IEC 61850 PIS-10 Software Stack

How to Update from PIS-10 Version 1 to Version 2

Application Note

Document 500-0034 v.1.01

Introduction

The PIS-10 IEC 61850 V2 software library includes edition 2 functionality and some minor changes to the API function calls and the callbacks generated by the PIS-10 software library. This document will guide you through each change in the API and how to update your existing V1 code to V2 and implement the new functions for V2. The Main API functions calls have not been changed, the majority of changes are based around the IEC61850 Controls functionality.

Please refer to our navigable online API User Documentation http://api.systemcorp.com.au for more information on other API functions and data structures.

IEC61850_Create(…)

The function call to *IEC61850_Create* has not changed from V1 to V2 but the *IEC61850_Parameters* structure that is passed into the create function has been modified. These changes are documented bellow

IEC61850_Ed1_Ed2_Flag:

The most important change is the new variable `enum IEC61850_Ed1_Ed2_Flag Ed1_Ed2_Flag;`
This variable in the structure allows you to set the behaviour of the stack as `IEC61850_Edition1` or `IEC61850_Edition2`.

Example:

```
tServerParam.Ed1_Ed2_Flag = IEC61850_Edition2; // set the PIS-10 to Edition2
```

uiClientMaxAssociations:

For client applications the variable *uiClientMaxConnections* has been removed and its functionality has been merged into `uiMaxAssociations.` This means that the client and server functionality is now consistent when setting the maximum associations the PIS-10 will allow.

Example:

~~`tClientParam.uiClientMaxConnections = MaxAssocVal;`~~`//V1 code on client`
The above code should be replaced with:

```
tClientParam.uiMaxAssociations = MaxAssocVal; //V2 code to set max associations on client
```

uiMMSTimeout:

For client applications it defines the amount of time the PIS-10 will wait before timing out on an MMS message.

Example:

```
tClientParam. uiMMSTimeout = 1500; //V2 code to set the MMS timeout to 1.5 seconds
```

ptReadMultipleCallback:

IEC61850_ReadMultipleCallback is a new optional callback for the server if this callback is set *IEC61850_ReadMultipleCallback* will be used in place of the *IEC61850_ReadCallback*. The functionality of this new callback is explained later on in this document.

Example:

```
tServerParam.ptReadMultipleCallback = ReadMultipleCallbackHandler; //Assign Multiple Read Callback
```

ptQuestionableCallback:

ptQuestionableCallback is a new optional callback for the server or client. The callback is used to notify the application when a data point is questionable due to a GOOSE data packet not being received before the TTL in the previously received GOOSE packet.

Example:

```
tServerParam.ptQuestionableCallback = QuestionableCallbackHandler; // Assign Questionable Callback
```

ptOprTestCallback:

ptOprTestCallback is a new **mandatory** callback for the server. This callback is called before the *IEC61850_ControlOperateCallback* callback and it allows user application to check the status of the hardware if it is in a controllable state.

Example:

```
tServerParam.ptOprTestCallback = OperativeTestCallbackHandler;// Assign Operative Test Callback
```

ptCmdTermCallback:

ptCmdTermCallback is a new optional callback for the client. This callback is called when a command termination message is received.

Example:

```
tClientParam.ptCmdTermCallback = CommandTerminationCallback; // Assign CMD termination Callback
```

ptErrorCallback:

ptErrorCallback is a new optional callback for the client. This callback is called when a “LastApplError” message is received.

Example:

```
tClientParam.ptErrorCallback = ErrorCallbackHandler; // Assign Error Callback Function
```

ptConnectionStatusCallback:

*ptConnectionStatusCallback*is a new optional callback for the client. This callback is called when a remote servers connection status changes

Example:

```
tClientParam.ptConnectionStatusCallback = ConnectionStatusCallback;// Assign Connection Status
Callback Function
```

ptTimestampCallback:

ptTimestampCallback is a new optional callback for the server and client. This callback is called when a timestamp is needed by the IEC61850 stack

Example:

```
tClientParam.ptTimestampCallback= TimestampCallbackHandler; // Assign Timestamp Callback Function
```

IEC61850_ControlSelectCallback(…)

The *IEC61850_ControlSelectCallback* has been slightly modified instead of having separate values for the **iSyncroCheck** and the **iInterlockCheck** they have been packaged inside a new structure called **ptSelectParameters** of type struct *IEC61850_CommandParameters*. This **ptSelectParameters** structure will contain **iSyncroCheck** and **iInterlockCheck** but also *bTestMode* so you know when a control is in test mode.

