer to the ASN1 encoded message buffer.

getGsRefRspEncode

Usage: This function encodes the GSE Management Gs Reference response, including the 14 byte Ethertype header. The parameters in the **(GSE_REF_RSP *)** and (**ETYPE_INFO ***) structures must be filled in before this function is called. The (**ST_UCHAR ***) argument passed to this function points to the encode buffer and **asn1DataBufLen** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **asn1DataLenOut** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:

```
ST_UCHAR *getGsRefRspEncode (GSE_REF_RSP *ctrl,
                             ST_UCHAR *asn1DataBuf,
                             ST_INT asn1DataBufLen,
                             ST_INT *asn1DataLenOut,
                             ETYPE_INFO *etype_info,
                             ST_UINT8 *dstMac,
                             ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to a **GSE_REF_RSP** structure.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **ETYPE_INFO** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**ST_UCHAR ***) pointer to the ASN1 encoded message buffer.

getGSSEDataOffsetReqEncode

Usage: This function encodes the GSE Management GSSE Data Offset request, including the 14 byte Ethertype header. The parameters in the (**`GSE_OFFSET_REQ *`**) and (**`ETYPE_INFO *`**) structures must be filled in before this function is called. The (**`ST_UCHAR *`**) argument passed to this function points to the encode buffer and **`asn1DataBufLen`** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **`asn1DataLenOut`** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:

```
ST_UCHAR *getGSSEDataOffsetReqEncode (GSE_OFFSET_REQ *ctrl,
                                      ST_UCHAR *asn1DataBuf,
                                      ST_INT asn1DataBufLen,
                                      ST_INT *asn1DataLenOut,
                                      ETYPE_INFO *etype_info,
                                      ST_UINT8 *dstMac,
                                      ST_UINT8 *srcMac);
```

Parameters:

`ctrl` This parameter points to a **`GSE_OFFSET_REQ`** structure. The structure contains a pointer to an array of visible reference strings. The **`numVStrings`** variable must be set to the number of visible reference strings in the array.

`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **`ETYPE_INFO`** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**`ST_UCHAR *`**) pointer to the ASN1 encoded message buffer.

getGSSEDataOffsetRspEncode

Usage: This function encodes the GSE Management GSSE Data Offset response, including the 14 byte Ethertype header. The parameters in the (**`GSE_OFFSET_RSP *`**) and (**`ETYPE_INFO *`**) structures must be filled in before this function is called. The (**`ST_UCHAR *`**) argument passed to this function points to the encode buffer and **`asn1DataBufLen`** parameter contains the length of the buffer. The function returns a pointer to the ASN1 encoded message and **`asn1DataLenOut`** points to the length of the ASN1 encoded buffer.

Include: gse_mgmt.h

Function Prototype:

```
ST_UCHAR *getGSSEDataOffsetRspEncode (GSE_OFFSET_RSP *ctrl,
                                      ST_UCHAR *asn1DataBuf,
                                      ST_INT asn1DataBufLen,
                                      ST_INT *asn1DataLenOut,
                                      ETYPE_INFO *etype_info,
                                      ST_UINT8 *dstMac,
                                      ST_UINT8 *srcMac);
```

Parameters:

`ctrl`	This parameter points to a **`GSE_OFFSET_RSP`** structure.
`asn1DataBuf`	This parameter points to the buffer to be used for the ASN1 message encoding.
`asn1DataBufLen`	This parameter contains the size of the ASN1 buffer in bytes.
`asn1DataLenOut`	This parameter points to the length of the ASN1 encoded message buffer when the function returns.
`etype_info`	This parameter points to an **`ETYPE_INFO`** structure.
`dstMac`	This parameter points to a destination MAC address.
`srcMac`	This parameter points to a source MAC address.

Return Value: The function returns a (**`ST_UCHAR *`**) pointer to the ASN1 encoded message buffer.

GSE Management Message Decode Functions

gse_mgmt_msg_decode

Usage: This function decodes GSE Management messages, including the Ethernet header. This function allocates a **`GSE_MGMT_MSG`** structure, fills it in, and returns a pointer to it. The function also fills in the **`ETPE_INFO`** structure referenced by the **`etypeInfo`** argument. The **`SN_UNITDATA`** structure must not be freed until after the **`GSE_MGMT_MSG`** structure is freed because **`GSE_MGMT_MSG`** contains pointers to data in the **`SN_UNITDATA`** structure.

Include: gse_mgmt.h

Function Prototype:

```
GSE_MGMT_MSG *gse_mgmt_msg_decode (SN_UNITDATA *sn_req,
                                   ETYPE_INFO *etypeInfo);
```

Parameters:

`sn_req`	This parameter points to a **`SN_UNITDATA`** structure.
`etype_info`	This parameter points to an **`ETYPE_INFO`** structure.

Return Value: The function returns a (**`GSE_MGMT_MSG *`**) pointer message data.

gse_mgmt_msg_free

Usage: This function frees al the resources allocated for the **GSE_MGMT_MSG** structure.

Include: gse_mgmt.h

Function Prototype: `ST_VOID gse_mgmt_msg_free (GSE_MGMT_MSG *gseMgmt);`

Parameters:

`gseMgmt` This parameter points to the **`GSE_MGMT_MSG`** structure which gets filled depending on the **`msgType`**.

Return Value: This function does not return a value.

IEC 61850 SampledValue Data and Functions

The following data structures and functions may be used to send and receive IEC 61850 SampledValue messages.

Sampled Value ASDU Data Structure (message contains multiple ASDU)

```
typedef struct
  {
  ST_UINT8 *SamplePtr;                    /* pointer to "Sample" data         */
  ST_INT SampleLen;                       /* length of "Sample" data in bytes */
  ST_CHAR svID [MAX_SMPVAL_SVID_LEN+1];   /* MsvID or UsvID - Vstring65       */
  ST_BOOLEAN DatSetPres;                  /* is "DatSet" present in ASDU?     */
  ST_CHAR DatSet [MAX_SMPVAL_OBJREF_LEN+1];  /* Vstring129 (ObjectReference)  */
                                          /* (Optional)                       */
  ST_UINT16 SmpCnt;
  ST_UINT32 ConfRev;
  ST_BOOLEAN SmpSynch;
  ST_BOOLEAN RefrTmPres;                  /* is "RefrTm" present in ASDU?     */
  MMS_UTC_TIME RefrTm;                    /* Optional                         */
  ST_BOOLEAN SmpRatePres;                 /* is "SmpRate" present in ASDU?    */
  ST_UINT16 SmpRate;                      /* Optional                         */
  } SMPVAL_ASDU;
```

Sampled Value Message Data Structure

