CIMPLICITY 11

Open Interface API Reference

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address:

doc@ge.com

Open Interface API Reference

Chapter 1. About CIMPLICITY Open Interface

The CIMPLICITY Base System functionality - Point Management, Alarm Management and Data Logging facilities, as well as a full-functioned User Interface - enables CIMPLICITY users to collect data for reporting and to visualize data via lists, graphic status displays and alarms. Standard data communications capability make CIMPLICITY software a factory floor tool that can provide services such as those listed below.

- Downtime reporting
- Production reporting
- Records of production counts at work stations
- Graphic monitoring of automatic data point values
- Fault reporting via direct point values and alarms

CIMPLICITY software's flexible system architecture and modular design allows for easy add-on of functionality. These add-ons can be created programmatically using CIMPLICITY's API tool kits.

Important: If you plan to work with the API tool kit, you must have Microsoft Visual Studio 2017 installed.

The CIMPLICITY API tool kits are:

API Tool Kit	Description
Login/Logout *(page 25)*	Create and customize a Login/Logout box and/or use API's to give programmed applications some control over the CIMPLICITY login/logout activities.
Alarm Management *(page 33)*	Provides an interface for application programs to generate CIMPLICITY alarms based on the specific requirements of the application.
Alarm Interested Process *(page 127)*	Create a process that receives alarm information from the CIMPLICITY Alarm Manager.
External Alarm State Management *(page 143)*	Create an External Alarm State Manager (XASMgr) to manage CIMPLICITY alarms.
Alarm Viewer *(page 169)*	Provides an interface for application programmers to develop full-featured custom alarm viewers to meet special needs.
Device Communications *(page 253)*	• Support communications to devices for which standard CIMPLICITY software communication enablers are not available. • Provides the libraries and build procedures you need to create communication enablers that perform the same functions as standard CIMPLICITY software communication enablers.

API Tool Kit	Description
Point Management *(page 367)*	Provides an interface between application programs and CIMPLICITY software's ability to monitor data point values.
CIMPLICITY to Windows Server *(page 496)*	Provides CIMPLICITY access to point data from other Microsoft Windows products such as Microsoft Excel through DDE.

Chapter 2. Login/Logout API

About the Login/Logout API

CIMPLICITY provides a default Login/Logout box. You can create and customize a Login/Logout box and/or use API's to give programmed applications some control over the CIMPLICITY login/ logout activities.

Code a Login Box

Code a Login Box

1. Code your own login box.
2. Install the customized login box to replace the CIMPLICITY default login box.

   The easiest way to change the login box is to modify the Visual Basic Project that is supplied. Coding your own login box is also simple if you understand:

   - OLE Automation
   - How to build an Automation Server using your development language (e.g. Visual C++, etc)

Example Login Box Installation

1. Begin the CIMPLICITY server installation from your CIMPLICITY DVD.
2. Select Application Option.
3. Check the Device Communications Toolk it (abbreviated as **Devcom Toolkit**) as one of the components to install.

   This will install an **oilogin.zip** file in your CIMPLICITY\HMI\API\OILOGIN directory.

   Oilogin.zip is a compressed zip file contains an example Visual Basic 5.0 project and the resultant binary. You may modify the Visual Basic Project or use the contained binary. The project implements three differently styled login pads that are suitable for use on touch screens.

Step 2: Unzip the OILOGIN.ZIP file:

Use PKUNZIP or an equivalent application to uncompress the example files.

Tips about Coding a Login Box

The CIMPLICITY Global Parameter **LOGIN_INTERFACE** provides the name of the Automation Interface implemented by your Automation Server.

The Automation Server may be:

- In-process (e.g. an OCX) or
- Out of process (e.g. and EXE.)

The Automation Interface must provide an implementation for the following method:

In Visual C++ the definition would be:

BOOL DoLogin(BSTR* Project, BSTR* UserId, BSTR* Password)

Where	Is	
Project	Input	The name of the project to log in to.
UserId	Output	The user id supplied.
Password	Output	The password supplied.
The Return Value is:		
True	If the User Id and Password are supplied	
False	If the user has quit trying to login.	

In the case of a failed login, **DoLogin** will be called again.

Install a Customized Login Box

Install a Customized Login Box

You will need to register the replacement logging control with Windows and then install it in CIMPLICITY in order for it to take over the logging process.

You can replace the CIMPLICITY default login box with your customized login box in five easy steps.

Step 1 *(page 27)*	Register the replacement logging control.
Step 2 *(page 27)*	Select the type of login box you want.
Step 3 *(page 28)*	Set a CIMPLICITY global parameter on the server.
Step 4 *(page 29)*	Set a CIMPLICITY global parameter on both the server and viewer.
Step 5 *(page 30)*	Stop and restart CIMPLICITY.

Step 1. Register the Replacement Login Control

1. Click the Windows **Start** button.
2. Select **Programs**.
3. Select **Command Prompt**.

   A DOS window opens.

4. Change to the directory where you extracted the files.

   Example

   cd\LoginAPI

5. Type from the command prompt:

   cimregdll CIMPLogin.ocx

   If no errors display, the control has been registered.

   Note: For a viewer you will need to copy the **CIMPLogin.ocx** file to the viewer computer and register it there.

Step 2. Select the Type of Login Box you want

1. Open the **Windows Control Panel**.
2. Click **System**.
3. Select the Environment tab.
4. Type **LOGIN_KEYBOARD** in the **Variable** field.
5. Type one of the following values in the **Value** field.

Value	Specifies
QWERTY	A standard qwerty keyboard
ALPHA	An alpha-numeric keyboard (A,B,C,D, 1, 2, 3…)
NONE	No keyboard just like standard login

If the environment variable is not set, Login defaults to a QWERTY style keyboard.

Step 3. Set a CIMPLICITY Global Parameter on the Server

1. Open the project in the CIMPLCITY Workbench.
2. Select **Tools**.
3. Select **Command Prompt**.
4. At the command prompt:

   Type `Cd master .`

   Press Enter.

   Type `Idtpop glb_parms .`

   A message similar to the following "loading rec definitions…" appears.

   a. Press Enter.
   b. Type `Notepad glb_parms.idt`

   Windows Notepad appears displaying the glb_parms.idt file.

5. Use Notepad to add the following new entry to the file.

   ```
   LOGIN_INTERFACE|1|Cimplogin.Login
   ```

6. Exit Notepad.

   The DOS window appears with the cursor at the MASTER prompt.

7. Type `scpop glb_parms`.
8. Press Enter.

   A message similar to the second "loading rec definition…" appears.

Step 4. Set a CIMPLICITY Global Parameter on both the Server and Viewer

1. Click the Windows **Start** button.
2. Select Programs.
3. Select Command Prompt.

   A DOS window opens.

4. Change to the CIMPLICITY Data directory:

   Example

   ```
   Cd\cimplicity\hmi\data
   ```

5. At the command prompt:

   Type `Idtpop glb_parms`.

   Press Enter.

   A message similar to the following "loading rec definitions…" appears.

   a. Type `Notepad glb_parms.idt`.
   b. Press Enter.

   Windows Notepad opens displaying the glb_parms.idt file.

6. Type:

   ```
   LOGIN_INTERFACE|1|Cimplogin.Login
   ```

7. Exit Notepad.

   The DOS window appears with the cursor at the data prompt.

8. Type `scpop glb_parms`.

9. Press Enter.

   A message similar to the second "loading rec definition…" appears.

10. Type `Exit`.

    The DOS window closes.

Step 5. Stop and Restart CIMPLICITY

The new login box should appear instead of the default login box.

Login/Logout API Support

Login/Logout API Support

Several APIs exist that give programmed applications some control over the CIMPLICITY login/logout activities. Versions exist in both C callable and basicscript callable forms.

C callable routines

The C callable routines located in rcm.dll are:

COR_I4 GEFCIMPLICITY_setuserpw *(page 31)* (TCHAR *proj, TCHAR *CIMPuser, TCHAR *pw)

COR_I4 GEFCIMPLICITY_remuserpw *(page 31)* (TCHAR *proj, TCHAR *CIMPuser, TCHAR *pw)

COR_I4 GEFCIMPLICITY_login(TCHAR *proj)

COR_I4 GEFCIMPLICITY_logout(TCHAR *proj)

Be aware that:

- These functions are defined in the header file <inc_path\cor_user_api.h>

- To use these API's link with rcm.lib.
- The return value can be one of the following.
- COR_SUCCESS
- COR_WARNING
- COR_FAILURE

Corresponding basic script extensions

CimSetProjectUserPassword(CIMPuser as string, pw as string, proj as string)

CimClearProjectUserPassword(CIMPuser as string, pw as string, proj as string

CimLogin *(page 32)* (proj as string)

CimLogout *(page 31)* (proj as string)

General Guidelines

The following routines can be called to avoid the CIMPLICITY invoked GUI.

Purpose	To enable an application to have logins reference the set data
Routines	`GEFCIMPLICITY_setuserpw()` or `CimSetProjectUserPassword()`
Description	Use the routine to set the CIMPLICITY user and corresponding password (if there is one) for the project from which resources are about to be requested, Do this before you request any CIMPLICITY resources. • When the application requests a CIMPLICITY resource requiring a login, these set parameters will be used to provide the login action with data. • These parameters remain set until cleared by an application. • Therefore, if the system timeouts should cause a logout, a subsequent request would again reference this data for login. A logout can happen after all requests for resources have been removed. **Note:** If the provided data is not valid, no attempt is made to obtain the login information via GUI or otherwise, and no error is given. These routines can be called multiple times to change the data that should be referenced on future login attempts without having to clear any data already set.
Purpose	To enable an application to stop having logins reference the set data
Routines	`GEFCIMPLICITY_remuserpw()` or `CimClearProjectUserPassword()`
Description	The routine: • Does not force a logout of any currently logged in user • Does disable the automatic attempt to login.
Purpose	To force a logout of the currently logged in user
Routine	`CimLogout`
Description	After `CimLogout` is called, either `CimLogin` must be called or the similar action initiated from the Login Panel, before data can be retrieved again.

Purpose	To cause CIMPLICITY to initiate a login sequence
Routine	`CimLogin`
Description	This most likely will be preceded by a call to `CimSetProjectUserPassword` and `CimLogout`, to force a change in the currently logged in CIMPLICITY user.

Simple Login/Logout Example using Basic Script

```
'specify name of project being addressed
Const proj As String = "onepoint"
'user configured in that project
const cimpuser as string = "testuser"
'password for that configured user
const cimpuserpw as string = "testpassword"
'point that will have a value configured within the previously specified
 project
const pointwithvalue as string = "pointwithvalue"
Sub gp
    trace "point value = " & pointget("\\" & proj & "\" & pointwithvalue)
end sub
sub nouser
    Call CimClearProjectUserPassword(cimpuser, cimpuserpw, proj)
end sub
sub testuser
    Call CimSetProjectUserPassword(cimpuser, cimpuserpw, proj)
end sub
sub main
'Run this program immediately after starting the project, stepping
'through line by line -
    call testuser ' specify configured user for login
    call gp
    call nouser ' clear "auto login" user
    Call CimLogout(proj) 'logout from last user
    Call CimLogin(proj) 'initiate a login sequence - GUI should be invoked
    'need to wait for previous login process to be completed before
 continuing
    'on or fetch of data may fail with error indicating user not logged
 in...
    call gp 'access data
    Call CimLogout(proj)
end sub
```

Chapter 3. Alarm Management API

About Alarm Management API

The Alarm Management API is included in the Integrator's Toolkit for GE Intelligent Platform's CIMPLICITY software. This Application Program Interface (API) provides an interface for application programs to generate CIMPLICITY alarms based on the specific requirements of the application.

The Alarm Management API functions are fully integrated with CIMPLICITY software's Base system functionality to enhance its already powerful monitoring capability in a full range of computer integrated manufacturing environments.

Integration of Alarm Management and Base System

You can visualize the integration of Alarm Management and the Base System as follows:

Your API program integrates with Alarm Management as follows:

Alarm Management API Contents

The following is a list of all files distributed with the Alarm Management API. The files are loaded into the directories indicated. The environment variable **%BSM_ROOT%** is the directory where the CIMPLICITY software was installed.

Include files **in %BSM_ROOT%\api\include\inc_path** are:

alarmapi.h	**cor_thread.h**
am_defs.h	**cor_time.h**
am_errors.h	**ddl.h**
amap_defs.h	**examgr.h**
amaru_err.h	**ipc.hpp**
amdd_cmd.h	**ipcerr.h**
amdd_cont.h	**mf_defs.h**

amdd_defs.h	**mfamupdi.h**
amip.h	**mfpmterm.h**
cor.h	**mfstatus.h**
cor_event.h	**netcom.h**
cor_mutex.h	**rcm.h**
cor_os.h	**rtr_bcst.h**
cor_stat.h	**sc_recs.h**

Source files in **%BSM_ROOT%\api\am_api** are:

amaru_demo.c	makefile amaru_demo_exe.vcxproj

Source files in **%BSM_ROOT%\api\lib** are:

amaru.lib	**fasrtl.lib**
amip.lib	**ipc.lib**
corutil.lib	**cim_mf.lib**
ddl.lib	**cim_sc.lib**
examgr.lib	

Notes on Internationalization for the Alarm Management API

- Work with strings.
- Recommended reading.

Work with strings

This API is written for the international environment. In an international environment, strings in CIMPLICITY software can be multi-byte strings. If you want your code to conform to international standards, it is recommended that you do the following when working with strings:

- Use the **TCHAR** macros found in **TCHAR.H** .
- Declare string buffers as **TCHAR[]** . Declare string pointers as **TCHAR*** or **LPTSTR** .
- Wrap string and character constants with the **_T()** macro.
- Use the **_tcs...()** functions in place of the **str...()** functions. For example, use **_tcslen()** in place of **strlen()** .

- Be careful when incrementing a pointer through a string. Remember that a logical character may occupy one or two **TCHAR** units. So replace code that looks like this:

```
char *cp;

for (cp = string; *cp != '\0'; ++cp)

{

}
```

with code that looks like this:

```
TCHAR const *cp;

for (cp = string; *cp != _T('\0'); cp = _tcsinc(cp))

{

}
```

- Avoid using a variable to hold the value of a logical character. Instead, use a pointer to a character in the string. In particular, avoid the **_tcsnextc()** macro, because the value it returns appears to be incompatible with some of the C runtime library functions.
- Use the functions **_tccpy()** and **_tccmp()** and string pointers instead of the = and == operators on characters.
- Use **GetStringTypeEx()** instead of the character classification macros such as **_istalpha()** .
- Use **CharUpper()** and **CharLower()** instead of **_toupper()** and **_tolower().**

Recommended Reading

Microsoft has several good papers on writing international code on its Developer Network DVD and its web site To find documentation on the web site, go to http://msdn.microsoft.com/default.asp and search for MBCS

For documentation on globalization, go to http://www.microsoft.com/globaldev/

The following book is also available:

- Schmitt, David A. International Programming for Microsoft® Windows®, ISBN 1-57231-956-9.

For more information about this book, go to http://mspress.microsoft.com/books/2323.htm.

Alarm Management API Overview

Alarm Management API Overview

CIMPLICITY software's Alarm Management module is responsible for maintaining the status of outstanding alarms, or predetermined conditions of interest detected by an application process. Alarm Management informs users of current alarm occurrences and sends information on alarm occurrences to interested processes. Alarm Management provides a set of services to generate new alarms and update the status of existing alarms. These services allow an application to interact with Alarm Management capability without knowing the message structure and message passing aspects of interfacing to an Alarm Management Resident Process.

The Alarm Management module consists of an Alarm Management Resident Process (AMRP), a configured number of Alarm Management Allocated Processes (AMAP), and a set of utilities linked into application programs. These utilities are referred to as the Alarm Management Application Resident Utilities (AMARU).

Operation Overview

- Overview
- Initialize communications with AMRP.
- Generate alarm update information.
- Send alarm information.
- Terminate communications with AMRP.

Overview

Alarm Management maintains an in-memory database containing information on the current user population and the current set of outstanding alarms. This database is updated as information on the current user population is received from User Registration, as new alarms are generated, and as the status of existing alarms is updated.

The status of alarms may be updated interactively by a user through the AMAP or by an application program sending a message to the AMRP. When the status of alarms changes, new alarm counts and dates are determined for each user. If a user's information changes, the new data is sent to the user's Alarm Viewer.

Any process that wants to communicate with AMRP to update alarm statuses must do the following:

- Initialize communications with AMRP
- Generate alarm update information

- Send alarm information
- Terminate communications with AMRP

Initialize Communications with AMRP

The **amaru_init** procedure initializes the utilities for communication with an Alarm Manager in the current Interprocess Communications (IPC) network. An application must invoke this procedure in order to communicate with Alarm Management. Typically, initialization is done once when the application process starts up.

Generate Alarm Update Information

Application programs require the following information in order to generate or update alarms:

ALARM_ID	The identifier of the alarm to be generated or updated.
FR_ID	The identifier of the factory resource the alarm is being generated or updated for.
REF_ID	A third identifier (the Reference ID) designating the instance of the ALARM_ID/FR_ID being generated or updated.
Alarm Parameters	The value of the run time parameters to be loaded into the alarm message.
Alarm Management ID	The identifier of the AMRP servicing the specified factory resource. This information is carried in the **alarm_mgr** field in the **fr.dat** configuration file and can be checked for the factory resource for which the alarm is being generated or updated.

Together, the ALARM_ID, FR_ID, and REF_ID uniquely identify an alarm occurrence. All operations on the same ALARM_ID, FR_ID, REF_ID combination occur on the same alarm.

Two steps are necessary in order to generate or update alarms. First, the request is added to an IPC buffer. Then the request is sent to the appropriate AMRP. The appropriate AMRP can be determined once the factory resource is known.

The **amaru_add_gen** procedure adds alarm generation information to the specified IPC buffer. The caller is informed if the buffer is full.

The **amaru_add_update** procedure adds alarm update information to the specified IPC buffer. The caller is informed if the buffer is full.

The AMRP accepts messages with multiple generation and update requests. Thus, if an application program has multiple alarms to generate or update, these requests can be packed into a single IPC message and sent as a whole. The Alarm Management routines support and favor this mode of operation as it reduces network message traffic.

Send Alarm Information

The **amaru_send_msg** procedure sends an IPC message to the specified AMRP. The procedure handles the redundancy aspects of the AMRP. If the specified AMRP is not available, an appropriate status is returned to the caller.

Alarm Management receives static information on the legal set of alarms from system configuration files at initialization. This configuration information specifies the user roles configured to see each alarm. Alarms are generated with respect to factory resources. When an alarm is generated, all users who have a view of the specified factory resource, and whose Role matches one of the roles the alarm is configured to be routed to, are informed of the new alarm occurrence.

Terminate Communications with AMRP

When an application no longer needs to communicate with Alarm Management, the **amaru_terminate** procedure is called to end communication with the AMRP.

Alarm Management API Features

The Alarm Management API lets you construct application programs that generate, reset, acknowledge, and delete alarms. The application programs can use the standard C language functions provided by the API in order to communicate with the Alarm Management module of CIMPLICITY software's Base System.

This API gives you transparent access to the CIMPLICITY alarm database in order generate new alarms or modify the status of existing alarms, regardless of the physical location of the application program.

A C language subroutine interface is available for application programs modifying alarm status. The C language interface is used on nodes where a CIMPLICITY environment is currently running.

Alarm Management API Getting Started

Alarm Management API Getting Started

The CIMPLICITY Alarm Management API lets application programs access the functions of CIMPLICITY software's Alarm Management application module. Using the API requires that you do the following:.

- Understand the subroutine interfaces and communications services provided by CIMPLICITY's Alarm Management capability. These functions are documented in Chapter 4 of this manual.

- Understand the configuration requirements and file formats for Alarm Management. The Alarm Management configuration data is described in the CIMPLICITY Base System User Manual (GFK-1180). The file formats are provided in Chapter 5 of this manual for your reference.
- Code appropriate applications programs.
- Compile and link the programs as explained in this section.

Alarm Management API Sample Program

Overview

A sample Microsoft Visual C++ project, named `amaru_demo_exe.vcxproj`, is provided to build the sample program. Use this project as a basis for constructing projects for your own applications.

Depending on how you installed Visual C++, the INCLUDE, LIB, and PATH environment variables may not be automatically set when you install MSDEV. If they are not set, you will have to set them manually or run the following to set them before building any user programs.

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft Visual
 Studio\Installer\vswhere.exe" -property installationPath`) do set VSPATH=
%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

When you run the demo program, it will generate an alarm, then reset it.

Build the Demo (Sample) Program

1. From the CIMPLICITY Workbench for your project, select **Command Prompt** from the **Tools** menu. This will ensure that your environment variables (in particular **%BSM_ROOT%** and **%SITE_ROOT%**) are set correctly.
2. In the Command Prompt window, issue the following commands (where < drive > is the disk where your CIMPLICITY software is installed):

   ```
   < drive >:
   cd %BSM_ROOT%\api
   ```

3. (If the environment variables are not set automatically) issue the following command to set them:

   ```
   for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
    Visual Studio\Installer\vswhere.exe" -property installationPath`) do
    set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
   ```

4. Launch Visual Studio:

   ```
   devenv CimplicityAPI.sln
   ```

5. `devenv CimplicityAPI.sln`
6. Open the Solution Explorer.
7. Right click amaru_demo_exe.

8. Select **Build** on the Popup menu.

Run the Demo (Sample) Program

The API process name must be stored in the PRCNAM environment variable for the program to run. The name is an arbitrary character string of up to 10 characters. To create PRCNAM, enter the following command in the Command Prompt window:

```
set PRCNAM=<name>
```

where < name > is the API process name.

To run the sample program, enter the following command in the Command Prompt window:

```
amaru_demo
```

⚠ **Important:** You must have a project running locally or the sample program will fail to run successfully.

Example: Send When Buffer Filled

This example shows coding necessary to add alarm generation/update requests to the IPC message buffer until the buffer is full. When the buffer is full, a message is sent.

```
        ... get parameters ... (load msg_fields)
     while (not_finished)
     {
        /* first call */
        amaru_add_gen (msg_wbody, MAX_MSG_SIZE, TRUE, ... );
        /* repeat */
        repeat
        {
           ... get parameters ...
        }
        until ( amaru_add_gen (msg_wbody,     , FALSE, ...) !=
                                               COR_SUCCESS);
        if (ret_stat.err_code == MF_INSUF_SPACE)
        {
           /* send message */
           amaru_send_msg (port_id, ... );
           /* unload returned message */
           ...
        }
     } /* end while */
```

Note: The alarm generation/update requests are added to the IPC message buffer. The most recent alarm segment is not added to the message buffer if there is not sufficient space for it. Therefore it must be kept and loaded into the next datagram as the first segment.

Example: Send an Alarm With Multiple Parameters

The following example shows how to send an alarm with multiple parameters:

```
    #define NUM_PARAMETERS
    AM_MSG_FIELD    msg_field[NUM_PARAMETERS];
    /* xxx is the union member corresponding to the field type */
    msg_field[0].ftype =     ;
    msg_field[0].field, xxx =     ;
```

```
    msg_field[NUM_PARAMETERS-1].field, xxx = ...;
    amaru_ADD_GEN (msg_wbody, MAX_MSG_SIZE,
                          ,
                   alarm_id,
                   fr_id,
                   user_or_serv_id,
                   resp_type,
                   ref_id,
                   key,
                   msg_field,
                   NUM_PARAMETERS,
                   &ret_stat) FALSE;   /* reset follows */
```

In this example, **msg_field** is a pointer to the beginning of an array of parameters. The types may be different, but they all belong to the same alarm.

Example: External Alarm Timestamps

By default, newly generated CIMPLICITY alarms are assigned a timestamp indicating the alarm generation time and duration. The Alarm Management API lets you provide your own timestamp, so that external alarms can be synchronized to a more accurate clock. The following example shows how to generate an alarm with an external timestamp:

```
COR_STAMP stamp;
int yyyy, mm, dd, hh, min, sec, tt100Nano;
;
   get parameters     (load msg_fields)
/* Setup time for the alarm */
yyyy = 1995;
mm = 01;     /* month = January */
```

```
dd = 10;     /* day of the month */
hh = 13;     /* hours - 24 hour clock */
min = 30;    /* minutes after 1pm */
sec = 12;    /* seconds after 1:30 */
tt100Nano = 5000000;     /* subseconds in 100 Nano second units*/
int ret;     /* return code */
/* Convert time into a CIMPLICITY timestamp */
cor_stamp_calcHR( &stamp, yyyy, mm, dd, yy, min,sec, tt100Nano);
/* set up to generate the alarm */
/* the alarm is fixed as shown */
amaru_add_gen_stamp(
                    alarm_write_body,                /* data pointer of write
 buffer */
                    alarm_write_len - IPC_HEAD_LEN, /* data length avail */
                    TRUE,                            /* TRUE if first message
 in buffer */
                                                     /* FALSE if not first
 message */
                    "$RTR_LINK_DOWN",                /* ID of alarm */
                    "$SYSTEM",                        /* ID of resource */
                    object_name,                     /* identifier of who
 sent the alarm */
                    NULL,                            /* reference ID or NULL
 goes here */
                    AM_CAPTURED_RESP,                /* AMRP sends response
 immediately */
                    0,                                /* key */
                    msg_field,                       /* the message struct
 array */
                    i,                                /* the number of fields
 in message */
                    FALSE,                            /* no reset follows */
     stamp,
   &ret_stat );
if (ret_stat.status != COR_SUCCESS)
{
   printf("%s\n",ret_stat.err_msg);
   goto end_of_program_am;
}
/*    send message to AMRP as before    */
```

There is also a corresponding function **amaru_add_update_stamp()** in the API which can be used to specify an external reset time for an outstanding alarm.

Field Definitions: Alarm Management API Application

Field Definitions: Alarm Management API Application

- Include files.
- General subroutines for the Alarm Management API.
- Application subroutines for the Alarm Management API.

Include Files

The following header files contain definitions used by Alarm Management procedures and therefore must be included in an application program that interfaces with Alarm Management.

be included in an application program that interfaces with Alarm Management.

```
#include <inc_path/cor.h>
#include <inc_path/cor_stat.h>
#include <inc_path/sc_recs.h>
#include <inc_path/netcom.h>
#include <inc_path/am_errors.h Error Codes Returned by AMRP >
```

Type definitions supporting the use of the AMARU procedures can be found in:

```
#include <inc_path/am_defs.h am_defs.h >
#include <inc_path/amaru_err.h Error Codes Returned by AMARU >
```

General Subroutines for the Alarm Management API

General Subroutines for Field Definitions: Alarm Management API

The general subroutines are used to get the current process name, suspend the process temporarily, deactivate the IPC port, and register with IPC.

- coprcnam
- Get current process name
- cor_logstatus
- Write error information to the CIMPLICITY status log.
- cor_long_long_from_stamp_utc
- Convert the `COR_STAMP` to a long long.
- cor_stamp
- Get t"Field Definitions:e current time of day as a timestamp
- cor_stamp_calc_utcHR
- Generate a UTC timestamp for a particular date and time
- cor_stamp_calcHR
- Generate a timestamp for a particular date and time.

- cor_stamp_cmp
- Compare the order in which one `COR_STAMP` occurs relative to another.
- cor_stamp_convert_to_ascii
- Convert the `COR_STAMP` to ASCII.
- cor_stamp_convert_to_ascii_utc
- Convert the UTC `COR_STAMP` to ASCII.
- cor_stamp_get_components_utcHR
- Convert a UTC timestamp into its various components.
- cor_stamp_get_componentsHR
- Convert a timestamp into its various components.
- cor_stamp_get_diff
- Calculate the difference between two `COR_STAMP`s.
- cor_stamp_get_diffHR
- Calculate the difference between two `COR_STAMP`s.
- cor_stamp_getfracHR
- Get the fractional part of a `COR_STAMP`.
- cor_stamp_is_set
- Return whether or not the `COR_STAMP` is set.
- cor_stamp_is_valid
- Return whether or not the `COR_STAMP` is valid.
- cor_stamp_reset
- Reset the `COR_STAMP` to the beginning of epoch time.
- cor_stamp_set_fracHR
- Set the fractional part of a `COR_STAMP`.
- ipc_deactivate
- Deactivate a port
- ipc_register
- Register wit"Field Definitions: IPC

coprcnam

Returns the current process name. The process name will be used by **ipc_register** along with the node name to define the physical address of this process.

The process name is extracted from the PRCNAM environment variable, and should be unique for each running application.

Syntax

```
void coprcnam (prcnam)
char *prcnam;
```

Input Arguments

None.

Output Arguments

prcnam	The name of the current process. Maximum length is 13 characters.

Return Value

None.

cor_logstatus

Write error information to the CIMPLICITY status log.

Syntax

```
void cor_logstatus( const TCHAR *proc,
    COR_BOOLEAN severe,
    COR_STATUS *status )
```

Input Arguments

`proc`	The name of the function where the error is detected.
`severe`	Is situation severe error (FALSE, TRUE, FATAL)
`status`	Pointer to the status block containing error info.

Output Arguments

None

Return Value

None

The value is sent to the status log, which is viewed through the Log Viewer application.

cor_long_long_from_stamp_utc

Convert the `COR_STAMP` to a long long.

Syntax

```
long long cor_long_long_from_stamp_utc(const COR_STAMP* time_stamp);
```

Input Arguments

`time_stamp`	The timestamp that will be converted to long long.

Output Arguments

NONE	

Return Value

long long

cor_stamp

Gets the current time of day as a CIMPLICITY timestamp.

Syntax

```
void cor_stamp ( stamp )
COR_STAMP *stamp;
```

Input Arguments

None.

Output Arguments

`stamp`	A pointer to caller-supplied storage for the result.

Return Value

None

cor_stamp_calc_utcHR

Generates a UTC timestamp for a particular date and time. Invalid input parameters will not cause this function to fail.

Syntax

```
int cor_stamp_calcHR( stamp, yyyy, mm, dd, hh, min, sec, tt100Nano )
COR_STAMP *stamp;
int yyyy;
int mm;
int dd;
int hh;
int min;
int sec;
int tt100Nano;
```

Input Arguments

`yyyy`	The year. Must be in the range 1970.9999.
`mm`	The month. Must be in the range 1.12.
`dd`	The day. Must be in the range 1.n, where n is the number of days in the month.
`hh`	The hour, specified as a 24-hour clock.
`min`	Minutes past the hour, in the range 0..59
`sec`	Seconds, in the range 0..59
`tt100Nano`	Fractional seconds, specified in 100 Nano seconds.

Output Arguments

`stamp`	A pointer to caller-supplied storage for the result in UTC time.

Return Value

int

cor_stamp_calcHR

Generate a timestamp for a particular date and time. Invalid input parameters will not cause this function to fail.

Syntax

```
int cor_stamp_calcHR( stamp, yyyy, mm, dd, hh, min, sec, tt100Nano )
COR_STAMP *stamp;
int yyyy;
int mm;
int dd;
int hh;
```

```
int min;
int sec;
int tt100Nano;
```

Input Arguments

`yyyy`	The year. Must be in the range 1970.9999.
`mm`	The month. Must be in the range 1.12.
`dd`	The day. Must be in the range 1.n, where n is the number of days in the month.
`hh`	The hour, specified as a 24-hour clock.
`min`	Minutes past the hour, in the range 0..59
`sec`	Seconds, in the range 0..59
`tt100Nano`	Fractional seconds, specified in 100 Nano seconds.

Output Arguments

`stamp`	A pointer to caller-supplied storage for the result in time.

Return Value

int

cor_stamp_cmp

Compare the order in which one `COR_STAMP` occurs relative to another (which stamp occurs first; which occurs second).

Syntax

```
COR_I4 cor_stamp_cmp(const COR_STAMP *time1, const COR_STAMP *time2);
```

Input Arguments

`stamp1`	First timestamp in the comparison.
`stamp2`	Second timestamp in the comparison.

Output Arguments

NONE	

Return Value

COR_I4

cor_stamp_convert_to_ascii

Converts the `COR_STAMP` to ASCII.

Syntax

```
void cor_stamp_convert_to_ascii( const COR_STAMP *ttime, TCHAR *tasc_time,
 int time_format);
```

Input Arguments

`ttime`	COR_STAMP that will be converted to ASCII.
`time_format`	The format value can be one of many constants defined in the: ..\<CIMPLICITY installation>\api\include \inc_path\cor.h file.

Output Arguments

`tasc_time`	Converted ASCII format.

Return Value

NONE

cor_stamp_convert_to_ascii_utc

Converts the UTC `COR_STAMP` to ASCII.

Syntax

```
void cor_stamp_convert_to_ascii_utc( const COR_STAMP *ttime, TCHAR
 *tasc_time, int time_format);
```

Input Arguments

`ttime`	UTC COR_STAMP that will be converted to ASCII.
`time_format`	The format value can be one of many constants defined in the: ..\<CIMPLICITY installation>\api\include \inc_path\cor.h file.

Output Arguments

`tasc_time`	Converted ASCII format.

Return Value

NONE

cor_stamp_get_components_utcHR

Convert a UTC timestamp into its various components.

Syntax

```
void cor_stamp_get_components_utcHR(stamp, yyyy, mm, dd, hh,
                                    min, sec, tt100Nano )
COR_STAMP *stamp;
int *yyyy;
int *mm;
int *dd;
int *hh;
int *min;
int *sec;
int *tt100Nano;
```

Input Arguments

`stamp`	A valid CIMPLICITY UTC timestamp.

Output Arguments

`yyyy`	The year, in the range 1970..9999.
`mm`	The month, in the range 1..12.
`dd`	The day of the month.
`hh`	The hour, specified as a 24-hour clock.
`min`	Minutes past the hour, in the range 0..59
`sec`	Seconds, in the range 0..59
`tt100Nano`	Fractional seconds, specified in 100 Nano seconds.

Return Value

NONE

cor_stamp_get_componentsHR

Convert a timestamp into its various components.

Syntax

```
void cor_stamp_get_componentsHR(stamp, yyyy, mm, dd, hh,
                                min, sec, tt100Nano )
COR_STAMP *stamp;
int *yyyy;
int *mm;
int *dd;
int *hh;
int *min;
int *sec;
int *tt100Nano;
```

Input Arguments

`stamp`	A valid CIMPLICITY timestamp.

Output Arguments

`yyyy`	The year, in the range 1970..9999.
`mm`	The month, in the range 1..12.
`dd`	The day of the month.
`hh`	The hour, specified as a 24-hour clock.
`min`	Minutes past the hour, in the range 0..59
`sec`	Seconds, in the range 0..59
`tt100Nano`	Fractional seconds, specified in 100 Nano seconds.

Return Value

NONE

cor_stamp_get_diff

Calculate the difference between two `COR_STAMP`s.

Syntax

```
COR_I4 cor_stamp_get_diff( const COR_STAMP *, const COR_STAMP * );
```

Input Arguments

`stamp1`	The first stamp in the calculation.
`stamp2`	The second stamp in the calculation.

Output Arguments

NONE	

Return Value

COR_I4

cor_stamp_get_diffHR

Calculates the difference between two `COR_STAMP`s.

Syntax

```
long long cor_stamp_get_diffHR( const COR_STAMP *, const COR_STAMP * );
```

Input Arguments

`stamp1`	The first stamp in the calculation.
`stamp2`	The second stamp in the calculation.

Output Arguments

NONE	

Return Value

long long

cor_stamp_getfracHR

Get the fractional part of a `COR_STAMP`.

Syntax

```
int cor_stamp_getfracHR(const COR_STAMP *stamp);
```

Input Arguments

`stamp`	cor_stamp that holds the fractional part to be retrieved.

Output Arguments

NONE	

Return Value

int

cor_stamp_is_set

Return whether or not the `COR_STAMP` is set.

Syntax

```
int cor_stamp_is_set( const COR_STAMP * );
```

Input Arguments

`stamp`	`COR_STAMP` that is being checked to determine if it is set.

Output Arguments

NONE	

Return Value

int

cor_stamp_is_valid

Return whether or not the `COR_STAMP` is valid.

Syntax

```
int cor_stamp_is_valid( const COR_STAMP * );
```

Input Arguments

`stamp`	`COR_STAMP` that is being checked for validity.

Output Arguments

NONE	

Return Value

int

cor_stamp_reset

Reset the `COR_STAMP` to the beginning of epoch time.

Syntax

```
void cor_stamp_reset(COR_STAMP *stamp);
```

Input Arguments

`stamp`	`COR_STAMP` that will be reset.

Output Arguments

NONE	

Return Value

VOID

cor_stamp_set_fracHR

Set the fractional part of a `COR_STAMP`.

Syntax

```
void cor_stamp_set_fracHR(COR_STAMP *stamp, int frac);
```

Input Arguments

`stamp`	`COR_STAMP` that will hold the fractional part to be set.
`frac`	The fractional part that will be set for the cor_stamp.

Output Arguments

NONE	

Return Value

VOID

ipc_deactivate

Terminate all activities associated with the IPC. If any RR messages are outstanding for the process that executes **ipc_deactivate** , a message is transmitted on its behalf. No further datagram messages can be sent to or from a process that executes this service.

Syntax

```
ipc_deactivate( ret_stat, port_index);
COR_STATUS *ret_stat;
int port_index;
```

Input Arguments:

`port_index`	The port index returned by `ipc_register`.

Output Arguments

`ret_stat`	Status structure

Return Value

Either COR_SUCCESS or COR_FAILURE. If COR_FAILURE is returned, an error code is reported in **ret_stat.err_code** .

ipc_register

Initialize IPC functions for datagram and logical link communications and register with the IPC router process. `ipc_register` must be called prior to using any other communications functions. Following successful execution of this function, an application can start sending and receiving datagram messages or establish logical link communications.

Syntax

```
int ipc_register (ret_stat, port_index, object_name, maxlnk,
                  bufsiz)
COR_STATUS *ret_stat;
int *port_index;
char *object_name;
int maxlnk;
int bufsiz;
```

Input Arguments

`object_name`	The name under which the process registers. The object_name in combination with the node name defines the physical address of the process. All other processes address the process with this name. Maximum length is 10 characters.
`maxlnk`	The maximum number of logical links this process can request. Each datagram port and each logical link connection counts as one link.
`bufsiz`	The maximum message size used by this process. `ipc_dg_alloc` may be used to determine this size. The maximum is MAXMSGSIZ, which is defined in `netcom.h`.

Output Arguments

`port_index`	Identifier for port index to be used in datagram functions.
`ret_stat`	Pointer to status structure. When the function is not successful, `ret_stat.err_code` contains one of the following values:

	IPC_ERR_BUFTOOBIG	103	requested buffer too big
	IPC_ERR_MAXLNKVIO	108	max link violation
	IPC_ERR_MBXASGN	109	mailbox assignment failed
	IPC_ERR_RTRLNKFAI	110	router link attempt failed
	IPC_ERR_INTERNDAT	111	internal data failure

Return Value

Either COR_SUCCESS, COR_WARNING, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in `ret_stat.err_msg` and `ret_stat.err_code`.

Application Subroutines for Alarm Management API

Application Subroutines for Field Definitions: Alarm Management API

The application subroutines are used to communicate with the Alarm Manager and send and receive alarm messages.

- amaru_init
- Initialize the interface with Alarm Manager
- amaru_add_gen
- Add alarm generation information to datagram buffer
- amaru_add_gen_stamp
- Add alarm generation information and timestamp to datagram buffer
- amaru_add_update
- Add alarm update information to IPC buffer
- amaru_add_update_ca
- Add alarm update information to the current IPC buffer with Change approval Information.
- amaru_add_update_stamp
- Add alarm update information and timestamp to datagram buffer
- amaru_send_msg
- Send datagram buffer with alarm messages to Alarm Manager
- amaru_alloc_buffer
- Create datagram buffer for alarm messages
- amaru_num_messages
- Find the number of messages in a datagram response buffer
- amaru_get_resp
- Retrieve the nth message from the datagram response buffer
- amaru_free_buffer
- Deallocate the datagram buffer
- amaru_terminate
- Terminate the interface with Alarm Manager

amaru_init

This initialization routine should be called at process start-up. It gathers the information necessary to communicate with the Alarm Manager in the current IPC network. This information is used by **amaru_send_msg** in order to determine the physical address of the destination AMRP.

Syntax

```
int amaru_init (ret_stat);
COR_STATUS *ret_stat;
```

Input Arguments

None.

Output Arguments

ret_stat	Pointer to status structure.

Return Value

Either COR_SUCCESS, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in **ret_stat.err_msg** and **ret_stat.err_code** .

There are no error messages generated by **amaru_init** itself. If the low-level routine **sc_open** or **sc_close** fails the error status is passed unchanged to the calling program. Typically, this happens if the system configuration data is not accessible.

amaru_add_gen

Call this subroutine to add alarm generation information to the current IPC buffer. It is the responsibility of the application program to allocate space for the message buffer. The routine may be called repeatedly to load multiple alarm generation segments into the message buffer.