Example:

```
enum eCommandAddCause SelectCallbackHandler( struct IEC61850_DataAttributeID * ptControlID, struct
IEC61850_DataAttributeData * ptSelectValue, struct IEC61850_CommandParameters* ptSelectParameters)
{
   enum eCommandAddCause eErrorCode = IEC61850_COMMAND_ERROR_NONE;
   /*Do Select Code Here*/
   return eErrorCode;
}
```

IEC61850_ControlOperateCallback(…)

The *IEC61850_ControlOperateCallback* has been slightly modified instead of having separate values for the **iSyncroCheck** and the **iInterlockCheck** they have been packaged inside a new structure called **ptOperateParameters** of type struct *IEC61850_CommandParameters*. This **ptOperateParameters** structure will contain **iSyncroCheck** and **iInterlockCheck** but also *bTestMode* so you know when a control is in test mode.

Example:

```
enum eCommandAddCause OperateCallbackHandler( struct IEC61850_DataAttributeID * ptControlID, struct
IEC61850_DataAttributeData * ptOperateValue, struct IEC61850_CommandParameters* ptOperateParameters)
{
   enum eCommandAddCause eErrorCode = IEC61850_COMMAND_ERROR_NONE;
   /*Do Operate Code Here*/
   return eErrorCode;
}
```

IEC61850_ControlCancelCallback(…)

The *IEC61850_ControlCancelCallback* has been slightly modified to include a new structure called **ptCancelParameters** of type struct *IEC61850_CommandParameters*. This **ptCancelParameters** structure will contain **iSyncroCheck**, **iInterlockCheck** and *bTestMode*.

Example:

```
enum eCommandAddCause CancelCallbackHandler( struct IEC61850_DataAttributeID * ptControlID, struct
IEC61850_CommandParameters* ptCancelParameters)
{
   enum eCommandAddCause eErrorCode = IEC61850_COMMAND_ERROR_NONE;
   /*Do Cancel Code Here*/
   return eErrorCode;
}
```

IEC61850_ControlOperativeTestCallback(…)

The *IEC61850_ControlOperativeTestCallback* is a new **Mandatory** callback that has been implemented as per the IEC61850 standard. The purpose of this callback is to test the hardware state to be sure it is ready to do an Operate. If the hardware is not in a ready state an error value should be returned.

Example:

```
enum eCommandAddCause OperativeTestCallbackHandler(void * ptUserData, struct
IEC61850_DataAttributeID* ptControlID, struct IEC61850_CommandParameters* ptOperativeTestParameters)
{
   enum eCommandAddCause eErrorCode = IEC61850_COMMAND_ERROR_NONE;
   /*Do Operative Test Code Here*/
   return eErrorCode;
}
```

IEC61850_ControlCommandTerminationCallback(…)

The *IEC61850_ControlCommandTerminationCallback* is a new optional callback that will be called when a Command Termination message is received by the client. This can be used to know if a control was successful or not.

Example:

```
enum eCommandAddCause CommandTerminationCallback(void * ptUserData, struct IEC61850_DataAttributeID
* ptControlID, struct IEC61850_DataAttributeData * ptCmdTermValue)
{
   enum eCommandAddCause eErrorCode = IEC61850_COMMAND_ERROR_NONE;
   /*Do Command Termination Code Here*/
   return eErrorCode;
}
```

IEC61850_ErrorCallback(…)

The *IEC61850_ IEC61850_ErrorCallback* is a new optional callback that will be called when a “LastApplError” message is received. The **ptErrorParamtrs** of type *struct IEC61850_ErrorParameters* contains the necessary information about the error that has been received including the error value and string that describes the error.