```
typedef struct
  {
  ST_UINT16 numASDU;                /* Num of ASDU concatenated into one APDU */
  SMPVAL_ASDU *asduArray;           /* array of "numASDU" structs             */
                                    /* allocated by smpval_msg_create OR      */
                                    /* allocated during decode                */
  ST_UCHAR securityBuf [MAX_SMPVAL_SECURITY_LEN];    /* security info         */
  ST_UINT securityLen;              /* len of security info                   */
  }SMPVAL_MSG;
```

Functions for sending Sampled Value Messages

smpval_msg_create

Usage: This function allocates and initializes a Sampled Value message structure to store all message data.

Function Prototype: `SMPVAL_MSG *smpval_msg_create (ST_UINT numASDU);`

Parameters:

`numASDU`	Number of ASDUs concatenated into one APDU.

Return Value: The function returns a pointer to a **`SMPVAL_MSG`** structure where all message data may be stored.

smpval_msg_destroy

Usage: This function frees up all memory associated with a **`SMPVAL_MSG`** structure.

Function Prototype: `ST_VOID smpval_msg_destroy (SMPVAL_MSG *smpvalMsg);`

Parameters:

`smpvalMsg`	This is a pointer to a **`SMPVAL_MSG`** structure returned from the **`smpval_msg_create`** function.

Return Value: This function does not return a value.

smpval_asdu_data_update

Usage: This function updates data stored for one ASDU to be sent in a SMPVAL message.

Function Prototype:

```
ST_RET smpval_asdu_data_update (SMPVAL_MSG *smpvalMsg,
                                ST_INT asduIdx,
                                ST_UINT8 *SamplePtr,
                                ST_INT SampleLen,
                                ST_CHAR *svID,
                                ST_INT SmpCnt,
                                ST_BOOLEAN DatSetPres,
                                ST_CHAR *DatSet,
                                ST_UINT32 ConfRev,
                                ST_BOOLEAN SmpSynch,
                                ST_BOOLEAN RefrTmPres,
                                MMS_UTC_TIME *RefrTm,
                                ST_BOOLEAN SmpRatePres,
                                ST_UINT16 SmpRate);
```

Parameters:

`smpvalMsg`	Pointer to a SMPVAL message info structure.
`asduIdx`	Index into array of ASDU for this SMPVAL message.
`SamplePtr`	Pointer to the actual sample data.
`SampleLen`	Length of the sample data in bytes.
`svID`	Pointer to the svID string (system-wide unique identifier) to send in this ASDU.
`SmpCnt`	Sample count value to send in this ASDU. This is incremented each time a new sampling value is taken.
`DatSetPres`	This flag is set to **`SD_TRUE`**, if the optional Data Set should be sent.
`DatSet`	Contains the optional Data Set if the **`DatSetPres`** argument is set to **`SD_TRUE`**
`ConfRev`	Configuration Revision value to send in this ASDU.
`SmpSynch`	**`SmpSynch`** value to send in this ASDU. If this is true, Sampled Values are resynchronized by a clock signal.
`RefrTmPres`	This flag is set to **`SD_TRUE`** if the optional Reference Time should be sent.
`RefrTm`	Contains the optional Reference Time if the **`RefrTmPres`** is set to **`SD_TRUE`**.
`SmpRatePres`	This flag is set to **`SD_TRUE`** if the optional Sample Rate should be sent.
`SmpRate`	Contains the optional Sample Rate is the **`SmpRatePres`** argument is set to **`SD_TRUE`**.

Return Value: `ST_RET` `SD_SUCCESS` If OK, or an error code.

smpval_msg_send

Usage: This function encodes and sends a complete SMPVAL message (APDU plus Ethertype header).

Function Prototype:
```
ST_RET smpval_msg_send (SMPVAL_MSG *smpvalMsg,
                        ETYPE_INFO *etypeInfo,
                        ST_UCHAR *dstMac);
```

Parameters:

`smpvalMsg`	Pointer to a SMPVAL message info structure.
`etypeInfo`	Pointer to the ethertype info.
`dstMac`	Pointer to the destination (Multicast) MAC address.

Return Value: **`SD_SUCCESS`** or an error code.

Functions for Receiving SMPVAL Messages

smpval_msg_decode

Usage: This function decodes a received SMPVAL message and fills in the **`SMPVAL_MSG`** structure.

Function Prototype:
```
SMPVAL_MSG *smpval_msg_decode (SN_UNITDATA *sn_req,
                               ETYPE_INFO *etypeInfo);
```

Parameters:

`sn_req`	Pointer to the message to decode.
`etypeInfo`	Pointer to an **`ETYPE_INFO`** structure of where to store decoded Ethertype info.

Return Value: Returns a pointer of type **`SMPVAL_MSG`**.

smpval_msg_free

Usage: This function must be called to free the **`SMPVAL_MSG`** structure returned from **`smpval_msg_decode`**.

Function Prototype:
```
ST_VOID smpval_msg_free (SMPVAL_MSG *smpvalMsg);
```

Parameters:

`smpvalMsg`	Pointer to a SMPVAL message info structure to be freed.

Return Value: Ignored

New Utility Function

reverse_bytes

Usage: This function copies data from source to destination but reverses the order of the bytes (i.e., converts Big-Endian to Little-Endian or vice versa). Users must set the SampledValue data with the correct byte order. This function should be useful for doing that on some platforms.

Function Prototype:

```
ST_VOID reverse_bytes (ST_UINT8 *dst,
                       ST_UINT8 *src,
                       ST_INT numbytes);
```

Parameters:

`dest`	Pointer to the destination buffer.
`src`	Pointer to the source buffer.
`numbytes`	Indicates the number of bytes to copy.

Return Value: Ignored

New Reporting Functions

mvl61850_brcb_entryid_init

Usage: This function sets the initial value of the EntryID attribute in an IEC 61850 BRCB.

Function Prototype:

```
ST_VOID mvl61850_brcb_entryid_init (MVLU_RPT_CTRL *rptCtrl,
                                    ST_UINT8 *EntryID);
```

Parameters:

`rptCtrl`	Pointer of type **`MVLU_RPT_CTRL`** to the Report Control data structure for the BRCB.
`EntryID`	Pointer to an array of 8 bytes containing the initial **`EntryID`** value.

Return Value: `ST_VOID`

mvl61850_create_rpt_ctrl

Usage: This function is used to create an MVL Report Control data structure for an IEC 61850 report.

Function Prototype:

```
MVLU_RPT_CTRL *mvl61850_create_rpt_ctrl (ST_CHAR *basrcbName,
                                          MVL_NVLIST_CTRL *dsNvl,
                                          MVL_VAR_ASSOC *base_va,
                                          ST_INT rcb_type,
                                          ST_INT buftim_action,
                                          ST_INT brcb_bufsize,
                                          ST_UINT32 ConfRev);
```

Parameters:

`basrcbName`	This is the variable name for the BASRCB used to control the report (e.g., **`POPF$BR$brcbST01`** or **`POPF$RP$urcbMX01`**).
`dsNvl`	This is the data set associated with the report control element. This must be a completely resolved Named Variable List. That is, all variable associations must be complete and valid. This will be the case when the data set NVL is created using **`mvlu_derive_rpt_ds`** or **`mvlu_rpt_nvl_add`**.
`base_va`	This is the MVL Variable Association for the Logical Node to which the BASRCB belongs.

`rcb_type`	The Report Control Block type (one of the following):
	`RCB_TYPE_IEC_BRCB`
	`RBC_TYPE_IEC_URCB`

`buftim_action`	This is the action to be taken of a variable changes twice before the **`BufTim`** timer expires (one of the following):
	`MVLU_RPT_BUFTIM_REPLACE`
	`MVLU_RPT_BUFTIM_SEND_NOW`

`brcb_bufsize`	This is the maximum amount of memory (in bytes) to allow for storing IEC-61850 Buffered Reports. It is used only if **`rcb_type = RCB_TYPE_IEC_BRCB`**.
`ConfRev`	This is the value to be stored in the **`ConfRev`** attribute of the RCB.

Return Value: This function returns a pointer to the **`MVLU_RPT_CTRL`** structure. NULL will be returned if an error occurs.

mvl61850_free_rpt_ctrl

Usage: This function is used to free a MVL Report Control element created via **`mvl61850_create_rpt_ctrl`**.

Function Prototype: `ST_VOID mvl61850_free_rpt_ctrl (MVLU_RPT_CTRL *rptCtrl);`

Parameters:

`rptCtrl`	A pointer to a **`MVLU_RPT_CTRL`** structure to be freed.

Return Value: `ST_VOID`

mvl61850_rpt_service

Usage: This function is used to provide MVL with report processing time. This processing consists of scanning for changes in any data included in the data set, and generating IEC 61850 reports as necessary. It should be called frequently.

Function Prototype: `ST_VOID mvl61850_rpt_service (ST_VOID);`

Parameters: NONE

Return Value: `ST_VOID`

Changes to Support more than one VMD

Many changes were made to allow more than one VMD, primarily to simplify creation of "client" programs. These changes allow a client application to read multiple SCL files, one for each server to which it connects, and to create a VMD for each server. These new VMDs may be used to simplify processing of data received from the servers (see **`config_iec_remote_vmd`** and related code in **client.c**). Previously, all functions assumed that there was only one VMD, accessed by the global variable **`mvl_vmd`**. To allow more than one VMD, it is not practical to use global variables, so many functions were changed so that the VMD is passed as an argument, and new functions were added.

These new functions were added:

`mvl_vmd_create`	This function creates a new VMD.
`mvl_vmd_type_id_create`	This function creates a type reserved for a particular VMD.
`mvl_vmd_type_id_destroy`	This function destroys a type reserved for a particular VMD.
`mvl_vmd_type_id_destroy_all`	This function destroys ***ALL*** types reserved for a particular VMD.
`mvl_vmd_type_ctrl_find`	This function finds a type reserved for a particular VMD.

These new functions <u>replace</u> existing functions:

NEW FUNCTION	REPLACES
`mvl_vmd_dom_add`	`mvl_dom_add`
`mvl_vmd_dom_remove`	`mvl_dom_remove`
`mvl_vmd_var_add`	`mvl_var_add`
`mvl_vmd_var_remove`	`mvl_var_remove`
`mvl_vmd_nvl_add`	`mvl_nvl_add`
`mvl_vmd_nvl_remove`	`mvl_nvl_remove`
`mvl_vmd_jou_add`	`mvl_jou_add`
`mvl_vmd_jou_remove`	`mvl_jou_remove`
`mvl_vmd_find_dom`	`mvl_find_dom`
`mvl_vmd_dom_find_last`	`mvl_dom_find_last`
`mvl_vmd_find_var`	`mvl_find_va`
`mvl_vmd_find_nvl`	`mvl_find_nvl`
`mvl_vmd_find_jou`	`mvl_find_jou`

These functions were changed to include a VMD argument:

`mvl_vmd_destroy`	(see details)
`u_mvl_get_va_aa`	(see details)

These "internal" functions (i.e., not normally called from user applications) replace existing functions:

NEW FUNCTION	REPLACES
`mvl_vmd_dom_insert`	`mvl_dom_insert`
`mvl_vmd_dom_delete`	`mvl_dom_delete`
`mvl_vmd_var_insert`	`mvl_var_insert`
`mvl_vmd_var_delete`	`mvl_var_delete`
`mvl_vmd_nvl_insert`	`mvl_nvl_insert`
`mvl_vmd_nvl_delete`	`mvl_nvl_delete`
`mvl_vmd_jou_insert`	`mvl_jou_insert`
`mvl_vmd_jou_delete`	`mvl_jou_delete`

These "internal" functions (i.e., not normally called from user applications) were changed to include a VMD argument:

```
mvl_nvl_create
_mvl_objname_to_va
```

Changes to eliminate global variable (`_mvl_curr_net_info`)

Some functions needed to access Application Association-specific (**`AA_SPEC`**) scope objects but they did not have an argument to specify the connection (**`MVL_NET_INFO *`**). Instead, they used a global pointer (**`_mvl_curr_net_info`**) to specify the connection. Unfortunately, this pointer was not always correct. To eliminate this problem, the correct pointer is now passed to these functions, and the global variable has been eliminated. ***This will require small changes in some user applications*** (see modified functions in **server.c** for example). Applications that link to the "mvlu" library do not require any changes because these functions are included in the library and they are only called from within the library.

These functions were changed to include a (`MVL_NET_INFO *`) argument:

```
u_gnl_ind_vars
u_gnl_ind_nvls
u_gnl_ind_jous
```

mvl_vmd_destroy

Usage: This function destroys all objects in a VMD and frees up the associated memory. If the VMD was created by mvl_vmd_create, the **`MVL_VMD_CTRL`** structure itself is also freed. If the VMD is the global VMD, **`mvl_vmd`**, most resources are freed, but the **`MVL_VMD_CTRL`** structure and the arrays it points to are not freed because they are allocated by "Foundry-generated" code.

Function Prototype:

```
ST_RET mvl_vmd_destroy (MVL_VMD_CTRL *vmd_ctrl);
```

Parameters:

`vmd_ctrl`	Pointer to the VMD to be destroyed.

Return Value: `ST_RET` `SD_SUCCESS` If OK, or an error code.

u_mvl_get_va_aa

Usage: The function will be called when a variable is being read or written and it is not present in the MVL Variable Association control tables. The user application can resolve the association and return a **`MVL_VAR_ASSOC`** if appropriate. Note that this function is only used when MVL is compiled with **`MVL_AA_SUPP`** and **`USE_MANUFACTURED_OBJ`** defined.

If **`*alt_access_done_out`** is set **`SD_TRUE`**, MVL will assume that the alternate access operation has been addressed by the called function.

Function Prototype:

```
MVL_VAR_ASSOC *u_mvl_get_va_aa (MVL_VMD_CTRL *vmd_ctrl,
                                ST_INT service,
                                OBJECT_NAME *obj,
                                MVL_NET_INFO *netInfo,
                                ST_BOOLEAN alt_access_pres,
                                ALT_ACCESS *alt_acc,
                                ST_BOOLEAN *alt_access_done_out);
```

Parameters:

`vmd_ctrl`	A pointer to the VMD in which to find the variable specified by **`obj`**.
`service`	The MMS service requiring VariableAccess look up. Values may be **`MMSOP_WRITE`**, **`MMSOP_MVLU_RPT_VA`**, **`MMSOP_INFO_RPT`**, or **`MMSOP_GET_VAR`**.
`obj`	The name and scope of the variable needing to be resolved by the application.
`netInfo`	A pointer to connection information, this provides the application with the means to resolve ApplicationAssociation specific variables. The structure **`MVL_NET_INFO`** is defined in **mvl_defs.h.**

u_mvl_get_va_aa (cont'd)

Parameters (cont'd):

`alt_access_pres`	Tells the application if AlternateAccess information is present. Values are **`SD_TRUE`** and **`SD_FALSE`**.
`alt_acc`	When the **`alt_access_pres`** parameter is set to **`SD_TRUE`**, this points to AlternateAccess information for the application to use when performing the VariableAccess.
`alt_access_done_out`	When the **`alt_access_pres`** parameter is set to **`SD_TRUE`**, this is set by the application if AlternateAccess is performed by the application.

Return Value:	`!=NULL`	The application successfully manufactured the variable association.
	`NULL`	An error meaning the application did not resolve the variable.

Enhanced Logging Features

SISCO provides library functions and new macros aiding application developers in implementing application logging. Following sample can be used for setting an application logging using a XML logging configuration file. The **logcfgx.c** must be linked to the application.

In an application header file:

```
 /* logging masks */
#define MYLOG_ERR        0x00000001
#define MYLOG_FLOW       0x00000002
#define MYLOG_DATA       0x00000004
extern ST_UINT my_debug_sel;
extern SD_CONST ST_CHAR *SD_CONST _mylog_err_logstr;
extern SD_CONST ST_CHAR *SD_CONST _mylog_flow_logstr;

/*  error log macros /
#define MY_LOG_ERR0(a)    SLOG_0 (my_debug_sel & MYLOG_ERR,_mylog_err_logstr, a)
#define MY_LOG_ERR1(a, b) SLOG_1 (my_debug_sel & MYLOG_ERR,_mylog_err_logstr, a, b)

/* Create new macros MY_LOG_ERR2, etc.,                  */
/* for each additional argument passed to the log macro. */
/*  error log continuation macros (do not include log message header) */
#define MY_LOG_ERRC0(a)    SLOGC_0 (my_debug_sel & MYLOG_ERR,_mylog_err_logstr, a)
#define MY_LOG_ERRC1(a, b) SLOGC_1 (my_debug_sel & MYLOG_ERR,_mylog_err_logstr, a, b)

/* program flow log macros */
#define MY_LOG_FLOW0(a)   SLOG_0 (my_debug_sel & MYLOG_FLOW,_mylog_flow_logstr, a)

/* hex logging */
#define MY_LOG_DATA(num, ptr)   SLOGH (my_debug_sel & MYLOG_DATA, num , ptr)
```

In an application's C or C++ file:

```
   #include “glbtypes.h”   /* SISCO’s file */
  #include “sysincs.h”     /* SISCO’s file */
  #include “glbsem.h”      /* SISCO’s file – needed for S_MT_SUPPORT define */
  #include “slog.h”        /* SISCO’s file */
  #include “myapp.h”

  SD_CONST static ST_CHAR *SD_CONST thisFileName = __FILE__;
  ST_UINT my_debug_sel = MYLOG_ERR;
/* log errors, other masks maybe set during program execution  */
  SD_CONST ST_CHAR *SD_CONST _mylog_err_logstr =  “MYLOG_ERR”;
  SD_CONST ST_CHAR *SD_CONST _mylog_flow_logstr = “MYLOG_FLOW”;

  LOGCFGX_VALUE_MAP myLogMaskMaps[] =
    {
      {"MYLOG_ERR",  MYLOG_ERR,  &my_debug_sel, _LOGCFG_DATATYPE_UINT_MASK, "Error"},
      {"MYLOG_FLOW", MYLOG_FLOW, &my_debug_sel, _LOGCFG_DATATYPE_UINT_MASK, "Flow"},
      {"MYLOG_DATA", MYLOG_DATA, &my_debug_sel, _LOGCFG_DATATYPE_UINT_MASK, "Data"}
    };
```

```
LOGCFG_VALUE_GROUP myLogMaskMapCtrl =
    {
    {NULL,NULL},
    "User",
    sizeof(myLogMaskMaps)/sizeof(LOGCFGX_VALUE_MAP),
    myLogMaskMaps
    };
/*
The code below, used to configure log masks from the logcfg.xml file, needs to be at the
beginning of main before any logging (MMS or application) is done.
    logCfgAddMaskGroup (&myLogMaskMapCtrl);
  #if defined(S_SEC_ENABLED)
    logCfgAddMaskGroup (&secLogMaskMapCtrl);
  #endif
    logCfgAddMaskGroup (&mmsLogMaskMapCtrl);
    logCfgAddMaskGroup (&acseLogMaskMapCtrl);
    logCfgAddMaskGroup (&tp4LogMaskMapCtrl);
    logCfgAddMaskGroup (&asn1LogMaskMapCtrl);
    logCfgAddMaskGroup (&sxLogMaskMapCtrl);
  #if defined(S_MT_SUPPORT)
    logCfgAddMaskGroup (&gsLogMaskMapCtrl);
  #endif
    logCfgAddMaskGroup (&sockLogMaskMapCtrl);
    logCfgAddMaskGroup (&memLogMaskMapCtrl);
    logCfgAddMaskGroup (&memDebugMapCtrl);

ret =  logcfgx_ex (sLogCtrl, "logcfg.xml", NULL, SD_FALSE, SD_FALSE);
if (ret != SD_SUCCESS)
  printf ("Parsing Log Configuration file failed '%s'...\n", “logcfg.xml”);

  /* sample of application logging */
  if  (type == expected_type)
    {
    MY_LOG_FLOW1 ("Received message type= %d", type);
    MY_LOG_DATA (num_bytes, data_ptr);
    }
  else
    MY_LOG_ERR1 ("Unexpected message received type= %d", type);
```

SISCO logging functions can be accessed directly but the SLOG macros are a more convenient and simpler way of writing logging code.

IPC Logging

SISCO revised its application logging to allow for collecting log messages over a TCP connection. If a developer implements logging using the **logcfg.xml** file, then the IPC logging can be enabled from there without any other programming. Otherwise, the following section describes how to hook up the IPC logging into an application.

IPC Logging in an Application

An application can enable IPC logging by setting the following flag in the SISCO’s global log control structure:

```
sLogCtrl->logCtrl |= LOG_IPC_EN;
```

This IPC logging flag can be enabled/disabled multiple times while the application is running. Although not required the application can call the initialization function **`slogIpcInit (sLogCtrl)`** when it is setting first time the **`LOG_IPC_EN`** flag. This will allow the IPC logging system to send application identification message to a Client as soon as socket connection is established, before any log messages are sent.

The default listening port designated by SISCO for an application is **`IPC_LOG_BASE_PORT (55147)`** defined in the **slog.h**.

Note that MMS Lite libraries have only the error log mask turned ON. If application wants to log various levels of MMS communication/processing it needs to turn the proper masks ON.
The application can change the default IPC logging parameters by modifying fields in the **`sLogCtrl->ipc (IPC_LOG_CTRL)`** structure:

port	Base port number where application will listen for socket connections from Client applications such as Hyper Terminal or Telnet. Default is IPC_LOG_BASE_PORT (55147).
portCnt	Number of listening ports starting with base port, that are available to multiple instances of the application. Default is 1.
portUsed	This is the listen port actually used by given instance of an application. Set in the **`slogIpcInit`** function.
maxConns	Maximum number of socket connections that can be accepted for IPC logging. The default is **`IPC_LOG_MAX_CONNECTIONS`** (10).
maxQueCnt	Maximum number of log messages that can be queued on any one connection. The default is **`IPC_LOG_MAX_QUEUE_CNT`** (100).
appId	This is pointer to a NULL terminated string identifying the application. The buffer holding this information must be persistent while the program is running. There is no size limit on the buffer. The application identification string is sent to a Client in the first message after socket connection has been established. The default is NULL pointer.

New Data Structures

Ethertype Info Data Structure

This structure contains Ethertype packet header information. It is used when sending and receiving Ethertype packets.