Syntax

```
int amaru_add_gen (bodyptr, bodylen, first_seg,
                   alarm_id, fr_id, user_or_serv_id,
                   ref_id, resp_type, key, msg_field,
                   num_fields, reset_follows, ret_stat);
char *bodyptr;
int bodylen;
COR_BOOLEAN first_seg;
char alarm_id[LONG_NAME_LEN+1];
char fr_id[FR_ID_LEN+1];
char user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1];
char ref_id[AM_REF_ID_LEN+1];
```

```
AM_RESP_TYPE resp_type;
AM_RESP_KEY key;
AM_MSG_FIELD msg_field[];
int num_fields;
COR_BOOLEAN reset_follows;
COR_STATUS *ret_stat;
```

Input arguments

bodyptr	Pointer to the beginning of the message body of the IPC buffer.
bodylen	Maximum length of the message body.
first_seg	Boolean value specifying whether the current generation request should be the first request in the message. TRUE implies first segment.
alarm_id	Identifier of the alarm to be generated.
fr_id	Identifier of the factory resource for which the alarm is being generated.
user_or_serv_id	Used for labeling logged alarms. Usually this is the **service_id** of the sending process. The AMAP sends the **user_id** of the connected terminal.
ref_id	Used to specify unique alarms when multiple alarm definitions are generated for the same Factory Resources. Alarm uniqueness is defined by the combination of **alarm_id, fr_id** , and **ref_id** .
	Note: The **ref_id** is not displayed directly on the Alarm Manager User Interface. The **ref_id** , when used, can be duplicated in a message field for display.
resp_type	Used to select the type of response desired from the AMRP. The application program has three choices:
	AM_CAPTURED_RESP - AMRP responds once the message has been captured (sent to a standby process or journalled).
	AM_FULL_RESP - AMRP responds once the message has been fully processed. A status segment is returned for each request in the original message.
	AM_NO_RESP - No response from AMRP to indicate that an alarm message has been received.
	Note: Only the **resp_type** in the first generation message of each segment is used. Therefore, it is not possible to intermix the type of responses desired within a single IPC message..
key	The key is useful when the **resp_type** is set to AM_FULL_RESP. The key allows the application program to match the status segments returned with the alarm generation/update requests. The key is specified by the application program and is returned "as is" by the AMRP.
msg_field	Pointer to the beginning of an array containing variable parameter information for the particular alarm. Each element of the array contains a type specifier and the actual field value. The structure `AM_MSG_FIELD` and the valid field types can be found in the include file **inc_path/** **am_defs.h** ***(page 95)*** .
num_fields	The number of variable fields in the **msg_field** array. There is a maximum of AM_MAX_FIELDS. Each element consists of a type specifier and the actual information.

reset_follows	Set to TRUE to indicate that on acknowledgment, the application will update the alarm message, clear the alarm, and retain the acknowledgment. Otherwise, set to FALSE.

Output Arguments

ret_stat	Pointer to status structure.

Return Value

Either COR_SUCCESS, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in **ret_stat.err_msg** and **ret_stat.err_code** .

amaru_add_gen does not directly generate error codes. It passes the status set by the message formatting routines back to the calling program. The error codes are defined in the include file **inc_path/am_errors.h** .

When an error occurs, the value of **ret_stat.status** is COR_FAILURE. **ret_stat.err_source** and **ret_stat.err_code** can be used to determine the type of error.

If the source is COR_MF_ERR and the code is MF_INSUF_SPACE, the alarm generation information is not added as the message is full. The application program should call **amaru_send_msg** , reset **first_seg** to TRUE, and then add the information to the now empty buffer.

amaru_add_gen_stamp

Call this subroutine to add alarm generation information to the current IPC buffer with an external alarm timestamp.

Syntax

```
int amaru_add_gen_stamp (bodyptr, bodylen, first_seg,
                         alarm_id, fr_id, user_or_serv_id,
                         ref_id, resp_type, key, msg_field,
                         num_fields, reset_follows, stamp,
                         ret_stat);
char *bodyptr;
int bodylen;
COR_BOOLEAN first_seg;
char alarm_id[LONG_NAME_LEN+1];
char fr_id[FR_ID_LEN+1];
char user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1];
char ref_id[AM_REF_ID_LEN+1];
AM_RESP_TYPE resp_type;
AM_RESP_KEY key;
AM_MSG_FIELD msg_field[];
int num_fields;
COR BOOLEAN reset_follows;
```

```
COR_STAMP stamp;
COR_STATUS *ret_stat;
```

Input Arguments

bodyptr	Pointer to the beginning of the message body of the IPC buffer.
bodylen	Maximum length of the message body.
first_seg	Boolean value specifying whether the current generation request should be the first request in the message. TRUE implies first segment.
alarm_id	Identifier of the alarm to be generated.
fr_id	Identifier of the factory resource for which the alarm is being generated.
user_or_serv_id	Used for labeling logged alarms. Usually this is the **service_id** of the sending process. The AMAP sends the **user_id** of the connected terminal.
ref_id	Used to specify unique alarms when multiple alarm definitions are generated for the same Factory Resources. Alarm uniqueness is defined by the combination of **alarm_id** , **fr_id** , and **ref_id.**
	Note: The **ref_id** is not displayed directly on the Alarm Manager User Interface. The **ref_id** , when used, can be duplicated in a message field for display.
resp_type	Used to select the type of response desired from the AMRP. The application program has three choices:
	AM_CAPTURED_RESP - AMRP responds once the message has been captured (sent to a standby process or journalled).
	AM_FULL_RESP - AMRP responds once the message has been fully processed. A status segment is returned for each request in the original message.
	AM_NO_RESP - No response from AMRP to indicate that an alarm message has been received.
	Note: Only the **resp_type** in the first generation message of each segment is used. Therefore, it is not possible to intermix the type of responses desired within a single IPC message..
key	The key is useful when the **resp_type** is set to AM_FULL_RESP. The key allows the application program to match the status segments returned with the alarm generation/update requests. The key is specified by the application program and is returned "as is" by the AMRP.
msg_field	Pointer to the beginning of an array containing variable parameter information for the particular alarm. Each element of the array contains a type specifier and the actual field value. The structure `AM_MSG_FIELD` and the valid field types can be found in include file **inc_path/ am_defs.h *(page 95)*** .
num_fields	The number of variable fields in the **msg_field** array. There is a maximum of AM_MAX_FIELDS. Each element consists of a type specifier and the actual information.
reset_follows	Set to TRUE to indicate that on acknowledgment, the application will update the alarm message, clear the alarm, and retain the acknowledgment. Otherwise, set to FALSE.
stamp	Specifies the time at which the alarm was generated. Until the alarm is cleared, its duration will be difference between the current time-of-day and this value.

Output Arguments

ret_stat	Pointer to status structure.

Return Value

Either COR_SUCCESS, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in **ret_stat** . Other than the timestamp parameter, this function should in all ways conform to the behavior of **amaru_add_gen()** .

amaru_add_update

Call this subroutine to add alarm update information to the current IPC buffer. It is the responsibility of the application program to allocate space for the message buffer. The routine may be called repeatedly to load multiple alarm update segments into the message buffer.

The purpose of the update function is to change the state of an outstanding alarm. The states can be changed as follows:

AM_ACKNOWLEDGED	The alarm has been acknowledged by an operator.
AM_CLEARED	The condition that generated the alarm is now gone.
AM_DELETED	The alarm should be deleted

These constants can be found in the **inc_path/am_defs.h** include file .

Note: The contents of the alarm message field of an alarm cannot be updated. If an application needs to display a different alarm message when a particular alarm has been cleared, the application program must generate a new alarm occurrence with the new message and then update the new alarm occurrence to the desired state.

Syntax

```
int amaru_add_update (bodyptr, bodylen, first_seg,
                      alarm_id, fr_id, user_or_serv_id,
                      ref_id, action, seq_num, resp_type,
                      key, ret_stat);
char *bodyptr;
int bodylen;
COR_BOOLEAN first_seg;
char alarm_id[LONG_NAME_LEN+1];
char fr_id[FR_ID_LEN+1];
char user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1];
char ref_id[AM_REF_ID_LEN+1];
AM_STATE_TYPE action;
int seq_num;
AM_RESP_TYPE resp_type;
```

```
AM_RESP_KEY key;
COR_STATUS *ret_stat;
```

Input Arguments

bodyptr	Pointer to the beginning of the message body of the IPC buffer.
Bodylen	Maximum length of the message body.
first_seg	Boolean value specifying whether the current generation request should be the first request in the message. TRUE implies first segment.
alarm_id	Identifier of the alarm to be generated.
fr_id	Identifier of the factory resource for which the alarm is being generated.
user_or_serv_id	Used for labeling logged alarms. Usually this is the **service_id** of the sending process. The AMAP sends the **user_id** of the connected terminal.
ref_id	Used to specify unique alarms when multiple alarm definitions are generated for the same Factory Resources. Alarm uniqueness is defined by the combination of **alarm_id** , **fr_id** , and **ref_id** .
	Note: The **ref_id** is not displayed directly on the Alarm Manager User Interface. The **ref_id** , when used, can be duplicated in a message field for display.
action	Specifies the update action to take.
seq_num	Only used by AMAP. Application programs should pass zero (0).
resp_type	Used to select the type of response desired from the AMRP. The application program has three choices:
	AM_CAPTURED_RESP - AMRP responds once the message has been captured (sent to a standby process or journalled).
	AM_FULL_RESP - AMRP responds once the message has been fully processed. A status segment is returned for each request in the original message.
	AM_NO_RESP - No response from AMRP to indicate that an alarm message has been received.
	Note: Only the **resp_type** in the first generation message of each segment is used. Therefore, it is not possible to intermix the type of responses desired within a single IPC message.
key	The key is useful when the **resp_type** is set to AM_FULL_RESP. The key allows the application program to match the status segments returned with the alarm generation/update requests. The key is specified by the application program and is returned "as is" by the AMRP.

Output Arguments

ret_stat	Pointer to status structure.

Return Value

Either COR_SUCCESS, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in **ret_stat.err_msg** and **ret_stat.err_code** .

amaru_add_update does not directly generate error codes. It passes the status set by the MF-routines back to the calling program. The error codes are defined in the **inc_path/mf_defs.h** include file and are shown in Chapter 7.

When an error occurs, the value of **ret_stat.status** is COR_FAILURE. **ret_stat.err_source** and **ret_stat.err_code** can be used to determine the type of error.

If the source is COR_MF_ERR and the code is MF_INSUF_SPACE, the alarm generation information is not added as the message if full. The application program should call **amaru_send_msg** , reset **first_seg** to TRUE, and then add the information to the now empty buffer.

amaru_add_update_ca

Call this subroutine to add alarm update information to the current IPC buffer with Change approval Information.

Syntax

```
Int  amaru_add_update_ca (bodyptr, bodylen, first_seg,
 alarm_id, fr_id, user_or_serv_id,
  ref_id, action, seq_num, resp_type,
key, changeapproval_obj, ret_stat);
char *bodyptr;
int bodylen;
COR_BOOLEAN first_seg;
char alarm_id[LONG_NAME_LEN+1];
char fr_id[FR_ID_LEN+1];
char user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1];
char ref_id[AM_REF_ID_LEN+1];
AM_STATE_TYPE action;
int seq_num;
AM_RESP_TYPE resp_type;
AM_RESP_KEY key;
ChangeapprovalInfo *changeapproval_obj,
COR_STATUS *ret_stat;
```

Input Arguments

`bodyptr`	Pointer to the beginning of the message body of the IPC buffer.
`bodylen`	Maximum length of the message body.

first_seg	Boolean value specifying whether the current generation request should be the first request in the message. TRUE implies first segment.	
alarm_id	Identifier of the alarm to be generated.	
fr_id	Identifier of the factory resource for which the alarm is being generated.	
user_or_serv_id	Used for labeling logged alarms. Usually this is the service_id of the sending process. The AMAP sends the user_id of the connected terminal.	
ref_id	Used to specify unique alarms when multiple alarm definitions are generated for the same Factory Resources. Alarm uniqueness is defined by the combination of alarm_id, fr_id, and ref_id.	
	Note: The ref_id is not displayed directly on the Alarm Manager User Interface. The ref_id, when used, can be duplicated in a message field for display.	
action	Specifies the update action to take.	
seq_num	Only used by AMAP. Application programs should pass 0.	
resp_type	Used to select the type of response desired from the AMRP. The application program has three choices:	
	AM_CAPTURED_RESP - AMRP	Responds once the message has been captured (sent to a standby process or journalled).
	AM_FULL_RESP - AMRP	Responds once the message has been fully processed. A status segment is returned for each request in the original message.
	AM_NO_RESP	No response from AMRP to indicate that an alarm message has been received.
	Note: Only the resp_type in the first generation message of each segment is used. Therefore, it is not possible to intermix the type of responses desired within a single IPC message..	
key	The key is useful when the resp_type is set to AM_FULL_RESP. The key allows the application program to match the status segments returned with the alarm generation/update requests. The key is specified by the application program and is returned as is by the AMRP.	
changeapproval_obj	A pointer to the change approval information for the alarm update operation.	

Output Arguments

ret_stat	Pointer to status structure.

Return Value:

Either COR_SUCCESS, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in ret_stat. Other than the changeapproval_obj parameter, this function should in all ways conform to the behavior of amaru_add_update().

amaru_add_update_stamp

Call this subroutine to add alarm update information to the current IPC buffer with an external alarm timestamp.

Syntax

```
int amaru_add_update_stamp (bodyptr, bodylen, first_seg,
                        alarm_id, fr_id, user_or_serv_id,
                        ref_id, action, seq_num, resp_type,
                        key, stamp, ret_stat);
char *bodyptr;
int bodylen;
COR_BOOLEAN first_seg;
char alarm_id[LONG_NAME_LEN+1];
char fr_id[FR_ID_LEN+1];
char user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1];
char ref_id[AM_REF_ID_LEN+1];
AM_STATE_TYPE action;
int seq_num;
AM_RESP_TYPE resp_type;
AM_RESP_KEY key;
COR_STAMP stamp;
COR_STATUS *ret_stat;
```

Input Arguments

bodyptr	Pointer to the beginning of the message body of the IPC buffer.
bodylen	Maximum length of the message body.
first_seg	Boolean value specifying whether the current generation request should be the first request in the message. TRUE implies first segment.
alarm_id	Identifier of the alarm to be generated.
fr_id	Identifier of the factory resource for which the alarm is being generated.
user_or_serv_id	Used for labeling logged alarms. Usually this is the **service_id** of the sending process. The AMAP sends the **user_id** of the connected terminal.
ref_id	Used to specify unique alarms when multiple alarm definitions are generated for the same Factory Resources. Alarm uniqueness is defined by the combination of **alarm_id** , **fr_id** , and **ref_id** .
	Note: The **ref_id** is not displayed directly on the Alarm Manager User Interface. The **ref_id** , when used, can be duplicated in a message field for display.
action	Specifies the update action to take.
seq_num	Only used by AMAP. Application programs should pass zero (0).
resp_type	Used to select the type of response desired from the AMRP. The application program has three choices:
	AM_CAPTURED_RESP - AMRP responds once the message has been captured (sent to a standby process or journalled).

	AM_FULL_RESP - AMRP responds once the message has been fully processed. A status segment is returned for each request in the original message.
	AM_NO_RESP - No response from AMRP to indicate that an alarm message has been received.
	Note: Only the **resp_type** in the first generation message of each segment is used. Therefore, it is not possible to intermix the type of responses desired within a single IPC message..
key	The key is useful when the **resp_type** is set to AM_FULL_RESP. The key allows the application program to match the status segments returned with the alarm generation/update requests. The key is specified by the application program and is returned "as is" by the AMRP.
stamp	Specifies the update time for the alarm. If this update causes the alarm to be reset, this timestamp will be used to compute the alarm duration.

Output Arguments

ret_stat	Pointer to status structure.

Return Value:

Either COR_SUCCESS, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in **ret_stat** . Other than the timestamp parameter, this function should in all ways conform to the behavior of **amaru_add_update()** .

amaru_add_update_stamp_ca

Call this subroutine to add alarm update information to the current IPC buffer with external alarm timestamp and change approval Information.

Syntax

```
Int  amaru_add_update_ca (bodyptr, bodylen, first_seg,
 alarm_id, fr_id, user_or_serv_id,
  ref_id, action, seq_num, resp_type,
key, changeapproval_obj, ret_stat);
char *bodyptr;
int bodylen;
COR_BOOLEAN first_seg;
char alarm_id[LONG_NAME_LEN+1];
char fr_id[FR_ID_LEN+1];
char user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1];
char ref_id[AM_REF_ID_LEN+1];
AM_STATE_TYPE action;
int seq_num;
AM_RESP_TYPE resp_type;
AM_RESP_KEY key;
COR_STAMP stamp;
ChangeapprovalInfo *changeapproval_obj,
```

```
COR_STATUS *ret_stat;
```

Input Arguments

`bodyptr`	Pointer to the beginning of the message body of the IPC buffer.	
`bodylen`	Maximum length of the message body.	
`first_seg`	Boolean value specifying whether the current generation request should be the first request in the message. TRUE implies first segment.	
`alarm_id`	Identifier of the alarm to be generated.	
`fr_id`	Identifier of the factory resource for which the alarm is being generated.	
`user_or_serv_id`	Used for labeling logged alarms. Usually this is the `service_id` of the sending process. The AMAP sends the `user_id` of the connected terminal.	
`ref_id`	Used to specify unique alarms when multiple alarm definitions are generated for the same Factory Resources. Alarm uniqueness is defined by the combination of `alarm_id`, `fr_id`, and `ref_id`.	
	Note: The `ref_id` is not displayed directly on the Alarm Manager User Interface. The `ref_id`, when used, can be duplicated in a message field for display.	
`action`	Specifies the update action to take.	
`seq_num`	Only used by AMAP. Application programs should pass 0.	
`resp_type`	Used to select the type of response desired from the AMRP. The application program has three choices:	
	`AM_CAPTURED_RESP - AMRP`	Responds once the message has been captured (sent to a standby process or journalled).
	`AM_FULL_RESP - AMRP`	Responds once the message has been fully processed. A status segment is returned for each request in the original message.
	`AM_NO_RESP`	No response from AMRP to indicate that an alarm message has been received.
	Note: Only the `resp_type` in the first generation message of each segment is used. Therefore, it is not possible to inter-mix the type of responses desired within a single IPC message..	
`key`	The key is useful when the `resp_type` is set to `AM_FULL_RESP`. The key allows the application program to match the status segments returned with the alarm generation/update requests. The key is specified by the application program and is returned "as is" by the AMRP.	
`stamp`	Specifies the update time for the alarm. If this update causes the alarm to be reset, this timestamp will be used to compute the alarm duration.	
`changeapproval_obj`	A pointer to the change approval information for the alarm update operation.	

Output Arguments

`ret_stat`	Pointer to status structure.

Return Value:

Either `COR_SUCCESS`, or `COR_FAILURE`. If the function returns anything other than `COR_SUCCESS`, additional error information can be found in `ret_stat`.

Other than the `changeapproval_obj` parameter, this function should in all ways conform to the behavior of `amaru_add_update()`.

amaru_send_msg

Call this subroutine to send the IPC-datagram message **containing alarm generation/updated requests filled in by the amaru_add_gen or the amaru _add_update** routine.

amaru_send_msg returns the status of the operation in the status message. Therefore, to determine if an **amaru_add_gen** or **amaru_add_update** call is successful, the status message must be checked.

Syntax

```
int amaru_send_msg (port_id, wrt_buf, read_buf,
                    read_buf_len, alarm_mgr_id, ret_stat);
int port_id;
char *wrt_buf;
char *read_buf;
int read_buf_len;
char *alarm_mgr_id;
COR_STATUS *ret_stat;
```

Input Arguments

port_id	Datagram Port over which message should be sent.
wrt_buf	Pointer to message buffer to be written.
read_buf	Pointer to message buffer where returned status should be sent.
read_buf_len	Maximum length of read buffer.
alarm_mgr_id	ID of the AMRP to receive the message. The application can determine the **alarm_mgr_id** from the configuration record of the factory resource the alarm is associated with (in **fr.dat**). Each factory resource has an associated AMRP.

Output Arguments

ret_stat	Pointer to status structure.

Return Value

Either COR_SUCCESS, or COR_FAILURE. If the function returns anything other than COR_SUCCESS, additional error information can be found in **ret_stat.err_msg** and **ret_stat.err_code** .

All error codes returned by internally used procedures **are passed unchanged to the calling program. If the AMRP with the specified alarm _mgr_id** cannot be found, a status of COR_FAILURE is set and the amaru_UNKNOWN_ALARM_MGR error code is returned. This error status is defined in **amaru_err.h** which is shown in Chapter 8.

amaru_alloc_buffer

Call this subroutine to create an IPC datagram buffer for alarm messages. Two calls should be made, one for a write buffer, and one for a read buffer.

Syntax

```
amaru_alloc_buffer ( alarm_write_buffer, alarm_write_len,
                     alarm_write_body);
IPCDG **alarm_write_buffer;
int *alarm_write_len;
char **alarm_write_body;
amaru_alloc_buffer( alarm_read_buffer, alarm_read_len,
                    alarm_read_body);
IPCDG **alarm_read_buffer;
int *alarm_read_len;
char **alarm_read_body;
```

Input Arguments

None.

Output Arguments

alarm_write_buffer	Address of datagram write buffer
alarm_write_len	Write buffer length
alarm_write_body	Pointer to address of write message within buffer
alarm_read_buffer	Address of datagram read buffer
alarm_read_len	Read buffer length
alarm_read_body	Pointer to address of read message within buffer

Return Value

None.

amaru_num_messages

Call this subroutine to monitor the number of response messages in the allocated datagram read buffer.

Syntax

```
amaru_num_messages ( alarm_read_body, ret_stat);
char *alarm_read_body;
COR_STATUS *ret_stat;
```

Input Arguments

alarm_read_body	Pointer to address of datagram body in allocated buffer. (Header information contains number of alarm messages in body.)

Output Arguments

returned value	Number of messages in body.
ret_stat	Pointer to status structure.

Return Value

None.

amaru_get_resp

This routine retrieves a designated response from the datagram read buffer. Given the value i, where 0<i< the total number of messages in the buffer, the routine retrieves the status pertaining to the Ith message.

Syntax

```
amaru_get_resp ( alarm_read_body, i, ret_stat);
char *alarm_read_body;
int i;
COR_STATUS *ret_stat;
```

Input Arguments

alarm_read_body	Pointer to address of datagram body in allocated buffer.
i	The number of the message to seek (0 < i < the total number of messages in the buffer)

Output Arguments

ret_stat	Pointer to status structure.

Return Value

None.

amaru_free_buffer

This routine is called by an application program when the number of AMARU datagram messages no longer needs monitoring. A call must be made for each buffer allocated.

Syntax

```
amaru_free_buffer (alarm_write_buffer);
IPCDG *alarm_write_buffer;
amaru_free_buffer (alarm_read_buffer);
IPCDG *alarm_read_buffer;
```

Input Arguments

alarm_write_buffer	Address of datagram write buffer.
alarm_read_buffer	Address of datagram read buffer.

Output Arguments

None.

Return Value

None.

amaru_terminate

Call this subroutine in your application program when communication with Alarm Management Resident Processes is no longer desired.

Syntax

```
amaru_terminate()
```

Input Arguments

None.

Output Arguments

None.

Return Value

None.

Alarm Management Configuration Files

Alarm Management Configuration Files

The system configuration files accessed by Alarm Management are:

- Alarm Definition file (`alarm_def`)
- Alarm Class file (`alarm_class`)
- Alarm Type file (`alarm_type`)
- Alarm Field file (`alarm_field`)
- Alarm Manager file (`alarm_mgr`)
- Alarm Routing file (`alarm_routing`)
- Alarm Interested Processes file (`alarm_intproc`)

The records defined in these files have the following relationships:

The following sections describe the configuration files that are accessed directly by Alarm Management.

Alarm Definition File (alarm_def)

Alarm Definition File (alarm_def)

The ALARM_DEF records designated in this configuration file define specific alarms. Each alarm is uniquely identified by its **alarm_id** .

Record Type ALARM_DEF

Filenames	**alarm_def.idt, alarm_def.dat,**
	alarm_def.idx
Edit Locations	**%SITE_ROOT%\master**

Entries in the **alarm_def.dat** are created via interactive configuration transactions.

Field Definitions for Alarm Definition File

Field Definitions for Alarm Definition File

Records in this file contain the following fields:

- ack_tout
- alarm_id
- alarm_msg
- alarm_type_id
- class_id
- clr_tout
- del_opt
- description
- help_fname
- log_file
- log_opt
- manual_clear_allowed
- max_stacked
- rep_tout

ack_tout

Maximum Field Length	long integer

Definition	Acknowledge Time-out option. The time in minutes that should elapse before the alarm is automatically acknowledged. A value of zero indicates the alarm should not be automatically acknowledged, and a value less than zero indicates the alarm should immediately be automatically acknowledged.

alarm_id

Maximum Field Length	256 characters
Definition	Unique identifier for the alarm

alarm_msg

Maximum Field Length:	80 characters
Definition	The alarm message format. The message contains the fixed text for the alarm and position holders for the run-time parameters.

alarm_type_id

Maximum Field Length	16 characters
Definition	The parameters in the alarm message. A group of alarm fields in an alarm message is specified by defining alarm types.

class_id

Maximum Field Length	5 characters
Definition	The class associated with this alarm

clr_tout

Maximum Field Length	long integer
Definition	Clear Time-out option. The time in minutes that should elapse before the alarm condition is automatically reset. A value of zero indicates the alarm should not be automatically reset, and a value less than zero indicates the alarm should immediately be reset.

del_opt

Maximum Field Length	2 characters
Definition	Code indicating when the alarm occurrence should be deleted from the system. Valid entries are:

	A	Acknowledge only
	R	Reset only
	AR	Acknowledge and Reset.

Description

Maximum Field Length	40 characters
Definition	A brief explanation of the alarm definition.

help_fname

Maximum Field Length	80 characters
Definition	The name of a text file containing user information on what to do when the alarm occurs.

log_file

Maximum Field Length	byte	
Definition	File to which alarms are to be logged:	
	0	ALARM_LOG
	1	EVENT_LOG.

log_opt

Maximum Field Length	4 characters
Definition	Code indicating whether or not, and if so, when, the alarm should be logged. The alarm can be logged when Generated, Acknowledged, Reset, or Deleted. To specify any combination of the above, this field contains any combination of the letters A,R,D,G.

manual_clear_allowed

Maximum Field Length	1 character	
Definition	Code indicating whether or not an operator is allowed to reset this alarm from the Alarm Management User Interface display. Valid entries are:	
	Y	Operator can reset the alarm condition
	N	Prohibits the operator from resetting the alarm condition.

max_stacked

Maximum Field Length	long integer
Definition	The maximum number of alarm occurrences tracked for stacked alarms. A value of zero indicates the alarm should not be stacked. The maximum number of stacked alarms is 19.

rep_tout

Maximum Field Length	long integer
Definition	Time in minutes before alarm is repeated. A value of zero means that the alarm will not be repeated.

Sample Configuration File (alarm_def)

The following is an example of the **alarm_def.idt** configuration file:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: ALARM_DEF LIST OF CONFIGURED ALARMS
*
*   0 ALARM_ID                Identifies the Alarm
*   1 CLASS_ID                Identifies the Class
*   2 alarm_type_id           Identifies the Alarm Type
*   3 alarm_msg               Raw Alarm Message
*   4 del_opt                 Delete on A-ACK,R-RESET,AR-ACK AND RESET
*   5 manual_clear_allowed    Manual Clear of Alarm flag
*   6 log_opt                 Log = GARD for Gen, Ack, Reset, Del
*   7 ack_tout                Time in minutes before Alarm is ACKed
*   8 clr_tout                Time in minutes before Alarm is Cleared
*   9 max_stacked             Maximum number of Alarms to Stack
*  10 help_fname              Help File Name if available
*  11 log_file                Standard or alt log file specification
*  12 description             Description of Alarm Definition
*  13 rep_tout                Time in minutes before Alarm is Repeated
*
$ALARM_DISABLED|$SYS|$PTM_AM|Alarm Detection disabled for: %s|-
A|N|G|-1|0|0||1||0
$ALARM_ENABLED|$SYS|$PTM_AM|Alarm Detection enabled for: %s|-
A|N|G|-0|0|0||1||0
$ALARM_MODIFIED|$SYS|$PTM_AM|Alarm limits modified for: %s|-
A|N|G|-1|0|0||1||0
$ALARM_RAWLIM|HIGH|$OOR_AL|Point %s has exceeded its Raw Limits|-
AR|N||0|0|0||0||0
$ALARM_RESTORED|$SYS|$PTM_AM|Alarm limits restored for: %s|-
A|N|G|-1|0|0||1||0
```

Alarm Class File (alarm_class)

Alarm Class File (alarm_class)

An ALARM_CLASS record in this configuration file defines an alarm class. The class_id uniquely identifies the alarm class. Alarm classes are user-defined groupings for alarms. They are useful to view alarms sorted by a user-defined criteria.

Record Type	ALARM_CLASS
Filenames	**Alarm_class.idt, alarm_class.dat,**
	Alarm_class.idx
Edit Locations	**%SITE_ROOT%\master**

Entries in the **alarm_class.dat** file are created via interactive configuration transactions.

Field Definitions for Alarm Class

Field Definitions for Alarm Class

Records in this file contain the following fields:File

- class_ack_bg
- class_ack_fg
- class_alarm_bg
- class_alarm_fg
- class_id
- class_normal_bg
- class_normal_fg
- class_order
- class_title

class_ack_bg

Maximum Field Length	integer
Definition	Ack state background color.

class_ack_fg

Maximum Field Length	integer

Definition	Ack state foreground color.

class_alarm_bg

Maximum Field Length	integer
Definition	Alarm state background color.

class_alarm_fg

Maximum Field Length	integer
Definition	Alarm state foreground color.

class_id

Maximum Field Length	5 characters
Definition	Unique identifier for the class.

class_normal_bg

Maximum Field Length	integer
Definition	Normal state background color.

class_normal_fg

Maximum Field Length	long integer
Definition	Normal state foreground color.

class_order

Maximum Field Length	long integer
Definition	Order in which alarms should appear when sorted by class.

class_title

Maximum Field Length	64 characters
Definition	Title to appear in the alarm setup display produced by the AMAP.

Sample Configuration File (alarm_class)

The following is an example of the **alarm_class.idt** configuration file:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: ALARM_CLASS LIST OF ALARM CLASSES
*
*   0 CLASS_ID                Identifier for the Class
*   1 class_title             Title used in class display
*   2 class_order             Order for class sort
*   3 class_alarm_fg          Alarm state foreground color
*   4 class_alarm_bg          Alarm state background color
*   5 class_normal_fg         Normal state foreground color
*   6 class_normal_bg         Normal state background color
*   7 class_ack_fg            Ack state foreground color
*   8 class_ack_bg            Ack state background color
*
$SYS|Point Mgmt Alarms|3|7|0|7|0|7|0
HIGH|High Priority Alarms|0|7|0|7|0|7|0
INFO|Information Alarm|5|7|3|7|0|7|0
LOW|Low Priority Alarms|2|7|0|7|0|7|0
MED|Medium Priority Alarms|1|7|0|7|0|7|0
WARN|Warning Alarm|2|0|15|0|2|7|0
```

Alarm Type File (alarm_type)

Alarm Type File (alarm_type)

An ALARM_TYPE record in this configuration file defines a type of alarm message. The alarm_type_id uniquely identifies the alarm message type. Alarm types are useful to define and refer to common types of alarm messages.

Record Type	ALARM_TYPE
Filenames	**Alarm_type.idt, alarm_type.dat,**
	Alarm_type.idx
Edit Locations	**%SITE_ROOT%\master**

Field Definitions for Alarm Type File

Field Definitions for Alarm Type File

Records in this file contain the following fields:

- alarm_type_id

- description

alarm_type_id

Maximum Field Length	16 characters
Definition	Unique identifier for the type of alarm message.

Description

Maximum Field Length	40 characters
Definition	String used to document the purpose of the alarm type.

Sample Configuration File (alarm_type)

The following is an example of the **alarm_type.idt** configuration file:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: ALARM_TYPE TYPE OF ALARM MESSAGE FORMATS
*
*   0 ALARM_TYPE_ID              Identifies Alarm Type
*   1 description                Description of Alarm Type
*
$AM_STATUS|Alarm status alarm
$AT_21|
$DEVICE|Device Comm. Toolkit - device down alarm
$DYN_CFG|Dynamic config change
$MAC_EVENT|Alarm w/ single field
$OOR_AL|Alarm for out of range points
$PTM_AM|Point Mgmt info msg: Point_id %s
$RTR_LINK_DOWN|Router down alarm
AMSI_ALARM|Device-Generated Alarm Message
MCP_PROC_DOWN|Process Manager Alarm
NO_FIELDS|Contains only the alarm message
```

Alarm Field File (alarm_field)

Alarm Field File (alarm_field)

An ALARM_FIELD record in this configuration file defines a field for a specific type of alarm message.)

Record Type	ALARM_FIELD
Filenames	**alarm_field.idt, alarm_field.dat,**

	alarm_field.idx
Edit Locations	**%SITE_ROOT%\master**

Field Definitions for Alarm Field File

Field Definitions for Alarm Field File

Records in this file contain the following fields:

- alarm_field_id
- alarm_type_id
- description
- field_format
- field_len
- field_num
- field_type
- field_use

alarm_field_id

Maximum Field Length	16 characters
Definition	Short, mnemonic description of field's purpose displayed as the field identifier during the configuration transaction for alarm definitions of this alarm type.

alarm_type_id

Maximum Field Length	16 characters
Definition	ID of the alarm type this field is associated with.

Description

Maximum Field Length	40 characters
Definition	String used to describe the purpose of the field.

field_format

Maximum Field Length	10 characters
Definition	C-format string for the field. The parameter appears in the alarm message according to the format string specified. Legal values are:

	%d	decimal
	%o	unsigned octal
	%x	unsigned hexadecimal
	%u	unsigned decimal
	%c	single character
	%s	string
	%e	scientific notation
	%f	floating point
	%g	use scientific notation or floating point, whichever is shorter

Note: All point values in alarm messages are passed as strings.

field_len

Maximum Field Length	long integer
Definition	Number of bytes in the variable parameter. Used for character string fields.

field_num

Maximum Field Length	long integer
Definition	Position of the field in the alarm message string.

field_type

Maximum Field Length	long integer	
Definition	Type of field. Legal values are:	
	0	Character
	1	String
	2	Integer
	3	Integer * 1
	4	Integer * 2
	5	Integer * 4
	6	Boolean
	7	Unsigned integer * 1

	8	Unsigned integer * 2
	9	Unsigned integer * 4
	10	Floating point

field_use

Maximum Field Length	long integer
Definition	An application-specific field used to identify application information on the content of the field. For instance, it is used in Point Management alarms to specify the Point Id that should be included in the alarm message. If set to '4,' Point Management inserts the configured engineering units label into the alarm message.

Sample Configuration File (alarm_field)

The following is an example of the **alarm_field.idt** configuration file:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: ALARM_FIELD ALARM FIELDS FOR EACH TYPE
*
*   0 ALARM_TYPE_ID       Alarm Type Identifier
*   1 field_num           Order field appears in alarm message
*   2 field_type          C-0,S-1,I-2,I1-3,4,5,B-6,U1-7,8,9
*   3 field_len           Number of characters in string fields
*   4 field_format        C Format string for the field
*   5 field_use           Field usage in application
*   6 description         Description of Alarm Field
*   7 am_field_id         Alarm field identifier
*
$AM_STATUS|0|5|10|%ld|0||GEN count
$AM_STATUS|1|5|10|%ld|0||UPD count
$AM_STATUS|2|5|10|%ld|0||LOG time
$AT_21|0|1|32|%s|2||
$AT_21|1|1|16|%s|1||
$DEVICE|0|1|72|%s|0|DEVCOM Toolkit alarm message|MESSAGE
$DYN_CFG|0|1|16|%s|0|User ID|DYNCFG USER
$DYN_CFG|1|1|16|%s|1|action|DYNCFG ACTION
$DYN_CFG|2|1|16|%s|2|entity type|DYNCFG TYPE
$DYN_CFG|3|1|32|%s|3|entity name|DYNCFG NAME
```

Records in this file whose **alarm_type_id** start with "$AL_" must not be modified. These are standard alarm types generated by CIMPLICITY software.

Alarm Manager File (alarm_mgr)

Alarm Manager File (alarm_mgr)

The record in this configuration file defines information global to a specific AMRP. The tuning parameters should rarely be modified. They exist to make Alarm Management more responsive to certain types of requests. The AMRP contains an effective priority scheme for determining what actions to perform next when a number of actions are outstanding. Each time an action is passed over for execution its priority is raised insuring that eventually the action is performed.

Record Type	ALARM_MGR
Filenames	**alarm_mgr.idt, alarm_mgr.dat,**
	alarm_mgr.idx
Edit Locations	**%SITE_ROOT%\master**

Field Definitions for Alarm Manager File

Field Definitions for Alarm Manager File

Note: Fields **segs_to_proc** through **process_jrnl_prio** define tuning parameters used to control AMRP processing. These parameters are reserved for GE Intelligent Platforms use. If you feel that they need adjusting on your system, please call CIMPLICITY Technical Support before making any changes.

Fields **dg_input_prio** through **process_jrnl_prio** define how AMRP activities are scheduled. Each type of activity is assigned an initial effective priority. Each time an item is not processed, its priority is increased by one. The priority range is from 1 to 20.

Records in this file contain the following fields:

- alarm_mgr_id
- alloc_segs_to_proc
- auto_pnum
- autoupd_tout
- count_
- count_type
- date_mask
- dg_input_prio
- dyn_am_cont_mask
- dyn_am_video_mask
- gen_auto_prio
- jrnl_max

- jrnl_max_act_limit
- jrnl_min
- jrnl_status
- jrnl_timer_period
- master_input_prio
- process_allocq_prio
- process_auto_prio
- process_jrnl_prio
- process_updq_
- segs_to_proc
- service_id
- upd_terms_prio
- ur_input_prio

alarm_mgr_id

Maximum Field Length	16 characters
Definition	The AMRP for which these parameters should be associated.

alloc_segs_to_proc

Maximum Field Length	long integer
Definition	Tuning parameter: the number of AMAP request segments to process each time slot.

auto_pnum

Maximum Field Length	long integer
Definition	Tuning parameter: number of Auto Alarms to process each time slot.

autoupd_tout

Maximum Field Length	long integer
Definition	Time out for processing Auto Alarms. Alarms which should be automatically Acknowledged or Reset are checked after each **autoupd_tout_minutes** . If their timers are expired, the alarms statuses are updated accordingly.