Example:

```
enum eCommandAddCause ErrorCallbackHandler(void * ptUserData, struct IEC61850_DataAttributeID *
ptDataAttributeID,  struct IEC61850_ErrorParameters * ptErrorParamtrs)
{
   enum eCommandAddCause eErrorCode = IEC61850_COMMAND_ERROR_NONE;
   /*Do Error Processing Code Here*/
   return eErrorCode;
}
```

IEC61850_DataPointQuestionableCallback(…)

The *IEC61850_DataPointQuestionableCallback* is a new optional callback for the server or client. The callback is used to notify the application when a data point is questionable due to a GOOSE data packet not being received before the TTL in the previously received GOOSE packet.

Example:

```
void QuestionableCallbackHandler(void * ptUserData, struct IEC61850_DataAttributeID *
ptDataAttributeID)
{
   /*Do Questionable Data Point Processing Code Here*/
}
```

IEC61850_ReadMultipleCallback(…)

The *IEC61850_ReadMultipleCallback* is a new optional callback for the server or client. The callback is called when a structure is read by a client. When a structure is read by a client then this callback will provide all data points inside the structure that have private IDs as an array of data points and array of private IDs along with a count that contains the number of points inside the arrays. The arrays are symmetrical this means that the data value in **ptReturnedValue** at element 0 will have a matching private ID in **ptDataAttributeID** at element 0.
If this callback is set then *IEC61850_ReadCallback* will **NOT** be called *IEC61850_ReadMultipleCallback* will be called instead.

Example:

```
enum IEC61850_CallbackReturnServiceErrorCodes ReadMultipleCallbackHandler(void * ptUserData, struct
IEC61850_DataAttributeID * ptDataAttributeID, struct IEC61850_DataAttributeData * ptReturnedValue,
int iCount)
{
   enum IEC61850_CallbackReturnServiceErrorCodes eErrorCode = IEC61850_CB_ERROR_NONE;
   /*Do Multiple Read Data Point Code Here*/
   return eErrorCode;
}
```

IEC61850_ConnectionStatusCallback (…)

The *IEC61850_ConnectionStatusCallback* is a new optional callback for the client. The callback is called when the connected state of a remote server has changed. A remote server can be in 5 possible states that are defined by `enum IEC61850_ServerStatusVal`.

Example:

```
void ConnectionStatusCallback(struct IEC61850_ServerStatusArray inConnectedServersArray)
{
   Integer32 i =0;

   for(i =0; i < inConnectedServersArray.ServerStatusArraySize; i++)
   {
      struct IEC61850_ServerStatus * ServerStatus = &(inConnectedServersArray.ServerStatusArray[i]);
      if(ServerStatus != NULL)
      {
         switch(ServerStatus->isConnected)
         {
          /*Do Connection Status Callback Code Here*/
         }
      }
   }
}
```

IEC61850_TimeStampCallback (…)

The *IEC61850_TimeStampCallback* is a new optional callback for the server and client. The callback is called when the IEC61850 stack needs to generate a timestamp. If the callback is set then the IEC61850 stack will call this function with a pointer to a timestamp `struct IEC61850_TimeStamp` the user code can then populate the structure with the current time and this time will be used. If the callback is not set then the IEC61850 stack will get the current time directly from the operating system.

Example:

```
void TimeStampCallback (struct IEC61850_TimeStamp *CurrentTime)
{
   /*Populate Timestamp structure Code Here*/
}
```

If you need assistance

Please refer to our navigable online API User Documentation http://api.systemcorp.com.au for more information on other API functions and data structures.

All technical questions must be sent to our support email address: **support@systemcorp.com.au**
Upon receiving your question(s), it will get logged in our support system and you will receive and acknowledgement which will include a ticket ID. Please refer to your ticket ID when you are following up about enquiry.