```
typedef struct {
  ST_UINT16 tci;        /* VLAN Tag Control Info.*/
  ST_UINT16 etypeID;    /* Ethertype ID.         */
  ST_UINT16 appID;      /* APP ID.               */
  } ETYPE_INFO;
```

SN_UNITDATA Data Structure

This structure contains Subnetwork packet information extracted from received packets (see **`clnp_snet_read`**) .

```
typedef struct {
  ST_UCHAR loc_mac [CLNP_MAX_LEN_MAC]; /*Buffer for local MAC addr        */
  ST_UCHAR rem_mac [CLNP_MAX_LEN_MAC]; /* Buffer for remote MAC addr      */
  /* WARNING: The "lpdu_len" param does NOT always contain the PDU length.*/
  /* It contains the "Length/Type" field of the MAC frame as defined in   */
  /* IEEE 802.3. Any value greater than or equal to 0x600 must be         */
  /* interpreted as the "Type" of the MAC frame.                          */
  /* Renaming this parameter would clarify the code, but too much         */
  /* existing code is already using it.                                   */
  ST_UINT16 lpdu_len;         /* IEEE 802.3 "Length/Type" field.          */
  ST_UCHAR *lpdu;             /* Pointer to LPDU buffer to send.          */
  } SN_UNITDATA;
```

MVL_MAX_DYN Data Structure

This structure contains the maximum number of various objects that may be “dynamically created” at runtime.

```
typedef struct
  {
  ST_INT aa_nvls;
  ST_INT aa_vars;
  ST_INT doms;
  ST_INT dom_nvls;
  ST_INT dom_vars;
  ST_INT journals;
  ST_INT types;
  ST_INT vmd_nvls;
  ST_INT vmd_vars;
  } MVL_MAX_DYN;

extern MVL_MAX_DYN mvl_max_dyn;
```

Parameters

`aa_nvls`	Indicates the maximum number of “dynamic” AA-specific NamedVariableLists.
`aa_vars`	Indicates the maximum number of “dynamic” AA-specific variables.
`doms`	Indicates the maximum number of “dynamic” domains
`dom_nvls`	Indicates the maximum number of “dynamic” domain-specific NamedVariableLists.
`dom_vars`	Indicates the maximum number of “dynamic” domain-specific variables.

`journals`	Indicates the maximum number of “dynamic” journals
`types`	Indicates the maximum number of “dynamic” data types
`vmd_nvls`	Indicates the maximum number of “dynamic” VMD-specific NamedVariableLists.
`vmd_vars`	Indicates the maximum number of “dynamic” VMD-specific variables.

New Functions

scl_info_destroy

Usage: This function destroys all the info stored in the **`SCL_INFO`** structure by **`scl_parse`**, and frees up the associated memory.

Function Prototype:

```
ST_VOID scl_info_destroy (SCL_INFO *scl_info);
```

Parameters:

`scl_info`	This is a pointer to a **`SCL_INFO`** structure to be destroyed.

Return Value: `ST_VOID` (Ignored)

datamap_cfg_read

Usage: This optional function reads a data mapping configuration file and maps the data for all variables in the global VMD, **`mvl_vmd`**.

Function Prototype:

```
ST_RET datamap_cfg_read (ST_CHAR *in_filename
                        ST_CHAR *out_filename);
```

Parameters:

`in_filename`	This is a pointer to the input configuration file name.
`out_filename`	This is a pointer to the output configuration file name

Return Value: `ST_RET` `SD_SUCCESS` or an error code.

Comments: The input configuration file is a simple ASCII text file. Each line contains the mapping for a single "leaf" in 3 columns, as follows:

COLUMN #1: Domain name

COLUMN #2: Leaf name

COLUMN #3: User text to be passed to leaf function

Note: All the mapping information is passed to every leaf function in a **`DATA_MAP`** structure. See **`u_custom_rd_ind`** in **userleaf.c** to see how this structure may be accessed. **Note:** If the user's leaf functions do not need the information configured by this function, then this function need not be called.

datamap_cfg_destroy

Usage: This function removes all mappings created by the **`datamap_cfg_read`** function and frees the associated buffers.

Critical Note: This must be called ***BEFORE*** the variables are removed, or it will be impossible to remove the mapping.

Function Prototype:

```
ST_VOID datamap_cfg_destroy ();
```

Parameters: None

Return Value: `ST_VOID` (Ignored)

mvlu_rpt_destroy_scan_ctrl

Usage: This function destroys a **`MVLU_RPT_SCAN_CTRL`** structure created by the **`mvlu_rpt_create_scan_ctrl`** function or the **`mvlu_rpt_create_scan_ctrl2`** function.

Function Prototype:

```
ST_VOID mvlu_rpt_destroy_scan_ctrl (MVLU_RPT_SCAN_CTRL *scanCtrl);
```

Parameters:

`scanCtrl` Indicates a pointer the **`MVLU_RPT_SCAN_CTRL`** structure to be destroyed.

Return Value: `ST VOID` (Ignored)

clnp_snet_read_hook_add

Usage: This function adds a subnetwork read hook function to do custom processing of received packets.

Function Prototype:

```
ST_RET clnp_snet_read_hook_add (ST_RET (*usr_fun)
                                (SN_UNITDATA *sn_req));
```

Parameters:

`usr_fun`	Pointer to a user function to be called when each subnetwork packet is received. The user function must return **`SD_SUCCESS`** if it processed the packet, or an error code if it did not process the packet (i.e., the main code should process the packet).

Return Value: `ST_RET` `SD_SUCCESS` or Error.

Note: See **cli_goose.c** for an example of how this function may be used for processing GOOSE messages.

clnp_snet_read_hook_remove

Usage: This function removes a subnetwork read hook function (i.e., stops custom processing).

Function Prototype:

```
ST_RET clnp_snet_read_hook_remove (ST_RET (*usr_fun)
                                   (SN_UNITDATA *sn_req));
```

Parameters:

`usr_fun`	Pointer to a user function added previously by calling **`clnp_snet_read_hook_add`**.

Return Value: `ST_RET` `SD_SUCCESS` or Error.

clnp_snet_frame_to_udt

Usage: This function extracts data from a raw frame and stores it in a **`SN_UNITDATA`** structure, needed by other subnetwork functions. This function should work on any platform, and should simplify the porting of the subnetwork interface to new platforms.

CRITICAL: The caller must initialize **`sn_req->lpdu`** to point to an allocated buffer before calling this function.

Note: To see how this function is used on Windows or LINUX, see **clnp_w32.c** or **clnp_linux.c**.

Function Prototype:

```
ST_RET clnp_snet_frame_to_udt (ST_UINT8 *frame_buf,
                               ST_INT frame_len,
                               SN_UNITDATA *sn_req,
                               ST_INT udata_max_len);
```

Parameters:

`frame_buf`	This is a pointer to the raw frame buffer.
`frame_len`	Indicates the length of the raw frame in bytes.
`sn_req`	Pointer to a structure used to store the result of the extraction.
`udata_max_len`	Indicates the maximum user data length. This length must match the size of the allocated buffer **`sn_req->lpdu`**, and should normally be set to **`ETHE_MAX_LEN_UDATA`** to allow for the largest possible Ethernet frame.

Return Value: `ST_RET` `SD_SUCCESS` or Error.

etype_hdr_decode

Usage: This function decodes the Ethertype header of a received frame. It assumes the **`sn_req->lpdu_len`** contains the IEEE 802.3 Length/Type field and **`sn_req->lpdu_len`** points at the IEEE 802.3 MAC Client Data. It returns a pointer to the APDU (**A**pplication **P**rotocol **D**ata **U**nit) and sets the length of the APDU

Note: The use of this function is demonstrated in **iec_rx.c**.

Function Prototype:

```
ST_UCHAR *etype_hdr_decode(SN_UNITDATA *sn_req,
                           ETYPE_INFO *info,
                           ST_INT *apduLen);
```

Parameters:

`sn_req`	Pointer to the subnetwork frame to be decoded.
`info`	Pointer to a structure to contain the decoded Ethertype header information
`apduLen`	Pointer to the length of the APDU (after the Ethertype header). The function sets the value of the integer pointed to by this argument

Return Value: `ST_UCHAR *` Pointer to the APDU (after Ethertype header)

ms_rt_el_tag_text

Usage: This function converts the **`el_tag`** member of a **`RUNTIME_TYPE`** structure to text.

Function Prototype:

```
ST_CHAR *ms_rt_el_tag_text (SD_CONST RUNTIME_TYPE *rt_type);
```

Parameters:

`rt_type`	Pointer to the **`RUNTIME_TYPE`** structure

Return Value: `ST_CHAR *` Pointer to a static string indicating the **`el_tag_value`**.

Example: **`If (rt_type>el_tag==RT_STR_START)`**, this function returns a pointer to a static string **`RT_STR_START`**.

get_next_string

Usage: This function returns a pointer to the next string found in the input buffer (string may be surrounded by “double quotes”), up to the next delimiter in the input. It ignores leading spaces or commas in the input.

Note: This function works much like the standard function **`strtok`**, but it allows extracting “quoted strings”, and it is much better at discarding extra delimiter characters.

Function Prototype:

```
ST_CHAR *get_next_string (ST_CHAR **ptrptr,
                          ST_CHAR *delimiters);
```

Parameters:

`ptrptr`	Pointer to a pointer to the current position in the input buffer. The current position is changed by this function.
`delimiters`	Pointer to a set of delimiter characters (like **`strtok`** or **`strpbrk`**).

Return Value: `ST_CHAR *` Pointer to the next string in the input. NULL on an error or the end of the input string.

Logging Functions

logcfgx_ex

Usage: This function defined in **logcfgx.c** parses specified **logcfg.xml** file to set up application logging parameters and masks. This function is prototyped in **slog.h**. SISCO libraries contain **`LOGCFG_VALUE_GROUP`** variables that specify logging masks for each component. All possible variables are listed in the **slog.h**. The sample in the *Enhanced Logging Features* section shows several calls to **`logCfgAddMaskGroup`**, passing the most important **`LOGCFG_VALUE_GROUP`** variables for MMS users. A user could add additional logging parameters to the provided sample, **logcfg.xml**. Additionally other program parameters could also be configured through this file when needed. The **`LOGCFGX_VALUE_MAP`** and **`LOGCFG_VALUE_GROUP`** need to be created in user application as shown in the *Enhanced Logging Features* section described previously.

Function Prototype:

```
ST_RET logcfgx_ex (LOG_CTRL *lc,
                   ST_CHAR   *logFileName,
                   ST_CHAR    *fileNamePrefix,
                   ST_BOOLEAN masksOnly,
                   ST_BOOLEAN saveTagVals);
```

Parameters:

`lc`	This is a pointer to the log control structure of type **`LOG_CTRL`**. Typically, it would be the global logging control **`sLogCtrl`**.
`LogFileName`	This is the logging configuration XML file name. This file must have structure as shown in the provided sample **logcfg.xml**.
`fileNamePrefix`	This is the optional prefix for XML parameter of type **`LOGCFG_DATATYPE_FILENAME`**. It may be used to specify base directory for any file name configured in the XML file.
`masksOnly`	This option, if set to **`SD_TRUE`**, will parse only the masks from the XML logging configuration file.
`saveTagVals`	This option, if set to **`SD_TRUE`**, will save values for all tags in the XML logging configuration file. This feature is used internally by SISCO.

Return Value: `ST_RET` `SD_SUCCESS` If the function is successful; otherwise `SD_FAILURE`

The **`dataType`** in a **`LOGCFGX_VALUE_MAP`** structure can be one of following defines:

```
_LOGCFG_DATATYPE_UINT_MASK
_LOGCFG_DATATYPE_UINT32_MASK
_LOGCFG_DATATYPE_RUINT32_MASK
_LOGCFG_DATATYPE_BOOLEAN
_LOGCFG_DATATYPE_INT
_LOGCFG_DATATYPE_LONG
_LOGCFG_DATATYPE_INT16
_LOGCFG_DATATYPE_INT32
_LOGCFG_DATATYPE_UINT
_LOGCFG_DATATYPE_ULONG
_LOGCFG_DATATYPE_UINT16
_LOGCFG_DATATYPE_UINT32
_LOGCFG_DATATYPE_STRING
_LOGCFG_DATATYPE_STRINGBUF
_LOGCFG_DATATYPE_DOUBLE
_LOGCFG_DATATYPE_FILENAME
_LOGCFG_DATATYPE_CALLBACK
```

logCfgAddMaskGroup

Usage: This function defined in **logcfgx.c** adds **`LOGCFG_VALUE_GROUP`** variables to the parsing engine for use during the XML logging configuration file parsing. This function is prototyped in **slog.h**.

Function Prototype:

```
ST_VOID logCfgAddMaskGroup (LOGCFG_VALUE_GROUP *logMaskGroup);
```

Parameters:

`logMaskGroup`	This is a pointer to the **`LOGCFG_VALUE_GROUP`** variable.

Return Value: `none`

slog_max_msg_size_set

Usage: This function changes the maximum size allowed for each log message.

Function Prototype:

```
ST_RET slog_max_msg_size_set (LOG_CTRL *lc,
                              ST_INT max_msg_size);
```

Parameters:

`lc`	This is a pointer to the **`LOG_CTRL`** structure used to control logging.
`max_msg_size`	Indicates the maximum size allowed for each log message.

Return Value: `ST_RET` `SD_SUCCESS` or Error.

slog_max_msg_size_get

Usage: This macro is used to get the maximum message size. This macro replaces a function, but is much faster.

Macro: `#define slog_max_msg_size_get(log_ctrl)(log_ctrl->max_msg_size)`

Parameters:

`log_ctrl`	This is a pointer to the **`LOG_CTRL`** structure used to control logging.

Return Value: `ST_INT` Maximum size allowed for a log message.

IEC 61850 Functions

mvl61850_ctl_chk_sbo

Usage: This function should be called from the "leaf" function when the IEC 61850 "SBO" attribute is being read (i.e., performing Control Model 'Select' Service when **ctlModel = sbo-with-normal-security**). It checks if the client is allowed to perform 'Select'. If 'Select' is allowed, it reserves a **`MVL_SBO_CTRL`** structure and returns a pointer to it.

Function Prototype:

```
MVL_SBO_CTRL *mvl61850_ctl_chk_sbo (MVLU_RD_VA_CTRL *mvluRdVaCtrl);
```

Parameters:

`mvluRdVaCtrl`	This is a pointer to a **`MVLU_RD_VA_CTRL`** structure. The caller should use the same pointer that was passed to the "leaf" function.

Return Value: If successful, a pointer to the **`MVL_SBO_CTRL`** structure (**`sbo_var`** member of the structure contains the name to send in the read response)
If failed, returns NULL.

CRITICAL NOTE: If the return is NOT NULL and the caller decides not to allow the 'Select" for some other reason, the caller must free the structure by calling **`mvlu_sbo_ctrl_free`** and passing this pointer as the argument.

---

mvl61850_ctl_chk_sbow

Usage: This function should be called from a "leaf" function when the IEC 61850 **SBOw** attribute is being written (i.e., performing Control Model 'Select' Service when **ctlModel = sbo-with-enhanced-security**). It checks is the client is allowed to perform 'Select'. If 'Select' is allowed, it reserves a **`MVL_SBO_CTRL`** structure and returns a pointer to it.

Function Prototype:

```
MVL_SBO_CTRL *mvl61850_ctl_chk_sbow (MVLU_WR_VA_CTRL *mvluWrVaCtrl);
```

Parameters:

`mvluWrVaCtrl`	This is a pointer to a **`MVLU_WR_VA_CTRL`** structure. The caller should use the same pointer that was passed to the "leaf" function.

Return Value: If successful, a pointer is returned to the **`MVL_SBO_CTRL`** structure.
If failed, returns NULL.

CRITICAL NOTE: If the return is NOT NULL and the caller decides not to allow the 'Select' for some other reason, the caller MUST free the structure by calling **`mvlu_sbo_ctrl_free`** and passing this pointer as the argument.

NOTE: The **SBOw** is a structure, so several leaf functions are called when it is written. It is most efficient if this function is only called from one of those leaf functions. The sample code in **userleaf.c** calls it only when the mandatory **Check** attribute is being written.

mvl61850_ctl_chk_state

Usage: This function checks the value of **`ctlModel`**, etc. to determine if the control is the right type and in the right state to allow writing of the **`Oper`** structure. It should be called from the user “leaf” function for writing components of the **`Oper`** structure.

Function Prototype:

```
ST_RET mvl61850_ctl_chk_state (MVLU_WR_VA_CTRL *mvluWrVaCtrl);
```

Parameters:

`mvluWrVaCtrl`	This is a pointer to a **`MVLU_WR_VA_CTRL`** structure. The caller should use the same pointer that was passed to the “leaf” function.

Return Value: `ST_RET`

`SD_SUCCESS` if **`Oper`** may be written.

Error code if **`Oper`** may *not* be written.

mvl61850_beh_stval_rd_ind

Usage: This function may be used as the “leaf” function for reading the **`stVal`** component of the **`Beh`** attribute in an IEC 61850 server application. It computes the value of **`stVal`** based on the values of **`Mod$stVal`** according to the rules in IEC 61850-7-4.

Note: See **userleaf.c** for an example of how to use this function in an IEC 61850 application.

Function Prototype:

```
ST_VOID mvl61850_beh_stval_rd_ind (MVLU_RD_VA_CTRL *mvluRdVaCtrl);
```

Parameters:

`mvluRDVaCtrl`	This is a pointer to a **`MVLU_RD_VA_CTRL`** structure.

Return Value: `ST_VOID` (Ignored)

mvl61850_ctl_command_termination

Usage: This function sends an IEC 61850 Command Termination request to implement controls with “enhanced security”.

Note: See **scl_srvr.c** for an example of how to use this function in an IEC 61850 application.

Function Prototype:

```
ST_RET mvl61850_ctl_command_termination (MVL_NET_INFO *net_info,
                                         ST_CHAR *oper_ref,
                                         ST_RET status,
                                         MVL61850_LAST_APPL_ERROR *last_appl_error);
```

Parameters:

`net_info`	This is a pointer to a **`MVL_NET_INFO`** structure.
`oper_ref`	This is the ObjectReference for the **`Oper`** attribute being controlled.
`status`	Indicates the completion status for the control (**`SD_SUCCESS`** or **`SD_FAILURE`**)
`last_appl_error`	This is a pointer to a **`MVL61850_LAST_APPL_ERROR`** structure containing the data to be sent in the **`LastApplError`** variable of the CommandTermination request if the completion status is **`SD_SUCCESS`**. If the completion status is **`SD_FAILURE`**, this data is not used.

Return Value: `ST_RET` `SD_SUCCESS` if CommandTermination was sent, or Error.

mvl61850_rpt_ctrl_destroy_all

Usage: This function destroys all IEC 61850 report controls.

Note: See **scl_srvr.c** for an example of how to use this function in an IEC 61850 application.

Function Prototype: `ST_VOID mvl61850_rpt_ctrl_destroy_all ( );`

Parameters: none

Return Value: none

mvlu_rpt_ctrl_destroy_all

Usage: This function destroys all UCA report controls.

Note: See **uca_srvr.c** for an example of how to use this function in a UCA application.

Function Prototype: `ST_VOID mvlu_rpt_ctrl_destroy_all ( );`

Parameters: none

Return Value: none

u_mvl61850_ctl_oper_begin

Usage: This call back function must be supplied by the user. It is called from MVL when the **`Oper`** structure is being written. It is called before any "leaf" functions are called for the **`Oper`** attributes.

Note: See **scl_srvr.c** for an example of how to use this function in an IEC 61850 application.

Function Prototype: `ST_VOID u_mvl61850_ctl_oper_begin (ST_CHAR *oper_ref);`

Parameters:

`oper_ref`	This is the ObjectReference for the **`Oper`** attribute being controlled.

Return Value: `ST_VOID` (Ignored)

u_mvl61850_ctl_oper_end

Usage: This call back function must be supplied by the user. It is called from MVL when the **`Oper`** structure is being written. It is called after all “leaf” functions are called for the **`Oper`** attributes.

Note: See **scl_srvr.c** for an example of how to use this function in an IEC 61850 application.

Function Prototype:

```
ST_VOID u_mvl61850_ctl_oper_end (MVL_NET_INFO *net_info,
                                 ST_CHAR *oper_ref,
                                 MVL_VAR_ASSOC *base_var);
```

Parameters:

`net_info`	This is a pointer to a **`MVL_NET_INFO`** structure that
`oper_ref`	This is the ObjectReference for the **`Oper`** attribute being controlled.
`base_var`	This is a pointer to a **`MVL_VAR_ASSOC`** structure for the Logical Node that contains the **`Oper`** attribute.

Return Value: `ST_VOID` (Ignored)

Modified Functions

scl2_datatype_create_all

Usage: Creates MMS Data types for all Logical Node Types (LNodeType) defined in SCL.

Function Prototype:
```
ST_RET scl2_datatype_create_all (MVL_VMD_CTRL *vmd_ctrl,
                                 SCL_INFO *sclInfo,
                                 ST_INT max_rt_num,
                                 ST_BOOLEAN use_names,
                                 ST_CHAR *name_prefix);
```

Parameters:

`vmd_ctrl`	A pointer to an **`MVL_VMD_CTRL`** structure in which to add types.
`sclInfo`	This is the main **`SCL_INFO`** structure where all the SCL info stored.
`max_rt_num`	Indicates the maximum number of `RUNTIME_TYPE` for each LNodeType.
`use_names`	This flag if **`SD_TRUE`**, will generate a name for each type.
`name_prefix`	This is a pointer to unique prefix to add to each type name. This is only used if **`use_names==SD_TRUE`**.

Return Value: `SD_SUCCESS` or error code

scl2_ld_create_all

Usage: Creates all Logical Devices from information extracted from the SCL file (stored in the **`SCL_INFO`** structure). This includes creating the Logical Device (MMS Domain), and within the Logical Device: all Logical Nodes (MMS variables), all Data Sets (MMS NamedVariableLists), and all ReportControlBlocks.

NOTE: *The functions **`scl_parse`** and **`scl2_datatype_create_all`** must be called first to read the SCL file and initialize the **`SCL_INFO`** structure passed to this function, and to create all MMS Data Types needed by the Logical Device.*

Function Prototype:

```
ST_RET scl2_ld_create_all (MVL_VMD_CTRL *vmd_ctrl,
                           SCL_INFO *sclInfo,
                           ST_UINT reportScanRate,
                           ST_INT brcb_bufsize,
                           ST_BOOLEAN is_client);
```

Parameters:

`vmd_ctrl`	Pointer to an **`MVL_VMD_CTRL`** structure in which to add Logical Devices.
`sclInfo`	Pointer to structure containing all information extracted from the SCL file (filled in by **`scl_parse`**).
`reportScanRate`	Report Scan Rate (in milliseconds). If this value is not 0 (zero) and Report Control Blocks (RCB) are configured in the SCL file, all members of the Report Data Set will automatically be scanned for data changes at this rate. If this value is 0 (zero), scanning will take place every time **`mvl61850_rpt_service`** is called.
`brcb_bufsize`	Buffer size to use for each buffered report (if configured).
`is_client`	If this flag is set, Client model is created (i.e., Control Blocks NOT created).

Return Value: `SD_SUCCESS` or error code

SCL Server Sample Application

The "SCL Server" sample application reads an SCL file (i.e., an input file conforming to the "**S**ubstation **C**onfiguration description **L**anguage" defined in IEC-61850-6). The SCL input is used to dynamically create all MMS objects. All source code and sample configuration files are found in the directory **\mmslite\mvl\ust\scl_srvr**.

This sample application reads the SCL file and creates MMS objects by calling the functions **`scl_parse`**, **`scl2_datatype_create_all`**, and **`scl2_ld_create_all`**. The configuration file, **startup.cfg** is read to get information to pass to **`scl_parse`** (see **startup.c**). This same information could be obtained in many different ways in a real application.

After all MMS objects are created, data mapping is done by calling **`datamap_cfg_read`** to read the file **datamap.cfg** (see **usermap.c**). The user should create this file to contain information to map "leafnames" to "user-defined text". The user-defined text" may then be used in leaf functions. A **`DATA_MAP`** structure (see **usermap.h)** is allocated for each entry (one line of this file) and the entry information is stored in the **`DATA_MAP`** structure. This structure may be accessed in the "read and write leaf functions" as demonstrated in **`u_custom_rd_ind`** and **`u_custom_wr_ind`** (see **userleaf.c**). These two example leaf functions may be customized by the developer to use the **`DATA_MAP`** information to control data access in any way appropriate for the user application. The expected format of this file is 3 columns separated by tabs. One line of the file is used to store information for one "leaf" by allocating one **`DATA_MAP`** structure and copying the information to it. The columns must contain the following:

COLUMN #1: Domain name
COLUMN #2: Leaf name
COLUMN #3: User text to be used by leaf function

To help to create the **datamap.cfg** file, every time the "SCL Server" application is executed, an output file **datamapout.cfg** is generated that contains all valid entries from this file plus "sample entries" for any "leaf" that is NOT configured in this file. The "sample entries" in **datamapout.cfg** may be copied to this input file and modified to contain the appropriate "User Text" in COLUMN #3.

IMPORTANT**:** Any time objects are added, deleted, or changed in the SCL Configuration file (**scl.xml**), this file should be updated with the appropriate data mapping.

NOTE: The use of the **datamap.cfg** input file is only one method to provide data mapping. The user is free to use other data mapping methods if necessary.

AA-Specific Variables in the Foundry ODF Input File

Objects with the `AA_SCOPE:` prefix are no longer allowed in the Foundry ODF input file. These objects are rarely needed. If they are needed, it is recommended that they be created dynamically using code similar to the following:

```
  {
      /* Add AA_SPEC variable named "rptCtrl" on each calling connection.*/
      /* To do same thing for called conns, change "calling" to "called".*/
      ST_INT16 calling_data [MAX_CALLING]; /*data to use for calling conns*/
      ST_INT type_id, j;
      OBJECT_NAME object_name;
      MVL_VAR_ASSOC *var_assoc;

      /* This simple example assumes number of conns within limits.     */
      assert (mvl_cfg_info->num_calling <= MAX_CALLING);

      /* Construct AA-Specific object name.           */
      object_name.object_tag = AA_SPEC;
      object_name.obj_name.vmd_spec = "rptCtrl";
      type_id = mvl_typename_to_typeid ("I16");
```

```
/* Create AA-Specific variable on every connection.   */
for (j = 0; j< mvl_cfg_info->num_calling; j++)
  var_assoc = mvl_vmd_var_add (&object_name,
                  &mvl_calling_conn_ctrl[j],    /* MVL_NET_INFO * */
                  type_id,
                  &calling_data[j], /* different data for each var*/
                  NULL,             /* MVL_VAR_PROC * */
                  SD_TRUE);   /* ALWAYS copy name to var_assoc    */
}
```

If an application cannot use dynamic objects for some reason, the old way may be used by simply compiling the Foundry output source code with **`OBSOLETE_AA_OBJ_INIT`** defined, but this approach is not recommended.

Known Software Anomalies

NamedVariableLists created by Foundry may not be used in Reports

The MMS-EASE *Lite* Reporting Subsystem does not work if the NamedVariableList (Data Set) is created by Foundry. The NamedVariableList must be created dynamically using **`mvlu_derive_rpt_ds`** or **`mvl_vmd_nvl_add`**, as in the sample applications.

Notes for Phar Lap TNT ETS Operating System Only

heap checking

The heap checking function **`_heapwalk`** does not work correctly on the Phar Lap TNT ETS operating system. You should disable the heap checking code by setting the following:

```
m_heap_check_enable = SD_FALSE;
```

This is usually done in **client.c**, **server.c**, or **uca_srvr.c**.