*count_*

Maximum Field Length	byte

Definition	Code identifying the display attribute used for the alarm count value when displayed by the CIMPLICITY Alarm Viewer.mask The **count_mask** values are:

	1	Blink
	2	Reverse video
	4	High intensity
	8	Underline
	16	Bell

	To combine attributes, add their values. For example, to indicate a bell, revise video, blinking count, the count mask value would be 19 (16+2+1). A zero can also be used to indicate the default reverse video attribute

count_type

Maximum Field Length	long integer
Definition	This parameter is used to specify the meaning of the alarm count on the Context Manager screens. The count can represent all alarms, all unacknowledged alarms, or all unacknowledged alarms still in an alarm state. Valid values are:

	0	All alarms
	1	All UNACK alarms
	2	All UNACK, UNCLEAR alarms

date_mask

Maximum Field Length	byte
Definition	Code identifying the display attribute for the alarm date when displayed by the CIMPLICITY Alarm Viewer. The **date_mask** values are:

	1	Blink
	2	Reverse video
	4	High intensity
	8	Underline
	16	Bell

	To combine attributes, add their values. For example, to indicate a bell, revise video, blinking count, the count mask value would be 19 (16+2+1). A zero can also be used to indicate the default reverse video attribute

dg_input_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for standard datagram input. This is mostly for messages to generate and update alarms. This value should be staticly high to insure the IPC queues are flushed often. This helps avoid lost messages due to congestion drops.

dyn_am_cont_mask

Maximum Field Length	long integer
Definition	Parameter defining what optional reference items appear on the screen during dynamic display mode in the Alarm Management user interface. The alarm record on this screen always contains the number of the alarm, time the alarm was logged, acknowledgment status, and alarm message. The optional information can be configured via this parameter according to the following values:

	4	Include Alarm ID with default information
	2	Include Resource ID with default information
	1	Include Alarm Class with default information
	0	Display only default information

	To include more than one optional item, add their values. For example, to include both Alarm and Resource IDs, enter the value of 6 (4+2). To specify no optional items, set this parameter to 0.

dyn_am_video_mask

Maximum Field Length	long integer
Definition	Parameter to set the video display for alarms in alarm state when the Alarm Management user interface is in dynamic display mode. Valid values are:

	4	Alarms in alarm state are displayed with high intensity attribute
	2	Alarms in alarm state are displayed with reverse video attribute

	If any other number is specified, alarms in alarm state are displayed in normal video attribute.

gen_auto_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for going through the list of alarms to be auto acknowledged or reset once the auto timer expires.

jrnl_max

Maximum Field Length	long integer
Definition	The maximum threshold for Journal Rollover (in blocks). If the journal file reaches this size before the rollover takes place, the rollover is done regardless of the current Alarm Management work load.

jrnl_max_act_limit

Maximum Field Length	long integer
Definition	The maximum activity limit for doing the journal rollover. **Note:** This strategy is only used when the block size of the journal file is between the minimum and maximum threshold.

jrnl_min

Maximum Field Length	long integer
Definition	The minimum threshold for Journal Rollover (in blocks). A rollover is the action of taking the current alarm database, writing it to a dump file, and then clearing the journal file. When the minimum threshold is reached, the rollover takes place as soon as a lull in alarm activity is detected.

jrnl_status

Maximum Field Length	byte	
Definition	Code indicating whether or not journalling is present. Valid values are:	
	0	journalling is not present
	1	journalling is present

jrnl_timer_period

Maximum Field Length	long integer
Definition	The number of minutes to wait before doing a rollover. This parameter is used in conjunction with the **jrnl_max_act_limit** which specifies an activity limit on alarms. If the activity limit is not reached during the timer period, the journal rollover takes place. This is how the "lull in alarm activity" is specified.

master_input_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for processing input from an active AMRP to a standby AMRP.

process_allocq_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for processing outstanding requests by AMAPs. A staticly high number favors users at AMAPs.

process_auto_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for processing auto alarms once they have expired.

process_jrnl_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for processing the request to perform a journal rollover.

*process_updq_*

Maximum Field Length	long integeprio r
Definition	Tuning parameter: the starting effective priority for processing outstanding alarm update and generation requests which have been added to the Alarm Management input queue.

segs_to_proc

Maximum Field Length	long integer

Definition	Tuning parameter: the number of Alarm Generation and Update Segments to Process each time slot. This is really a measure of how long the AMRP will be performing this task. A small number causes the AMRP to check its queues often. A larger number causes the AMRP to process more messages before checking its queues again.

service_id

Maximum Field Length	32 characters
Definition	The service identifier of the AMRP as configured in **service.dat** . The AMRP determines its **service_id** from IPC XLATE and then uses this information to key into the ALARM_MGR file to read its global configuration data.

upd_terms_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for updating user terminals with new alarm count and date information. A low value gives priority to alarms being updated and generated during a burst of alarms. A high value gives priority to updating user terminals as alarms are generated and updated.

ur_input_prio

Maximum Field Length	long integer
Definition	Tuning parameter: the starting effective priority for input regarding user population changes from User Registration.

Sample Configuration File (alarm_mgr)

The following is an example of the **alarm_mgr.idt** configuration file:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: ALARM_MGR LIST OF ALARM MANAGERS
*
*  0 alarm_mgr_id        Identifier for this alarm manager
*  1 SERVICE_ID          Identifier for this alarm manager
*  2 jrnl_status         Journalling ON / OFF
*  3 jrnl_min            Min threshold (in blocks)
*  4 jrnl_max            Max threshold (in blocks)
*  5 jrnl_timer_period   Timer for jrnl activity check
*  6 jrnl_max_act_limit  Maximum activity in time period
*  7 autoupd_tout        Time period for processing auto alarms
*  8 auto_pnum           Number auto alarms to process at a time
*  9 dyn_am_cont_mask    Fields displayed in dynamic display mess
```

```
* 10 count_type           async field alarm count type
* 11 dyn_am_video_mask    Video attribute for dynamic display stat
* 12 segs_to_proc         Number of Update Segments to process
* 13 alloc_segs_to_proc   Number of AP Segments to process
* 14 dg_input_prio        static priority of Datagram input
* 15 ur_input_prio        static priority of UR input
* 16 master_input_prio    static priority of Server AMRP input
* 17 upd_terms_prio       static priority of Async Fields
* 18 gen_auto_prio        static priority of gen auto exp list
* 19 process_auto_prio    static priority of expired auto alarms
* 20 process_updq_prio    static priority of alarm update request
* 21 process_allocq_prio static priority of AP requests
* 22 process_jrnl_prio    static priority of Jrl Rollover
* 23 count_mask           Display Attribute alarm async count
* 24 date_mask            Display Attribute alarm async date/time
*
AMRP|AMRP|0|50|100|1|20|5|25|6|0|2|5|4|8|4|1|2|1|1|2|3|10|7|7
```

Alarm Routing File (alarm_routing)

Alarm Routing File (alarm_routing)

The records in this configuration file define the types of users who should receive information about specific alarms.

Record Type	ALARM_ROUTING
Filenames	**alarm_routing.idt, alarm_routing.dat,**
	alarm_routing.idx
Edit Locations	**%SITE_ROOT%\master**

Entries in the **alarm_routing.dat** file are created via interactive configuration transactions.

Field Definitions for Alarm Routing File

Field Definitions for Alarm Routing File

Records in this file contain the following fields:

- alarm_id
- role_id

alarm_id

Maximum Field Length	256 characters

Definition	Identifier of the alarm.

role_id

Maximum Field Length	16 characters
Definition	Identifier of the user role to route the alarm to.

Sample Configuration File (alarm_routing)

The following is an example of the **alarm_routing.idt** configuration file:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: ALARM_ROUTING ROLES TO RECEIVE ALARMS
*
*   0 ALARM_ID              Identifies the Alarm
* 1 ROLE_ID                 Role to receive info on the Alarm
*
$ADDRESS_CONFLICT|SYSMGR
$ADDRESS_CONFLICT|USER
$ALARM_RAWLIM|SYSMGR
$ALARM_RAWLIM|USER
$AM_STATUS|SYSMGR
$DEVICE|SYSMGR
$DEVICE|USER
$DEVICE_DOWN|SYSMGR
$DEVICE_DOWN|USER
$DEVICE_FAILOVER|SYSMGR
$DEVICE_FAILOVER|USER
$RTR_LINK_DOWN|OPER
$RTR_LINK_DOWN|SYSMGR
$RTR_LINK_DOWN|USER
$UNKNOWN_FAULT|SYSMGR
$UNKNOWN_FAULT|USER
```

Alarm Interested Processes File (alarm_intproc)

Alarm Interested Processes File (alarm_intproc)

The record in this configuration files specifies an application process interested in alarms associated with specific factory resources.

Record Type	ALARM_INTPROC
Filenames	**alarm_intproc.idt, alarm_intproc.dat,**
	alarm_intproc.idx

Edit Locations	**%SITE_ROOT%\master**

Field Definitions for Alarm Interested Processes File

Field Definitions for Alarm Interested Processes File

Records in this file contain the following fields:

- class_id
- fr_id
- log_file
- service_id

class_id

Maximum Field Length	5 characters
Definition	The name of a configured alarm class. If specified, the alarms sent to the named process is limited to only those having this class. If this field is blank, alarms of any class are sent.

fr_id

Maximum Field Length	16 characters
Definition	The identifier of the factory resource the process is interested in.

log_file

Maximum Field Length	byte	
Definition	Code indicating which alarms are sent to an interested process based on the type of logging configured for the alarm: Valid values are:	
	0	Send alarms that are logged to the standard log file.
	1	Send alarms that are logged to the alternate log file.
	2	Send any alarm.

service_id

Maximum Field Length	32 characters

Definition	The service identifier as configured in **service.dat** of the process interested in the alarms. The service identifier is used in a call to **ipc_xlate** to get the physical address of the interested process.

Sample Configuration File (alarm_intproc)

The following is an example of the **alarm_intproc.idt** configuration file:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: ALARM_INTPROC LIST OF PROCESSES RECEIVING ALARMS
*
*   0 FR_ID               Factory Resource Identifier
*   1 SERVICE_ID          Service Identifier
*   2 class_id            Identifies the Class
*   3 log_file            Standard or alt log file specification
DEVICE_1|MAINT_MGMT|HIGH|0
DEVICE_@|MAINT_MGMT|LOW|0
```

Alarm Management Definitions Header Files

Alarm Management Definitions Header Files

- am_defs.h
- amaru_proto.h

am_defs.h

The following is a partial listing of this file showing you important definitions you need to know:

```
#ifndef _AM_DEFS_H
#define _AM_DEFS_H
#include <inc_path/ddl.h>
#include <inc_path/rtr_bcst.h>
#include <inc_path/counter.h>
#include <inc_path/ptm_defs.h>
#include <inc_path/am_defs_constants.h>
#include <malloc.h>

TCHAR* amrp_get_longname(const TCHAR* short_name, TCHAR* long_name, int
 long_name_len);
TCHAR* amrp_get_long_ref_id(const TCHAR* alarm_id, const TCHAR* ref_id,
 TCHAR* long_name, int long_name_len);

#define ALARM_GET_LONG_NAME(short_name) amrp_get_longname(short_name,
 (TCHAR*)_alloca((MAX_ALM_ID_LEN+1) * sizeof(TCHAR)), MAX_ALM_ID_LEN+1)
```

```
#ifndef ADHOC_DEFS_H
   #include <inc_path/adhoc_defs.h>
#endif

/*************************************/
/* Internal format of Datagram Address */
/*************************************/

typedef TCHAR AM_DADDR[DATAGRAM_NODESIZE+1+OBJECT_NAME_LEN+1];
#define AM_BLANK_MSG _T("                                                  ")

/*****************************************************************/
/* Interaction between AMRP and process Generating or Updating    */
/* Alarm Status                                                   */
/*****************************************************************/
/* Types of responses a process can request.  Full implies, wait  */
/* until after processing is complete and sent a response on the  */
/* success or failure of the request.                             */
/*****************************************************************/
typedef int AM_RESP_TYPE;
#define AM_FULL_RESP      0
#define AM_CAPTURED_RESP  1
#define AM_NO_RESP        2

/*********************************************************************/
/* Typedef for Key for identifying which segment is being responded to */
/*********************************************************************/

typedef int AM_RESP_KEY;
/*****************************************************************/
/* Alarm States                                                   */
/* WARNING - the order here must not be changed.  For efficiency   */
/* the module AM_TERM_INFO assumes the order as specified. If this */
/* is changed, active terminal alarm information will be incorrect.*/
/*                    */
/* AM_REPEATED is not an actual state, but is used for auto-repeat */
/* updates.                    */
/* AM_COMMENT_DEL is not an actual state, used for logging info.   */
/* AM_UPDATE is not a state but used to indicate that the alarm might have
 updated values */
/*****************************************************************/

typedef int AM_STATE_TYPE;
#define AM_GENERATED          0
#define AM_ACKNOWLEDGED       1
#define AM_CLEARED            2
#define AM_DELETED            3
#define AM_NONEXISTENT        4
#define AM_COMMENT_MSG        5
#define AM_NOSTATE            6
#define AM_REPEATED           7
#define AM_COMMENT_DEL        8
#define AM_UPDATE_ALARM       9
```

```
#define AM_SHELVED_ONESHOT   10
#define AM_SHELVED_TIMEOUT   11
#define AM_UNSHELVED         12
#define AM_CONFIRMED         13
#define AM_UNCONFIRMED       14
#define AM_BLOCKED_FLTR      15
#define AM_UNBLOCKED_FLTR    16
#define AM_SHELVED_ALL       17
/*****************************************************************/
/* number of permanent states - I.E. state in which in an alarm  */
/* can remain                                                    */
/*****************************************************************/
#define AM_NUM_PERMANENT_STATES 9  /* gen, ack, clear, shelved, unshelved,
 confirmed, unconfirmed, blocked, unblocked */
#define AM_STATE_TYPE_NOT_PERMAMENT COR_MAXU1
// see amaru_state_type_to_mask and amaru_bit_index_to_state_type
#define AM_GENERATED_BIT         0
#define AM_ACKNOWLEDGED_BIT      1
#define AM_CLEARED_BIT           2
#define AM_DELETED_BIT           AM_STATE_TYPE_NOT_PERMAMENT
#define AM_NONEXISTENT_BIT       AM_STATE_TYPE_NOT_PERMAMENT
#define AM_COMMENT_MSG_BIT       AM_STATE_TYPE_NOT_PERMAMENT
#define AM_NOSTATE_BIT           AM_STATE_TYPE_NOT_PERMAMENT
#define AM_REPEATED_BIT          AM_STATE_TYPE_NOT_PERMAMENT
#define AM_COMMENT_DEL_BIT       AM_STATE_TYPE_NOT_PERMAMENT
#define AM_UPDATE_ALARM_BIT      AM_STATE_TYPE_NOT_PERMAMENT
#define AM_SHELVED_ONESHOT_BIT   AM_STATE_TYPE_NOT_PERMAMENT
#define AM_SHELVED_TIMEOUT_BIT   AM_STATE_TYPE_NOT_PERMAMENT
#define AM_UNSHELVED_BIT         3
#define AM_CONFIRMED_BIT         4
#define AM_UNCONFIRMED_BIT       5
#define AM_BLOCKED_FLTR_BIT      6
#define AM_UNBLOCKED_FLTR_BIT    7
#define AM_SHELVED_ALL_BIT       8
// see amaru_state_type_to_array_index and amaru_state_array_index_to_type
#define AM_GENERATED_ARRAY_INDEX         0
#define AM_ACKNOWLEDGED_ARRAY_INDEX      2 // for historical reasons, these
 are swapped
#define AM_CLEARED_ARRAY_INDEX           1 // for historical reasons, these
 are swapped
#define AM_DELETED_ARRAY_INDEX           AM_STATE_TYPE_NOT_PERMAMENT
#define AM_NONEXISTENT_ARRAY_INDEX       AM_STATE_TYPE_NOT_PERMAMENT
#define AM_COMMENT_MSG_ARRAY_INDEX       AM_STATE_TYPE_NOT_PERMAMENT
#define AM_NOSTATE_ARRAY_INDEX           AM_STATE_TYPE_NOT_PERMAMENT
#define AM_REPEATED_ARRAY_INDEX          AM_STATE_TYPE_NOT_PERMAMENT
#define AM_COMMENT_DEL_ARRAY_INDEX       AM_STATE_TYPE_NOT_PERMAMENT
#define AM_UPDATE_ALARM_ARRAY_INDEX      AM_STATE_TYPE_NOT_PERMAMENT
#define AM_SHELVED_ONESHOT_ARRAY_INDEX   AM_STATE_TYPE_NOT_PERMAMENT
#define AM_SHELVED_TIMEOUT_ARRAY_INDEX   AM_STATE_TYPE_NOT_PERMAMENT
#define AM_UNSHELVED_ARRAY_INDEX         3
#define AM_CONFIRMED_ARRAY_INDEX         4
#define AM_UNCONFIRMED_ARRAY_INDEX       5
#define AM_BLOCKED_FLTR_ARRAY_INDEX      6
```

```
#define AM_UNBLOCKED_FLTR_ARRAY_INDEX    7
#define AM_SHELVED_ALL_ARRAY_INDEX       8
/**********************/
/* Legal Filter Types */
/**********************/

typedef int AM_FILTER_TYPE;
#define AM_TIME_FILTER  0
#define AM_STATE_FILTER 1
#define AM_FR_FILTER    2
#define AM_CLASS_FILTER 3
#define AM_FLD_FILTER   4 // this also includes FR and CLASS filters in new
 setups

/* alarm block state
Alarm can remain in any of the following states.
AM_ALARM
AM_BLOCKED
AM_NO_ALARM
*/

typedef int AM_BLOCK_TYPE;
#define AM_ALARM           0
#define AM_BLOCKED         1
#define AM_NO_ALARM        2
// flags that can modify how an alarm gen is handled
#define AM_GEN_UPDATE_ALARM (1 << 0)
#define AM_GEN_IF_EXISTING  (1 << 1)
#define AM_GEN_CREATE       (1 << 2)
#define AM_GEN_ACK          (1 << 3)
#define AM_GEN_RESET        (1 << 4)
#define AM_GEN_MSGCHANGE    (1 << 5)
#define AM_GEN_IGNORE_TIME  (1 << 6)

/****************************************/
/* Type Definitions for Filter Parameters */
/****************************************/
#define SIZEOF_AM_FILTER COR_MAX(CLASS_ID_LEN+1,FR_ID_LEN+1)

/****************************************/
/* Proficy Process Systems               */
/****************************************/
//#define PPS_REF_ID          _T("ProficyProcessSystemsAlarm #")
#define PPS_REF_ID          _T("PROFICYPROCESSSYSTEMSALARM #")

// Values to indicate which field will be filtered on
typedef unsigned char AM_FIELD_ID;
//Values defined in am_defs_constants.h
// Value to indicate how a field will be matched
typedef unsigned char AM_FIELD_FILTER_TYPE;
//Values defined in am_defs_constants.h
// There can only be one instance of this type of filter in a segment
```

```
typedef struct am_time_state_filter
{
   COR_STAMP start_time;      // 0 if this filter is not applied
   COR_U2 alarm_state_mask;  // bit mask for alarm states to filter
} AM_TIME_STATE_FILTER;

typedef struct am_setup_field_filter
{
   AM_FIELD_ID fieldIndex;
   AM_FIELD_FILTER_TYPE matchType;
   TCHAR *filterString;
   void *regExp;
   TCHAR *upperCaseFilterString;
} AM_SETUP_FIELD_FILTER;

// Multiple field filters in a segment are represented by this structure,
// which represents the same data as AM_SETUP_FIELD_FILTER,
// but is used in segment repeat group with variable-length, NULL-
terminated string
typedef struct am_field_filter
{
   AM_FIELD_ID fieldIndex;
   AM_FIELD_FILTER_TYPE matchType;
   TCHAR filterString[1];
} AM_FIELD_FILTER;

#define am_field_filter_calc_size(filter_string) (offsetof(AM_FIELD_FILTER,
 filterString) + (_tcslen(filter_string) + 1) * sizeof(TCHAR))
typedef union am_filter
{
   COR_STAMP      start_time;
   TCHAR          class_id[CLASS_ID_LEN+1];
   AM_STATE_TYPE  alarm_state;
   TCHAR          fr_id[FR_ID_LEN+1];
   COR_I1         _pad[DO_ALIGN(SIZEOF_AM_FILTER,4)]; /* alignment data */
} AM_FILTER;

typedef struct am_filter_parm
{
   AM_FILTER_TYPE type;
   AM_FILTER filter;
} AM_FILTER_PARM;

typedef struct am_stacked_info
{
   COR_STAMP gentime;
   AM_STATE_TYPE alarm_state;
   TCHAR alarm_msg[ALARM_MSG_LEN+1];
   COR_I1 alarmLevel;
   COR_I2 severity;
} AM_STACKED_INFO;

//SCR23024. Added to support alarm clear time in AMV's StackView option.
```

```
typedef struct am_stacked_info_ex
{
   COR_STAMP gentime;
   AM_STATE_TYPE alarm_state;
   TCHAR alarm_msg[EXPANDED_ALARM_MSG_LEN+1];
   COR_I1 alarmLevel;
   COR_I2 severity;
   COR_STAMP clrtime;
} AM_STACKED_INFO_EX;

typedef struct am_stacked_info_exV1
{
   COR_STAMP gentime;
   AM_STATE_TYPE alarm_state;
   TCHAR alarm_msg[ALARM_MSG_LEN+1];
   COR_I1 alarmLevel;
   COR_I2 severity;
   COR_STAMP clrtime;
} AM_STACKED_INFO_EX_V1;

typedef struct am_stacked_info2
{
   COR_STAMP gentime;
   AM_STATE_TYPE alarm_state;
   COR_U2 alarm_msg_ofs ; //[ALARM_MSG_LEN+1];
//          COR_I1 _pad;                  /* alignment data */
   COR_I2 severity;
   COR_U1 alarmLevel;
} AM_STACKED_INFO2 ;

typedef LCID AM_LOCALEID;
typedef unsigned short AM_UNICODESIZE;
typedef unsigned short AM_UNICODENUMBER;
struct amUnicodeStringsHeader
{
   AM_UNICODESIZE size;
   AM_UNICODENUMBER number;
   // repeated number of times
   // AM_LOCALEID langid;
   // WCHAR language[];
   // BYTE unicodeMsg[length]
};

/*********************************************************************/
/* Maximum Number of Stacked Alarms (in addition to the most recent) */
/*********************************************************************/
#define AM_MAX_STACKED 19

/************************************/
/* Max length of Alarm Comment      */
/************************************/
#define old_AM_COMMENT_LEN 72
typedef struct old_am_comment_info
```

```
{
   COR_STAMP gentime;
   TCHAR alarm_comment[old_AM_COMMENT_LEN+1];
   COR_I1 _pad[3];                          /* alignment data */
} old_AM_COMMENT_INFO;
#define AM_COMMENT2_LINE_LEN 72
#define AM_COMMENT2_LEN (AM_COMMENT2_LINE_LEN*10)

typedef struct am_comment_info2
{
   COR_STAMP gentime;
   TCHAR alarm_comment[AM_COMMENT2_LEN+1];
   COR_I1 _pad[3];                          /* alignment data */
} AM_COMMENT2_INFO;
#define AM_COMMENT_ACT_ADD           _T('A')
#define AM_COMMENT_ACT_DELETE        _T('D')

/*****************************************************************/
/* Maximum Number of Alarm Comments                              */
/*****************************************************************/
#define AM_MAX_ALARM_COMMENTS 20

/*********************************/
/* Maximum size of a string parameter */
/*********************************/
#define AM_MAX_STR 73

/*****************************************/
/* Log and Delete Option Character Meanings */
/*****************************************/
#define AM_ACK_CHAR  _T('A')
#define AM_CLR_CHAR  _T('R')
#define AM_DEL_CHAR  _T('D')
#define AM_GEN_CHAR  _T('G')
#define AM_SHELVE_CHAR  _T('S')
#define AM_TIMEDSHELVE_CHAR  _T('T')

/*****************************************************************/
/* Defines for alarm visibility type                             */
/* WARNING - the order here must not be changed.  For efficiency   */
/* the module AM_TERM_INFO assumes the order as specified. If this */
/* is changed, active terminal alarm information will be incorrect.*/
#define AM_ALL_ALARMS  0  /* all alarms visible by this user */
#define AM_UNACK_ONLY  1  /* all unacknowledged alarms visible by this user
  */
#define AM_UNACK_UNCLR 2  /* all unacknowledged, uncleared alarms visible
 by */
/* this user */

/*********************************/
/* Defines for Message Formatting */
/*********************************/
#define AM_DG       _T('D')
```

```
#define AM_NON_DG   _T('L')
#define AM_LL       _T('L')

/**************************/
/* Resident Process States */
/**************************/
typedef int AMRP_STATE_TYPE;

/**********************/
/* Legal AMRP States */
/**********************/
#define AMRP_ACTIVE    0
#define AMRP_STANDBY     1

/***************************/
/* Legal Alarm Field Types */
/***************************/
typedef int AM_FIELD_TYPE;
#define AM_CHAR 0
#define AM_STR 1
#define AM_INT 2
#define AM_COR_I1 3
#define AM_COR_I2 4
#define AM_COR_I4 5
#define AM_COR_BOOLEAN 6
#define AM_COR_U1 7
#define AM_COR_U2 8
#define AM_COR_U4 9
#define AM_FLOAT 10
#define AM_SEVERITY 11  // this is a special field that holds the alarm
 severity level
#define AM_LEVEL 12  // this is a special field that holds the alarm
 state(Alarm - hi, lo: Warning - hi, lo)
#define AM_OPC_CATEGORY 13
#define AM_OPC_CONDITION 14
#define AM_OPC_SUBCONDITION 15
#define AM_STR_CONCAT 16
#define AM_POINT_VALUE 17
#define AM_STR_TRANSLATE 18
#define AM_USER_BITS 20
#define AM_POINT_VALUE1 21
#define AM_POINT_VALUE2 22
#define AM_POINT_VALUE3 23
#define AM_POINT_VALUE4 24
#define AM_POINT_VALUE5 25
#define AM_POINT_VALUE6 26
#define AM_POINT_VALUE7 27
#define AM_POINT_VALUE8 28
#define AM_POINT_VALUE9 29
#define AM_POINT_ID1 31
#define AM_POINT_ID2 32
#define AM_POINT_ID3 33
#define AM_POINT_ID4 34
```

```
#define AM_POINT_ID5 35
#define AM_POINT_ID6 36
#define AM_POINT_ID7 37
#define AM_POINT_ID8 38
#define AM_POINT_ID9 39
#define AM_COR_U8 40
#define AM_COR_I8 41
#define AM_STR_SHORT 42
#define AM_EXP_MSG_LEN      EXPANDED_ALARM_MSG_LEN  //( ALARM_MSG_LEN ) /*
 from sc_recs.h */

#define AM_MAX_FIELDS   6 /* warning, if this number is changed, so must
 */
/* the case statement in procedure am_msg_expand */
// optional fields that can be enabled thru configuration now
#define AM_POINT_VALUE_FIELD _T("point_val")
#define AM_USER_BITS_FIELD _T("user_bits")
#define AM_POINT_VALUE1_FIELD _T("point_val_1")
#define AM_POINT_ID1_FIELD _T("point_id_1")
#define AM_POINT_VALUE2_FIELD _T("point_val_2")
#define AM_POINT_ID2_FIELD _T("point_id_2")
#define AM_POINT_VALUE3_FIELD _T("point_val_3")
#define AM_POINT_ID3_FIELD _T("point_id_3")
#define AM_POINT_VALUE4_FIELD _T("point_val_4")
#define AM_POINT_ID4_FIELD _T("point_id_4")
#define AM_POINT_VALUE5_FIELD _T("point_val_5")
#define AM_POINT_ID5_FIELD _T("point_id_5")
#define AM_POINT_VALUE6_FIELD _T("point_val_6")
#define AM_POINT_ID6_FIELD _T("point_id_6")
#define AM_POINT_VALUE7_FIELD _T("point_val_7")
#define AM_POINT_ID7_FIELD _T("point_id_7")
#define AM_POINT_VALUE8_FIELD _T("point_val_8")
#define AM_POINT_ID8_FIELD _T("point_id_8")
#define AM_POINT_VALUE9_FIELD _T("point_val_9")
#define AM_POINT_ID9_FIELD _T("point_id_9")
// Defines for the OPC A&E Server, do not use 0
#define AM_OPC_TYPE_CONDITION 1
#define AM_OPC_TYPE_SIMPLE    2
#define AM_OPC_TYPE_TRAKING   3

#define AM_STATE_LEN    3
#define AM_TIME_LEN    22
/*********************************************************/
/* Typedefs neccessary to define unformatted message fields */
/*********************************************************/
#define SIZEOF_AM_FIELD (AM_MAX_STR+1)

typedef union am_field
{
   COR_I4 cori4_val;
   COR_I2 cori2_val;
   COR_I1 cori1_val;
   COR_I8 cori8_val;
```

```
   COR_U4 coru4_val;
   COR_U2 coru2_val;
   COR_U1 coru1_val;
   COR_U8 coru8_val;
   COR_BOOLEAN corbool_val;
   int ival;
   float fval;
   __int64 val64;
   unsigned __int64 valu64;
   TCHAR chval;
   TCHAR str[AM_MAX_STR+1];
   /* This structure must be aligned because it is contained
    * in the AM_GEN_SEG message segment, which may be passed
    * in a hybrid environment. */
   COR_I1 _pad[DO_ALIGN(SIZEOF_AM_FIELD,8)];
} AM_FIELD;

typedef struct am_msg_field
{
   AM_FIELD_TYPE ftype;
   AM_FIELD field;
} AM_MSG_FIELD;

typedef struct am_field_parms
{
   int fld_sz ;
   void *pfld_data ;
   unsigned char fld_type ;
} AM_FIELD_PARMS ;
typedef struct alarm_ca_alarmoper_req
{
   TCHAR performuserid[USER_ID_LEN + 1 ];
   TCHAR performpassword[PASSWORD_LEN+1];
   TCHAR performcomment[CHANGEAPPROVAL_COMMENT_LEN + 1];
   TCHAR verifyuserid[USER_ID_LEN + 1];
   TCHAR verifypassword[PASSWORD_LEN+1];
   TCHAR verifycomment[CHANGEAPPROVAL_COMMENT_LEN + 1];
   TCHAR location[MAX_COMPUTERNAME_LENGTH + 1];
} ALARM_CA_ALARM_OPER_REQ;
/************************************/
/* Max length of reference id string */
/************************************/
//REF_ID_LEN - AM_REF_ID_LEN must be at least as large as REF_ID_LEN!!!
#define AM_REF_ID_LEN REF_ID_LEN
#define AM_REF_ID_LEN32 32 //for compat. with older mf items / systems
#define AM_NULL_REF_ID -1
#define AM_NULL_ID -1
/*********************************/
/* Length of buffer in am_log    */
/*********************************/
#define AM_LOG_BUF_SIZE 156

// Lower 4 bits in log_file of ALARM_DEF
```

```
#define AM_LOG_ALARM_FILE     0
#define AM_LOG_ALTERNATE_FILE 1
#define AM_LOG_BOTH_FILES     2
#define LOBITS(n)             (n & 0x0F)
#define HIBITS(n)             ((n & 0xF0) >> 4)

/* The character that begins a site-wide setup name. */
#define AM_SITE_WIDE_CHAR  _T('$')
/* These help us create aligned messages. */
#define SETUP_ID_MFLEN DO_ALIGN(SETUP_ID_LEN+1,4)
#define USER_ID_MFLEN  DO_ALIGN(USER_ID_LEN+1,4)
/* The name that is displayed when the user is using an ad hoc
 * setup.  (This must be SETUP_ID_LEN characters or less).
 *                         0123456789ABCDEF
 */
#define AMAP_ADHOC_SETUP _T("<AD HOC SETUP>")
#define AM_HELPTEXT_LEN 72
#define AM_HELPTEXT_NUM_LINES 59
// Flags for the AM_GEN_LIST_EX2 struct alarmRequestFlags field
#define ALM_REQ_DELETED_ALARMS 0x01
#define ALM_REQ_OPC_ALARMS     0x02
/*****************************************************************/

typedef struct
{
   int              seg_num ;
   AM_STATE_TYPE    action ;
   COR_I4           seq_num ;
   TCHAR            *alarm_id;
   TCHAR            *fr_id;
   TCHAR            *ref_id;
   TCHAR            *user_or_serv_id;
   /* These items only in update_ex_seg, AM_VERS_1 */
   COR_STAMP        upd_time ;
   COR_I4           version ;
   char *conc_alarm_id;
   TCHAR *perform_user_id;
   TCHAR *perform_user_password;
   TCHAR *perform_user_comments;
   TCHAR *verify_user_id;
   TCHAR *verify_user_password;
   TCHAR *verify_user_comments;
} AMU_UPDATE_INFO ;
/* segment/feature Versioning information */
/* initiated with creation of am_list_ex, am_upd_inf_ex, am_gen_list_ex */
#define AM_VERS_0   0
#define AM_VERS_1   1
#define AM_VERS_2               2
#define AM_VERS_3   3 /*lengthened, perm., comments */
#define AM_VERS_4   4 /* added alarm severity to the AM_STACKED_INFO
 structure */
#define AM_VERS_5   5 /* added alarm blinking */
#define AM_VERS_6   6 /* lengthened alarm IDs */
```

```
#define AM_VERS_6_1 7 // added support for OPC
#define AM_VERS_6_1_1 8 // added support for unicode messages
#define AM_VERS_6_1_3 9 // added OPC CV
#define AM_VERS_6_1_4 10 // COR_STAMP
#define AM_VERS_6_1_4_1 11 // fix COR_STAMP reset time usage
#define AM_VERS_6_1_5_1  12    //SCR23024
#define AM_VERS_6_3   13 // OPC changes for events
#define AM_VERS_7_0   14 // ackStamp and deletedStamp
#define AM_VERS_8_1 15 // some alarm parameters split to provide individual
 parameter for each alarm level
#define AM_VERS_8_2 16 //Change approval support for alarms.
#define AM_VERS_9_0 17 //Long name support for alarms
#define AM_VERS_10_0 18 // Shelving support for alarms
#define AM_VERS_10_99 19 // description in alarm data
#define AM_VERS_OFFSET 1000 // AM version 6_0 sends back the client version
 in AM_LIST_EX2
// so if the version is > AM_VERS_OFFSET subtract the offset and you get
 the correct version
/* If you add a new version, bump the current version, as all code
** using this should be compatible
** with any earlier versions (provided fields are NOT removed.)
*/
#define AM_VERS_CURRENT   AM_VERS_10_99
// Prototypes
void am_alloc_init(void);
void am_alloc_gen_list(TCHAR (*alloc_addr)[30],TCHAR (*cm_addr)[30],TCHAR
 dyn_disp_mode,int *total_alarms, COR_BOOLEAN comments_wanted, int version,
 AM_LOCALEID langId, WCHAR *language, COR_BOOLEAN allLanguages, COR_U2
 alarmRequestFlags, struct cor_status *ret_stat);
void am_alloc_snd_list(struct datagram_address *sender,struct
 ipc_time_stamp *stamp,TCHAR rr_flag,TCHAR (*alloc_addr)[30],int
 max_to_send,struct cor_status *ret_stat);
void am_alloc_creseg(int size,int seg_type,int rep_count,TCHAR
 need_all,TCHAR **seg_ptr,int *ret_count,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr_flag,struct cor_status
 *ret_stat);
void am_alloc_rem_proc(TCHAR (*alloc_addr)[30],struct cor_status
 *ret_stat);
void am_alloc_add_proc(TCHAR (*alloc_addr)[30],TCHAR (*cm_addr)[30],struct
 ipc_time_stamp *stamp,struct cor_status *ret_stat);
void am_alloc_rem_all_ap_alarms(long *alloc_proc_ptr);
void am_alloc_unexp_term(struct datagram_address *sender);
void am_allocq_add(TCHAR *msg_ptr,struct datagram_address *sender,struct
 ipc_time_stamp *stamp,TCHAR rr_flag, int ipc_version);
void am_allocq_process(void);
void am_allocq_rem_req(long *alloc_req_ptr);
void am_allocq_add_resp(long *alloc_req_ptr,struct cor_status
 *ret_stat,TCHAR total_valid,int total_alarms);
void am_allocq_rem_all_resp(long *alloc_req_ptr);
void am_allocq_send_resp(long *alloc_req_ptr,int msg_status);
void am_allocq_send_slave(TCHAR *body,int segment);
long *am_ap_add_alarm(long *alloc_proc_ptr,long *alarm_occur_ptr,int
 insert_location,long *ap_pos_ptr);
```

```
void am_ap_rem_alarm(long *ap_alarm_ptr);
void am_auto_init(void);
void am_auto_start_timer(void);
void am_auto(void);
void am_auto_add_clr(long *alarm_def_ptr,long *alarm_occur_ptr, int
 which_level);
void am_auto_add_ack(long *alarm_def_ptr,long *alarm_occur_ptr, int
 which_level);
void am_auto_add_repeat(long *alarm_def_ptr,long *alarm_occur_ptr, int
 which_level);
void am_auto_rem_ack(long *alarm_occur_ptr, int which_level);
void am_auto_rem_clr(long *alarm_occur_ptr, int which_level);
void am_auto_rem_repeat(long *alarm_occur_ptr, int which_level);
void am_auto_gen_explist(void);
void am_auto_process_explist(void);
void am_auto_slave_init(void);
void am_auto_slave_flush(void);
void am_auto_slave_upd_add(TCHAR *alarm_id,TCHAR *fr_id,TCHAR *ref_id,int
 action,int seq_num);
void am_build_alarm_list(long *alloc_proc_ptr,long *active_term_ptr,struct
 cor_status *ret_stat);
void am_build_by_fr(long *alloc_proc_ptr,long *active_term_ptr);
void am_build_by_class(long *alloc_proc_ptr,long *active_term_ptr);
void am_build_by_state(long *alloc_proc_ptr,long *active_term_ptr);
void am_build_by_time(long *alloc_proc_ptr,long *active_term_ptr);
COR_BOOLEAN am_check_ff(long *alloc_proc_ptr,long *alarm_occur_ptr,long
 *active_term_ptr, long *alarm_info);
COR_BOOLEAN am_check_fr(long *alloc_proc_ptr,long *alarm_occur_ptr,long
 *active_term_ptr, long *alarm_info);
COR_BOOLEAN am_check_class(long *alloc_proc_ptr,long *alarm_def_ptr, long
 *alarm_info);
COR_BOOLEAN am_check_role(long *role_ptr,long *alarm_def_ptr);
COR_BOOLEAN am_check_state(struct allocated_process * alloc_proc_ptr,
                           struct alarm_info * alarm_info_ptr, //pointer to
 DMS alarm info record
                           struct alarm_occurrence * alarm_occurrence_ptr);
COR_BOOLEAN am_check_time(long * alloc_proc_ptr, long * alarm_info_ptr);
COR_BOOLEAN am_should_severity_override_filter(struct alarm_info
 *alarm_info_ptr);
long * am_create_alarm_info();
void am_set_alarm_info_severity(long* alarm_info_ptr, COR_I2 severity);
void am_cmd_init(void);
void am_cmd_get(int *cmd,TCHAR **msg_ptr,struct datagram_address
 *sender,struct ipc_time_stamp **stamp,TCHAR *rr_flag, int *ipc_version);
void am_cmd_get_dg(int *cmd,TCHAR **msg_ptr,struct datagram_address
 *sender,struct ipc_time_stamp **stamp,TCHAR *rr_flag, int *ipc_version,int
 curr_state,struct cor_status *ret_stat);
void am_cmd_get_ur(int *cmd,TCHAR **msg_ptr,struct cor_status *ret_stat);
void am_cmd_get_master(int *cmd,TCHAR **msg_ptr,DADDR *psender,struct
 cor_status *ret_stat);
void am_ipc_link_term(struct cor_status *ret_stat,int link_index);
void am_comm_init(void);
void am_comm_term(void);
```

```
void am_comment_alarm(const TCHAR *alarm_id,const TCHAR *fr_id,const
 TCHAR *ref_id,const TCHAR *user_or_serv_id,int action,const TCHAR
 *alarm_comment,struct cor_time_stamp *gentime,struct cor_status *ret_stat,
 COR_BOOLEAN skip_log);
void am_comment_add(long *alarm_occur_ptr,long *fr_ptr,long
 *alarm_fr_ptr,long *ref_ptr,const TCHAR *alarm_id,const TCHAR
 *fr_id,const TCHAR *ref_id,const TCHAR *user_or_serv_id,const TCHAR
 *alarm_comment,struct cor_time_stamp *gentime,struct cor_status *ret_stat,
 long *alarm_def_ptr);
void am_conc_proc_alarm_gen(struct testContext* pContext, struct AlarmInfo*
 pAlarmInfo);
void am_conc_proc_alarm_almmoddel(struct testContext *pContext, TCHAR
 *pAlmInf);
void am_config_terminate(void);
void am_config_init(void);
void am_config_get_type_info(TCHAR *sc_type_ptr,struct cor_status
 *ret_stat);
void am_config_type_info(void);
void am_config_class(void);
void am_config_role(void);
void am_config_fr_info(void);
COR_BOOLEAN am_config_get_info(TCHAR *sc_def_ptr,struct cor_status
 *ret_stat, LPCTSTR alarmId, int alarm_num,TCHAR dynamic_cfg, LPCTSTR
 roleAlarm);
COR_BOOLEAN am_config_get_info_ex(TCHAR *sc_def_ptr,struct cor_status
 *ret_stat, struct cor_statusV1 *ret_v1stat,LPCTSTR alarmId, int
 alarm_num,TCHAR dynamic_cfg, LPCTSTR roleAlarm);
void am_config_create_alarm(LPCTSTR alarmId, LPCTSTR modelAlarm);
void am_config_alarm_info(struct cor_status *ret_stat);
void am_config_init_alarm_info(void);
void am_config_global_data(void);
void am_config_link_field(long *alarm_field_ptr,long *alarm_type_ptr);
void am_config_link_class(long *class_ptr);
void am_config_init_ref_info(void);
void am_config_ptm_ack(void);
int am_config_user_setups(void);
void am_daddr_to(TCHAR *dest,struct datagram_address *src);
void am_daddr_from(struct datagram_address *dest,TCHAR *src);
void am_dd_rem_alloc_proc(long *alloc_proc_ptr);
void am_dd_info_upd(long *alarm_def_ptr,long *alarm_occur_ptr,long
 *alarm_info_ptr,int prev_state,int action);
void am_dd_info_upd_EX(RECORD_PTR alarm_def_ptr,
                       RECORD_PTR alarm_occur_ptr,
                       RECORD_PTR alarm_info_ptr,
                       AM_STATE_TYPE prev_state,
                       AM_STATE_TYPE action,
                       BOOL sendDeleted);
void am_dd_info_process(long *alarm_def_ptr,long *alloc_proc_ptr,long
 *active_term_ptr,long *alarm_occur_ptr,long *alarm_info_ptr,long
 *fr_ptr,int prev_state,int action);
void am_dd_info_upd_load(long *alloc_proc_ptr,long *alarm_def_ptr,long
 *alarm_occur_ptr,long *alarm_info_ptr,int prev_state,int action,long
 *fr_ptr,struct cor_status *ret_stat);
```

```
void am_dd_flush(void);
void am_debug_on(void);
void am_debug_off(void);
void am_driver_reset(void);
void am_driver_inc_gen(void);
void am_driver_inc_upd(void);
void am_driver_inc_log(int diff);
void am_driver_auto(void);
void am_dyncfg_proc(TCHAR *body_ptr,struct datagram_address *sender,struct
 ipc_time_stamp *stamp,TCHAR rr);
void am_dyncfg_delete_proc(TCHAR *body_ptr,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr);
void am_filter_mod_parms(long *alloc_proc_ptr,int primary_filter,TCHAR
 first_filter_msg,struct am_filter_parm *filter_parms,int
 num_filter_parms,struct cor_status *ret_stat);
void am_filter_mod_parms_ex(long *alloc_proc_ptr, int primary_filter, const
 AM_TIME_STATE_FILTER* ts_filter, TCHAR first_filter_msg, const unsigned
 char* field_filters, int num_filters, int total_filter_bytes, struct
 cor_status *ret_stat);
void am_filter_rem_frs(long *alloc_proc_ptr);
void am_filter_rem_ffs(long *alloc_proc_ptr);
void am_filter_rem_classes(long *alloc_proc_ptr);
void am_filter_set_default(long *alloc_proc_ptr);
void am_filter_default_states(long *alloc_proc_ptr);
void am_filter_default_ff(long *alloc_proc_ptr);
void am_filter_default_fr(long *alloc_proc_ptr);
void am_filter_default_classes(long *alloc_proc_ptr);
void am_find_master(TCHAR *am_ptm_resync);
//fld_fmt identifies:
//1==structured format for alarm msg parameters
//2==linear memory format for alarm msg parameters

void am_gen_alarm(int fld_fmt, const TCHAR *alarm_id,const TCHAR
 *fr_id,const TCHAR *ref_id,const TCHAR *user_or_serv_id,int seq_num,
                   void/*struct am_msg_field*/ *alarm_fields,TCHAR
 reset_follows,int num_fields,
                   COR_STAMP *gentime, long **palarm_occurrence, COR_U2
 alarm_gen_flags, struct cor_status *ret_stat);
void am_gen_new_alarm(int fld_fmt, long *alarm_def_ptr,long *fr_ptr,long
 *alarm_fr_ptr,long *ref_ptr,
                       long *service_ptr,void /*struct am_msg_field*/
 *alarm_fields,int num_fields,
                       const TCHAR *alarm_id,const TCHAR *fr_id,const TCHAR
 *ref_id,const TCHAR *user_or_serv_id,int seq_num,
                       COR_STAMP *gentime, long **palarm_occurrence, COR_U2
 genFlags, struct cor_status *ret_stat);
void am_gen_existing_alarm(int fld_fmt, long *alarm_def_ptr,long
 *alarm_occur_ptr,long *fr_ptr,long *alarm_fr_ptr,
                            long *ref_ptr,long *service_ptr,void /*struct
 am_msg_field*/ *alarm_fields,
                            TCHAR reset_follows,int num_fields,const TCHAR
 *alarm_id,const TCHAR *fr_id,const TCHAR *ref_id,
```

```
                            const TCHAR *user_or_serv_id,int seq_num,
 COR_STAMP *gentime, COR_U2 genFlags,
                            struct cor_status *ret_stat);
void am_gen_update_alarm(int fld_fmt, long *alarm_def_ptr,long
 *alarm_occur_ptr,long *fr_ptr,long *alarm_fr_ptr,
                           long *ref_ptr,long *service_ptr,void /*struct
 am_msg_field*/ *alarm_fields,
                           TCHAR reset_follows,int num_fields,const TCHAR
 *alarm_id,const TCHAR *fr_id,const TCHAR *ref_id,
                           const TCHAR *user_or_serv_id,int seq_num,
 COR_STAMP *gentime,
                           struct cor_status *ret_stat);
void am_get_alarms(struct cor_status *ret_stat,int link_index);
void am_init(void);
void am_intproc_upd(long *fr_ptr,long *alarm_def_ptr,long
 *alarm_info_ptr,int prev_state,int action,const TCHAR *alarm_id,const
 TCHAR *fr_id,const TCHAR *ref_id,const TCHAR *service_sought,COR_I4
 seq_num,BOOL blocked);
void am_jrnl_init(void);
int am_jrnl_open(int file,struct cor_status *ret_stat);
void am_jrnl_write_delta(TCHAR *msg_ptr,int msglen);
int am_jrnl_timer(int action,struct cor_status *ret_stat);
int am_jrnl_rollover(struct cor_status *ret_stat);
void am_jrnl_check_rollover(void);
int am_jrnl_write_dump(struct cor_status *ret_stat);
int am_jrnl_read_delta(struct cor_status *ret_stat);
int am_jrnl_read_dump(struct cor_status *ret_stat);
int am_jrnl_reset_delta(struct cor_status *ret_stat);
int am_jrnl_rename_dump(struct cor_status *ret_stat);
int am_jrnl_recover(int cond,struct cor_status *ret_stat);
void am_jrnl_error(long code,TCHAR *proc_id,struct cor_status *ret_stat);
void am_jrnl_recov_err(int cond,struct cor_status *ret_stat);
void am_jrnl_terminate(void);
void am_log_upd(long *alarm_def_ptr,long *alarm_info_ptr,int prev_state,int
 action,const TCHAR *alarm_id,
                const TCHAR *fr_id,const TCHAR *ref_id,const TCHAR *log_id,
 void /*struct am_msg_field*/ *alarm_fields, int num_fields, COR_STAMP
 *stamp );
void am_log_ca_serv_upd(long *alarm_def_ptr,long *alarm_info_ptr,int
 action,int actStatus,const TCHAR *alarm_id,const TCHAR *user_or_serv_id,
                        TCHAR *location);
int am_log_check(long *alarm_def_ptr,int log_on_action, COR_I2
 alarm_level);
void am_log_init(void);
void am_log_flush(void);
void am_log_terminate(void);
void am_log_load_data_field(void);
void am_master_init(void);
void am_master_slave_add(TCHAR *msg_ptr,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr_flag);
void am_master_slave_rem(long *slave_ptr);
void am_master_upd_slaves(TCHAR *msg_ptr,int msg_len);
```

```
void am_master_upd_slaves_wsender(TCHAR *msg_ptr,int msg_len,DADDR
 *psender, COR_STATUS *pret_stat);
void am_master_transition(TCHAR *msg_ptr,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr_flag);
void am_msg_expand(long *alarm_def_ptr,long *alarm_info_ptr,struct
 am_msg_field *alarm_fields,int num_fields,TCHAR *category,TCHAR
 *condition,TCHAR *subcondition, TCHAR *opcCv, struct cor_status
 *ret_stat);
void am_msg_gen2_expand(long *alarm_def_ptr,long *alarm_info_ptr, void /
*struct am_msg_field*/ *alarm_fields,int num_fields,TCHAR *category,TCHAR
 *condition,TCHAR *subcondition, TCHAR *opcCv, struct cor_status
 *ret_stat);
int am_msg_cvt(TCHAR *msg,int endian,int fp);
void am_new_master(struct ipc_dad_lst *list_proc_ptr,int cnt,struct
 cor_status *ret_stat);
void am_occur_find(const TCHAR *alarm_id,const TCHAR *fr_id,const
 TCHAR *ref_id,const TCHAR *user_or_serv_id,long **alarm_occur_ptr,long
 **alarm_def_ptr,long **fr_ptr,long **alarm_fr_ptr,long **ref_ptr,long
 **service_ptr,struct cor_status *ret_stat);
void am_occur_rem(long *alarm_def_ptr,long *alarm_occur_ptr,long
 *fr_ptr,long *alarm_fr_ptr,long *ref_ptr,long *service_ptr);
void am_ptm_inform(TCHAR resync);
void refresh_alarm_info_to_ptm();
void am_ptm_ack_info(const TCHAR *user_or_serv_id,const TCHAR *ref_id,int
 seq_num);
void am_ptm_ack_info_flush(void);
void am_ptm_alm_info(RECORD_PTR fr_ptr,RECORD_PTR alarm_def_ptr,RECORD_PTR
 alarm_info_ptr,AM_STATE_TYPE prev_state,AM_STATE_TYPE action,const TCHAR
 alarm_id[ALARM_ID_LEN+1],const TCHAR fr_id[FR_ID_LEN+1],const TCHAR
 ref_id[AM_REF_ID_LEN+1],COR_I4 seq_num);
int am_ptm_process_que(struct datagram_address *sender);
void am_purge_init(void);
int am_purge_or_delete_alarms(const TCHAR *user_or_serv_id,int action,
 struct cor_status *ret_stat);
void am_report_error_init(void);
void am_report_error(const TCHAR proc_name[],COR_STATUS *ret_stat,int
 severity);
void am_report_error_wparam(const TCHAR proc_name[], COR_STATUS
 *ret_stat,int severity,const TCHAR param[],const TCHAR param2[]);
void am_report_error_wparam_ex(const TCHAR proc_name[], COR_STATUS
 *ret_stat,COR_STATUSV1 *ret_v1stat,int severity,const TCHAR param[],const
 TCHAR param2[]);
void am_sched_init(struct am_sched_data *am_sched_array);
void am_sched_next_action(struct am_sched_data *am_sched_array,int
 *next_action);
void am_sched_sort_sched(struct am_sched_data *am_sched_array);
int am_seq_set(TCHAR *msg_ptr,struct cor_status *ret_stat);
int am_setup_exists(TCHAR *user_id,TCHAR *setup_id,struct cor_status
 *ret_stat);
void am_setup_makegroupkey(TCHAR *user_id,TCHAR *setup_id,TCHAR
 *key_value);
int am_setup_delete(TCHAR *user_id,TCHAR *setup_id,struct cor_status
 *ret_stat);
```

```
int am_setup_delete_FF(TCHAR *user_id,TCHAR *setup_id,struct cor_status
 *ret_stat);
int am_setup_delete_helper(int record_id,int key_id,TCHAR *key_value,struct
 cor_status *ret_stat);
int am_setup_do_save(long *alloc_proc_ptr,TCHAR *user_id,TCHAR
 *setup_id,COR_BOOLEAN deleteFF,struct cor_status *ret_stat);
void am_setup_invalidate_list(void);
int am_setup_set_default(TCHAR *user_id,TCHAR *setup_id,struct cor_status
 *ret_stat);
void am_setup_clear(TCHAR *alloc_addr,long *alloc_req_ptr,struct cor_status
 *ret_stat);
int am_setup_count(long *alloc_proc_ptr,TCHAR *user_id);
int am_setup_send_filter(TCHAR *alloc_addr,long *alloc_req_ptr,long
 *alloc_proc_ptr,TCHAR *setup_id);
int am_setup_send_filter_v10(TCHAR *alloc_addr,long *alloc_req_ptr,long
 *alloc_proc_ptr,TCHAR *setup_id);
int am_setup_send_filter_current(TCHAR *alloc_addr,long *alloc_req_ptr,long
 *alloc_proc_ptr,TCHAR *setup_id);
void am_setup_send_filter_add_n_send_v10(struct datagram_address
 *sender,struct ipc_time_stamp *stamp,long *alloc_proc_ptr,TCHAR
 **seg_ptr,int *count,int *curr_repeat,int *repeats,struct am_filter_parm
 *filter_parm,const TCHAR* setup_id,struct cor_status *ret_stat);
void am_setup_send_filter_add_n_send_current(struct datagram_address
 *sender,struct ipc_time_stamp *stamp, COR_U2 alarm_state_mask,
 COR_STAMP* alarm_timestamp, long *alloc_proc_ptr,TCHAR **seg_ptr,int
 *count_repeat_bytes,int *curr_repeat_byte,int *curr_filter_count,int
 *repeats, AM_FIELD_FILTER* filed_filter,const TCHAR *filter_string, const
 TCHAR *setup_id,struct cor_status *ret_stat);
int am_setup_list(TCHAR *alloc_addr,long *alloc_req_ptr,TCHAR
 *user_id,struct cor_status *ret_stat);
int am_setup_load(TCHAR *alloc_addr,long *alloc_req_ptr,TCHAR
 *user_id,TCHAR *setup_id,struct cor_status *ret_stat);
int am_setup_save(TCHAR *alloc_addr,TCHAR *user_id,TCHAR *setup_id,TCHAR
 first_filter_msg,TCHAR last_filter_msg,int primary_filter,struct
 am_filter_parm *filter_parms,int num_filter_parms,struct cor_status
 *ret_stat);
int am_setup_save_ex(TCHAR *alloc_addr, TCHAR *user_id, TCHAR *setup_id,
 TCHAR first_filter_msg, TCHAR last_filter_msg, int primary_filter, const
 AM_TIME_STATE_FILTER* ts_filter, const unsigned char *field_filters, int
 num_filters, int total_filter_bytes, struct cor_status *ret_stat);
void am_shutdown(void);
void am_slave_snd_not_master(TCHAR *msg_ptr,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr_flag);
void am_snd_alarm_info(int link_id,struct cor_status *ret_stat);
void am_snd_alarm_info_creseg(int size,int seg_type,int rep_count,TCHAR
 need_all,TCHAR **seg_ptr,int *ret_cnt,int link_id,struct cor_status
 *ret_stat);
void am_snd_segerr_stat(TCHAR *msg_ptr,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr_flag);
void am_snd_stat(struct cor_status *stat,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr_flag);
void am_snd_stat(struct cor_status *stat,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR rr_flag);
```

```
void am_snd_stat_ex(struct cor_status *stat,struct datagram_address
 *sender,struct ipc_time_stamp *stamp,TCHAR snd_as_rr, TCHAR resp_2_rr);
void am_term_init(void);
void am_term_process_ur(TCHAR *msg_ptr);
void am_term_add(TCHAR *msg_ptr,TCHAR *seg_ptr, int seg_type);
void am_term_rem(struct datagram_address *cm_addr);
void am_term_info_upd(long *alarm_def_ptr,long *alarm_occur_ptr,long
 *alarm_info_ptr,int prev_state);
void am_term_info_process(long *alarm_def_ptr,long *role_ptr,long
 *active_term_ptr,long *alarm_info_ptr,int prev_state);
void am_term_info_prc_actions(long *active_term_ptr,long
 *alarm_info_ptr,long *role_ptr,int *action_set);
void am_term_info_summary(long *active_term_ptr,long *role_ptr,TCHAR
 need_count);
void am_upd_alarm(const TCHAR *alarm_id,const TCHAR *fr_id,const TCHAR
 *ref_id,const TCHAR *user_or_serv_id,int action,
                  int seq_num, COR_STAMP *upd_time, struct cor_status
 *ret_stat,COR_STAMP *unshelve_time,TCHAR *comment, void *objCaData);
void am_upd_ack(long *alarm_def_ptr,long *alarm_occur_ptr,long *fr_ptr,long
 *alarm_fr_ptr,long *ref_ptr,
                long *service_ptr,const TCHAR *alarm_id,const TCHAR
 *fr_id,const TCHAR *ref_id,const TCHAR *user_or_serv_id,
                COR_STAMP *upd_time, TCHAR *occur_deleted, void *objCaData,
 struct cor_status *ret_stat);
void am_upd_clr(long *alarm_def_ptr,long *alarm_occur_ptr,long *fr_ptr,long
 *alarm_fr_ptr,long *ref_ptr,
                long *service_ptr,const TCHAR *alarm_id,const TCHAR
 *fr_id,const TCHAR *ref_id,const TCHAR *user_or_serv_id,
                COR_STAMP *upd_time, TCHAR *occur_deleted, void *objCaData,
 struct cor_status *ret_stat);
void am_upd_repeat(long *alarm_def_ptr,long *alarm_occur_ptr,long
 *fr_ptr,long *alarm_fr_ptr,
                  long *ref_ptr,long *service_ptr,const TCHAR
 *alarm_id,const TCHAR *fr_id,const TCHAR *ref_id,
                  const TCHAR *user_or_serv_id, COR_STAMP *upd_time, TCHAR
 *occur_deleted,
                  struct cor_status *ret_stat);
void am_upd_del(long *alarm_def_ptr, long *alarm_occur_ptr, long *fr_ptr,
 long *alarm_fr_ptr,
                long *ref_ptr, long *service_ptr, const TCHAR *alarm_id,
 const TCHAR *fr_id, const TCHAR *ref_id,
                const TCHAR *user_or_serv_id, COR_STAMP *upd_time,  void
 *objCaData, struct cor_status *ret_stat);
void am_upd_ca_serv_ack(long *alarm_def_ptr,long *alarm_occur_ptr,long
 *fr_ptr,long *alarm_fr_ptr,long *ref_ptr,
                        long *service_ptr,const TCHAR *alarm_id,const TCHAR
 *fr_id,const TCHAR *ref_id,const TCHAR *user_or_serv_id,
                        COR_STAMP *upd_time, TCHAR *occur_deleted,TCHAR
 *location,struct cor_status *ret_stat);
void am_upd_ca_serv_clr(long *alarm_def_ptr,long *alarm_occur_ptr,long
 *fr_ptr,long *alarm_fr_ptr,long *ref_ptr,
                        long *service_ptr,const TCHAR *alarm_id,const TCHAR
 *fr_id,const TCHAR *ref_id,const TCHAR *user_or_serv_id,
```

```
                        COR_STAMP *upd_time, TCHAR *occur_deleted,TCHAR
 *location,struct cor_status *ret_stat);
void am_upd_ca_serv_del(long *alarm_def_ptr, long *alarm_occur_ptr, long
 *fr_ptr, long *alarm_fr_ptr,
                        long *ref_ptr, long *service_ptr, const TCHAR
 *alarm_id, const TCHAR *fr_id, const TCHAR *ref_id,
                        const TCHAR *user_or_serv_id, COR_STAMP
 *upd_time,TCHAR *location, struct cor_status *ret_stat);
void am_upd_active_terms(void);
void am_updq_add(TCHAR *msg_ptr,struct datagram_address *sender,struct
 ipc_time_stamp *stamp,TCHAR rr_flag,TCHAR jrnl_recover);
void am_updq_process(struct cor_time_stamp *gentime);
void am_updq_add_resp(long *upd_req_ptr,int key,struct cor_status
 *cor_stat,int seq_num);
void am_updq_rem_req(long *upd_req_ptr);
void am_updq_rem_all_resp(long *upd_req_ptr);
void am_updq_send_resp(long *upd_req_ptr);
void am_ur_caradd(struct cor_status *ret_stat);
void am_ur_findnew(void);
int amrp_state(void);
void amrp_setstate(int new_state);
void amrp_remove_master_logical(void);
long *am_find_extmgr(struct datagram_address *pdaddr, TCHAR proc_name[]) ;
COR_I4   am_conc_pass_update_extmgr(DADDR *extmgr_daddr, int seg_type,
 TCHAR *seg_ptr, COR_I4  seg_len, COR_I4 rpt_len, COR_STATUS *ret_stat) ;
void am_process_ext_asmgr_req(TCHAR *msg_ptr, DADDR *sender, IPC_stamp
 *stamp, COR_BOOLEAN was_rr_flag) ;
COR_I4 am_init_extmgr() ;
void am_dd_comment_upd ( long *alarm_occur_ptr, long *alarm_comm_ptr, TCHAR
 comment_action ) ;
void am_dd_reset_term (RECORD_PTR alloc_proc_ptr);
int am_operhlp_load(AM_DADDR alloc_addr, RECORD_PTR alloc_req_ptr, TCHAR
 *alarmId, COR_STATUS *ret_stat);
COR_BOOLEAN am_proc_dead_asmgr (DADDR * psender);
COR_I4 am_init_updq_mutex(COR_STATUS *ret_stat);
int am_operhlp_filename(const TCHAR alarm_id[ALARM_ID_LEN + 1],
                        TCHAR helpFile[HELP_FNAME_LEN+1],
                        COR_STATUS *ret_stat);
int am_operhlp_count_lines(const TCHAR *help_file, int *count, COR_STATUS
 *retstat);
//scr 23546
BOOL am_remove_operhlp_extension(const TCHAR* i_help_file, TCHAR*
 o_help_file_no_ext);
BOOL am_gen_set_block_state(RECORD_PTR alarm_def_ptr,const TCHAR* fr_id);
void am_alarm_dump(TCHAR *msg_ptr);
void am_unblock_get_group_ptr(RECORD_PTR alarm_def_ptr,const TCHAR
 *fr_id,RECORD_PTR* org_blk_alarm_ptr,RECORD_PTR* blk_group_ptr);
void am_config_alarm_blk(void);
COR_I4 am_comment_file_erase(const TCHAR *palarm_id, const TCHAR *pres_id,
 const TCHAR *pref_id, COR_STATUS *pretstat);
COR_I4 am_comment_file_add(const TCHAR *palarm_id, const TCHAR
 *palarm_comment, COR_STAMP *pgentime
```

```
                              , const TCHAR *pres_id, const TCHAR *pref_id,
 COR_STATUS *pretstat);
COR_I4 am_comment_file_fetch(const TCHAR *palarm_id, RECORD_PTR
 alarm_occur_ptr
                               , const TCHAR *pres_id, const TCHAR *pref_id,
 COR_STATUS *pretstat);
void am_remove_all_alarm_comments(RECORD_PTR alarm_occur_ptr) ;
void am_comment_delete (
   COR_STATUS    *pret_stat,
   const TCHAR          *alarm_id,
   const TCHAR          *fr_id,
   const TCHAR          *ref_id,
   const TCHAR          *user_or_serv_id,
   AM_STATE_TYPE action,
   AM_COMMENT2_INFO *pcomment_info,
   COR_STAMP     *gentime,
   RECORD_PTR     alarm_occur_ptr,
   RECORD_PTR     alarm_def_ptr
) ;
void am_inc_alarm_count(RECORD_PTR alarm_def_ptr,int count);
void am_inc_alarm_acked_count(RECORD_PTR alarm_def_ptr,int count);
void am_inc_alarm_reset_count(RECORD_PTR alarm_def_ptr,int count);
void am_process_req_alarm_list(TCHAR* msg_ptr,DADDR* sender,COR_BOOLEAN
 rr_flag);
void am_process_req_alarm_state(TCHAR* msg_ptr,DADDR* sender,COR_BOOLEAN
 rr_flag);
int am_proc_alm_setup(TCHAR *setUpId);
int am_proc_alm_user(TCHAR *UserId);

RECORD_PTR am_config_new_user_id(TCHAR *user_id);
int GetAlarmLevelIndex(COR_I2 alarm_level);
void am_alloc_alarm_state(TCHAR (*alloc_addr)[30], TCHAR (*cm_addr)[30],
 RECORD_PTR alloc_req_ptr, const TCHAR* alarm_id, const TCHAR* fr_id, const
 TCHAR* ref_id, COR_STATUS* retstat);
void AddFilterItem(RECORD_PTR alloc_proc_ptr, int fieldIndex, int
 matchFlag, TCHAR *filterString);

#endif // _AM_DEFS_H
```

amaru_proto.h

This file contains function prototypes for the AMARU library. The following is a listing of this file:

```
#ifndef _AMARU_PROTO_H
#define _AMARU_PROTO_H
#ifdef WIN32
#include <inc_path/ddl.h>
#include <inc_path/sc_recs.h>
#include <inc_path/am_defs.h>
#ifndef AmaruExport
#define AmaruExport
```

```
#endif
#ifdef __cplusplus
  extern "C" {
#endif
AmaruExport int amaru_add_update(
                          TCHAR *bodyptr, int bodylen, TCHAR first_seg,
                          const TCHAR *alarm_id, const TCHAR *fr_id,
                          const TCHAR *user_or_serv_id, const TCHAR *ref_id,
                          int action, int seq_num, int resp_type, int key,
                          struct cor_status *ret_stat );
AmaruExport int amaru_add_update_stamp(
                          TCHAR *bodyptr, int bodylen, TCHAR first_seg,
                          const TCHAR *alarm_id, const TCHAR *fr_id,
                          const TCHAR *user_or_serv_id, const TCHAR *ref_id,
                          int action, int seq_num, int resp_type, int key,
                          COR_STAMP stamp, struct cor_status *ret_stat );
AmaruExport int amaru_add_gen(
                          TCHAR *bodyptr, int bodylen, TCHAR first_seg,
                          const TCHAR *alarm_id, const TCHAR *fr_id,
                          const TCHAR *user_or_serv_id, const TCHAR *ref_id,
                          int resp_type, int key, struct am_msg_field
 *msg_field,
                          int num_fields, TCHAR reset_follows,
                          struct cor_status *ret_stat );
AmaruExport int amaru_add_gen_stamp(
                          TCHAR *bodyptr, int bodylen, TCHAR first_seg,
                          const TCHAR *alarm_id, const TCHAR *fr_id,
                          const TCHAR *user_or_serv_id, const TCHAR *ref_id,
                          int resp_type, int key, struct am_msg_field
 *msg_field,
                          int num_fields, TCHAR reset_follows,
                          COR_STAMP stamp, struct cor_status *ret_stat );
AmaruExport void amaru_alloc_buffer(struct IPC_datagram **buffer ,int
 *length,TCHAR **body_ptr );
AmaruExport void amaru_free_buffer(struct IPC_datagram *buffer);
AmaruExport int amaru_init(struct cor_status *ret_stat);
AmaruExport int amaru_purge(TCHAR *user_or_serv_id,int dg_port_index,struct
 IPC_datagram *send_buffer_ptr,TCHAR *send_body_ptr,int
 send_buffer_len,struct IPC_datagram *recv_buffer_ptr,TCHAR
 *recv_body_ptr,int recv_buffer_len,struct cor_status *ret_stat);
AmaruExport void amaru_get_resp(TCHAR *body_ptr,int i,struct cor_status
 *ret_stat);
AmaruExport int amaru_send_msg(int port_id,struct IPC_datagram
 *wrt_buf,struct IPC_datagram *read_buf,int read_buf_len,TCHAR
 *alarm_mgr_id,struct cor_status *ret_stat);
AmaruExport void amaru_terminate();
AmaruExport int amaru_num_messages(TCHAR *body_ptr,struct cor_status
 *ret_stat);
AmaruExport int amaru_client_init(const TCHAR *system, COR_STATUS
 *retstat);
AmaruExport COR_BOOLEAN amaru_is_initialized();
#define AMARU_STYLE_LOCAL_AM 0x01
#define AMARU_STYLE_ACTIVE_AM_IF_SERVER 0x02
```

```
AmaruExport void amaru_set_style(int style);
AmaruExport int amaru_send_comment (
                                    TCHAR           *bodyptr,    /* pointer
 to message body */
                                    int             bodylen,     /* length
 of message body */
                                    TCHAR           *alarm_id,
                                    TCHAR           *fr_id,
                                    TCHAR           *user_or_serv_id,
                                    TCHAR           *ref_id,
                                    TCHAR         *alarm_comment,
                                    COR_STATUS *ret_stat);
AmaruExport int amaru_setup_default (
                                TCHAR            *setup_id,
                                TCHAR            *user_id,
                                int            dg_port_index,
                                IPCDG  *send_buffer_ptr,
                                TCHAR  *send_body_ptr,
                                int  send_buffer_len,
                                IPCDG  *recv_buffer_ptr,
                                TCHAR  *recv_body_ptr,
                                int  recv_buffer_len,
                                COR_STATUS *ret_stat);
AmaruExport int amaru_setup_delete (
                                TCHAR            *setup_id,
                                TCHAR            *user_id,
                                int            dg_port_index,
                                IPCDG          *send_buffer_ptr,
                                TCHAR          *send_body_ptr,
                                int            send_buffer_len,
                                IPCDG          *recv_buffer_ptr,
                                TCHAR          *recv_body_ptr,
                                int            recv_buffer_len,
                                COR_STATUS     *ret_stat);
AmaruExport int amaru_setup_save_start (
                                     TCHAR            *setup_id,
                                     TCHAR            *user_id,
                                     int  num_rpts,
                                     AM_FILTER_TYPE primary_filter,
                                     int  dg_port_index,
                                     IPCDG  *send_buffer_ptr,
                                     TCHAR  *send_body_ptr,
                                     int  send_buffer_len,
                                     IPCDG  *recv_buffer_ptr,
                                     TCHAR  *recv_body_ptr,
                                     int  recv_buffer_len,
                                     COR_STATUS *ret_stat);
AmaruExport int amaru_setup_save_repeat (
                                   AM_FILTER_TYPE  filter_type,
                                   COR_STAMP       *start_time,
                                   TCHAR           *class_id,
                                   AM_STATE_TYPE    alarm_state,
                                   TCHAR            *fr_id,
```

```
                                        COR_STATUS *ret_stat);
/* Private internal functions - For GE Intelligent Platform's use only. */
int amaru_tls_initialize_process();
int amaru_tls_terminate_process();
int amaru_tls_initialize_thread(RECORD_PTR data);
int amaru_tls_terminate_thread();
int amaru_rcm_init(COR_STATUS *retstat);
int amaru_rcm_init_root(RECORD_PTR amaru_root, COR_STATUS *retstat);
RECORD_PTR amaru_tls_get();
void amaru_force_terminate();
void amaru_force_terminate_root(RECORD_PTR amaru_root);
AmaruExport int amaru_client_init_root(const TCHAR *sysname,
                                                    RECORD_PTR *amaru_root,
 COR_STATUS *retstat);
AmaruExport int amaru_purge_root (
TCHAR  user_or_serv_id [COR_MAX(USER_ID_LEN, SERVICE_ID_LEN) + 1],
int  dg_port_index,
IPCDG  *send_buffer_ptr,
TCHAR  *send_body_ptr,
int  send_buffer_len,
IPCDG  *recv_buffer_ptr,
TCHAR  *recv_body_ptr,
int  recv_buffer_len,
RECORD_PTR amaru_root,
COR_STATUS *ret_stat);
AmaruExport int amaru_setup_default_root (
TCHAR            *setup_id,
TCHAR            *user_id,
int  dg_port_index,
IPCDG  *send_buffer_ptr,
TCHAR  *send_body_ptr,
int  send_buffer_len,
IPCDG  *recv_buffer_ptr,
TCHAR  *recv_body_ptr,
int  recv_buffer_len,
RECORD_PTR amaru_root,
COR_STATUS *ret_stat);
AmaruExport int amaru_setup_delete_root (
TCHAR            *setup_id,
TCHAR            *user_id,
int  dg_port_index,
IPCDG  *send_buffer_ptr,
TCHAR  *send_body_ptr,
int  send_buffer_len,
IPCDG  *recv_buffer_ptr,
TCHAR  *recv_body_ptr,
int  recv_buffer_len,
RECORD_PTR amaru_root,
COR_STATUS *ret_stat);
AmaruExport int amaru_setup_save_start_root (
TCHAR            *setup_id,
TCHAR            *user_id,
int  num_rpts,
```

```
AM_FILTER_TYPE primary_filter,
int  dg_port_index,
IPCDG  *send_buffer_ptr,
TCHAR  *send_body_ptr,
int  send_buffer_len,
IPCDG  *recv_buffer_ptr,
TCHAR  *recv_body_ptr,
int  recv_buffer_len,
RECORD_PTR amaru_root,
COR_STATUS *ret_stat);
AmaruExport int amaru_setup_save_repeat_root (
AM_FILTER_TYPE  filter_type,
COR_STAMP       *start_time,
TCHAR            *class_id,
AM_STATE_TYPE   alarm_state,
TCHAR            *fr_id,
RECORD_PTR amaru_root,
COR_STATUS *ret_stat);
AmaruExport int amaru_add_gen_root (
TCHAR         *bodyptr,          /* pointer to message body */
int           bodylen,           /* length of message body */
COR_BOOLEAN   first_seg,         /* first segment ?  */
const TCHAR   *alarm_id,         /* [ALARM_ID_LEN+1] */
const TCHAR   *fr_id,            /* [FR_ID_LEN+1] */
const TCHAR   *user_or_serv_id,  /* [COR_MAX(USER_ID_LEN,
 SERVICE_ID_LEN)+1] */
const TCHAR   *ref_id,           /* [AM_REF_ID_LEN+1] */
AM_RESP_TYPE  resp_type,
AM_RESP_KEY   key,
AM_MSG_FIELD  msg_field[],
int           num_fields,
COR_BOOLEAN   reset_follows,
RECORD_PTR    amaru_root,
COR_STATUS    *ret_stat);
AmaruExport int amaru_add_gen_stamp_root (
TCHAR         *bodyptr,          /* pointer to message body */
int           bodylen,           /* length of message body */
COR_BOOLEAN   first_seg,         /* first segment ?  */
const TCHAR   *alarm_id,         /* [ALARM_ID_LEN+1] */
const TCHAR   *fr_id,            /* [FR_ID_LEN+1] */
const TCHAR   *user_or_serv_id,  /* [COR_MAX(USER_ID_LEN,
 SERVICE_ID_LEN)+1] */
const TCHAR   *ref_id,           /* [AM_REF_ID_LEN+1] */
AM_RESP_TYPE  resp_type,
AM_RESP_KEY   key,
AM_MSG_FIELD  msg_field[],
int           num_fields,
COR_BOOLEAN   reset_follows,
COR_STAMP     gentime,
RECORD_PTR    amaru_root,
COR_STATUS    *ret_stat);
AmaruExport int amaru_add_update_root (
TCHAR        *bodyptr,                  /* pointer to message body */
```

```
int           bodylen,                   /* length of message body */
COR_BOOLEAN   first_seg,                  /* first segment ?  */
const TCHAR alarm_id[ALARM_ID_LEN+1],
const TCHAR fr_id[FR_ID_LEN+1],
const TCHAR user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1],
const TCHAR ref_id[AM_REF_ID_LEN+1],
AM_STATE_TYPE action,
COR_I4       seq_num,
AM_RESP_TYPE resp_type,
AM_RESP_KEY  key,
RECORD_PTR amaru_root,
COR_STATUS    *ret_stat);
AmaruExport int amaru_add_update_ex_root (
TCHAR        *bodyptr,                    /* pointer to message body */
int          bodylen,                     /* length of message body */
COR_BOOLEAN  first_seg,                   /* first segment ?  */
const TCHAR alarm_id[ALARM_ID_LEN+1],
const TCHAR fr_id[FR_ID_LEN+1],
const TCHAR user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1],
const TCHAR ref_id[AM_REF_ID_LEN+1],
AM_STATE_TYPE action,
COR_I4       seq_num,
AM_RESP_TYPE resp_type,
AM_RESP_KEY  key,
RECORD_PTR amaru_root,
const TCHAR *conc_alarm_id,
COR_STATUS   *ret_stat);
AmaruExport int amaru_pass_root (
TCHAR        *bodyptr,                   /* pointer to message body */
int          bodylen,                    /* length of message body */
COR_BOOLEAN  first_seg,                  /* first segment ?  */
COR_I4       seg_type ,
COR_I4       seg_len ,
TCHAR        *seg_ptr ,
AM_RESP_TYPE resp_type,
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat) ;
AmaruExport int amaru_add_update_stamp_root(
TCHAR        *bodyptr,                    /* pointer to message body */
int           bodylen,                    /* length of message body */
COR_BOOLEAN first_seg,                   /* first segment ?    */
const TCHAR alarm_id[ALARM_ID_LEN+1],
const TCHAR fr_id[FR_ID_LEN+1],
const TCHAR user_or_serv_id[COR_MAX(USER_ID_LEN, SERVICE_ID_LEN)+1],
const TCHAR ref_id[AM_REF_ID_LEN+1],
AM_STATE_TYPE action,
COR_I4       seq_num,
AM_RESP_TYPE resp_type,
AM_RESP_KEY  key,
COR_STAMP upd_time,
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat);
AmaruExport int amaru_send_msg_root(
```

```
int    port_id,     /* IPC port where message is sent */
IPCDG  *wrt_buf,    /* Buffer containing message to be sent */
IPCDG  *read_buf,   /* Buffer to contain message received   */
int    read_buf_len,      /* Maximum Length of read buffer  */
TCHAR  *alarm_mgr_id,     /* Identifier of Alarm Manager    */
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat);
AmaruExport int amaru_send_comment_root (
TCHAR        *bodyptr,                  /* pointer to message body */
int          bodylen,                   /* length of message body */
TCHAR        *alarm_id,
TCHAR        *fr_id,
TCHAR        *user_or_serv_id,
TCHAR        *ref_id,
TCHAR            *alarm_comment,
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat);
AmaruExport int amaru_send_comment_ex_root (
TCHAR        *bodyptr,                  /* pointer to message body */
int          bodylen,                   /* length of message body */
TCHAR        *alarm_id,
TCHAR        *conc_alarm_id,
TCHAR        *fr_id,
TCHAR        *user_or_serv_id,
TCHAR        *ref_id,
TCHAR            *alarm_comment,
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat);
AmaruExport int amaru_send_comment2_root (
TCHAR        *bodyptr,                  /* pointer to message body */
int          bodylen,                   /* length of message body */
TCHAR        action,                    /* 'A'dd, 'D'elete */
TCHAR        *alarm_id,
TCHAR        *conc_alarm_id,
TCHAR        *fr_id,
TCHAR        *user_or_serv_id,
TCHAR        *ref_id,
TCHAR            *alarm_comment,
const COR_STAMP       *pgentime,
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat);
AmaruExport int amaru_local_send_msg(
int    port_id,     /* IPC port where message is sent */
IPCDG  *wrt_buf,    /* Buffer containing message to be sent */
IPCDG  *read_buf,   /* Buffer to contain message received */
int    read_buf_len,      /* Maximum Length of read buffer */
const TCHAR   *alarm_mgr_id,      /* Identifier of Alarm Manager         */
COR_STATUS    *ret_stat) ;
AmaruExport int amaru_local_send_msg_root(
int    port_id,    /* IPC port where message is sent  */
IPCDG  *wrt_buf,   /* Buffer containing message to be sent */
IPCDG  *read_buf,  /* Buffer to contain message received   */
int    read_buf_len,/* Maximum Length of read buffer   */
```

```
const TCHAR *alarm_mgr_id,      /* Identifier of Alarm Manager          */
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat) ;
AmaruExport int amaru_local_pass_msg(
int    port_id,     /* IPC port where message is sent */
IPCDG  *wrt_buf,    /* Buffer containing message to be sent */
IPCDG  *read_buf,   /* Buffer to contain message received   */
int    read_buf_len,      /* Maximum Length of read buffer  */
const TCHAR  *alarm_mgr_id,      /* Identifier of Alarm Manager          */
COR_BOOLEAN     rr,
COR_STATUS   *ret_stat) ;
AmaruExport int amaru_local_pass_msg_root(
int    port_id,     /* IPC port where message is sent */
IPCDG  *wrt_buf,    /* Buffer containing message to be sent */
IPCDG  *read_buf,   /* Buffer to contain message received   */
int    read_buf_len,      /* Maximum Length of read buffer  */
const TCHAR  *alarm_mgr_id,      /* Identifier of Alarm Manager          */
COR_BOOLEAN     rr,
RECORD_PTR amaru_root,
COR_STATUS    *ret_stat) ;
AmaruExport void amaru_terminate_root(RECORD_PTR amaru_root);
AmaruExport int amaru_delete (
TCHAR        user_or_serv_id[],
int          dg_port_index,
IPCDG        *send_buffer_ptr,
TCHAR        *send_body_ptr,
int          send_buffer_len,
IPCDG        *recv_buffer_ptr,
TCHAR        *recv_body_ptr,
int          recv_buffer_len,
COR_BOOLEAN     local_send ,
struct cor_status *ret_stat) ;
AmaruExport int amaru_delete_root (
TCHAR        user_or_serv_id[],
int          dg_port_index,
IPCDG        *send_buffer_ptr,
TCHAR        *send_body_ptr,
int          send_buffer_len,
IPCDG        *recv_buffer_ptr,
TCHAR        *recv_body_ptr,
int          recv_buffer_len,
RECORD_PTR amaru_root,
COR_BOOLEAN     local_send ,
struct cor_status *ret_stat) ;
AmaruExport COR_I4 amu_request_external_state_management(
int     port_id ,   /* IPC port where message is sent */
IPCDG   *wrt_buf ,  /* Buffer containing message to be sent */
int     wbodylen ,
IPCDG   *read_buf , /* Buffer to contain message received */
int     read_buf_len , /* maximum length of read buffer */
const TCHAR   *alarm_mgr_id , /* Identifier of alarm manager */
COR_BOOLEAN enable_support ,
const TCHAR   *proc_name ,
```

```
const TCHAR   *port_name ,    /* IPC port name for comm to AMRP */
const TCHAR   *port_name2 ,    /* IPC port name to rcv updates */
const TCHAR   *user_or_srv_id , /* assoc this id with this req. channel */
COR_STATUS  *ret_stat ) ;
AmaruExport COR_I4 amu_request_external_state_management_root(
int     port_id ,   /* IPC port where message is sent */
IPCDG   *wrt_buf ,  /* Buffer containing message to be sent */
int     wbodylen ,
IPCDG   *read_buf , /* Buffer to contain message received */
int     read_buf_len , /* maximum length of read buffer */
const TCHAR   *alarm_mgr_id , /* Identifier of alarm manager */
COR_BOOLEAN enable_support ,
const TCHAR   *proc_name ,
const TCHAR   *port_name ,    /* IPC port name for comm to AMRP */
const TCHAR   *port_name2 ,    /* IPC port name to rcv updates */
const TCHAR   *user_or_srv_id , /* assoc this id with this req. channel */
RECORD_PTR amaru_root ,
COR_STATUS  *ret_stat );
AmaruExport COR_I4 amu_get_extmgr_upd(
AMU_UPDATE_INFO *pamu_update_info ,
TCHAR           *body_ptr ,
int             i ,
COR_STATUS      *ret_stat ) ;
AmaruExport int amaru_add_verify_gen_stamp(
                        TCHAR *bodyptr, int bodylen, TCHAR first_seg,
                        const TCHAR *alarm_id, const TCHAR *fr_id,
                        const TCHAR *user_or_serv_id, const TCHAR *ref_id,
                        int resp_type, int key, struct am_msg_field
 *msg_field,
                        int num_fields, COR_BOOLEAN havegentime, COR_STAMP
 genstamp,
                        int cur_state, COR_I4 cleared_time,
                        struct cor_status *ret_stat );
AmaruExport int amaru_add_verify_gen_stamp_root (
TCHAR         *bodyptr,          /* pointer to message body */
int           bodylen,           /* length of message body */
COR_BOOLEAN   first_seg,         /* first segment ?    */
const TCHAR   *alarm_id,         /* [ALARM_ID_LEN+1] */
const TCHAR   *fr_id,            /* [FR_ID_LEN+1] */
const TCHAR   *user_or_serv_id,  /* [COR_MAX(USER_ID_LEN,
 SERVICE_ID_LEN)+1] */
const TCHAR   *ref_id,           /* [AM_REF_ID_LEN+1] */
AM_RESP_TYPE  resp_type,
AM_RESP_KEY   key,
AM_MSG_FIELD  msg_field[],
int           num_fields,
COR_BOOLEAN   havegentime,
COR_STAMP     gentime,
int           cur_state,
COR_I4        cleared_time,
RECORD_PTR    amaru_root,
COR_STATUS    *ret_stat);
#define DECLARE_AMARU_ROOT RECORD_PTR amaru_root = amaru_tls_get()
```

```
#ifdef __cplusplus
  } /* C++ */
#endif
#ifdef __cplusplus
#include <inc_path/ipc.hpp>
  AmaruExport int amaru_remote_send_msg(CIPC *system,DADDR
 *daddr,COR_STATUS *retstat);
  AmaruExport int amaru_remote_send_msg(CIPC *ipc, const TCHAR *system,
 COR_STATUS *retstat);
AmaruExport int amaru_remote_send_msg_root(CIPC *ipc, DADDR *daddr,
 RECORD_PTR amaru_root,
                                     COR_STATUS *retstat);
AmaruExport int amaru_remote_send_msg_root(CIPC *ipc, const TCHAR *system,
 RECORD_PTR amaru_root,
                                     COR_STATUS *retstat);
inline
AmaruExport int amaru_add_comment2_root (
TCHAR        *bodyptr,                  /* pointer to message body */
int          bodylen,                   /* length of message body  */
TCHAR        *alarm_id,
TCHAR        *conc_alarm_id,
TCHAR        *fr_id,
TCHAR        *user_or_serv_id,
TCHAR        *ref_id,
TCHAR            *alarm_comment,
const COR_STAMP        *pgentime,
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat,
TCHAR        action=AM_COMMENT_ACT_ADD /* 'A'dd, 'D'elete */
)
{ return amaru_send_comment2_root(bodyptr, bodylen, action, alarm_id,
                          conc_alarm_id, fr_id, user_or_serv_id,
                          ref_id, alarm_comment, pgentime, amaru_root,
 ret_stat) ;
}
inline
AmaruExport int amaru_del_comment2_root (
TCHAR        *bodyptr,                  /* pointer to message body */
int          bodylen,                   /* length of message body */
TCHAR        *alarm_id,
TCHAR        *conc_alarm_id,
TCHAR        *fr_id,
TCHAR        *user_or_serv_id,
TCHAR        *ref_id,
TCHAR            *alarm_comment,
const COR_STAMP     *pgentime,
RECORD_PTR amaru_root,
COR_STATUS   *ret_stat,
TCHAR        action=AM_COMMENT_ACT_DELETE          /* 'A'dd, 'D'elete */
)
{ return amaru_send_comment2_root(bodyptr, bodylen, action, alarm_id,
                          conc_alarm_id, fr_id, user_or_serv_id,
```

```
                              ref_id, alarm_comment, pgentime, amaru_root,
 ret_stat) ;
}
#endif
#endif
#endif
```

Error Codes Returned by AMRP

This section lists the error codes returned by the Alarm Management Resident Process (AMRP). The error messages are located in **inc_path/am_errors.h** .

Number	Defined Constant	Description
0	AM_SUCCESS	Normal successful completion
1	AM_NONEXISTENT_ALARM_DEF	Alarm definition not configured
2	AM_NONEXISTENT_CLASS	Alarm class not configured
3	AM_NONEXISTENT_FR	Factory resource not configured
4	AM_ROUTE_NONEXISTENT_ROLE	Role not configured
5	AM_ALARM_DEF_IN_USE	Tried to remove alarm from cache still in use
6	AM_ILLEGAL_SEGMENT	Illegal segment type
7	AM_ALREADY_CLEARED	Alarm already reset
8	AM_ILLEGAL_CURRENT_STATE	Illegal current state
9	AM_ALREADY_ACKNOWLEDGED	Alarm already acknowledged
10	AM_ILLEGAL_ACTION	Illegal action in update
11	AM_SEQ_NUM_MISMATCH	Generation sequence numbers don't match
12	AM_NO_ALARM_OCCURRENCE	Alarm does not exist
13	AM_UR_NONEXISTENT_ROLE	Role does not exist for User registration
14	AM_ALREADY_LOGGED_IN	User already logged in
15	AM_NOT_LOGGED_IN	User not logged in
16	AM_UNEXPECTED_CASE	Unexpected case
17	AM_ILLEGAL_ALARM_TRANSITION	Illegal alarm state transition
18	AM_UNKNOWN_STATE	Unknown alarm state
19	AM_ILLEGAL_FILTER	Illegal filter
20	AM_TERMINAL_NOT_LOGGED_IN	Terminal not logged in
21	AM_NONEXISTENT_ALLOCATED_PROC	Allocated process does not exist

22	AM_FIELD_NUM_MISMATCH	Number of fields does not match alarm def
23	AM_UNKNOWN_REQUEST	Unknown request
24	AM_STATE_CMD_MISMATCH	AM command state mismatch
25	AM_UR_CARADD_ERR	Communication error with User registration
26	AM_UR_LL_FAILURE	Communication to User registration lost
27	AM_IPC_WRITE_PORT_ERR	ipc_write_port error
28	AM_IPC_XLATE_ERR	ipc_xlate error
29	AM_FIELD_TYPE_MISMATCH	Field type does not match alarm def
30	AM_MASTER_SLAVE_DEADLOCK	Active / Standby deadlock
31	AM_ALREADY_SLAVE	Process is already standby
32	AM_NOT_MASTER	Process is not master
33	AM_JRNL_SLAVES_EXIST	Standby processes exist
34	AM_JRNL_RECOV_ERR	Temporary dump file exists
35	AM_JRNL_UNKNOWN_ERR	Unknown state number for recover
36	AM_JRNL_NOT_DATAGRAM	Not the regular IPC-buffer is used
37	AM_NO_FRS	No resources associated with this alarm manager
39	AM_NO_DEFAULT_SETUP	No default setup
40	AM_NONEXISTENT_SETUP	Non-existent setup
41	AM_NO_HELP_AVAIL	No help file available
42	AM_HFILE_READ_ERR	Help file read error
43	AM_INVALID_PRIM_FILTER	The primary filter in the setup is invalid
44	AM_DUP_EXTMGR_ALARM	Another XASMgr has gen'd this alarm - Can't distinguish
45	AM_INV_EXTMGR_ACTION	Requested XASMgr action invalid

Error Codes Returned by AMARU

This section lists the error codes returned by the Alarm Management Resident Utilities (AMARU). The error messages are located in **inc_path/amaru_err.h** .

Number	Defined Constant	Description
1010	AMARU_UNKNOWN_ALARM_MGR	Unknown Alarm Manager

Chapter 4. Alarm Interested Process API

About the Alarm Interested Process API

You can use the Alarm Interested Process API (AMIP API) to create a process that receives alarm information from the CIMPLICITY Alarm Manager. You can then process the information as you wish. For example, you can write the information to a log file or send it to an output device.

Contents of the Alarm Interested Process API

The following is a list of all files distributed with the Alarm Interested Process API. The files are loaded into the directories indicated. The environment variable **%BSM_ROOT%** is the directory where the CIMPLICITY software was installed.

Include files in %BSM_ROOT%\api\include\inc_path are:

alarmapi.h	cor_thread.h
am_defs.h	cor_time.h
am_errors.h	ddl.h
amap_defs.h	examgr.h
amaru_err.h	ipc.hpp
amdd_cmd.h	ipcerr.h
amdd_cont.h	mf_defs.h
amdd_defs.h	mfamupdi.h
amip.h	mfpmterm.h
cor.h	mfstatus.h
cor_event.h	netcom.h
cor_mutex.h	rcm.h
cor_os.h	rtr_bcst.h
cor_stat.h	sc_recs.h

Source files in %BSM_ROOT%\api\amip are:

amip.cpp	makefile

Source files in %BSM_ROOT%\api\amipsample are:

makefile	amipsample.cpp	amipsample_exe.vcxproj

Source files in %BSM_ROOT%\api\lib are:

amaru.lib	fasrtl.lib
amip.lib	ipc.lib
corutil.lib	cim_mf.lib
ddl.lib	cim_sc.lib
examgr.lib	

Notes on Internationalization for the Alarm Interested Process API

- Work with strings
- Recommended reading

Work with strings

This API is written for the international environment. In an international environment, strings in CIMPLICITY software can be multi-byte strings. If you want your code to conform to international standards, It is recommended that you do the following when working with strings:

- Use the TCHAR macros found in TCHAR.H.
- Declare string buffers as `TCHAR[]`. Declare string pointers as `TCHAR*` or `LPTSTR.`
- Wrap string and character constants with the `_T()` macro.
- Use the `_tcs...()` functions in place of the `str...()` functions. For example, use `_tcslen()` in place of `strlen()`.
- Be careful when incrementing a pointer through a string. Remember that a logical character may occupy one or two **TCHAR** units. So replace code that looks like this:

```
char *cp;

for (cp = string; *cp != '\0'; ++cp)

{

    …

}
```

with code that looks like this:

```
TCHAR const *cp;

for (cp = string; *cp != _T('\0'); cp = _tcsinc(cp))

{

    …

}
```

- Avoid using a variable to hold the value of a logical character. Instead, use a pointer to a character in the string. In particular, avoid the `_tcsnextc()` macro, because the value it returns appears to be incompatible with some of the C runtime library functions.
- Use the functions `_tccpy()` and `_tccmp()` with string pointers instead of using the `=` and `==` operators on characters.
- Use `GetStringTypeEx()` instead of the character classification macros such as `_istalpha()`.
- Use `CharUpper()` and `CharLower()` instead of `_toupper()` and `_tolower()`.

Recommended reading

Microsoft has several good papers on writing international code on its Developer Network CD and its web site. To find documentation on the web site, go to http://msdn.microsoft.com/default.asp and search for MBBCS.

For documentation on globalization, go to http://www.microsoft.com/globaldev/.

The following book is also available:

- Schmitt, David A., Internationalization Programming for Microsoft® Windows®, ISBN 1-57231-956-9

AMIP API Class Reference

AMIP API Class Reference

The Alarm Interested Process API gives you the classes command handlers, and notify handlers you need to create your own application.

Classes
Command Handlers

Classes

Classes

The classes included in the AMIP API encapsulates alarm messages and inter-process communications.

- CAlarmUpdateInfo
- CAMIPBase

CAMIPBase

Encapsulates interprocess communications messages. The structure is:

```
{
public:
    CAMIPBase();
    ~CAMIPBase();
enum Error
    {
        errorNo=0,               // no error
        errorIpcRegister,        // IPC Regisration Failure
        errorMemAlocateFailed,   // Mem Alloc failed
        errorReserveEventFlags, // event flag could not be reserved
        errorInit,               // Error in initialization
        errorStart,              // Error in Starting the process
        errorReadPort,           // Error in Reading port
        errorIPC,                // IPC Error
        errorRouterDied,         // router died
        errorParnterDied,        // partner process died
        errorDGUnload,           // error in datagram unload
    };
    BOOL            Start(BOOL bAsync=FALSE);
    BOOL            Stop();
    BOOL            IsRunning();
    int                  GetLastError();
// virtual functions
    virtual void         OnAlarmInfo(CAlarmUpdateInfo *pAlarmInfo)=0;
    virtual void         OnShutdown()=0;
    virtual void         OnStatus(COR_STATUS *pStatus)=0;
    virtual void         NotifyStopOnError();
};
```

This is the abstract base class from which you drive an AMIP object. An AMIP object provides member functions for informing Alarm update to the application.

The constructor of CAMIPBase takes care of connecting to CIMPLICITY project.

`Start()` starts the process of informing alarm update. This process can be synchronous or asynchronous.

- The synchronous `Start()` waits until the process is shutdown.
- The asynchronous `Start()` can be stopped by `Stop()`.

`Stop()` stops the process of informing alarm update.

`GetLastError()` return the last error value. The error values are :

Error Code	Description
errorNo	No error
errorIpcRegister	IPC Registration Failure
errorMemAlocateFailed	Memory allocation failed
errorReserveEventFlags	Event flag could not be reserved
errorInit	Error in initialization
errorStart	Error in starting the process
errorReadPort	Error in Reading port
errorIPC	IPC Error
errorRouterDied	Router terminated
errorParnterDied	Partner process terminated
errorDGUnload	Error in datagram unload

CAlarmUpdateInfo

Encapsulate the alarm update information. The structure is:

```
class CAlarmUpdateInfo
{
public:
    CAlarmUpdateInfo();
    ~CAlarmUpdateInfo(){};
    CAlarmUpdateInfo(AM_UPD_INF am_upd_inf);
    AM_STATE_TYPE prev_state;
    AM_STATE_TYPE requested_action;
    AM_STATE_TYPE final_state;
    COR_STAMP   gentime;        // traditional timestamp
    COR_I4      generated_time; // alarm duration start time
    COR_I4      cleared_time;   // alarm duration end time
    COR_I4      amrp_sync;      // alarm duration AMRP sync
    char        alarm_id[ALARM_ID_LEN+1];
    char        fr_id[FR_ID_LEN+1];
```

```
    char        ref_id[AM_REF_ID_LEN+1];
    char        class_id[CLASS_ID_LEN+1];
    char        alarm_msg[ALARM_MSG_LEN+1];
    COR_I1      log_file;
};
```

Command Handlers

Command Handlers

The command handlers included in the Alarm Interested Process API let you process an alarm message, process a status block, perform shutdown operations and handle critical errors.

- OnAlarmInfo
- OnStatus
- OnShutdown()
- NotifyStopOnError()

OnAlarmInfo

This handler is called whenever a new alarm or event message is received by the AMIP API application.(CAlarmUpdateInfo *pAlarmInfo)

The `pAlarmInfo` is a pointer to the Alarm Information structure defined in the CAlarmUpdateInfo class.

Place all the code you need to process a message in this command handler.

OnStatus

This handler is called whenever an internal error message is received. Supply code to log the message to either the CIMPLICITY Status Log for your project or a user defined status log. (COR_STATUS *pStatus)

The `pStatus` is a pointer to the COR_STATUS status block. The COR_STATUS structure is defined in cor_stat.h.

OnShutdown()

This handler is called whenever a shutdown is requested.

Place all the code you need to handle your application shutdown in this command handler.

NotifyStopOnError()

This handler is called whenever a critical internal error has occurred. Supply code to perform all actions necessary to perform a clean exit.

The possible critical error values for the AMIP API are defined in the CAMPIBase class as is the GetLastError() function you can use to get the last error value.

Use the Alarm Interested Process (AMIP) API

Use the Alarm Interested Process (AMIP) API

Important: In order to use the Alarm Interested Process API, you must have the following software installed on your computer:

- Microsoft Visual C++ Version 2017.
- CIMPLICITY Base System and Alarm Management API software.

Step 1 *(page 134)*	Create the Sample Program Executable
Step 2 *(page 135)*	Run the Sample Program
Step 3 *(page 138)*	Write your AMIP Application
Step 4 *(page 139)*	Integrate your AMIP Application with a CIMPLICITY Project

About the Sample Program

The Alarm Interested Process API contains a sample program that demonstrates the use of this API. You can find the source file and make file for this program in the %BSM_ROOT%api \AMIPSAMPLE directory. It contains:

amipsample.cpp	Sample application source code.
makefile	Creates the sample executable.
Amipsample_exe.vcxproj	Visual Studio project to create the sample executable.

This sample program receives alarm messages from the Alarm Manager and writes the data to the sample.log file in the root directory of the drive where the program is run.

You can run this program in synchronous or asynchronous mode.

- In synchronous mode, the program waits in `CAMIPSample` until it receives an alarm message, then processes the message.
- In asynchronous mode, the program places the call in `CAMIPSample`, then is free to do other things until a message is received.

You can use the project provided with the sample as a basis for constructing projects for your own applications.

Note: Depending on how you install Visual C++, the INCLUDE, LIB, and PATH environment variables may not be automatically set when you install MSDEV. If they are not set, you will have to set them manually or run the following to set them before building any user programs:

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft Visual
 Studio\Installer\vswhere.exe" -property installationPath`) do set VSPATH=
%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

You can any existing or any newly created project to create *(page 134)* and run *(page 135)* the executable for the sample program.

Step 1. Create the Sample Program Executable

1. Click Start on the Windows task bar.
2. Select (All) Programs>Proficy HMI SCADA - CIMPLICITY version>**Demo Project**.

   The Demo project opens in the CIMPLICITY Workbench.

3. Click Tools>Command prompt on the Workbench menu bar.

   A Command Prompt window opens.

4. Type the following commands:

   `<drive>:`

   `cd %BSM_ROOT%api`

   Where

<drive> is the disk where your CIMPLICITY software is installed.

5. (If the environment variables are not set automatically) issue the following command to set them:

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
 Visual Studio\Installer\vswhere.exe" -property installationPath`) do
 set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

   This ensures that your environment variables (in particular %BSM_ROOT% and %SITE_ROOT%) are set correctly.

6. Start Visual Studio:

```
devenv CimplicityAPI.sln
```

7. Open Solution Explorer.
8. Right-click amipsample_exe to build the project.
9. Select **Build** on the Popup menu.

Step 2. Run the Sample Program

Step 2. Run the Sample Program

Follow the steps below to run amipsample.exe, in a project:

1. Configure an Alarm Log Printer.
2. Run the process from the Command Prompt window.
3. Use the Car Wash demo screen to generate alarms for the sample program to process.

   Note: You can also use this procedure to test your application executable.

Step 2.1 *(page 136)*	Configure an Alarm Log Printer for the Sample Program.
Step 2.2 *(page 136)*	Start the Sample Program.

Step 2.3 *(page 137)*	Generate Alarms for the Sample Program to Process.

Step 2.1. Configure an Alarm Log Printer for the Sample Program

1. Select the Alarm Printer icon in the Workbench left pane.
2. Open a New Alarm Log Printer dialog box..
3. Enter the printer name in the **Name** field
4. Click OK.

   The Alarm Printer dialog box opens.

5. Check the check boxes required for your system and enter fields to do the following.

Option			Description
Log events			Sends logged events to the specified printer or file.
Log alarms			Enables alarm logging options. Types of logged alarm that can be checked are: • Generated alarms • Acknowledged alarms • Reset alarms • Deleted alarms
		Checked	Sends logged alarm data to the specified printer or file. **Note:** Check all logged alarm check boxes if you are working with the SAMPLE program.
		Clear	Ignores the logged alarms.
	All alarm classes		
		Checked	Sends data for all alarm classes in the selected categories to the specified printer or file.
		Clear	Enables Alarm Class. Sends data for the selected alarm class in the selected categories to the specified printer or file.
Output			Name of the device or path and file to which the data will be sent. Example Enter c:\test.log

6. Click OK.

   Result: The Alarm Printer dialog box closes. The printer name displays in the Workbench right pane.

7. Click Project>Configuration Update on the Workbench menu bar.

Step 2.2. Start the Sample Program

1. Start the project.
2. Open the CIMPLICITY Process Control window.
3. Do the following.

A	Select the project in the Project drop-down list.
B	Click Connect.
C	Select MASTER_SAMPLE.
D	Click Stop Process.

4. Close the process Control window.
5. Click Tools>Command prompt... on the Workbench menu bar.
6. Enter the following command in the Command Prompt window:

   ```
   SET PRCNAM=SAMPLE
   ```

   to set the `PRCNAM` environment variable.

   Note: The name you use here must match the name you entered in the New Alarm Log Printer dialog.

7. Run the sample program.
   - To run the program in synchronous mode, type:

   ```
   AMIPSAMPLE
   ```

   - To run the program in asynchronous mode, type:

   ```
   AMIPSAMPLE ASYNC
   ```

Step 2.3. Generate Alarms for the Sample Program to Process

1. Click Start on the Windows task bar.
2. Select (All) Programs>Proficy HMI SCADA - CIMPLICITY version>CimView.
3. Select the Water_Delivery.cim screen in the project name>screens directory.
4. Click Start.
5. Click Auto.

After a few minutes, you can check the test.log file to verify that alarms are being logged.

Step 3. Write your AMIP Application

Step 3. Write your AMIP Application

Use amipsample.cpp as a template for writing your application. You will need to provide your own application code for:

- `main`
- `OnAlarmInfo`
- `OnStatus`
- `OnShutdown`
- `NotifyStopOnError`

Step 3.1 *(page 138)*	Compile and Link Your AMIP Application.
Step 3.2 *(page 139)*	Test your AMIP Application.

Step 3.1. Compile and Link your AMIP Application

1. From the Start menu, open the CIMPLICITY menu.
2. Select your project.
3. In the CIMPLICITY Workbench for your project, select Command prompt... from the Tools menu.

   This ensures that your environment variables (in particular %BSM_ROOT% and %SITE_ROOT%) are set correctly.

4. In the Command Prompt window, issue the following commands:

   <drive>`:`

   `cd` <directory>

   where <drive> is the disk where your CIMPLICITY software is installed, and <directory> is your application project directory.

5. If the environment variables are not set automatically, issue the following command to set them:

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
 Visual Studio\Installer\vswhere.exe" -property installationPath`) do
 set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

6. Start Visual Studio:

   `devenv <solution>`

   where:

   `<solution>` is the Visual Studio solution containing your project.

7. Build your project.

Step 3.2. Test your AMIP Application

You can test your application using any project. Just execute the same steps *(page 135)* for your application that you did for the sample program.

Step 4. Integrate your AMIP Application with a CIMPLICITY Project

Step 4. Integrate your AMIP Application with a CIMPLICITY Project

After you have verified that your application works correctly, you can integrate it into your CIMPLICITY project.

The necessary steps to integrate your AMIP application into a CIMPLICITY project involves three distinct parts.

Step 4.1 *(page 140)*	Create a configuration file for your application.
Step 4.2 *(page 141)*	Modify the system registry to inform CIMPLICITY software of the application.
	When these procedures are followed correctly, the process will start when your CIMPLICITY project starts.
Step 4.3 *(page 142)*	Configure the project for your AMIP process.

After you complete these procedures, your AMIP process runs automatically when you start your CIMPLICITY project.

Step 4.1. Create the Configuration File

1. Place your AMIP application executable in the %BSM_ROOT%exe directory (if you used the default directory during installation, this is will be c:\Program Files (x86)\Proficy\Proficy CIMPLICITY\EXE.

2. Create a file called <appname>.RP in %BSM_ROOT%bsm_data, where <appname> can be any name of your choosing (the name will be used again in the system registry).

   Note: The <appname>.RP file is an ASCII file in CIMPLICITY standard IDT format.

3. Edit <appname>.RP using the Notepad.

4. Enter the following on the first line:

   ```
   |-*
   ```

5. On the second line, enter information in the following fields. Separate the fields with vertical bars (`|`)

Node Location	Enter MASTER.
Process ID	The process identifier. Create short identifier for the process
Image Name	Name of the executable.
Service ID	The Service identifier. Use the Process ID.
Subsystem ID	The subsystem identifier. Use Process ID.
Object Name	The object name. Use Process ID.
Priority	The priority of the process. Set this to 20.
PM Flags	Process management flags to indicate if check in is required. Use 0.
Max. per Node	Maximum number of this process which can run on a single node.
Multiple Per System	Can this process run on more than one node. Enter 0.
Startup Type	Use RP. The process will be started after device communication processes.
Description	Description of the process. Whatever you desire.

   The contents of a sample APPNAME.RP file follow:

```
|-*
ANY|SAMP_APP|BSM_ROOT:[EXE]SAMPAPP.EXE|SAMP_APP|SAMP_APP|-
SAMP_APP|20|0|1|1|0|RP|SAMPLE APPLICATION
```

Step 4.2. Update the System Registry

You will need to add some entries to the Registry to allow the CIMPLICITY Configuration program to recognize your new application.

Warning: Making changes to the registry is very dangerous and should be done with care.

1. Run regedit.exe (or regedt32.exe.
2. Select the following in order:

   HKEY_LOCAL_MACHINE

   SOFTWARE

   GE Fanuc

   CIMPLICITY

   HMI

   Version (where version is your currently installed version)

   Products

3. Select New Key on the Edit menu to add a key for your new product. The Key Name must match the prefix you gave to your .RP file (it does not have to be an IC product number).
4. Select New String Value on the Edit menu
   a. Enter **Name** in the field provided
   b. Press Enter twice.
   c. In the **Value data** field, enter the name that you want to be displayed for the option when you create a project.
5. Select New String Value on the Edit menu.
   a. Enter **Serial Number** in the field provided
   b. Press Enter.
6. Select New String Value on the Edit menu.,
   a. Enter Type in the field provided
   b. Press Enter twice.
   c. In the **Value data** field, enter App.
7. Exit from the registry.

Step 4.3. Configure the Project

1. Open your project's CIMPLICITY Workbench.
2. Select Properties... from the Project menu, or click the Settings button on the Toolbar.
3. In the General tab of the Project Properties dialog, select the new application option.
4. Click OK to close the dialog.
5. Select Command Prompt... from the Tools menu.
6. Issue the following commands:

   <drive>`:`

   `cd %BSM_ROOT%MASTER`

   `IDTPOP ALARM_INTPROC`

   This generates the alarm_intproc.idt file.

7. Open alarm_intproc.idt file using a text editor (such as Notepad).
8. Add the following line to the end of the file:

   `|`<service_id>`|$ALL$|0|B .`

   where <service_id> is the Service ID you entered in the <appname>`.RP` file.

9. Save this file and exit the text editor.
10. Issue the following command:

    `SCPOP ALARM_INTPROC`

11. Exit the Command Prompt window.
12. Perform a configuration update of the project.

Chapter 5. External Alarm State Management API

About the External Alarm State Management API

You can use the External Alarm State Management API to create an External Alarm State Manager (XASMgr) to manage CIMPLICITY alarms. An External Alarm State Manager can generate alarms within CIMPLICITY alarm management and maintains complete control over the state transitions of those alarms.

State transitions of CIMPLICITY alarms are normally driven by responses from CIMLPLICITY software or custom alarm viewers. When one of these viewers, under user direction, generates a response to an alarm, the CIMPLICITY Alarm Manager normally transitions the alarm based on its internal criteria.

If alarms are generated by an XASMgr process, the CIMPLICITY Alarm Manager does not transition these alarms, but passes their user responses to the XASMgr that generated the alarm. The XASMgr then decides whether the response to the alarm should be applied. If it decides the response can be applied, it passes the response back to the CIMPLICITY Alarm Manager.

Note: The Repeat, Acknowledge, and Reset timeouts are not applied by the Alarm Manager to alarms generated by an XASMgr. If you require this functionality for alarms managed by an XASMgr, the XASMgr must provide that functionality and send appropriate action requests to the Alarm Manager.

The following C++ classes are included in this API to aid you in utilizing CIMPLICITY Alarm Management:

- AlarmGen
- CExternalAlarmManager

Use the AlarmGen class to generate alarms for and send responses to the Alarm Manager. Use the CExternalAlarmManager class to develop your XASMgr process.

Notes on Internationalization for the External Alarm State Management API

- Work with strings.
- Recommended reading.

Work with strings

This API is written for the international environment. In an international environment, strings in CIMPLICITY software can be multibyte strings. If you want your code to conform to international standards, it is recommended that you do the following when working with strings:

- Use the TCHAR macros found in TCHAR.H.
- Declare string buffers as `TCHAR[]`. Declare string pointers as `TCHAR*` or `LPTSTR`.
- Wrap string and character constants with the `_T()` macro.
- Use the `_tcs...()` functions in place of the `str...()` functions. For example, use `_tcslen()` in place of `strlen()`.
- Be careful when incrementing a pointer through a string. Remember that a logical character may occupy one or two `TCHAR` units. So replace code that looks like this:

```
char *cp;

for (cp = string; *cp != '\0'; ++cp)

{

        …

}
```

with code that looks like this:

```
TCHAR const *cp;

for (cp = string; *cp != _T('\0'); cp = _tcsinc(cp))

{

        …

}
```

- Avoid using a variable to hold the value of a logical character. Instead, use a pointer to a character in the string. In particular, avoid the _tcsnextc() macro, because the value it returns appears to be incompatible with some of the C runtime library functions.
- Use the functions `_tccpy`() and _tccmp() and string pointers instead of the = and == operators on characters.
- Use `GetStringTypeEx()` instead of the character classification macros such as `_istalpha()`.
- Use `CharUpper()` and `CharLower()` instead of `_toupper()` and _tolower().

Recommended Reading

Microsoft has several good papers on writing international code on its Developer Network DVD and its web site. To find documentation on the web site, go to http://msdn.microsoft.com/default.asp and search for MBBCS.

For documentation on globalization, go to http://www.microsoft.com/globaldev/.

The following book is also available:

- Schmitt, David A., Internationalization Programming for Microsoft® Windows®, ISBN 1-57231-956-9

For more information about this book, go to http://mspress.microsoft.com/books/2323.htm.

External Alarm State Management Getting Started

External Alarm State Management Getting Started

In order to use the External Alarm State Management API for the current CIMPLICITY release, you must have the following software installed on your computer:

- Microsoft Visual Studio C++ 2017.
- CIMPLICITY Base System and Alarm Management API software.

1 *(page 149)*	Create an XASMgr API application.
2 *(page 163)*	Write an XASMgr application.
3 *(page 164)*	Compile and link the XASMgr application.
4 *(page 165)*	Test the XASMgr application.
5 *(page 165)*	Integrate the XASMgr application with a CIMPLICITY project.

External Alarm State Management API Contents

The following is a list of all files distributed with the Alarm Management API. The files are loaded into the directories indicated. The environment variable `%BSM_ROOT%` is the directory where the CIMPLICITY software was installed.

Include files in `%BSM_ROOT%\api\include\inc_path` are:

alarmapi.h	cor_thread.h
am_defs.h	cor_time.h
am_errors.h	ddl.h
amap_defs.h	examgr.h
amaru_err.h	ipc.hpp
amdd_cmd.h	ipcerr.h
amdd_cont.h	mf_defs.h
amdd_defs.h	mfamupdi.h
amip.h	mfpmterm.h
cor.h	mfstatus.h
cor_event.h	netcom.h
cor_mutex.h	rcm.h
cor_os.h	rtr_bcst.h
cor_stat.h	sc_recs.h

Source files in `%BSM_ROOT%\api\amxasmgr` are:

makefile	amxassample.cpp	amxassample_exe.vcxproj

Source files in `%BSM_ROOT%\api\lib` are:

amaru.lib	fasrtl.lib
amip.lib	ipc.lib
corutil.lib	cim_mf.lib
ddl.lib	cim_sc.lib
examgr.lib	rcm.lib

External Alarm State Management API Sample Program

The External Alarm State Management API contains a sample program that demonstrates the use of this API. You can find the source file and make file for this program in the `%BSM_ROOT%\api\amxasmgr` directory. It contains:

`amxassample.cpp`	The sample application source code.
`Amxassample_exe.vcxproj`	Visual Studio project that creates the sample executable.
`makefile`	`Makefile` that creates the sample executable.

This sample program generates up to ten alarms and processes Acknowledge, Clear, and Delete requests passed to it by the Alarm Manager.

Note: The program generates each of the alarms you create, then repeats the first three alarms every thirty seconds.

You can use the project provided with the sample as a basis for constructing projects for your own applications.

Note: Depending on how you installed Visual C++, the INCLUDE, LIB, and PATH environment variables may not be automatically set when you install MSDEV. If they are not set, you will have to set them manually or run the following to set them before building any user programs.

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft Visual
 Studio\Installer\vswhere.exe" -property installationPath`) do set VSPATH=
%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

- You can use any project to create and run the executable for the sample program.
- Create a sample program.
- Run a sample program.

Create a Sample Program

1. From the Start menu, select the CIMPLCIITY menu.
2. Select a project.
3. In the CIMPLICITY Workbench, select Tools>Command Prompt.

   This ensures that your environment variables (in particular `%BSM_ROOT%` and `%SITE_ROOT%`) are set correctly.

4. In the Command Prompt window, issue the following commands:

   <drive>:

```
cd %BSM_ROOT%\api
```

where <drive> is the disk where your CIMPLICITY software is installed

5. If the environment variables are not set automatically, issue the following command to set them:

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
Visual Studio\Installer\vswhere.exe" -property installationPath`) do
set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

6. Start Visual Studio:

```
devenv CimplicityAPI.sln
```

7. Open the Solution Explorer.
8. Right-click amxassample_exe.
9. Select **Build** on the Popup menu.

   The sample program in the project is amxassample.exe.

10. Create the **R1** resource and assign it to all available users.
11. Create 10 alarms. When you create each alarm, enter/select the following values.

Property	Value	
Alarm IDs	Name the alarms as follows. LONG_NAME_POINT_12345678912345678912345678 9_TEST1 LONG_NAME_POINT_12345678912345678912345678 9_TEST2 LONG_NAME_POINT_12345678912345678912345678 9_TEST3 LONG_NAME_POINT_12345678912345678912345678 9_TEST4 LONG_NAME_POINT_12345678912345678912345678 9_TEST5 TEST6 TEST7 TEST8 TEST9 TEST10	
Alarm Definitions	Alarm Class	Select High.
	Alarm Type	Leave blank.
	Maximum Stacked	Set to 20.
	Alarm Message	Enter text (e.g. for Test 1 enter TEST1 Alarm).
Alarm Routing	Route to all users. Note: Add the USER role to the Configured Roles For Alarm list.	
Alarm Options	Deletion Requirements	Check both check boxes.
	Automatic Actions	Select None.
	Manual Reset Allowed	Check the check box.

12. Update the project's configuration.
13. Start the project.
14. When project startup is complete, select Command prompt... from the Tools menu.
15. To run the sample program. type:

    AMXASSAMPLE

16. The sample program starts generating alarms.

    Note: The program generates all ten alarms, and then repeats the first three alarms at thirty second intervals.

17. You can acknowledge, reset and delete these alarms through an Alarm Viewer.
    The Alarm Manager sends the action you request to the sample program. The sample program then asks you if you want to perform the action you requested. If you confirm the action, the sample program sends a message to the Alarm Manager to execute the action. If you do not confirm the action, it is not performed.

1. Create an XASMgr API Application

1. Create an XASMgr API Application

The External Alarm State Management API gives you the classes *(page 149)* , command handlers *(page 157)* , and notify handlers you need to create your own application.

- Classes
- AlarmGen methods
- CExternalAlarmManager Methods and Command Handlers

Classes

Classes

The classes included in the XASMGR API encapsulate the methods and handlers used to generate and process alarm messages.

- AlarmGen
- CExternalAlarmManager

AlarmGen

Encapsulates the sending of alarm generation and response messages to the Alarm Manager. The structure is:

```
class AlarmGen (page 151)
{
public:
  AlarmGen (page 153) ();
  int InitializeAmGen (page 156) (int portIndex);
  void ResetFields (page 157) ();
  void AddField (page 152) (LPCTSTR value);
  void AddField (page 152) (int value);
  GenerateAlarm (page 155) (LPCTSTR alarmId,LPCTSTR fr,LPCTSTR refId);
  GenerateAlarmStamp (page 155) (LPCTSTR alarmId,LPCTSTR fr,LPCTSTR
                      refId,COR_STAMP stamp);
  AckAlarm (page 152) (LPCTSTR alarmId,LPCTSTR fr,LPCTSTR refId);
  ResetAlarm (page 156) (LPCTSTR alarmId,LPCTSTR fr,LPCTSTR refId);
  DeleteAlarm (page 154) (LPCTSTR alarmId,LPCTSTR fr,LPCTSTR refId);

  AckCAAlarm(LPCTSTR alarmId,LPCTSTR fr,LPCTSTR refId, ChangeapprovalInfo*
 ptrCAObj );
  ResetCAAlarm (LPCTSTR alarmId,LPCTSTR fr,LPCTSTR refId,
 ChangeapprovalInfo* ptrCAObj);
  DeleteCAAlarm (LPCTSTR alarmId,LPCTSTR fr,LPCTSTR refId,
 ChangeapprovalInfo* ptrCAObj );
}
```

CExternalAlarmManager

Encapsulates the methods and handlers used to generate and process external alarms.

```
class EXTMGRAPIEXPORT CExternalAlarmManager (page 157) : public AlarmGen
{
public:
    CExternalAlarmManager();
    virtual ~CExternalAlarmManager();
    BOOL            Start (page 158) (BOOL bAsync=FALSE);
    BOOL            Stop (page 159) ();
    int GenerateAlarmStampWithVerify (page 163) (LPCTSTR alarmId,
                                      LPCTSTR fr,
                                      LPCTSTR refId,
                                      int state,
                                      COR_STAMP stamp,
                                      COR_I4 clearedTime);
    virtual void    OnAlarmAck (page 160) (LPCTSTR alarmId,
                                LPCTSTR frId, LPCTSTR refId);
    virtual void    OnAlarmClear (page 161) (LPCTSTR alarmId,
                                  LPCTSTR frId,
                                  LPCTSTR refId);
```

```
    virtual void    OnAlarmDel (page 161) (LPCTSTR alarmId,
                                LPCTSTR frId, LPCTSTR refId);
    virtual void    OnShutdown (page 159) ()=0;
    virtual void    OnInited (page 159) ();
    BOOL            IsRunning (page 159) () const {return m_bRunning;};
    void            report_error (page 162) (LPCTSTR name, int x,
                                  COR_STATUS *stat);
}
```

This is the abstract base class from which you drive an XASMgr object. An XASMgr object provides member functions for processing alarm messages.

The constructor of `CExternalAlarmManager` takes care of connecting to a CIMPLICITY project.

`Start()` starts the process of generating alarms and monitoring their state changes. This process can by synchronous or asynchronous.

- The synchronous `Start()` waits until the process shuts down.
- The asynchronous `Start` can be stopped by `Stop()`.

`Stop()` stops the process of generating alarms and monitoring alarm state changes.

AlarmGen Methods

AlarmGen Methods

The AlarmGen methods provide a partial wrapper around basic functionality that exists in the AMARU library C functions. This wrapper is intended to simplify the effort required on the part of a developer to generate alarms for and send alarm responses to the CIMPLICITY Alarm Manager.

Methods include:

- AckAlarm
- AckCAAlarm
- AddField
- AddSeverity
- AlarmGen
- DeleteAlarm
- DeleteCAAlarm
- GenerateAlarm
- GenerateAlarmStamp
- InitializeAmGen
- ResetAlarm
- ResetCAAlarm
- ResetFields

AckAlarm

Syntax

```
AckAlarm (LPCTSTR alarmId, LPCTSTR fr, LPCTSTR refId )
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.

Remarks

This method sends a message to the CIMPLICITY Alarm Manager to acknowledge the alarm identified by the parameters.

AckCAAlarm

Syntax

```
AckCAAlarm (LPCTSTR alarmId, LPCTSTR fr, LPCTSTR refId , ChangeapprovalInfo* ptrCAObj );
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.
ptrCAObj	Change approval information for an alarm.

Remarks

This method sends a message to the CIMPLICITY Alarm Manager to acknowledge the alarm identified by the parameters.

AddField

Syntax

`void AddField (LPCTSTR` value `);` or `void AddField (int` value `);`

Parameters

value	The field value for the alarm message.

Remarks

`AddField(LPCSTR value)` adds a new string parameter to an internal holding area. This parameter is passed as an alarm message parameter with the next call to `GenerateAlarm()` or `GenerateAlarmStamp()`.

`AddField(int value)` adds a new integer parameter to an internal holding area. This parameter is passed as an alarm message parameter with the next call to `GenerateAlarm()` or `GenerateAlarmStamp()`.

This alarm field remains effective for all subsequent `GenerateAlarm()` or `GenerateAlarmStamp()` calls until `ResetFields` is called.

AddSeverity

Syntax

`void AddSeverity (int` value `);`

Parameters

value	The severity for the alarm message.

Remarks

`AddSeverity` adds the alarm severity to an internal holding area. This parameter is passed as the alarm message severity with the next call to `GenerateAlarm()` or `GenerateAlarmStamp()`.

This alarm severity remains effective for all subsequent `GenerateAlarm()` or `GenerateAlarmStamp()` calls until either another `AddSeverity` or `ResetFields` is called.

AlarmGen

Syntax

```
InitializeAlarmGen ( );
```

Remarks

This default constructor does basic initialization, but does not render the class ready for use. You must call the `InitializeAmGen()` method before you actually use an instance of the class.

DeleteAlarm

Syntax

```
DeleteAlarm (LPCTSTR alarmId, LPCTSTRfr, LPCTSTR refId );
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.

Remarks

This method sends a message to the CIMPLICITY Alarm Manager to delete the alarm identified by the parameters.

DeleteCAAlarm

Syntax

```
DeleteCAAlarm (LPCTSTR alarmId, LPCTSTR fr, LPCTSTR refId, ChangeapprovalInfo*
ptrCAObj )
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter `""` in this argument.
ptrCAObj	Change approval information for the alarm.

Remarks

This method sends a message to the CIMPLICITY Alarm Manager to delete the alarm identified by the parameters.

GenerateAlarm

Syntax

```
GenerateAlarm (LPCTSTR alarmId, LPCTSTR fr, LPCTSTR refId );
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.

Remarks

This method sends an alarm generation message to the CIMPLICITY Alarm Manager. Included in the message will be any alarm message parameters created by calls to `AddField()` or `AddSeverity()` since initialization or the last `ResetField()` call. The Alarm Manager uses the current time for the alarm generation `timestamp.GenerateAlarm`.

GenerateAlarmStamp

Syntax

```
GenerateAlarmStamp (LPCTSTR alarmId, LPCTSTR fr, LPCTSTR refID, COR_STAMP timestamp
);
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.
timestamp	The date and time to be used for the alarm generation time.

Remarks

This method sends an alarm generation message to the CIMPLICITY Alarm Manager. Included in the message will be any alarm message parameters created by calls to `AddField()` or `AddSeverity()` since initialization or the last `ResetField()` call. The Alarm Manager uses the timestamp given by this method for the alarm generation timestamp.

InitializeAmGen

Syntax

`InitializeAmGen (int` portIndex `);`

Parameters

portIndex	The port index used for receiving messages from other CIMPLICITY processes (in particular, the Alarm Manager). If you omit this parameter or pass it as a -1, the method internally acquires its own port index.

Remarks

This method prepares an instance of an `AlarmGen` class for use.

ResetAlarm

Syntax

`ResetAlarm (LPCTSTR` alarmId`, LPCTSTR` fr`, LPCTSTR` refId `);`

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.

Remarks

This method sends a message to the CIMPLICITY Alarm Manager to reset (clear) the alarm identified by the parameters.

ResetCAAlarm

Syntax

`ResetCAAlarm (LPCTSTR` alarmId`, LPCTSTR` fr`, LPCTSTR` refId `, ChangeapprovalInfo*` ptrCAObj `)`

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.
ptrCAObj	Change approval information for an alarm.

Remarks

This method sends a message to the CIMPLICITY Alarm Manager to reset(or Clear) the alarm identified by the parameters.

ResetFields

Syntax

```
ResetFields ( );
```

This method resets the fields to be passed with a generated alarm that requires parameters in its message (alarm parameters are determined by the Alarm Type definition). It also resets the alarm severity.

If you call `GenerateAlarm()` or `GenerateAlarmStamp()` immediately following a call to `ResetFields()`, no alarm parameters or severity are passed with the alarm generation message to the CIMPLICITY Alarm Manager. Whether or not this is correct depends on the definition of the alarm being generated.

CExternalAlarmManager Methods and Command Handlers

CExternalAlarmManager Methods and Command Handlers

The `CExternalAlarmManager` class hides most low-level interaction with the CIMPLICITY Alarm Manager, including the basic initiation of CIMPLCIITY IPC services, registering with the Alarm Manager as an XASMgr, and the details of generating alarms and receiving alarm responses.

Methods include:

- Start

- Stop
- OnInited
- OnShutdown
- IsRunning
- OnCAAlarmAck
- OnAlarmAck
- OnAlarmClear
- OnCAAlarmClear
- OnAlarmDel
- OnCAAlarmDel
- report_error
- GenerateAlarmStampWithVerify

Start

Syntax

`BOOL Start(BOOL` bAsync=FALSE `);`

Parameters

bAsync	Flag that determines whether synchronous or asynchronous processing will be used by the XASMgr. TRUE = Asynchronous processing will be used. FALSE = Synchronous processing will be used.

Remarks

It is anticipated that an XASMgr is going to be a multi-threaded application. The start method provides two means for this multi-threading to be accomplished.

- If you pass the parameter as FALSE, functionality continues to run in the main thread and will not return until the CIMPLICITY project shuts down.

If you choose this method, then you will probably either already have created your own thread, or anticipate doing it within `OnInited`. The need for your own thread is anticipated as it is expected that the alarms you will be managing are probably occurring in an asynchronous fashion, and you will need to interact independently with external devices in a fashion that could not be supported by the unpredictability of callbacks from the `CExternalAlarmManager` internals.

- If you pass the parameter as TRUE, then `CExternalAlarmManager` creates its own thread of execution, and the Start call returns after that thread has been spawned.

The next normal action to occur will be a call to the `OnInited` method. At the point `CExternalAlarmManger` calls this method all preparations have been successfully completed to allow this XASMgr to generate alarms. Do not attempt to generate or otherwise modify alarms

before `OnInited` has been called. If you do, those actions will not be handled appropriately, as initialization has not been completed.

Stop

Syntax

```
BOOL Stop( );
```

Remarks

This method stops asynchronous processing.

OnInited

Syntax

```
void OnInited( );
```

Remarks

This handler is called after the XASMgr process finishes initializing process communications.

Place all the code you need to handle XASMgr initiation in this command handler.

OnShutdown

Syntax

```
void OnShutdown( );
```

Remarks

This handler is called whenever CIMPLICITY interprocess communications indicates to the XASMgr that the project is shutting down.

Place all the code you need to handle your application shutdown in this command handler.

IsRunning

Syntax

```
BOOL isRunning ( )
```

This method verifies that the project is running and interprocess communications are working. It returns TRUE if everything is all right, and returns FALSE if it detects a problem.

OnAlarmAck

Syntax

```
void OnAlarmAck ( LPCTSTR alarmID, LPCTSTR frID, LPCTSRT refID );
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.

Remarks

This handler is called when an ACK request is initiated by a CIMPLICITY user.

OnCAAlarmAck

Syntax

```
void OnCAAlarmAck ( LPCTSTR alarmID, LPCTSTR frID, LPCTSRT refID , ChangeapprovalInfo* ptrCAObj)
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.
ptrCAObj	Change approval information for an alarm

Remarks

This handler is called when an `ACK` request is initiated by a CIMPLICITY user.

OnAlarmClear

Syntax

```
void OnAlarmClear (LPCTSTR alarmID, LPCTSTR frID, LPCTSRT refID );
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.

Remarks

This handler is called when a RESET request is initiated by a CIMPLICITY user.

OnCAAlarmClear

Syntax

```
void OnCAAlarmClear ( LPCTSTR alarmID, LPCTSTR frID, LPCTSRT refId , ChangeapprovalInfo* ptrCAObj)
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.
ptrCAObj	Change approval information for an alarm.

Remarks

This handler is called when an `RESET` request is initiated by a CIMPLICITY user.

OnAlarmDel

Syntax

```
void OnAlarmDel (LPCTSTR alarmID, LPCTSTR frID, LPCTSRT refID );
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.

Remarks

This handler is called when a DELETE request is initiated by a CIMPLICITY user.

OnCAAlarmDel

Syntax

`void OnCAAlarmDel ( LPCTSTR` alarmID`, LPCTSTR` frID`, LPCTSRT` refID `, ChangeapprovalInfo*` ptrCAObj`);`

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.
ptrCAObj	Change approval information for an alarm.

Remarks

This handler is called when a `DELETE` request is initiated by a CIMPLICITY user.

report_error

Syntax

`void report_error (LPCTSTR` name`, int` x`, COR_STATUS` *stat `);`

Parameters

name	The name of the function where the error is detected.
x	A code for the error.
*stat	The pointer to the COR_STATUS status block. The COR_STATUS structure is defined in cor_stat.h.

Remarks

This method generates a Status Log message.

GenerateAlarmStampWithVerify

Syntax

```
int GenerateAlarmStampWithVerify (LPCTSTR alarmId, LPCTSTR fr,
    LPCTSTR refId, int state, COR_STAMP stamp,
    COR_I4 cleared_time );
```

Parameters

alarmId	The Alarm ID for the alarm.
fr	The Resource ID for the alarm.
refId	The optional Reference ID for the alarm. If you have no Reference ID, enter "" in this argument.
state	The current state of the alarm. Valid values are: AM_GENERATED AM_ACKNOWLEDGED AM_CLEARED
stamp	The timestamp when the alarm was originally generated. You may pass a pointer to a valid timestamp or a null value.
cleared_time	If the state is AM_CLEARED, this field contains the clear time. Otherwise, pass a null value.

Remarks

This method is similar to the other GenerateAlarm calls. but invokes special processing within the Alarm Manager. If the alarm specified in the method is not currently generated, the alarm will be generated. A subsequent check is made of the alarm state of the alarm, comparing it to the requested state. If the state of the alarm is different from the state specified in this method, then the Alarm Manager applies the state specified by this method to the alarm. If no changes occur as a result of this call, nothing will be logged by the Alarm Manager. If changes occur, they will be logged as configured.

2. Write an XASMgr Application

- Required custom application code.
- Thread synchronization.

Required custom application code

Use amxasamxasamxassample.cpp as a template for writing your application. You will need to provide your own application code for:

- `main`
- `OnInited`
- `OnAlarmAck`
- `OnAlarmClear`
- `OnAlarmDel`
- `OnShutdown`

`Thread Synchronization`

Your implementation of `CExternalAlarmManager` will have to manage its own thread synchronization. The need for this synchronization will arise in situations where a callback method may need to access XASMgr specific structures at the same time the corresponding thread may need to access them.

The base internals have no knowledge of what you create, and cannot help out in this area. The responsibility for this synchronization rests completely with the XASMgr developer, and failure to deal with this issue by appropriate use of semaphores, mutexes or other Microsoft Win32 synchronization objects will likely result in an unreliable, unpredictable XASMgr implementation.

The sample program, as implemented, does not have need of or demonstrate these techniques. Please refer to Microsoft documentation and samples for management of multiple threads within a process potentially requiring simultaneous access to common data.

3. Compile and Link the XASMgr Application

1. From the Start menu, open the CIMPLICITY menu.
2. Select your project.
3. In the CIMPLICITY Workbench for your project, select Tools>Command Prompt.

   This ensures that your environment variables (in particular `%BSM_ROOT%` and `%SITE_ROOT%`) are set correctly.

4. In the Command Prompt window, issue the following commands:

   <drive>:

   `cd` <directory>

where <drive> is the disk where your CIMPLICITY software is installed, and <directory> is your application project directory.

5. If the environment variables are not set automatically, issue the following command to set them:

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
 Visual Studio\Installer\vswhere.exe" -property installationPath`) do
 set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

6. Now build the executable:

```
nmake
```

4. Test the XASMgr Application

You can test your application using any project. Just execute the same steps for your application that you did for the sample program.

5. Integrate the XASMgr Application with a CIMPLICITY Project

5. Integrate the XASMgr Application with a CIMPLICITY Project

After you have verified that your application works correctly, you can integrate it into your CIMPLICITY project.

The necessary steps to integrate your XASMgr application into a CIMPLICITY project involves three distinct steps:

5.1 *(page 165)*	Create the configuration file.
5.2 *(page 166)*	Update the system registry.
5.3 *(page 167)*	Configure the project.

After you complete these three procedures, your XASMgr process runs automatically when you start your CIMPLICITY project.

5.1. Create the Configuration File

1. Place your XASMgr application executable in the `%BSM_ROOT%exe` directory (if you used the default directory during installation, this is will be c:\Program Files (x86)\Proficy\Proficy CIMPLICITY\EXE.

2. Create a file called <appname>.RP in `%BSM_ROOT%bsm_data`, where <appname> can be any name of your choosing (the name will be used again in the system registry).

   The <appname>.RP file is an ASCII file in CIMPLICITY standard IDT format.

3. Edit <appname>.RP using the Notepad.

4. Enter the following on the first line:

   ```
   |-*
   ```

5. On the second line, enter information in the following fields. Separate the fields with vertical bars (|)

Node Location	Enter `MASTER`.
Process ID	The process identifier. Create short identifier for the process
Image Name	Name of the executable.
Service ID	The Service identifier. Use the Process ID.
Subsystem ID	The subsystem identifier. Use Process ID.
Object Name	The object name. Use Process ID.
Priority	The priority of the process. Set this to 20.
PM Flags	Process management flags to indicate if check in is required. Use 0.
Max. per Node	Maximum number of this process which can run on a single node.
Multiple Per System	Can this process run on more than one node. Enter 0.
Startup Type	Use RP. The process will be started after device communication processes.
Description	Description of the process. Whatever you desire.

   The contents of a sample APPNAME.RP file follow:

   ```
   |-*
   ANY|SAMP_APP|BSM_ROOT:[EXE]SAMPAPP.EXE|SAMP_APP|SAMP_APP|-
   SAMP_APP|20|0|1|1|0|RP|SAMPLE APPLICATION
   ```

5.2. Update the System Registry

You will need to add some entries to the Registry to allow the CIMPLICITY Configuration program to recognize your new application.

❗ **Warning:** Warning: Making changes to the registry is very dangerous and should be done with care.

1. Run regedit.exe or regedt32.exe
2. Select the following.

   32-bit systems

   `HKEY_LOCAL_MACHINE>SOFTWARE>GE Fanuc>CIMPLICITY>HMI>Version>Products`

   64-bit systems

   `HKEY_LOCAL_MACHINE>SOFTWARE>Wow6432Node>GE Fanuc>CIMPLICITY>HMI>Version>Products`

3. Select New Key on the Edit menu to add a key for your new product. The **Key Name** must match the prefix you gave to your .RP file (it does not have to be an IC product number).
4. Select New String Value on the Edit menu
   a. Enter "Name" in the field provided.
   b. Press Enter twice.
   c. In the **Value data** field, enter the name that you want to be displayed for the option when you create a project.
5. Select New String Value on the Edit menu.
   a. Enter `Serial Number` in the field provided
   b. Press Enter.
6. Select New String Value on the Edit menu.
   a. Enter `Type` in the field provided
   b. Press Enter twice.
   c. In the **Value data** field, enter `App`.
7. Exit from the registry.

5.3. Configure the Project

1. Open your project in the CIMPLICITY Workbench.
2. Select Properties... from the Project menu, or click the Settings button on the Toolbar.
3. On the General tab of the Project Properties dialog, select the new application options.
4. Click **OK** to close the dialog.

5. Perform a project configuration update.

Chapter 6. Alarm Viewer API

About the Alarm Viewer API

The Alarm Viewer Application Program Interface (AMV API) is included in the Integrator's Toolkit product option for GE Intelligent Platform's CIMPLICITY software. This API is fully integrated with CIMPLICITY software's Base System functionality to enhance its already powerful monitoring capability in a full range of computer integrated manufacturing environments.

The Alarm Viewer API provides an interface for application programmers to develop full-featured custom alarm viewers to meet special needs.

Alarm Viewer and CIMPLICITY Functionality

The CIMPLICITY Base System functionality -- Point Management, Alarm Management, and Data Logging facilities as well as a full-functioned User Interface -- enables CIMPLICITY users to collect data for reporting and to visualize data via lists, graphic status displays, and alarms. Standard data communications capabilities make CIMPLICITY software a factory floor tool that can provide services such as those listed below.

- Downtime reporting
- Production reporting
- Records of production counts at work stations
- Graphic monitoring of automatic data point values
- Fault reporting via direct point values and alarms

CIMPLICITY software's flexible system architecture and modular design allows for easy add-on of functionality.

Alarm Viewer Management API Overview

Alarm Viewer Management API Overview

CIMPLICITY software's Alarm Management module is responsible for maintaining the status of outstanding alarms, or predetermined conditions of interest, detected by an application process. Alarm Management informs users of current alarm occurrences and sends information on alarm

occurrences to interested processes. Alarm Management provides a set of services to generate new alarms and update the status of existing alarms. These services allow an application to interact with Alarm Management capability without knowing the message structure and message passing aspects of interfacing to an Alarm Management Resident Process.

The Alarm Management module consists of an Alarm Management Resident Process (AMRP), and a configured number of Alarm Management Allocated Processes (AMAP).

Alarm Viewer API Operation Overview

The CIMPLICITY Alarm Manager Resident Process (AMRP) is responsible for maintaining a centralized database of current alarms. Users may view and act on these alarms with the Alarm Viewer or an Alarm Viewer control embedded in a CimView screen.

These standard viewers:

- Display alarms sorted by various criteria.
- Filter alarms by time, alarm class, alarm state, and resource.
- Allow a user to acknowledge, delete, and comment on alarms.

In addition, sets of sorting and filtering preferences can be created, saved, loaded, and edited. The Alarm Viewer API (AMV API) allows a process engineer or system integrator to develop full-featured custom alarm viewers to meet special needs.

Important: The Alarm Viewer API must send the short alarm length (32 characters or fewer) to perform operations even if the alarm is assigned a longer ID. Both the short, and, if it exists, the long alarm names are available via the API.

No changes need to be made to existing applications that use this API; the short alarm name will be in the `alarmid` field used by the `AlarmInfo` structure in previous versions (Applications will already be using the correct alarmid field.). If an application requires communication with CIMPLICITY v9.0 servers that have projects with long alarm names, the only change required will be for the application to use the new `long_name` field (from the AlarmInfo structure) when displaying alarm names to the user.

Alarm Viewer API Features

The AMV API is implemented as a set of C++ classes which encapsulate various aspects of the connection between an alarm viewer process, such as the standard Alarm Viewer, and the AMRP. The classes and their functions are summarized below.

Class	Description
CAmvAlarm	An individual alarm instance

Class	Description
CAmvClassFilter	An Alarm Class (for example, HIGH)
CAmvClassFilterList	The set of classes used to filter alarms for a connection.
CAmvResourceFilter	A Resource
CAmvConn	The class through which the connection between an alarm viewer and the Alarm Manager is implemented.
CAmvFieldFilter	The set of fields used to filter alarms for a connection.
CAmvFieldFilterList	A way to loop through the field filters currently in use.
CAmvResourceFilterList	The set of resources used to filter alarms for a connection.
CAmvSetupList	A way to access Alarm Viewer saved setups
CAmvStateFilter	The set of alarm states used to filter alarms for a connection.
CAmvStateFilterList	A way to loop through the state filters currently in use.
CAmvTimeFilter	The time used to filter alarms for a connection. If enabled, only alarms after the specified time are passed.

Connections between alarm viewers and the alarm manager are encapsulated in the `CAmvConn` (Alarm Viewer Connection) class. When a `CAmvConn` object is constructed, you provide a number of callback functions. The constructor starts a new thread to process alarm manager communication. This thread calls your functions to process alarms.

Notes on Internationalization for the Alarm Management API

- Work with strings.
- Recommended reading.

Work with strings

This API is written for the international environment. In an international environment, strings in CIMPLICITY software can be multibyte strings. If you want your code to conform to international standards, It is recommended that you do the following when working with strings:

- Use the `TCHAR` macros found in TCHAR.H.
- Declare string buffers as `TCHAR[]`. Declare string pointers as `TCHAR*` or `LPTSTR`.
- Wrap string and character constants with the `_T()` macro.
- Use the `_tcs  ()` functions in place of the `str  ()` functions. For example, use `_tcslen()` in place of `strlen()`.
- Be careful when incrementing a pointer through a string. Remember that a logical character may occupy one or two `TCHAR` units. So replace code that looks like this:

```
char *cp;

for (cp = string; *cp != '\0'; ++cp)

{

    …

}
```

with code that looks like this:

```
TCHAR const *cp;

for (cp = string; *cp != _T('\0'); cp = _tcsinc(cp))

{

    …

}
```

- Avoid using a variable to hold the value of a logical character. Instead, use a pointer to a character in the string. In particular, avoid the `_tcsnextc()` macro, because the value it returns appears to be incompatible with some of the C runtime library functions.
- Use the functions `_tccpy()` and `_tccmp()` and string pointers instead of the `=` and `==` operators on characters.
- Use `GetStringTypeEx()` instead of the character classification macros such as `_istalpha()`.
- Use `CharUpper()` and `CharLower()` instead of `_toupper()` and `_tolower()s`.

Recommended Reading

Microsoft has several good papers on writing international code on its Developer Network DVD and its web site. To find documentation on the web site, go to http://msdn.microsoft.com/default.asp and search for MBCS.

The following book is also available:

- Schmitt, David A., International Programming for Microsoft® Windows®, ISBN 1-57231-956-9.

For more information about this book, go to http://mspress.microsoft.com/books/2323.htm.

Alarm Viewer API Getting Started

Alarm Viewer API Getting Started

The CIMPLICITY Alarm Viewer API lets application programs access the functions of CIMPLICITY software's Alarm Management Application Module. Using the API requires that you do the following:

- Understand the subroutine interfaces and communications services provided by CIMPLICITY software's Alarm Management capability.
- Understand the configuration requirements and file formats for Alarm Management.
- Code appropriate applications programs.
- Compile and link the programs as explained in this section.

How the Alarm Viewer API Works

As an aid in understanding how the `CAmvConn` uses the callback functions, this section gives a brief description of the order and context of execution. The order provided here represents a very high-level description of operation and includes only those details believed relevant to writing a custom alarm viewer.

```
WHEN constructor called
   Start trying to form connection
WHEN connection formed
   CALL DoConnectionFormed(context);
WHEN connection lost to AMRP
   Call LostAM(context);
WHEN connection lost to RCM
   Call DoRcmError(context, state);
WHEN AMRP sends count or date change
   Call UpdateCount(context, countInfo);
WHEN viewer calls RequestAlarms()
   Ask AMRP for all current alarms
WHEN viewer requests dynamic mode
   Call MaxAlarms(context);
WHEN viewer calls UpdateList()
   FOR EACH alarm with pending action
      send update to AMRP
   ENDFOR
WHEN AMRP sends current alarms (in static AND dynamic mode)
   CALL SetDisplayRedraw(context, FALSE);
  CALL ClearDisp(context);
   FOR EACH alarm from AMRP
      build AlarmInfo
```

```
      CALL dispFunc(context, AlarmInfo);
   ENDFOR
   Call SetDisplayRedraw(context, TRUE);
IF in dynamic mode
   WHEN alarm notification received from AMRP
      build AlarmInfo
      SWITCH (notification type)
         CASE generate:
             Call NotifyAlmGen(context, AlarmInfo);
             break;
         CASE modify:
             Call NotifyAlmMod(context, AlarmInfo, action);
             break;
          CASE delete:
             Call NotifyAlmDel(context, AlarmInfo);
             break
      ENDSWITCH
ENDIF
```

Application Subroutine Interface Contents

The following is a list of all files distributed with the Alarm Management API. The files are loaded into the directories indicated. The environment variable %BSM_ROOT% points to the base directory where CIMPLICITY software was installed.

Include files in %BSM_ROOT%\api\include\inc_path are:

```
am_defs.h
am_errors.h
amaru_err.h
amaru_proto.h
cor.h
cor_event.h
cor_os.h
cor_stat.h
ddl.h
ipcerr.h
netcom.h
sc_recs.h
```

Source files in %BSM_ROOT\api\amvtest are:

```
amvtest_exe.vcxproj
makefile
amvtest.h
main.cpp
amvtest.cpp
```

Libraries in %BSM_ROOT%\api\lib are:

```
amaru.lib
cim_mf.lib
ddl.lib
corutil.lib
fasrtl.lib
ipc.lib
cim_sc.lib
```

Build an Alarm Manager Connection

Build an Alarm Manager Connection

Each `CAmvConn` instance can only communicate with one Alarm Manager (for one project) at a time. However, your program can construct and maintain multiple `CAmvConn` objects to view alarms from multiple projects at once.

CAmvConn Syntax

CAmvConn Syntax

The syntax for the constructor is:

```
CAmvConn(
   void* who
   void (*dispFunc)(struct testContext *context,
                            struct AlarmInfo* pAI),
   void (*ClearDisp)(struct testContext *context),
   void (*LostAM)(struct testContext *context),
   int (*MaxAlarms)(struct testContext *context),
   void (*SetDisplayRedraw)(struct testContext *context, int val),
   void (*UpdateCount)(struct testContext *context,
                           RCM_ALARM_DATA *alarmData),
   void (*DoRcmError)(struct testContext *context, int state),
   void (*DoConnectionFormed)(struct testContext *context),
   void (*NotifyAlmGen)(struct testContext *context,
                           struct AlarmInfo* pAI) = 0,
   void (*NotifyAlmMod)(struct testContext *context,
                           struct AlarmInfo* pAI,
                           int alm_mod_action) = 0,
void (*NotifyAlmDel)(struct testContext *context,
                           struct AlarmInfo* pAI) = 0);
```

Constructor Members Summary

The following table summarizes the members of the constructor and their purpose:

Note that some functions are optional. If you do not need to implement an optional function, pass NULL or 0 in place of its pointer.

Argument	Description
Who	This pointer is maintained but never used by the `CAmvConn`. Typically, this is a pointer to the object which owns the `CAmvConn` instance. The callback functions can use this pointer to access members of the owner object.
DispFunc	`CAmvConn` calls this function to notify the viewer of new alarm data.
ClearDisp	`CAmvConn` calls this function when it removes all alarms from its local storage, for example in preparations for receipt of a new static list of alarms. The viewer should remove all its locally-maintained information about current alarms and typically clear the displayed list of alarms.
LostAM	`CAmvConn` calls this function when communication with the Alarm Manager (AM) is lost.
MaxAlarms	This is an optional function. `CAmvConn` calls this function in dynamic mode to query the viewer about how many alarms it can handle. Typically, this routine returns a very large value.
SetDisplayRedraw	`CAmvConn` calls this function with val set to FALSE before calling `dispFunc`, and again with val set to TRUE after calling `dispFunc`. Typically, when called with val set to TRUE, this function is responsible for redrawing the alarm view or sending a message to the application to cause a redraw.
UpdateCount	`CAmvConn` calls this function to notify the viewer that the alarm count or date of last change has changed.
DoRcmError	`CAmvConn` calls this function when a Remote Connection Manager error occurs. All locally-maintained information about alarms and the connection should be reset.
DoConnectionFormed	`CAmvConn` calls this function when the connection to the AMRP is complete. Due to the multi-threaded nature of the `CAmvConn` processing, this can be some time after the `CAmvConn` constructor completes.
NotifyAlmGen	`CAmvConn` calls this function in dynamic mode to notify the viewer when a new alarm has been generated or an existing alarm has been regenerated.
NotifyAlmMod	`CAmvConn` calls this function in dynamic mode to notify the viewer of the change of state of an existing alarm (for example, when an alarm is acknowledged.)
NotifyAlmDel	This is an optional function.
	`CAmvConn` calls this function in dynamic mode to notify the viewer when an alarm has been deleted.

testContext Structure

All of the callback functions take as their first argument a structure describing the context of the call:

```
struct testContext {
   void *who
   void *amapConn;
   void (*dispFunc)(struct testContext *context,
                    struct AlarmInfo* pAI);
   void (*ClearDisp)(struct testContext *context);
   void (*LostAM)(struct testContext *context);
   int (*MaxAlarms)(struct testContext *context);
```

```
   void (*SetDisplayRedraw)(struct testContext *context, int val);
   void (*UpdateCount)(struct testContext *context,
                       RCM_ALARM_DATA *alarmData);
   void (*DoRcmError)(struct testContext *context, int state);
   void (*DoConnectionFormed)(struct testContext *context);
   SAmapCallbacks AmapCallbacks ;
};
```

The following table summarizes the fields on the **testContext** structure and their meaning:

Field	Description
Who	A pointer to the object which owns the CAmvConn instance
AmapConn	AMAP connection instance
DispFunc	Function that adds an alarm
ClearDisp	Function that removes all alarms
LostAM	Function to notify of loss of AM connection
MaxAlarms	Returns the maximum number of dynamic alarms
SetDisplayRedraw	Should display be updated?
UpdateCount	Function to notify of new alarm count or update time
DoRcmError	Function to notify of lost of RCM connection
DoConnectionFormed	Function to notify of connection completion
AmapCallbacks	Structure with pointers to dynamic mode callback functions

By casting the who pointer to the appropriate type, members of the owner class can be accessed. By casting the `amapConn` pointer as a `CAmvConn*`, members of the connection can be accessed. For example:

```
_tprintf(_T("Project:\t%s\n"),
          (CAmvConn*)(testContext->amapConn)->GetConnectedSystem());
```

Thus the same callback functions can be used to process all the connections for your viewer and can determine which connection they are being called for through the amapConn pointer in the context.

SAmapCallbacks Structure

The `SAmapCallbacks` member of the context lists the dynamic-mode functions:

```
Typedef struct tagAmapCallbacks {
   struct testContext *client_data;
   void (*NotifyAlmGen)(struct testContext *context,
                        struct AlarmInfo* pAI);
void (*NotifyAlmMod)(struct testContext *context,
                        struct AlarmInfo* pAI,
                        int alm_mod_action);
```

```
void (*NotifyAlmDel)(struct testContext *context,
                          struct AlarmInfo* pAI);
} SAmapCallbacks, *PSAmapCallbacks ;
```

The following table summarizes the fields on the `SAmapCallbacks` structure and their meaning:

Field	Description
Client_data	A pointer to the object which owns the CAmvConn instance
NotifyAlmGen	Function to notify the viewer when a new alarm has been generated or an existing alarm has been regenerated.
NotifyAlmMod	Function to notify the viewer of the change of state of an existing alarm.
NotifyAlmDel	Function to notify the viewer when an alarm has been deleted.

AlarmInfo Structure

Callback functions which must process alarm information also take a pointer to an `AlarmInfo` structure:

```
struct AlarmInfo {
   TCHAR systemName[RTR_SYSNAME_SIZE+1];
   TCHAR classId[CLASS_ID_LEN+1];
   TCHAR frId[FR_ID_LEN+1];
   TCHAR alarmId[ALARM_ID_LEN+1];
   TCHAR refId[ AM_REF_ID_LEN+1 ];
   COR_STAMP genTime;
   TCHAR durationTime[COR_ASCII_TIME_LEN+1];
   int numComments;
   int numStacked;
   TCHAR message[ALARM_MSG_LEN+1];
   TCHAR action[2+1];
   int state;
   TCHAR stateStr[19+1];
   TCHAR ackState[4+1];
   COR_U1 fgColor;
   COR_U1 bgColor;
   int classOrder;
   long *alarmRecord;
   TCHAR del_opt[DEL_OPT_LEN+1];
   TCHAR manual_clear_allowed;
   COR_I4 seq_num;
   COR_I4 generated_time;
   COR_I4 cleared_time;
   COR_BOOLEAN ConcedAlarm;
   COR_I4 amrp_sync;
   COR_I4 max_stacked;
   TCHAR systemResId[FR_ID_LEN+1];
   TCHAR alarm_description[SC_DESCRIPTION_LEN+1];
.
.
```

```
.
#if AI_PTRS
    AM_STACKED_INFO  *pstacked_info;
#else
    AM_STACKED_INFO     pstacked_info[AM_MAX_STACKED];
#endif
    COR_BOOLEAN        info_inited;
#if AI_PTRS
    AM_COMMENT2_INFO *pstacked_com;
#else
    //AM_COMMENT2_INFO  pstacked_com[AM_MAX_ALARM_COMMENTS];
#endif
    COR_BOOLEAN       comnt_inited;
    TCHAR long_name[LONG_NAME_LEN+1];
.
.
.
};
```

Please note that this structure has some methods, and cannot be treated simply as a series of bytes; it must be constructed and destructed appropriately.

The following table summarizes the fields on the AlarmInfo structure and their meaning:

Field	Description
SystemName	Name of the project the alarm is from.
ClassID	The alarm's class ID.
FrId	The alarm's resource ID.
AlarmId	The alarm's ID. This value should not be printed, use `long_name`.
RefId	The alarm's reference ID.
GenTime	The time the alarm was generated in COR_STAMP format.
DurationTime	The alarm's duration time, presented as an ASCII string.
NumComments	The number of comments on the alarm.
NumStacked	The number of stacked alarm instances.
Message	The alarm message.
Action	Pending actions on the alarm.
State	The numeric state of the alarm: AM_GENERATED, AM_ACKNOWLEDGED, AM_CLEARED.
StateStr	The string representing the current state of the alarm: "Alarm" or "Normal."
AckState	Has the alarm been acknowledged? "Y" or "N."
FgColor	Foreground color for the current alarm state.
BgColor	Background color for the current alarm state.

Field	Description
ClassOrder	The priority of the alarm.
AlarmRecord	Reserved for GE Intelligent Platforms use.
Del_opt	Deletion requirements for the alarm.
Manual_clear_allowed	Can alarm be manually cleared.
Seq_num	Reserved for GE Intelligent Platforms use.
Generated_time	The time the alarm was generated.
Cleared_time	The time the alarm was cleared.
ConcedAlarm	Reserved for GE Intelligent Platforms use.
Amrp_sync	Time the AMRP sent the alarm to the Alarm Viewer.
Max_stacked	The maximum number of instances of an alarm that can be stacked.
SystemResId	Reserved for GE Intelligent Platforms use.
Pstacked_info	Pointer to an array of alarm data.
Pstacked_com	Pointer to an array of comment data.
Info_inited	Reserved for GE Digital use.
Comnt_inited	Reserved for GE Digital use.
long_name	Full Alarm ID
alarm_description	The description of the alarm.

Alarm Viewer API Sample Program

Alarm Viewer API Sample Program

This topic describes how to Build and Run the Demo (sample) program. A sample Microsoft Visual C++ project, amvtest_exe.vcxproj, is provided to build the sample program. Use this project as a basis for constructing projects for your own applications.

Note: Depending on how you installed Visual C++, the INCLUDE, LIB, and PATH environment variables may not be automatically set when you install MSDEV. If they are not set, you will have to set them manually or run the following to set them before building any user programs.

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft Visual
 Studio\Installer\vswhere.exe" -property installationPath`) do set VSPATH=
%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

When you run the demo program, it requests class, resource and alarm data from the AMRP, then displays the requested information.

To build the sample program, do the following:

1. Click Tools>Command Prompt on your project's CIMPLICITY Workbench menu bar.

   This will ensure that your environment variables (in particular %BSM_ROOT% and %SITE_ROOT%) are set correctly.

2. In the Command Prompt window, issue the following commands:

   `cd` <drive>

   `cd %BSM_ROOT%\api`

   Where <drive> is the disk where your CIMPLICITY software is installed.

3. If the environment variables are not set automatically, issue the following command to set them:

   ```
   for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
    Visual Studio\Installer\vswhere.exe" -property installationPath`) do
    set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
   ```

4. Now start Visual Studio:

   devenv CimplicityAPI.sln

5. Open the Solution Explorer.

6. Right click amvtest_exe.

7. Select **Build** on the Popup menu.

   Run the Demo (Sample) Program

   The API process name must be stored in the PRCNAM environment variable for the program to run. The name is an arbitrary character string of up to 10 characters. To create PRCNAM, enter the following command in the Command Prompt window:

   `set PRCNAM=`<name>

   where <name> is the API process name.

   To run the sample program, enter the following command in the Command Prompt window:

`amvtest`

You will be prompted for a project name and asked if you want to run in dynamic mode.

Once a connection to the AMRP has been formed, the test program will print out the Classes and Resources for which alarms will be processed. It will then request a list of current alarms from AMRP and print them out.

If running in dynamic mode, the program will wait for updates from AMRP and print them out as they are received.

To end the sample program, type `EXIT` and press return.

Alarm Viewer API Sample Program Files

The sample alarm viewer is contained in three files:

File	Description		
Amvtest.h	Contains the definition of two classes used in the sample		
	`CAmvTest`	The viewer class. It has a alarm manager connection object (CAmvConn) as one of its members and member functions which are used as AMV API callback functions.	
		The data members of the **CAmvTest** class are:	
		Member	**Description**
		m_listAlarms	The list of current alarms.
		m_pAmvConn	The connection to the alarm manager.
		m_bConnected	A flag indicating if there is a connection.
		m_bDynamic	A flag indicating if the connection is in dynamic mode.
		The `CAmvTest` constructor is responsible for initializing `m_pAmvConn` by allocating and initializing an instance of the `CAmvConn` class from the API. The static member functions in the second section are the AMV API callbacks. The `CAmvTest` constructor uses them when constructing a new `CAmvConn`. The `CAmvTest` destructor deletes this instance when the `CAmvTest` is deleted. The other `CAmvTest` methods are: RunTest - The "main program" for the class. It forms a connection, prints the class and resource filters, requests alarms, prints current alarms, and, for dynamic mode, sits waiting for alarm updates from AMRP. PrintClassFilters - Prints the class filters for the current connection. PrintResourceFilters - Prints the resource filters for the current connection.	
	`CAlarmList`	A class derived from MFC's CPtrList. It is used by CAmvTest to store alarms.	
		The `CAlarmList` class depends on `CPtrList` for most of its functionality. See the Microsoft Visual C++ documentation for more information on `CPtrList` methods. `CAlarmList` adds methods for functions specific to the alarm viewer application:	

<table>
<tr><th>File</th><th colspan="3">Description</th></tr>
<tr><td></td><td></td><td colspan="2">The <code>CAlarmList</code> class depends on <code>CPtrList</code> for most of its functionality. See the Microsoft Visual C++ documentation for more information on <code>CPtrList</code> methods. <code>CAlarmList</code> adds methods for functions specific to the alarm viewer application:</td></tr>
<tr><td></td><td></td><td>Member</td><td>Description</td></tr>
<tr><td></td><td></td><td>CAlarmList</td><td>Class constructor.</td></tr>
<tr><td></td><td></td><td>~CAlarmList</td><td>Class destructor.</td></tr>
<tr><td></td><td></td><td>SetRedraw</td><td>Control whether or not DrawList() actually draws.</td></tr>
<tr><td></td><td></td><td>DrawList</td><td>Print the alarms in the list.</td></tr>
<tr><td></td><td></td><td>DeleteAll</td><td>Take each alarm out of the list and delete it</td></tr>
<tr><td>Amvtest.cpp</td><td colspan="3">Contains the implementation of the two classes</td></tr>
<tr><td>Main.cpp</td><td colspan="3">A short program that creates an instance of one of the objects and executes it's test method.</td></tr>
<tr><td></td><td><code>CAlarmList</code></td><td colspan="2">A class derived from MFC's CPtrList. It is used by CAmvTest to store alarms.</td></tr>
</table>

All three files can be found in the %BSM_ROOT%\api\amvtest directory.

AMV API Class Reference

AMV API Class Reference

The CIMPLICITY Alarm Viewer API lets application programs access the functions of CIMPLICITY software's Alarm Management Application Module.

The classes included in the AMV API include:

- CAmvAlarm Class
- CAmvClassFilter
- CAmvClassFilterList
- CAmvConn
- CAmvFieldFilter
- CAmvFieldFilterList
- CAmvResourceFilter
- CAmvResourceFilterList
- CAmvSetupList
- CAmvStateFilter
- CAmvStateFilterList
- CAmvTimeFilter

CAmvAlarm

CAmvAlarm Class

Each alarm generated in the CIMPLICITY system is represented by an instance of the `CAmvAlarm` class. The alarm has a number of properties including:

- An indication of the static severity of the alarm. Standard classes are HIGH, MEDIUM, and LOW
- The resource for the point or device which the alarm is for
- The times the alarm was generated and cleared
- Under what circumstances the alarm may be deleted from the database
- An indication of whether or not the alarm may be manually cleared
- Help file name - Alarms may have help files which can be displayed to operators to aid in resolving the problem which caused the alarm
- Number of stacked instances of this alarm

Note: A number of AMV API routines return pointers to `CAmvAlarm` objects. You should never need to create one in your application.

CAmvAlarm Class Definition

CAmvAlarm Class Definition

The following class definition represents the `CAmvAlarm` class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvAlarm {
public:
   COR_I4 curr_stacked (page 188) ;
   AM_STACKED_INFO stacked_data[AM_MAX_STACKED+1] (page 193) ;
   COR_I4 curr_comment (page 188) ;
   AM_COMMENT_INFO stacked_com[AM_MAX_ALARM_COMMENTS] (page 193) ;
   COR_I4 generated_time (page 190) ;
   COR_I4 cleared_time (page 187) ;
   COR_I4 amrp_sync (page 185) ;
   COR_I4 amrp_sync_offset (page 186) ;
   COR_I4 max_stacked (page 192) ;
   TCHAR* ID(TCHAR* id_buf) (page 191) ;
   TCHAR* DeleteOptions() (page 189) ;
   COR_BOOLEAN ManualClearAllowed() (page 192) ;
   CAmvClassFilter* Class() (page 186) ;
};
```

CAmvAlarm Class Member Overview

The following table briefly describes the members of the `CAmvAlarm` class. The sections that follow go into detail of the syntax and semantics of using the members.

Member		Description
	Amrp_sync	The time the AMRP sent the alarm to the viewer.
	Amrp_sync_offset	The difference between the time the AMRP sent the alarm and the time the AMV received the alarm. Assuming staticly fast inter-process communication, this is the number of seconds difference between the clocks on the systems where the AMRP and viewer reside.
	Class()	Class filter for the alarm.
	Cleared_time	Time the alarm was cleared.
	Curr_comment	Alarm comments.
	Curr_stacked	Number of alarm instances currently stacked.
	DeleteOptions()	State requirements for alarm deletion.
	Generated_time	Time the alarm was generated.
	ID()	Alarm ID.
	ManualClearAllowed()	Can alarm be manually cleared.
	Stacked_com	Number of alarm comments.
	Stacked_data	Current and stacked alarm instances.

CAmvAlarm::amrp_sync

This field contains the time the AMRP sent the alarm to the viewer.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->amrp_sync;
```

Data Type

```
COR_I4
```

Example

This example calculates the number of hours and minutes an alarm was or has been in alarm state.

```
COR_I4 duration;
COR_I4 minutes;
COR_I4 seconds;
```

```
long ourtime;
cor_time_get_current_local(&ourtime);
if (alarm_ptr->cleared_time == 0)
   duration = ourtime +
      alarm_ptr->amrp_sync_offset -
         alarm_ptr->generated_time;
else
   duration = alarm_ptr->cleared_time -
      alarm_ptr->generated_time;
minutes = duration / 60;
seconds = duration - ( minutes * 60 );
```

See Also

CAmvAlarm::amrp_sync_offset *(page 186)*

CAmvAlarm::amrp_sync_offset

This field contains the difference between the time the AMRP sent the alarm and the time the alarm viewer received the alarm.

Assuming staticly fast inter-process communication, this is the number of seconds difference between the clocks on the systems where the AMRP and viewer reside.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->amrp_sync_offset;
```

Data Type

```
COR_I4
```

See Also

CAmvAlarm::amrp_sync *(page 185)*

CAmvAlarm::Class()

Class for the alarm.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->Class();
```

Data Type

```
CAmvClassFilter*
```

Example

This example prints an alarm ID and its class.

```
CAmvAlarm* alarm_ptr;
TCHAR id_buf[CLASS_ID_LEN+1];
_tprintf(_T("Alarm %s is class %s\n"),
         alarm_ptr->ID(),
         alarm_ptr->Class()->ID(id_buf));
```

See Also

CAmvClassFilter *(page 195)*

CAmvAlarm::cleared_time

This field contains the time the alarm was cleared.

Comments

If the alarm has not been cleared, this field contains a zero (0).

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->cleared_time;
```

Data Type

```
COR_I4
```

Example

This example calculates the number of hours and minutes an alarm was or has been in alarm state.

```
COR_I4 duration;
COR_I4 minutes;
COR_I4 seconds;
long ourtime;
cor_time_get_current_local(&ourtime);
if (alarm_ptr->cleared_time == 0)
   duration = ourtime +
      alarm_ptr->amrp_sync_offset -
```

```
         alarm_ptr->generated_time;
else
   duration = alarm_ptr->cleared_time -
      alarm_ptr->generated_time;
minutes = duration / 60;
seconds = duration - ( minutes * 60 );
```

See Also

CAmvAlarm::generated_time *(page 190)*

CAmvAlarm::curr_comment

This field contains the number of operator comments on the alarm.

Comments

The data member **stacked_com[]** has **curr_comment** elements in it.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->curr_comment;
```

Data Type

```
COR_I4
```

Example

This example enables or disables a button to display operator comments based on the presence of comments for the alarm.

```
CAmvAlarm* alarm_ptr;
int i;
if (alarm_ptr->curr_comment > 0) {
   EnableCommentButton();
} else {
   DisableCommentButton();
}
```

See Also

CAmvAlarm::stacked_com *(page 193)*

CAmvAlarm::curr_stacked

If an alarm happens multiple times before it is acknowledged by an operator, the instances of the alarm may be "stacked" so that the history of the alarm may be viewed.

This field contains the current number of stacked alarms.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->curr_stacked;
```

Data Type

```
COR_I1
```

Example

This example prints an alarm messages and, if there are any stacked alarms, it prints the number of stacked alarm instances.

```
CAmvAlarm* alarm_ptr;
_tprintf(_T("%s"), alarm_ptr->->stacked_data[0].alarm_msg);
if (alarm_ptr->curr_stacked > 1) {
   _tprintf(_T(" (%d)"), alarm_ptr->curr_stacked);
}
_tprintf(_T("\n"));
```

See Also

CAmvAlarm::stacked_data *(page 193)*

CAmvAlarm::DeleteOptions()

This field contains the criteria for deleting an alarm instance. The criteria can be one or both of the following:

AM_ACK_CHAR	Alarm was acknowledged by operator.
AM_CLR_CHAR	Alarm was cleared (reset).

Comments

- Alarms are deleted only if they meet the specified deletion criteria. The criteria are that the alarm has been cleared (returned to normal state) or acknowledged.
- An alarm with delete options of AM_ACK_CHAR (acknowledged) is deleted when the operator acknowledges it, even if it has not been cleared.

- An alarm with delete options of AM_CLR_CHAR (cleared) is automatically deleted when it clears even if it has not been acknowledged.
- An alarm with delete options of AM_ACK_CHAR and AM_CLR_CHAR must be cleared and acknowledged before it can be deleted.

Syntax

```
CAmvAlarm* alarm_ptr
alarm_ptr->DeleteOptions()
```

Data Type

```
TCHAR*
```

Example

This example sets a verbose prompt for the user to tell them what the deletion requirements are.

```
if ((_tcschr(alarm_ptr->DeleteOptions(), AM_ACK_CHAR) != NULL)
&&  (_tcschr(alarm_ptr->DeleteOptions(), AM_CLR_CHAR) != NULL))
     Requirement = CAmvStateFilter::ack_clear_msg();
else if
  ((_tcschr(alarm_ptr->DeleteOptions(), AM_ACK_CHAR) != NULL)
  && (_tcschr(alarm_ptr->DeleteOptions(), AM_CLR_CHAR) == NULL))
  Requirement = CAmvStateFilter::ack_only_msg();
else if
  ((_tcschr(alarm_ptr->DeleteOptions(), AM_ACK_CHAR) == NULL)
  && (_tcschr(alarm_ptr->DeleteOptions(), AM_CLR_CHAR) != NULL))
  Requirement = CAmvStateFilter::clear_only_msg();
```

See Also

```
            CAmvStateFilter::ack_clear_msg (page 241)
 , CAmvStateFilter::ack_only_msg (page 241)
 , CAmvStateFilter::clear_only_msg (page 242)
```

CAmvAlarm::generated_time

Time the alarm was generated.

Comments

Time is recorded in seconds since midnight (00:00) on 1 January 1970.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->generated_time;
```

Data Type

```
COR_I4
```

Example

This example calculates the number of hours and minutes an alarm was or has been in alarm state.

```
COR_I4 duration;
COR_I4 minutes;
COR_I4 seconds;
long ourtime;
cor_time_get_current_local(&ourtime);
if (alarm_ptr->cleared_time == 0)
   duration = ourtime +
      alarm_ptr->amrp_sync_offset -
         alarm_ptr->generated_time;
else
   duration = alarm_ptr->cleared_time -
      alarm_ptr->generated_time;
minutes = duration / 60;
seconds = duration - ( minutes * 60 );
```

See Also

CAmvAlarm::cleared_time *(page 187)*

CAmvAlarm::ID(id_buf)

This field contains the pointer to the buffer that contains the Alarm ID string.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->ID(TCHAR* id_buf);
```

Data Type

```
TCHAR*
```

Example

This example prints the ID of an alarm.

```
CAmvAlarm* alarm_ptr;
TCHAR id_buf[ALARM_ID_LEN+1];
_tprintf(_T("%s\n"), alarm_ptr->ID(id_buf);
```

CAmvAlarm::ManualClearAllowed()

This field indicates whether an operator can reset this alarm from the Alarm Viewer display. It contains one of the following values:

TRUE	The operator can reset the alarm.
FALSE	The operator cannot reset the alarm.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->ManualClearAllowed();
```

Data Type

```
COR_BOOLEAN
```

Example

This example displays an error message if manual reset is not allowed.

```
if (!alarm_ptr->ManualClearAllowed()) {
   MessageBox("Alarm may not be manually cleared\n");
}
```

CAmvAlarm::max_stacked

This field contains the maximum number of alarms that can be stacked.

Syntax

```
CAmvAlarm* alarm_ptr;
alarm_ptr->max_stacked;
```

Data Type

```
COR_I4
```

CAmvAlarm::stacked_com

This field contains an array of AM_COMMENT_INFO structures.

Each AM_COMMENT_INFO structure records a comment entered for an alarm by a user along with the time the comment was generated.

The number of comments currently associated with the alarm is found in **CAmvAlarm::curr_comment** .

Syntax

```
AM_COMMENT_INFO stacked_com[AM_MAX_ALARM_COMMENTS];
```

Data Type

```
typedef struct am_comment_info {
   COR_STAMP gentime;
   TCHAR alarm_comment[AM_COMMENT_LEN+1];
} AM_COMMENT_INFO;
```

Example

This example prints the comments for an alarm and the time the comments were created.

```
CAmvAlarm* alarm_ptr;
int i;
for (i=0; i < alarm_ptr->curr_comment; i++)
{
   _tprintf(_T("%ld %ld\t%s"),
            alarm_ptr->stacked_com[i].genTime.yyyymmdd,
            alarm_ptr->stacked_com[i].genTime.hhmmsstt,
            alarm_ptr->stacked_com[i].alarm_comment);
}
```

See Also

CAmvAlarm::curr_comment *(page 188)*

CAmvAlarm::stacked_data

This field contains an array of AM_STACKED_INFO structures.

Each AM_STACKED_INFO structure records the generation time, current alarm state, and alarm message for an instance of the alarm.

The number of instances currently associated with the alarm is found in **CAmvAlarm::curr_stacked** .

Comments

stacked_data[0] is always the most recent alarm.

Syntax

```
AM_STACKED_INFO stacked_data[AM_MAX_STACKED+1
```

Data Type

```
typedef struct am_stacked_info {
   COR_STAMP gentime;
   AM_STATE_TYPE alarm_state;
   TCHAR alarm_msg[ALARM_MSG_LEN+1];
} AM_STACKED_INFO;
```

Example

This example prints the stacked instances for an alarm and the time the instances occurred.

```
CAmvAlarm* alarm_ptr;
int i;
for (i=0; i < alarm_ptr->curr_stacked; i++)
{
   _tprintf(_T("%ld %ld\t%s"),
            alarm_ptr->stacked_data[i].genTime.yyyymmdd,
            alarm_ptr->stacked_data[i].genTime.hhmmsstt,
            alarm_ptr->stacked_data[i].alarm_msg);
}
```

See Also

```
            CAmvAlarm::curr_stacked (page 188)
```

CAmvClassFilter

CAmvClassFilter

A `CAmvClassFilter` controls whether alarms of the corresponding class will be passed from AMRP to the viewer.

CAmvClassFilter Class Definition

CAmvClassFilter Class Definition

The following class definition represents the `CAmvClassFilter` class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvClassFilter {
public:
  TCHAR class_title[CLASS_TITLE_LEN+1] (page 199) ;
  int class_order (page 198) ;
  COR_I2  class_alarm_fg (page 197) ;
  COR_I2  class_alarm_bg (page 197) ;
  COR_I2  class_normal_fg (page 198) ;
  COR_I2  class_normal_bg (page 197) ;
  COR_I2  class_ack_fg (page 196) ;
  COR_I2  class_ack_bg (page 196) ;
  TCHAR* ID(TCHAR* id_buf) (page 200) ;
  BOOL IsEnabled (page 201) ;
  void Enable (page 200) ;
  void Disable (page 199) ;
};// CAmvClassFilter
```

CAmvClassFilter Class Member Overview

The following table briefly describes the members of the `CAmvClassFilter` class. The sections that follow go into detail of the syntax and semantics of using the members.

Member		Description
	class_ack_bg	Background color for acknowledged alarms
	class_ack_fg	Foreground color for acknowledged alarms
	class_alarm_bg	Background color for alarm message in alarm state
	class_alarm_fg	Foreground color for alarm message in alarm state
	class_normal_bg	Background color for cleared, unacknowledged alarms
	class_normal_fg	Foreground color for cleared, unacknowledged alarms

	class_order	Alarm class priority. 0 is highest.
	class_title	Alarm class description (for example, "High Priority Alarms")
	Disable()	Don't have AMRP send alarms for this class to the viewer
	Enable()	Have AMRP send alarms for this class to the viewer
	ID()	Class ID (for example, "HIGH")
	IsEnabled()	Will alarms for this class be sent to the viewer by the AMRP

CAmvClassFilter::class_ack_bg

This member specifies the background color for alarms that are in the acknowledged state.

Comments

This color is a configured CIMPLICITY Alarm Class in the Workbench.

This member is read-only.

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_ack_bg;
```

Data Type

```
COR_I2
```

CAmvClassFilter::class_ack_fg

This member specifies the foreground color for alarms which are in the acknowledged state.

Comments

This color is a configured CIMPLICITY Alarm Class in the Workbench.

This member is read-only

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_ack_fg;
```

Data Type

```
COR_I2
```

CAmvClassFilter::class_alarm_bg

This member specifies the background color for alarms which are in the alarm state.

Comments

This color is a configured CIMPLICITY Alarm Class in the Workbench.

This member is read-only

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_alarm_bg;
```

Data Type

```
COR_I2
```

CAmvClassFilter::class_alarm_fg

This member specifies the foreground color for alarms which are in the alarm state.

Comments

This color is a configured CIMPLICITY Alarm Class in the Workbench.

This member is read-only

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_alarm_fg;
```

Data Type

```
COR_I2
```

CAmvClassFilter::class_normal_bg

This member specifies the background color for alarms that are in the normal state.

Comments

This color is a configured CIMPLICITY Alarm Class in the Workbench.

This member is read-only

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_normal_bg;
```

Data Type

```
COR_I2
```

CAmvClassFilter::class_normal_fg

This member specifies the foreground color for alarms that are in the normal state.

Comments

This color is a configured CIMPLICITY Alarm Class in the Workbench.

This member is read-only

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_normal_fg;
```

Data Type

```
COR_I2
```

CAmvClassFilter::class_order

The static priority of the alarm class. Zero is the highest priority.

Comments

This is a read-only field.

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_order;
```

Data Type

```
int
```

CAmvClassFilter::class_title

The alarm class description as configured in Alarm Classes from the Workbench.

Comments

This member is read-only and should not be modified.

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->class_title;
```

Data Type

```
TCHAR[CLASS_TITLE_LEN+1]
```

Example

This example prints the ID and description of all the alarm classes for which alarms will be displayed.

```
CAmvConn* AmvConn;
CAmvClassFilter* class_ptr;
TCHAR id_buf[CLASS_ID_LEN+1];
for (class_ptr = AmvConn->ClasssFilters->First();
     class_ptr != NULL;
     class_ptr = AmvConn->ClassFilters->Next(class_ptr)) {
   if (class_ptr->IsEnabled()) {
     _tprintf(_T("%s\t%s\n"),
                class_ptr->ID(id_buf),
                class_ptr->class_title);
   }
}
```

CAmvClassFilter::Disable()

Disable the alarm class filter so that AMRP does not send alarms of this class to the viewer.

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->Disable();
```

Data Type

```
void
```

See Also

```
CAmvClassFilter::Enable() (page 200)
```

CAmvClassFilter::Enable()

Enable the alarm class filter so that AMRP sends alarms of this class to the viewer.

Syntax

```
CAmvClassFilter* class_ptr;
class_ptr->Enable();
```

Data Type

```
void
```

See Also

```
CAmvClassFilter::Disable() (page 199)
```

CAmvClassFilter::ID()

Retrieves the class ID for the alarm class and puts it in the user-supplied buffer. Returns a pointer to that buffer.

Syntax

```
CAmvClassFilter* class_ptr;
TCHAR id_buf[CLASS_ID_LEN+1];
```

```
class_ptr->ID(id_buf);
```

Data Type

```
TCHAR*
```

CAmvClassFilter::IsEnabled()

Returns TRUE if alarms for this class will be sent to the viewer, FALSE otherwise.

Syntax

```
CAmvFilter* filter_ptr;
filter_ptr->IsEnabled()
```

Data Type

```
BOOL
```

CAmvClassFilterList

CAmvClassFilterList

The `CAmvClassFilterList` class provides methods to loop through all the class filters currently in use.

The list is built from data supplied by the AMRP. Filters cannot be added to, or removed from, the list.

CAmvClassFilterList Class Definition

CAmvClassFilterList Class Definition

The following class definition represents the `CAmvClassFilterList` class.

```
class CAmvClassFilterList {
public:
   CAmvClassFilter* First (page 202) (COR_STATUS* ret_stat);
   AmvClassFilter* Next (page 203) (CAmvClassFilter* filter,
   COR_STATUS* ret_stat);
};
```

CAmvClassFilterList Class Member Overview

The following table briefly describes the members of the `CAmvClassFilterList` class. The sections that follow go into detail of the syntax and semantics of using the members.

Member		Description
	First	Get a pointer to the first filter in the list
	Next	Get a pointer to the next filter in the list

CAmvClassFilterList::First()

Returns a pointer to the first class filter in the list.

If the filter list or the connection is in an invalid state, the return value is NULL and the ret_stat->status is COR_FAILURE.

Syntax

```
CAmvClassFilter* class_ptr;
CAmvClassFilterList* class_list;
class_ptr = class_list->First(&ret_stat);
```

Data Type

```
CAmvClassFilter*
```

Example

Print ID for all known classes.

```
CAmvConn* AmvConn;
CAmvClassFilter* class_ptr;
TCHAR id_buf[CLASS_ID_LEN+1];
for (class_ptr = AmvConn->ClasssFilters->First(&ret_stat);
   class_ptr != NULL && ret_stat.status == COR_SUCCESS;
   class_ptr = AmvConn->ClassFilters->Next(class_ptr,
```

&ret_stat))

```
{
   _tprintf(_T("%s\n"), class_ptr->ID(id_buf));
}
```

See Also

CAmvClassFilterList::Next() *(page 203)* , CAmvClassFilter *(page 195)*

```
```

CAmvClassFilterList::Next()

Returns a pointer to the next class filter in the list.

Syntax

```
CAmvClassFilter* class_ptr;
CAmvClassFilterList* class_list;
class_ptr = class_list->Next(class_ptr, &ret_stat);
```

Data Type

```
CAmvClassFilter*
```

See Also

```
            CAmvClassFilterList::First() (page 202) , CAmvClassFilter
 (page 195)
```

CAmvConn

CAmvConn

The connection between an alarm viewer and the Alarm Manager is implemented via the `CAmvConn` class. When you create a new instance of this class, you provide a number of call-back functions which are called in response to events such as new data from the Alarm Manager or loss of connection.

CAmvConn Class Definition

CAmvConn Class Definition

The following class definition represents the `CAmvConn` class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvConn {
public:
   CAmvConn(
      void* who
      void (*dispFunc)(struct testContext *context,
                       struct AlarmInfo *pAI),
```

```
      void (*ClearDisp)(struct testContext *context),
      void (*LostAM)(struct testContext *context),
      int (*MaxAlarms)(struct testContext *context),
      void (*SetDisplayRedraw)(struct testContext *context,
                               int val),
      void (*UpdateCount)(struct testContext *context,
                          RCM_ALARM_DATA *alarmData),
      void (*DoRcmError)(struct testContext *context, int state),
      void (*DoConnectionFormed)(struct testContext *context),
      void (*NotifyAlmGen)(struct testContext *context,
                          struct AlarmInfo* pAI) = 0,
void (*NotifyAlmMod)(struct testContext *context,
                          struct AlarmInfo* pAI,
                          int alm_mod_action) = 0,
void (*NotifyAlmDel)(struct testContext *context,
                          struct AlarmInfo* pAI) = 0);
  ~CAmvConn();
  /* Connection management */
  void ResetConnection (page 213) ;
  void BreakConnection (page 207) ;
  void FormConnection (page 209) (LPCTSTR system, COR_STATUS *ret_stat);
// The following make calls to the amap layer
  void UpdateList (page 223) (COR_STATUS* ret_stat);
  void SetAction (page 215) (const AlarmInfo *pAI, const TCHAR *action,
                  COR_STATUS *ret_stat);
void AddComment (page 206) (CAmvAlarm* alarmRecord, LPCTSTR comment,
                  COR_STATUS* ret_stat);
void SetStateInfo (page 218) (int alarmState, TCHAR **state,
                    TCHAR **ackState);
void OperHelpRequest (page 212) (TCHAR *alarmId,
                        void AddHelpLine(void *arg,
                                         TCHAR *str),
                        void *helpText,
                        COR_STATUS *ret_stat);
  /* Setup management */
  void SetupList (page 219) (COR_STATUS *ret_stat);
  CAmvSetupList (page 234) * Setups;
  /* Mode control */
  void SuspendDynamic (page 222) (COR_STATUS *ret_stat);
  void ResumeDynamic (page 214) (COR_STATUS *ret_stat);
  void RequestAlarms (page 213) (COR_STATUS *ret_stat);
  void SetToStatic (page 219) (COR_STATUS *ret_stat);
  void SetToDynamic (page 219) (COR_STATUS *ret_stat);
BOOL IsAlarmManagerConnected (page 211) ();
   LPCTSTR GetConnectedSystem() (page 210) ;
   BOOL ShouldReconnect() (page 221) ;
   enum AMAP_CONNECT {NOT_CONNECTED, CONNECTED, RESYNCING};
   AMAP_CONNECT IsConnected() (page 211) ;
  CAmvStateFilterList (page 244) * StateFilters;
  CAmvTimeFilter (page 247) * TimeFilter;
  CAmvClassFilterList (page 201) * ClassFilters;
  CAmvResourceFilterList (page 232) * ResourceFilters;
  AM_FILTER_TYPE PrimaryFilter;
```

```
  void SetPrimaryFilter (page 218) (AM_FILTER_TYPE type);
CAmvFieldFilterList* FieldFilters;
};//CAmvConn
```

CAmvConn Class Member Overview

Member	Description
AddComment()	Send a new alarm comment for the alarm to AMRP.
AMAP_CONNECT	An enum which specifies values returned by `IsConnected()`.
BreakConnection()	Disconnect from AMRP.
CAmvConn()	The constructor for the class. You must provide pointers to call-back functions for various events.
ClassFilters	A pointer to a CAmvClassFilterList which can be used to work with class filters for the connection.
ConfigurationAllowed	Returns TRUE if the user is allowed to create, modify, or delete viewer setups.
DeleteAllowed	Returns TRUE if the user is allowed to delete alarms.
FieldFilters	A pointer to a CAmvFieldFilterList which can be used to work with field filters for the connection.
FormConnection()	Open a connection to a project.
GetConnectedSystem()	Get the name of the project the connection is talking to.
IsAlarmManagerConnected()	BOOLEAN connection status.
IsConnected()	Multi-state connection status.
OperHelpRequest()	Display operator help for an alarm.
PrimaryFilter()	Get the current sort order for alarms.
RequestAlarms()	Get a list of all current alarms.
ResetConnection()	Disconnect from AMRP and RCM.
ResourceFilters	A pointer to a CAmvResourceFilterList which can be used to work with resource filters for the connection.
ResumeDynamic()	Resume suspended alarm updates.
SetAction()	Mark the alarm with up to two actions to be sent to AMRP the next time `UpdateList()` is called.
SetPrimaryFilter()	Set the sort order for alarms.
SetStateInfo()	Get the state and acknowledgment strings from a numeric alarm state.
SetToDynamic()	Set the connection to dynamic mode.
SetToStatic()	Set the connection to static mode.
SetupList()	Ask the AMRP to update the list of saved setups.

Member	Description
Setups	A pointer to a CAmvSetupList object which can be used to create, edit, activate, and save viewer setups for the connection.
ShouldReconnect()	TRUE if the connection should try to reestablish communication if they are lost.
StateFilters	A pointer to a CAmvStateFilterList which can be used to work with alarm state filters for the connection.
SuspendDynamic()	Temporarily suspend incoming alarm updates.
TimeFilter	A pointer to a CAmvTimeFilter which can be used to control the time filtering of alarms for the connection.
UpdateList()	Send alarm actions (acknowledge, delete) to AMRP.

CAmvConn::AddComment()

Send a new alarm comment to the AMRP. On return, **ret_stat->status** is COR_SUCCESS if no errors were encountered.

If this method fails, you will need to call **CAmvConn::BreakConnection()** .

Syntax

```
void AddComment(const AlarmInfo *pAI, LPCTSTR comment,
                COR_STATUS* ret_stat);
```

Data Type

```
void
```

Example

Add a comment to an alarm. If successful, update the display.

```
COR_STATUS ret_stat;
amvConn->AddComment(alarminfo_ptr, CommentStr, &ret_stat);
if(ret_stat.status == COR_SUCCESS) {
   ListComments();
} else {
   MessageBox(IDS_ERRLOSTAM);
}
```

CAmvConn::AMAP_CONNECT

Values returned by **CAmvConn::IsConnected()**

Syntax

```
enum AMAP_CONNECT { NOT_CONNECTED, CONNECTED, RESYNCING };
```

See Also

CAmvConn::IsConnected() *(page 211)*

CAmvConn::BreakConnection()

Disconnect from the AMRP.

Comments

Call this class member to break, and possibly reestablish, a connection to the AMRP when a communication error is detected.

If **CAmvConn::AddComment(), CAmvConn::OperHelpRequest(), or CAmvConn::SetupList()** fails, you will need to call **CAmvConn::BreakConnection().**

Syntax

```
CAmvConn* amvConn;
amvConn->BreakConnection()
```

Data Type

```
void
```

CAmvConn::CAmvConn()

Construct a new alarm management connection and provide callback functions to be called as events occur.

Comments

Much of the work of the Alarm Viewer is done through the callback functions. Depending on application requirements, the callbacks can directly handle the event or send messages so that the event is handled in the main message loop of the program.

Syntax

```
CAmvConn(
```

```
   void* who
   void (*dispFunc)(struct testContext *context,
                    struct AlarmInfo* pAI),
   void (*ClearDisp)(struct testContext *context),
   void (*LostAM)(struct testContext *context),
   int (*MaxAlarms)(struct testContext *context),
   void (*SetDisplayRedraw)(struct testContext *context,
                            int val),
   void (*UpdateCount)(struct testContext *context,
                       RCM_ALARM_DATA *alarmData),
void (*DoRcmError)(struct testContext *context,
                      int state),
   void (*DoConnectionFormed)(struct testContext *context),
   void (*NotifyAlmGen)(struct testContext *context,
                        struct AlarmInfo* pAI) = 0,
   void (*NotifyAlmMod)(struct testContext *context,
                        struct AlarmInfo* pAI,
                        int alm_mod_action) = 0,
   void (*NotifyAlmDel)(struct testContext *context,
                        struct AlarmInfo* pAI) = 0);
```

Data Types

CAmvConn uses the textContext *(page 176)*, SAmapCallbacks *(page 177)*, and AlarmInfo *(page 178)* structures.

See Also

Building an Alarm Management Connection, sample program *(page 182)*

CAmvConn::ClassFilters

Through this pointer, the individual class filters for the connection may be accessed.

Syntax

```
CAmvConn* amvConn;
CAmvClassFilterList* ClassFilters;
ClassFilters = amvConn->ClassFilters;
```

Data Type

```
CAmvClassFilterList*
```

See Also

```
            CAmvClassFilterList (page 201)
```

```
```

CAmvConn::ConfigurationAllowed()

Returns TRUE if the user is allowed to create, modify, or delete viewer setups.

Syntax

```
CAmvConn* amvConn;
amvConn->ConfigurationAllowed()
```

Data Type

```
BOOL
```

CAmvConn::DeleteAllowed()

Returns TRUE if the user is allowed to delete alarms.

Syntax

```
CAmvConn* amvConn;
amvConn->DeleteAllowed()
```

Data Type

```
BOOL
```

CAmvConn::FieldFilters()

Returns TRUE if the user is allowed to delete alarms.

Syntax

```
CAmvConn* amvConn;
CAmvFieldFilterList* FieldFilters;
FieldFilters = amvConn->FieldFilters;
```

Data Type

```
CAmvFieldFilterList*
```

CAmvConn::FormConnection()

Initialize a connection to the AMRP for the specified project.

Syntax

```
void FormConnection(LPCTSTR project, COR_STATUS* ret_stat)
```

Data Type

```
void
```

Example

This example allocates and initializes a new connection object then tries to connect to a project.

```
COR_STATUS ret_stat;
CAmvConn *amvConn = new CAmvConn(this,
                                 S_CallbackAddAlarm,
                                 S_CallbackResetContent,
                                 S_CallbackLostAM,
                                 S_MaxAlarms,
                                 S_SetDisplayRedraw,
                                 S_UpdateCount,
                                 S_DoRcmError,
                                 S_DoConnectionFormed,
                                 S_CallbackNotifyAlmGen,
                                 S_CallbackNotifyAlmMod,
                                 S_CallbackNotifyAlmDel);
amvConn->FormConnection(project, &ret_stat);
if(retstat.status != COR_SUCCESS) {
   delete amvConn;
   MessageBox(ret_stat.err_msg);
   return;
}
```

See Also

CAmvConn::GetConnectedSystem() *(page 210)*

CAmvConn::GetConnectedSystem()

Retrieve the name of the project that viewer is connected to.

Syntax

```
LPCTSTR GetConnectedSystem()
```

Data Type

```
LPCTSTR
```

Example

```
CAmvConn* amvConn;
_tprintf(_T("Connected to %s\n"),
        amvConn->GetConnectedSystem());
```

See Also

CAmvConn::FormConnection() *(page 209)*

CAmvConn::IsAlarmManagerConnected()

Returns TRUE if there is a connection to an alarm manager.

Syntax

```
IsAlarmManagerConnected()
```

Data Type

```
BOOL
```

CAmvConn::IsConnected()

Return the current connection status.

Syntax

```
AMAP_CONNECT IsConnected();
```

Data Type

```
enum AMAP_CONNECT { NOT_CONNECTED, CONNECTED, RESYNCING };
```

Example

This example displays the current connection status.

```
TCHAR StatusString[16];
```

```
switch (IsConnected()) {
    case NOT_CONNECTED:
            strcpy(StatusString, "Not Connected");
            break;
    case CONNECTED:
            strcpy(StatusString, "Connected");
            break;
    case RESYNCING:
            strcpy(StatusString, "Resyncing");
            break;
}
fprintf(stdout, "Status: %s\n", StatusString);
```

CAmvConn::OperHelpRequest()

Display operator help for an alarm. The **AddHelpLine()** callback function gets called for each line in the help file. It is passed the **helpText** pointer in **arg** and a pointer to the line of help text in **str** .

If this method fails, you will need to call **CAmvConn::BreakConnection()** .

Syntax

```
void OperHelpRequest(TCHAR *alarmId,
                     void AddHelpLine(void *arg,
                                      TCHAR *str),
                     void *helpText,
                     COR_STATUS *ret_stat);
```

Data Type

```
void
```

Example

```
static void S_AddHelpLine(void *arg, TCHAR *str)
{
   CListBox *box=(CListBox *)arg;
   box->AddString(str);
}

CAmvAlarm* alarm_ptr;
TCHAR id_buf[ALARM_ID_LEN+1];
CListBox HelpText;
amvConn->OperHelpRequest(alarm_ptr->ID(id_buf),
                         S_AddHelpLine,
                         (void *)HelpText,
                         &ret_stat);
```

CAmvConn::PrimaryFilter()

Returns the type of sorting currently being used by AMRP when sending static alarm lists.

The filter types are:

AM_TIME_FILTER	Alarms are sorted by time
AM_STATE_FILTER	Alarms are sorted by state
AM_FR_FILTER	Alarms are sorted by resource
AM_CLASS_FILTER	Alarms are sorted by class

Syntax

```
AM_FILTER_TYPE PrimaryFilter()
```

Data Type

```
typedef int AM_FILTER_TYPE;
```

CAmvConn::RequestAlarms()

Request AMRP to send a list of alarms.

Syntax

```
RequestAlarms(COR_STATUS *ret_stat);
```

Data Type

```
void
```

Example

Get the current list of alarms from AMRP.

```
ResetAlarmList();
amvConn->RequestAlarms(&ret_stat);
if (ret_stat.status != COR_SUCCESS) {
   MessageBox(IDS_ERRLOGMSG);
}
```

CAmvConn::ResetConnection()

This method cleans up the connection when communication is lost or when the **CAmvConn** is deleted.

Call this method to handle DO_IPCHASDIED messages.

Syntax

```
CAmvConn* amvConn;
amvConn->ResetConnection()
```

Data Type

```
void
```

Example

Handle DO_IPCHASDIED message by resetting connection.

```
afx_msg LRESULT IpcHasDied(WPARAM wParam, LPARAM lParam)
{
   amvConn->ResetConnection();
}
```

CAmvConn::ResourceFilters

Through this pointer, the individual resource filters for the connection may be accessed.

Syntax

```
CAmvConn* amvConn;
CAmvResourceFilterList* ResourceFilters;
ClassFilters = amvConn->ResourceFilters;
```

Data Type

```
CAmvResourceFilterList*
```

See Also

CAmvResourceFilterList *(page 232)*

CAmvConn::ResumeDynamic()

Temporarily suspend incoming dynamic updates.

Comments

When the viewer is showing the user modal dialogs like property sheets for setup parameters, dynamic updates should be suspended.

Syntax

```
ResumeDynamic(COR_STATUS *ret_stat);
```

Data Type

```
void
```

Example

Suspend dynamic updates while changing setups.

```
if (IsStatic()) {
   EditSetups()
else
   amvConn->SuspendDynamic();
   EditSetups();
   amvConn->ResumeDynamic();
}
```

See Also

```
CAmvConn::SuspendDynamic() (page 222)
```

CAmvConn:SetAction()

Mark the alarm with up to two actions to be sent to the AMRP the next time **UpdateList()** is called.

Comments

Up to two actions may be set in a two-character string. The valid characters are:

AM_ACK_CHAR	Acknowledge the alarm
AM_CLR_CHAR	Clear the alarm
AM_DEL_CHAR	Delete the alarm

A blank character string indicates no action.

Clearing an alarm requires that the user be allowed to manually clear alarms. This may be checked with **CAmvAlarm::ManualClearAllowed()** .

Deleting an alarm requires that the user be allowed to manually delete alarms. This may be checked with **CAmvAlarm::DeleteAllowed()** .

Syntax

```
SetAction(const AlarmInfo *pAI, const TCHAR *action,
          COR_STATUS *ret_stat);
```

Data Type

```
void
```

Example

Delete an alarm, if allowed. Return TRUE if successful, FALSE otherwise.

```
Int DeleteAlarm(CAmvConn* amvConn,
                struct AlarmInfo *pAlarmInfo);
{
   TCHAR action[3];
   if (amvConn->DeleteAllowed())
     {
       _tcsncpy(action, _T(" ");
       action[0] = AM_DEL_CHAR;
       amvConn->SetAction(pAlarmInfo, action, &status);
       amvConn->UpdateList(&status);
       return TRUE;
     }
     else
     {
       return FALSE;
     }
}
```

See Also

CAmvConn::UpdateList() *(page 223)*
, CAmvAlarm::ManualClearAllowed() *(page 192)*
,
CAmvConn::DeleteAllowed() *(page 209)*

CAmvConn:SetAction() - Change Approval

(When change approval is required) Mark the alarm with up to two actions to be sent to the AMRP the next time UpdateList() is called.

Comments

Up to two actions may be set in a two-character string. The valid characters are:

AM_ACK_CHAR	Acknowledge the alarm
AM_CLR_CHAR	Clear the alarm
AM_DEL_CHAR	Delete the alarm

A blank character string indicates no action.

Clearing an alarm requires that the user be allowed to manually clear alarms. This may be checked with **CAmvAlarm::ManualClearAllowed()** .

Deleting an alarm requires that the user be allowed to manually delete alarms. This may be checked with **CAmvAlarm::DeleteAllowed()** .

Syntax

```
SetAction(const AlarmInfo *pAI, const TCHAR *action,
          COR_STATUS *ret_stat, ALARM_CA_ALARM_OPER_REQ *caInfo, COR_STATUS
 *ret_stat);
```

Data Type

```
void
```

Example

Delete an alarm, if allowed. Return TRUE if successful, FALSE otherwise.

```
Int DeleteAlarm(CAmvConn* amvConn,
                struct AlarmInfo *pAlarmInfo);
{
   TCHAR action[3];
    //Fill Change approval information ( Perform/ Perform Verify Users
 information)
  ALARM_CA_ALARM_OPER_REQ caInfo;

     if (amvConn->DeleteAllowed())
     {
       _tcsncpy(action, _T(" ");
       action[0] = AM_DEL_CHAR;
```

```
        if(AMValarmInfo->GetChangeApprovalInfo())
        {
             amvConn->SetAction(pAlarmInfo, action, caInfo, &status);
        }
        Else
        {
             amvConn->SetAction(pAlarmInfo, action, &status);
        }
        amvConn->UpdateList(&status);
        return TRUE;
      }
      else
      {
        return FALSE;
      }
}
```

CAmvConn::SetPrimaryFilter

Set the sort order for alarm lists coming from AMRP.()

The filter type choices are:

AM_TIME_FILTER	Alarms are sorted by time
AM_STATE_FILTER	Alarms are sorted by state
AM_FR_FILTER	Alarms are sorted by resource
AM_CLASS_FILTER	Alarms are sorted by class

Comments

This controls the order of alarms as provided to the DispFunc() callback. In some cases, it may be better to not worry about sorting in the connection and to let the display list do the sorting.

A change in sort order will not take place until **Setups->FilterSetup()** is called.

Syntax

```
SetPrimaryFilter(AM_FILTER_TYPE type)
```

Data Type

```
void
```

CAmvConn::SetStateInfo()

Get the alarm state and acknowledge state strings from the numeric state.

Syntax

```
SetStateInfo(AM_STATE_TYPE alarmState, TCHAR **state,
             TCHAR **ackState);
```

Data Type

```
void
```

CAmvConn::SetToDynamic()

Set the connection to dynamic mode so alarm events are sent by AMRP as they occur.

Comments

In dynamic mode, each alarm event (generate, acknowledge, clear) is sent as it occurs and the connection calls the Notify callbacks.

Syntax

```
amvConn->SetToDynamic(COR_STATUS *ret_stat);
```

Data Type

```
void
```

CAmvConn::SetToStatic()

Set the connection to static mode so that alarm updates are only sent by AMRP in response to **RequestAlarms()** calls.

Syntax

```
amvConn->SetToStatic(COR_STATUS *ret_stat);
```

Data Type

```
void
```

CAmvConn::SetupList()

Retrieves the list of saved viewer setups from AMRP.

If this method fails, you will need to call **CAmvConn::BreakConnection()** .

Comments

Retrieving saved setups can be time consuming because of the inter-process communication and disk access involved, and because of the potentially large amount of data exchanged. Therefore, setups are not retrieved until they are requested with this function. **CAmvConn::SetupList()** must be called for the connection before the **CAmvSetupList** member functions of the **CAmvConn::Setups** are called.

Syntax

```
CAmvConn* amvConn;
amvConn->SetupList(COR_STATUS *ret_stat);
```

Data Type

```
void
```

Example

Retrieve saved setups from AMRP and print the setup IDs.

```
CAmvConn* amvConn;
int I;
amvConn->SetupList();
for (i = 0; i < amvConn->Setups->Number(); i++) {
   _tprintf(_T("%s\n"), amvConn->Setups->Setup(i));
}
```

See Also

CAmvConn::Setups *(page 220)*

CAmvConn::Setups

Use this pointer to access the saved Alarm Viewer setups for the connection.
CAmvConn::SetupList() must be called for the connection before the CAmvSetupList member functions of the **CAmvConn::Setups** are called.

Syntax

```
CAmvConn* amvConn;
CAmvSetupList* amvSetups;
amvSetups = amvConn->Setups;
```

Data Type

```
CAmvSetupList*
```

Example

Load a setup, save it under a new ID, and make it the default.

```
CAmvConn* amvConn;
amvConn->Setups->DoLoad(setup_id, &ret_stat);
amvConn->Setups->Update(new_setup_id);
amvConn->Setups->DoSave(new_setup_id, &ret_stat);
amvConn->Setups->DoSetD(new_setup_id, &ret_stat);
amvConn->Setups->FilterSetup();
```

See Also

CAmvConn::SetupList() *(page 219)*

CAmvConn::ShouldReconnect()

Resets the connection to the Alarm Management Resident Process.

Syntax

```
ShouldReconnect()
```

Data Type

```
BOOL
```

Example

Reset connection on loss of communication to AM.

```
void CAmvTest::LostAM(CAmvConn* amvConn)
{
  BOOL tryReconnect = FALSE;
  if(amvConn->ShouldReconnect())
```

```
  {
    tryReconnect = TRUE;
  }
  ResetConnectionDisplay(amvConn);
  if(!tryReconnect)
  {
    amvConn->SetStateString(IDS_LOSTSERVER);
  }
}
```

CAmvConn::StateFilters

Through this pointer, the individual state filters for the connection may be accessed.

Syntax

```
CAmvConn* amvConn;
CAmvStateFilterList* StateFilters;
StateFilters = amvConn->StateFilters
```

Data Type

```
CAmvStateFilter*
```

See Also

CAmvStateFilterList *(page 244)*

CAmvConn::SuspendDynamic()

Suspend dynamic updates.

Syntax

```
SuspendDynamic(COR_STATUS *ret_stat);
```

Data Type

```
void
```

Example

Suspend dynamic updates while changing setups.

```
if (IsStatic()) {
```

```
   EditSetups()
else
   amvConn->SuspendDynamic();
   EditSetups();
   amvConn->ResumeDynamic();
}
```

See Also

CAmvConn::ResumeDynamic() *(page 214)*

CAmvConn::TimeFilter

A pointer to the time filter which restricts what alarms are sent from AMRP to the viewer.

Syntax

```
CAmvConn* amvConn;
CAmvTimeFilter* TimeFilter;
TimeFilter = amvConn->TimeFilter;
```

Data Type

```
CAmvTimeFilter*
```

Example

Enable time filtering for the viewer

```
amvConn->TimeFilter->Enable();
```

See Also

CAmvTimeFilter *(page 247)*

CAmvConn::UpdateList()

Notify AMRP of all user-requested actions such as acknowledging or deleting an alarm.

Syntax

```
CAmvConn* amvConn;
```

```
amvConn->UpdateList(COR_STATUS* ret_stat)
```

Data Type

```
void
```

See Also

CAmvConn::SetAction() *(page 215)* , CAmvSetupList::FilterSetup() *(page 238)*

CAmvFieldFilter

CAmvFieldFilter

A CAmvFieldFilter controls whether alarms that match the field filter will be passed from AMRP to the viewer.

CAmvFieldFilter Class Definition

CAmvFieldFilter Class Definition

The following class definition represents the CAmvFieldFilter class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvFieldFilter {
public:
  AM_FIELD_ID field_id;
  AM_FIELD_FILTER_TYPE type;
  TCHAR filter_string[ALM_SETUP_FF_LEN+1];
};// CAmvFieldFilter
```

CAmvFieldFilter Class Member Overview

The following table briefly describes the members of the CAmvFieldFilter class. The sections that follow provide details about the syntax and semantics of using the members.

Member	Description
field_id	ID of the field that will be checked for a match.
type	Type of match that will be performed.

Member	Description
filter_string	The text string that will be used to perform the match.

CAmvFieldFilter::field_id

This member specifies the ID of the field that will be checked for a match.

Comments

The following values indicate the field that will be filtered:

```
#define AM_FF_FIELD_UNUNSED 0
#define AM_FF_FIELD_ALARM_ID 1
#define AM_FF_FIELD_ALARM_MSG 2
#define AM_FF_FIELD_ALARM_DESCRIPTION 3
```

Syntax

```
CAmvFieldFilter* field_filter;
field_filter->field_id = AM_FF_FIELD_ALARM_ID;
```

Data Type

```
COR_U1
```

CAmvFieldFilter::type

This member specifies the type of match that will be performed.

Comments

The following values indicate the type of match that will be performed:

```
#define AM_FF_MATCH_TYPE_SUBSTRING 0
#define AM_FF_MATCH_TYPE_WILDCARD  1
#define AM_FF_MATCH_TYPE_REGEX     2 //Uses ECMAScript grammar
```

Syntax

```
CAmvFieldFilter* field_filter;
field_filter->type = AM_FF_MATCH_TYPE_REGEX;
```

Data Type

```
COR_U1
```

CAmvFieldFilter::filter_string

This member specifies the text string that will be used to perform the match.

Comments

This is the string used in the match. All matches ignore case.

```
AM_FF_MATCH_TYPE_SUBSTRING - For this type, the string has to match part of
 the target string.
AM_FF_MATCH_TYPE_WILDCARD - For this type, the string has to exactly match
 the target string using * and ? as wildcard characters.
AM_FF_MATCH_TYPE_REGEX - For this type, the string has to exactly match the
 target string using an ECMAScript-compliant regular expression.
```

Syntax

```
CAmvFieldFilter* field_filter;
_tcscpy(field_filter->filter_string, _T("*ID*"));
```

Data Type

```
TCHAR
```

CAmvFieldFilterList

CAmvFieldFilterList

The CAmvFieldFilterList class provides methods to loop through all the field filters currently in use. The list is built from data supplied by AMRP.

CAmvFieldFilterList Class Definition

CAmvFieldFilterList Class Definition

The following class definition represents the CAmvFieldFilterList class

```
class AMAPEXPORT CAmvFieldFilterList : private CAmvFilter {
 friend class CAmap;
public:
  CAmvFieldFilter* First(COR_STATUS* ret_stat);
```

```
  CAmvFieldFilter* Next(CAmvFieldFilter* filter, COR_STATUS* ret_stat);
  CAmvFieldFilter* AddFieldFilter(AM_FIELD_ID field_id,
 AM_FIELD_FILTER_TYPE filter_type, LPCTSTR filter_string);
  // Removes it from the list and frees the record.
  // This will break First/Next traversal, so do not do it as part of a
 First/Next loop.
  void RemoveFieldFilter(CAmvFieldFilter* filter_field);
  void RemoveAllFieldFilters();
};// CAmvFieldFilterList
```

CAmvFieldFilterList Class Member Overview

The following table briefly describes the members of the CAmvFieldFilterList class. The sections that follow provide details about the syntax and semantics of using the members.

Member	Description
First	Returns a pointer to the first filter in the list.
Next	Returns a pointer to the next filter in the list.
AddFieldFilter	Adds a field filter to the list.
RemoveFieldFilter	Removes the field filter from the list.
RemoveAllFieldFilters	Removes all field filters from the list.

CAmvFieldFilterList::First()

This member returns a pointer to the first field filter in the list. If the filter list or the connection is in an invalid state, the return value is NULL and ret_stat->status is COR_FAILURE.

Syntax

```
CAmvFieldFilter* field_ptr;
CAmvFieldFilterList* field_list;
field_ptr = field_list->First(&ret_stat);
```

Data Type

```
CAmvFieldFilter*
```

Example

To remove the first field filter:

```
COR_STATUS ret_stat;
CAmvConn* AmvConn;
CAmvFieldFilter* field_ptr;
field_ptr = AmvConn->FieldFilters->First(&ret_stat);
```

```
if(field_ptr != NULL && ret_stat.status == COR_SUCCESS)
  AmvConn->FieldFilters->RemoveFieldFilter(field_ptr);
```

CAmvFieldFilterList::Next()

This member returns a pointer to the next field filter in the list.

Syntax

```
CAmvFieldFilter* field_ptr;
CAmvFieldFilterList* field_list;
field_ptr = field_list->Next(field_ptr, &ret_stat);
```

Data Type

```
CAmvFieldFilter*
```

CAmvFieldFilterList::AddFieldFilter()

This member adds a field filter to the list.

Syntax

```
AM_FIELD_ID field_id;
AM_FIELD_FILTER_TYPE filter_type;
LPCTSTR filter_string;
CAmvFieldFilter* field_ptr;
CAmvFieldFilterList* field_list;
field_ptr = field_list->Next(field_id, filter_type, filter_string);
```

Data Type

```
CAmvFieldFilter*
```

Example

To add a sub-string filter for alarm IDs:

```
CAmvConn* AmvConn;
CAmvFieldFilter* field_ptr;
field_ptr = AmvConn->FieldFilters->AddFieldFilter(AM_FF_FIELD_ALARM_ID,
 AM_FF_MATCH_TYPE_SUBSTRING, _T("MyId");
```

CAmvFieldFilterList::RemoveFieldFilter()

This member removes the specified field filter from the list and frees the record.

This procedure will break the First/Next traversal, so do not perform it as part of a First/Next loop.

Syntax

```
CAmvFieldFilter* field_ptr;
CAmvFieldFilterList* field_list;
field_list->RemoveFieldFilter(field_ptr);
```

Data Type

```
void
```

Example

To remove the first field filter:

```
COR_STATUS ret_stat;
CAmvConn* AmvConn;
CAmvFieldFilter* field_ptr;
field_ptr = AmvConn->FieldFilters->First(&ret_stat);
if(field_ptr != NULL && ret_stat.status == COR_SUCCESS)
  AmvConn->FieldFilters->RemoveFieldFilter(field_ptr);
```

CAmvFieldFilterList::RemoveAllFieldFilters()

This member removes all field filters from the list and frees the record.

Syntax

```
CAmvFieldFilterList* field_list;
field_list->RemoveFieldFilter();
```

Data Type

```
void
```

Example

To remove all field filters:

```
CAmvConn* AmvConn;
AmvConn->FieldFilters->RemoveAllFieldFilters(field_ptr);
```

CAmvResourceFilter

CAmvResourceFilter

The `CAmvResourceFilter` class controls whether alarms with the corresponding resource will be passed from AMRP to the viewer.

CAmvResourceFilter Class Definition

CAmvResourceFilter Class Definition

The following class definition represents the `CAmvResourceFilter` class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvResourceFilter {
public:
   TCHAR* ID(TCHAR* id_buf) (page 231) ;
   BOOL IsEnabled() (page 231)
   void Enable() (page 231) ;
   void Disable() (page 230) ;
};// CAmvResourceFilter
```

CAmvResourceFilter Class Member Overview

The following table briefly describes the members of the `CAmvResourceFilter` class. The sections that follow go into detail about the syntax and semantics of using the members.

Member		Description
	Disable()	Don't have AMRP send alarms for this resource to the viewer
	Enable()	Have AMRP send alarms for this resource to the viewer
	ID()	Resource ID
	IsEnabled()	Will alarms for this resource be sent to the viewer by the AMRP

CAmvResourceFilter::Disable()

Disables the resource filter so that AMRP does not send alarms for this resource to the viewer.

Syntax

```
CAmvResourceFilter* resource_ptr;
resource_ptr->Disable();
```

Data Type

```
void
```

See Also

```
CAmvResourceFilter::Enable() (page 231)
```

CAmvResourceFilter::Enable()

Enables the resource filter so that AMRP sends alarms for this resource to the viewer.

Syntax

```
CAmvResourceFilter* resource_ptr;
resource_ptr->Disable();
```

Data Type

```
void
```

CAmvResourceFilter::ID()

Retrieves the ID for the resource and puts it in the user-supplied buffer. Returns a pointer to that buffer.

Syntax

```
CAmvResourceFilter* resource_ptr;
TCHAR id_buf[FR_ID_LEN+1];
resource_ptr->ID(id_buf);
```

Data Type

```
TCHAR*
```

CAmvResourceFilter::IsEnabled()

Returns TRUE if AMRP will send alarms for this resource.

Syntax

```
CAmvResourceFilter* resource_ptr;
resource_ptr->IsEnabled();
```

Data Type

```
BOOL
```

CAmvResourceFilterList

CAmvResourceFilterList

The `CAmvResourceFilterList` class provides methods to loop through all the resource filters currently in use.

The list is built from data supplied by the AMRP. Filters cannot be added to, or removed from, the list.

CAmvResourceFilterList Class Definition

CAmvResourceFilterList Class Definition

The following class definition represents the `CAmvResourceFilterList` class.

```
class CAmvResourceFilterList {
public:
   CAmvResourceFilter* First (page 232) (COR_STATUS* ret_stat);
   CAmvResourceFilter* Next (page 233) (CAmvResourceFilter* filter,
                           COR_STATUS* ret_stat);
};
```

CAmvResourceFilterList Class Member Overview

The following table briefly describes the members of the `CAmvResourceFilterList` class. The sections that follow go into detail of the syntax and semantics of using the members.

Member		Description
	First	Get a pointer to the first filter in the list
	Next	Get a pointer to the next filter in the list

CAmvResourceFilterList::First()

Return a pointer to the first class filter in the list.

Comments

If the filter list or the connection is invalid, NULL is returned and **ret_stat >status** is set to COR_FAILURE.

Syntax

```
CAmvResourceFilter* resource_ptr;
CAmvResourceFilterList* resource_list;
resource_ptr = resource_list->First(&ret_stat);
```

Data Type

```
CAmvResourceFilter*
```

Example

```
CAmvConn* amvConn;
CAmvResourceFilter* resource_ptr;
TCHAR id_buf[FR_ID_LEN+1];
for (resource_ptr =
        amvConn->ResourceFilters->First(&ret_stat);
     resource_ptr != NULL && ret_stat.status == COR_SUCCESS;
     resource_ptr =
       amvConn->ResourceFilters->Next(resource_ptr,
                                      &ret_stat))
{
   _tprintf(_T("%s\n"), resource_ptr->ID(id_buf));
}
```

See Also

CAmvResourceFilterList::Next() *(page 233)* , CAmvResourceFilter *(page 230)*

CAmvResourceFilterList::Next()

Returns a pointer to the next resource filter in the list.

Comments

If the filter list or the connection is found to be invalid, NULL is returned and ret_stat->status is set to COR_FAILURE.

Syntax

```
CAmvResourceFilter* resource_ptr;
CAmvResourceFilterList* resource_list;
COR_STATUS ret_stat;
resource_ptr = resource_list->Next(resource_ptr, &ret_stat);
```

Data Type

```
CAmvResourceFilter*
```

See Also

CAmvResourceFilterList::First() *(page 232)* , CAmvResourceFilter *(page 230)*

CAmvSetupList

CAmvSetupList

The `CAmvSetupList` class provides the methods necessary to create, edit, save, and activate saved alarm viewer setups.

There is a pointer to an instance of this class on the `CAmvConn` class. You should not create instances of this object in your program.

CAmvSetupList Class Definition

CAmvSetupList Class Definition

The following class definition represents the `CAmvSetupList` class.

```
class CAmvSetupList {
public:
```

int Number();

```
    TCHAR* Setup(int i) (page 239) ;
```

```
    int DoSetD (page 236) (TCHAR* setupID, COR_STATUS* ret_stat);
    void DoDel (page 236) (TCHAR* setupID, COR_STATUS* ret_stat);
    void DoClear (page 235) (COR_STATUS* ret_stat);
    void DoLoad (page 236) (TCHAR* setupID, COR_STATUS* ret_stat);
    void DoSave (page 236) (TCHAR* setupID, COR_STATUS* ret_stat);
    void Update (page 240) (TCHAR* setupID);
    int Exists (page 237) (TCHAR* setupID, COR_STATUS* ret_stat);
    void FilterSetup (page 238) (COR_STATUS* ret_stat);
    const TCHAR* SetupID (page 239) (TCHAR *setupID = NULL);
};
```

CAmvSetupList Class Member Overview

The following table briefly describes the members of the `CAmvResourceFilterList` class. The sections that follow go into detail of the syntax and semantics of using the members.

Member		Description
	DoClear	Clear setups and return to unfiltered display
	DoDel	Delete the specified setup
	DoLoad	Load the specified setup and make it current
	DoSave	Save the specified setup
	DoSetD	Set the default setup for this user
	Exists	See if the named setup exists
	FilterSetup	Get setup information from AMRP
	Number	Get the number of setups
	Setup	Get the ID of the specified setup
	SetupID	Get the ID of the current setup
	Update	Rename the current setup

CAmvSetupList::DoClear()

Clear the currently loaded setup and return to unfiltered viewing.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->DoClear(COR_STATUS* ret_stat);
```

Data Type

```
void
```

CAmvSetupList::DoDel()

Delete the named setup from the database of saved setups.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->DoDel(TCHAR* setupID, COR_STATUS* ret_stat);
```

Data Type

```
void
```

CAmvSetupList::DoLoad()

Load the named setup from the database and make it the current setup, filtering by resource, state, time, and class as appropriate.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->DoLoad(TCHAR* setupID, COR_STATUS* ret_stat);
```

Data Type

```
void
```

CAmvSetupList::DoSave()

Save the named setup to the database.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->DoSave(TCHAR* setupID, COR_STATUS* ret_stat);
```

Data Type

```
void
```

CAmvSetupList::DoSetD()

Set the named setup as the default for this user. The next time the user starts a viewer, this setup will be loaded automatically.

Comments

Returns COR_SUCCESS if the configuration update is successful, COR_FAILURE otherwise.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->DoSetD(TCHAR* setupID, COR_STATUS* ret_stat);
```

Data Type

```
int
```

Example

This example tries to set the default setup and checks the return value.

```
if (amvConn->Setups->DoSetD(SetupID, &ret_stat) != COR_SUCCESS)
{
   MessageBox(ret_stat.err_msg);
}
```

CAmvSetupList::Exists()

Check to see if the named setup exists in the database.

Comments

Returns COR_SUCCESS if the AMRP was successfully queried regarding setup existence. Returns COR_FAILURE otherwise.

If the return value is COR_SUCCESS, **ret_stat->err_code** is TRUE if the setup exists, FALSE if it does not.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->Exists(TCHAR* setupID, COR_STATUS* ret_stat);
```

Data Type

```
int
```

Example

This example looks for a setup ID.

```
if (amvConn->Setups->Exists(testID, &ret_stat) != COR_SUCCESS)
{
    MessageBox("Could not communicate with AMRP");
} else
{
    Exists = ret_stat->err_code;
}
```

CAmvSetupList::FilterSetup()

Send the filter parameters for the currently selected setup to the AMRP and put them in effect.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->FilterSetup(TCHAR* setupID,
                             COR_STATUS* ret_stat);
```

Data Type

```
void
```

See Also

CAmvConn: UpdateList() CAmvConn::UpdateList() *(page 223)*

CAmvSetupList::Number()

Return the number of setups available to this user.

Syntax

```
CAmvConn* amvConn;
i = amvConn->Setups->Number()
```

Data Type

```
int
```

Example

List the IDs of the available setups.

```
amvConn->SetupList();
for (i = 0; i < amvConn->Setups->Number(); i++)
{
   _tprintf(_T("%s\n"), amvConn->Setups->Setup(i));
}
```

CAmvSetupList::Setup()

Return the ID of the specified setup.

Syntax

```
CAmvConn* amvConn;
TCHAR* str;
str = amvConn->Setups->Setup(int i);
```

Data Type

```
TCHAR*
```

See Also

```
CAmvSetupList:Number() CAmvSetupList::Number() (page 238)
```

CAmvSetupList::SetupID()

Get a pointer to the ID of the current setup or retrieve the ID of the current setup into a user-supplied buffer.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->SetupID();
amvConn->Setups->SetupID(TCHAR setupID[]);
```

Data Type

```
const TCHAR*
```

Example

Print the name of the current setup:

```
_tprintf(_T("Setup is %s\n"), amvConn->Setups->SetupID());
```

CAmvSetupList::Update()

Rename the current setup with the specified ID.

Syntax

```
CAmvConn* amvConn;
amvConn->Setups->Update(TCHAR setupID[]);
```

Data Type

```
Void
```

CAmvStateFilter

CAmvStateFilter

The `CAmvStateFilter` class controls whether alarms with the corresponding state will be passed from AMRP to the Viewer.

CAmvStateFilter Class Definition

CAmvStateFilter Class Definition

The following class definition represents the `CAmvStateFilter` class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvStateFilter {
public:
BOOL IsEnabled() (page 243) ;
void Enable() (page 242) ;
   void Disable() (page 242) ;
   AM_STATE_TYPE State() (page 243) ;
   static TCHAR* ack_clear_msg() (page 241) ;
   static TCHAR* ack_only_msg() (page 241) ;
   static TCHAR* clear_only_msg() (page 242) ;
};// CAmvStateFilter
```

CAmvStateFilter Class Member Overview

Member	Description

	ack_clear_msg()	The text string to describe cleared, acknowledged alarms
	ack_only_msg()	The text string to describe acknowledged alarms.
	clear_only_msg()	The text string to describe cleared, unacknowledged alarms.
	Disable()	Don't send alarms in this state to the viewer.
	Enable()	Send alarms in this state to the viewer.
	IsEnabled()	Will alarms in this state be sent to the viewer by AMRP.
	State()	The state number for the ID.

CAmvStateFilter::ack_clear_msg()

This function returns a user-readable string that describes cleared and acknowledged alarms.

Syntax

```
TCHAR* p = ack_clear_msg();
```

Data Type

```
TCHAR*
```

See Also

CAmvAlarm:DeleteOptions() CAmvAlarm::DeleteOptions() *(page 189)*

CAmvStateFilter::ack_only_msg()

This function returns a user-readable string that describes acknowledged alarms.

Syntax

```
TCHAR* p = ack_only_msg();
```

Data Type

```
TCHAR*
```

See Also

CAmvAlarm::DeleteOptions() *(page 189)*

```
```

CAmvStateFilter::clear_only_msg()

This function returns a user-readable string that describes cleared, unacknowledged alarms.

Syntax

```
TCHAR* p = clear_only_msg();
```

Data Type

```
TCHAR*
```

See Also

```
CAmvAlarm::DeleteOptions() (page 189)
```

CAmvStateFilter::Disable()

Disable the state filter so that AMRP does not send alarms in this state to the viewer.

Syntax

```
CAmvStateFilter* State_ptr;
State_ptr->Disable();
```

Data Type

```
void
```

See Also

```
CAmvStateFilter::Enable() (page 242)
```

CAmvStateFilter::Enable()

Enable the state filter so that AMRP sends alarms in this state to the viewer.

Syntax

```
CAmvStateFilter* state_ptr;
state_ptr->Disable();
```

Data Type

```
void
```

See Also

CAmvStateFilter::Disable() *(page 242)*

CAmvStateFilter::IsEnabled()

Returns TRUE if AMRP will send the viewer alarms in this state.

Syntax

```
CAmvStateFilter* state_ptr;
state_ptr->IsEnabled();
```

Data Type

```
BOOL
```

CAmvStateFilter::State()

Gets the state type for the filter.

The state type is one of the following:

AM_GENERATED	Alarms with Generated state.
AM_ACKNOWLEDGED	Alarms with Acknowledged state.
AM_CLEARED	Alarms with Cleared state.
AM_DELETED	Alarms with Deleted state.
AM_NONEXISTENT	Invalid filter or connection.

Comments

If the filter or connection is found to be invalid, AM_NONEXISTENT is returned and ret_stat->status is set to COR_FAILURE.

Syntax

```
AM_STATE_TYPE state;
CAmvStateFilter* state_ptr;
state = state_ptr->State(&ret_stat)
```

Data Type

```
typedef int AM_STATE_TYPE;
```

CAmvStateFilterList

CAmvStateFilterList

The `CAmvStateFilterList` class provides methods to loop through all the state filters currently in use.

The list is built from data supplied by the AMRP. Filters cannot be added to, or removed from the list.

CAmvStateFilterList Class Definition

CAmvStateFilterList Class Definition

The following class definition represents the `CAmvStateFilterList` class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvStateFilterList {
   CAmvStateFilter* First (page 245) (COR_STATUS* ret_stat);
   CAmvStateFilter* Next (page 246) (CAmvStateFilter* filter,
                          COR_STATUS* ret_stat);
   BOOL IsFiltered (page 246) (AM_STATE_TYPE state);
   void ClearAll() (page 245) ;
   void SetFilter (page 247) (AM_STATE_TYPE state);
};// CAmvStateFilterList
```

CAmvStateFilterList Class Member Overview

The following table briefly describes the members of the `CAmvStateFilterList` class. The sections that follow go into detail of the syntax and semantics of using the members.

Member		Description
	ClearAll()	Clear all state filters.
	ClearFilter()	
	First()	Get a pointer to the first state filter
	IsFiltered()	Is the specified state filter enabled.
	Next()	Get a pointer to the next state filter
	SetFilter()	Filter the specified state

CAmvStateFilterList::ClearAll()

Disable all state filters.

Syntax

```
CAmvConn* amvConn;
amvConn->StateFilters->ClearAll()
```

Data Type

```
void
```

CAmvStateFilterList::ClearFilter()

Disable filtering of the specified state

Syntax

```
CAmvConn* amvConn;
amvConn->StateFilters->ClearFilter(AM_STATE_TYPE state)
```

Data Type

```
void
```

CAmvStateFilterList::First()

Get the first state filter for the connection.

Comments

If the filter list or the connection is found to be invalid, NULL is returned and **ret_stat->status** is set to **COR_FAILURE** .

Syntax

```
CAmvConn* amvConn;
CAmvStateFilter* state_ptr;
COR_STATUS ret_stat;
state_ptr = amvConn->StateFilters->First(&ret_stat);
```

Data Type

```
CAmvStateFilter*
```

See Also

CAmvStateFilterList::Next() *(page 246)*

CAmvStateFilterList::IsFiltered()

Returns TRUE if alarms in the specified state will be sent to the viewer.

Syntax

```
CAmvConn* amvConn;
amvConn->StateFilters->IsFiltered(AM_STATE_TYPE state)
```

Data Type

```
BOOL
```

CAmvStateFilterList::Next()

Return the next state filter in the list.

Comments

If the filter list or the connection is found to be invalid, NULL is returned and **ret_stat->status** is set to **COR_FAILURE** .

Syntax

```
CAmvConn* amvConn;
CAmvStateFilter* state_ptr;
state_ptr = amvConn->StateFilters->Next(state_ptr, &ret_stat);
```

Data Type

```
CAmvStateFilter* (page 240)
```

CAmvStateFilterList::SetFilter()

Enable the filter for the specified state.

The state is one of the following:

AM_GENERATED	Alarms in Generated state.
AM_ACKNOWLEDGED	Alarms in Acknowledged state.
AM_CLEARED	Alarms in Cleared state.
AM_DELETED	Alarms in Deleted state.

Syntax

```
CAmvConn* amvConn;
amvConn->StateFilters->SetFilter(AM_STATE_TYPE state)
```

Data Type

```
void
```

CAmvTimeFilter

CAmvTimeFilter

The time filter tells the AMRP not to send alarms before a specified time.

There is a `CAmvTimeFilter` on the `CAmvConn` class. You should never have to create one of your own.

CAmvTimeFilter Class Definition

CAmvTimeFilter Class Definition

The following class definition represents the `CAmvTimeFilter` class. For clarity, it has been simplified from the actual class definition. All the user-accessible members are listed.

```
class CAmvTimeFilter {
public:
   BOOL IsEnabled() (page 249) ;
void Enable() (page 249) ;
   void Disable(); (page 248)
   void SetTimeStamp (page 249) (COR_STAMP* ts)
   COR_STAMP* const TimeStamp() (page 250)
};// CAmvTimeFilter
```

CAmvTimeFilter Class Member Overview

The following table briefly describes the members of the `CAmvTimeFilter` class. The sections that follow go into detail of the syntax and semantics of using the members.

Member	Description
Disable()	Do not limit alarms sent to the viewer by time.
Enable()	Limit alarms sent to the viewer by time.
IsEnabled()	Will alarms sent to the viewer by AMRP be limited by the timestamp specified.
SetTimeStamp()	Set the time used to filter alarms.
TimeStamp()	Get the time used to filter alarms.

CAmvTimeFilter::Disable()

Disable time filtering so that all alarms, no matter how old, are sent to the viewer.

Syntax

```
CAmvConn* amvConn;
amvConn->TimeFilter->Disable()
```

Data Type

```
void
```

See Also

CAmvTimeFilter::Enable() *(page 249)*

CAmvTimeFilter::Enable()

Enable time filtering so that only alarms after the specified time are sent to the viewer.

Syntax

```
CAmvConn* amvConn;
amvConn->TimeFilter->Enable()
```

Data Type

```
void
```

CAmvTimeFilter::IsEnabled()

Returns TRUE if alarms are being filtered by time.

Syntax

```
CAmvConn* amvConn;
amvConn->TimeFilter->IsEnabled()
```

Data Type

```
BOOL
```

CAmvTimeFilter::SetTimeStamp()

Set the time used to filter alarms.

Syntax

```
CAmvConn* amvConn;
amvConn->TimeFilter->SetTimeStamp(COR_STAMP* ts)
```

Data Type

```
void
```

Example

Set the time filter to pass only alarms occurring after 8am on 13 February 1991

```
COR_STAMP ts;
ts->yyyymmdd = 19910213;
ts->hhmmsstt = 08000000;
amvConn->TimeFilter->SetTimeStamp(&ts);
```

See Also

CAmvTimeFilter::TimeStamp() *(page 250)*

CAmvTimeFilter::TimeStamp()

Get a pointer to the timestamp used to filter alarms.

Syntax

```
CAmvConn* amvConn;
amvConn->TimeStamp->TimeStamp()
```

Data Type

```
COR_STAMP*
```

See Also

CAmvTimeFilter::SetTimeStamp() *(page 249)*

Data Types Used by Alarm Viewer API

Alarm Viewer API Data Types

The `CAmvConn` methods use the following data types:

- AlarmInfo
- SAmapCallbacks
- textContext

Standard Data Types

Standard Data Types

The **CAmvConn** methods also use the following CIMPLICITY standard data types:

- AM_STACKED_INFO
- COR_BOOLEAN
- COR_I2
- COR_I4
- COR_STAMP
- COR_STATUS
- RCM_ALARM_DATA

AM_STACKED_INFO

The AM_STACKED_INFO structure contains information about stacked alarms. Its format is:

```
typedef struct am_stacked_info
{
   COR_STAMP     gentime;
   AM_STATE_TYPE alarm_state;
   TCHAR         alarm_msg[ALARM_MSG_LEN+1];
   COR_I1        _pad[3];           /* alignment data */
} AM_STACKED_INFO;
```

COR_BOOLEAN

The COR_BOOLEAN definition is used for Boolean data types. Its format is:

```
typedef char COR_BOOLEAN;
```

COR_I2

The COR_I2 definition is used for two-byte signed integer values. Its format is:

```
typedef short COR_I2;
```

COR_I4

The COR_I4 definition is used for four-byte signed integer values. Its format is:

```
typedef int COR_I4;
```

COR_STAMP

The COR_STAMP structure contains timestamp information. Its definition is:

```
typedef struct cor_time_stamp {
   COR_U4 dummy1;
   COR_U4 dummy2;
} COR_STAMP;
```

COR_STATUS

The COR_STATUS structure contains status information about a subroutine or function called in the code. Its format is.

```
typedef struct cor_status {
   COR_I4      status;       /* success, failure, or warning */
   COR_I4      err_source;   /* what detected the error */
   COR_I4      err_code;     /* what is the error code */
   COR_I4      err_ref;      /* where did it occur */
   COR_BOOLEAN err_reported; /* has it been logged yet ? */
   TCHAR       err_msg[80];  /* any text message */
   TCHAR       _fill[3];     /* alignment data */
} COR_STATUS;
```

RCM_ALARM_DATA

The RCM_ALARM_DATA structure is used to record the current alarm count, the date the latest alarm was generated, and the display attributes for the alarm count and alarm date displays. Its format is:

```
typedef struct {
   int alarm_count;
   COR_I2 alarm_count_attr;
   COR_I2 alarm_date_attr;
   TCHAR alarm_date[16];
} RCM_ALARM_DATA;
```

Chapter 7. Device Communications Toolkit

About the Device Communications Toolkit

The Device Communications (Devcomm) Toolkit:

- Enables you to support communications to devices for which standard CIMPLICITY software communication enablers are not available. It provides you with the libraries and build procedures you need to create communication enablers that perform the same functions as standard CIMPLICITY software communication enablers.
- Is intended for application development programmers responsible for creating code that will access devices for which standard CIMPLICITY software communication enablers are not available.
- Requires familiarity with system design, programming concepts, the Visual C++ programming and CIMPLICITY product features.

The Device Communications Toolkit supports a set of functions that enables a custom communication enabler to:

- Update the point database
- Schedule point polling
- Process unsolicited data
- Modify device data via set points
- Update device status (device available/device unavailable)
- Generate alarms when a device is not available for communications
- Generate communication enabler-specific alarms

When you create a custom communication enabler, the set of functions you support may be limited by the device, the protocol used in communicating with the device, or the scope of your implementation.

Communications Enabler Overview

- Communications Enabler Visual representation.
- Custom Communication Enabler overview.

Communications Enabler Visual Representation

The following illustration shows you how a custom communication enabler is integrated into the standard CIMPLICITY software environment.

Custom Communication Enabler Overview

1. Build Custom Communication Enablers
   a. Determine the functions to be supported.
   b. Determine the device models to be supported and the memory locations to be made accessible for each device-supported model.
   c. Write specialized routines for each supported function.
   d. Use the build procedures provided by the Device Communications Toolkit to compile the specialized routines and create an executable.

2. Create Configuration

   Before you can run the executable, you must create configuration files that identify:

   - The name of the executable
   - The port(s) the enabler can use
   - The device model(s) the enabler will support
   - The memory locations the enabler will support for each supported model (optional)

3. Execute the Customized Communications Enabler

   After you create the executable and configuration files and place them in the appropriate locations, you may configure the enabler in CIMPLICITY software and run it.

Device Communications Subroutines in the Enabler Sample

The CIMPLICITY Device Communications Toolkit API provides a customizable interface between CIMPLICITY software and device communications.

Following is a sample of how the user-customized subroutines fit into the overall execution of the enabler.

The order provided here represents a very high-level description of operation and includes only those details believed relevant to customizing an enabler.

```
INITIALIZATION
Setup internal data structures
  READ process configuration data for enabler
  READ port configuration data for enabler
  CALL user_init()
```

```
...
CALL user_open_port()
CALL user_protocol_info();
...
FOREACH CIMPLICITY DEVICE related to this ENABLER
   READ CIMPLICITY configuration data for device
  IF supported.use_default_domain_count == TOOLKIT_NO
      CALL user_device_set_max_device_domain_count()
   ENDIF
 CALL user_device_info()
   READ point configuration data for device
   IF supported.model_req == TOOLKIT_YES
     CALL user_cpu_model()
   ENDIF
   FOREACH DEVICE_POINT
     CALL user_valid_point()
     IF problem with device or point and point was validated
       CALL user_remove_point()
     ENDIF
   ENDFOR
   IF supported.det_dev_status == TOOLKIT_YES
     CALL user_device_okay()
   ENDIF
ENDFOR
...
COMPLETE INITIALIZATION BY ESTABLISHING COMMUNICATIONS
WITH INTERESTED CIMPLICITY SUBSYSTEMS
...
WHEN
   UNSOLICITED_DATA_RECEIVED:
      IF support.unsolic_req == TOOLKIT_YES
         IF user_accept_unsolicited_data() returns TRUE
            CALL user_process_unsolicited_data()
         ENDIF
      ENDIF
   TIME_TO_SCAN_POINT_VALUES:
      IF support.read_req == TOOLKIT_YES
         SETUP parameters for read
         CALL user_read_data()
         CALL user_cvt_data_from_device()
      ENDIF
   SET_POINT:
      IF support.write_req == TOOLKIT_YES
         SETUP parameters for write
         CALL user_cvt_data_to_device()
         CALL user_write_data()
      ENDIF
   WRITE_POINT_QUALITY:
      IF support.unsolicited_quality_data == TOOLKIT_YES
         IF supported.extended_user_bits == TOOLKIT_YES
             CALL user_write_point_quality2()
         ELSE
            CALL user_write_point_quality()
```

```
            ENDIF
         ENDIF
      DEMAND_STATUS_UPDATE:
         IF (point is no longer in demand)
            CALL user_on_demand_response()
         ELSE
            CALL user_on_demand_response()
         ENDIF
      TIME_TO_RETRY_FAILED_DEVICES:
         FOREACH uninitialized device
           IF supported.use_default_domain_count == TOOLKIT_NO
               CALL user_device_set_max_device_domain_count()
            CALL user_device_info()
            IF supported.model_req == TOOLKIT_YES
               CALL user_cpu_model()
            ENDIF
            FOREACH DEVICE_POINT
               CALL user_valid_point()
            ENDFOR
            IF supported.det_dev_status == TOOLKIT_YES
               CALL user_device_okay()
            ENDIF
         ENDFOR
      NEW_POINT_DYNAMICALLY_CONFIGURED:
         IF existing point
            CALL user_remove_point()
         CALL user_valid_point()
         If not found valid
            CALL user_remove_point()
         IF poll-once point
            CALL user_read_data()
      CHECK_DEVICE_STATUS:
         IF ((support.host_redundancy == TOOLKIT_YES) &&
            (currently secondary node))
            CALL user_heartbeat_device
         ENDIF
      SHUTDOWN:
         Cleanup
         CALL user_term()
      USER_EVENT_1:
         CALL user_proc_event_1()
      USER_EVENT_2:
         CALL user_proc_event_2()
      USER_EVENT_3:
         CALL user_proc_event_3()
      USER_EVENT_4:
         CALL user_proc_event_4()
      USER_EVENT_5:
         CALL user_proc_event_5()
      USER_EVENT_6:
         CALL user_proc_event_6()
      USER_EVENT_7:
         CALL user_proc_event_7()
```

```
    USER_EVENT_8:
       CALL user_proc_event_8()
    USER_EVENT_9:
       CALL user_proc_event_9()
    USER_EVENT_10:
       CALL user_proc_event_10()
 END WHEN
```

Notes on Internationalization for the Device Communications Toolkit

- Work with strings.
- Recommended reading.

Work with strings

In an international environment, strings in CIMPLICITY software can be multibyte strings. If you want your code to conform to international standards, It is recommended that you do the following when working with strings:

- Use the **TCHAR** macros found in **TCHAR.H** .
- Declare string buffers as **TCHAR[]** . Declare string pointers as **TCHAR*** or **LPTSTR** .
- Wrap string and character constants with the **_ T()** macro.
- Use the **_tcs ()** functions in place of the **str ()** functions. For example, use **_tcslen()** in place of **strlen()** .
- Be careful when incrementing a pointer through a string. Remember that a logical character may occupy one or two **TCHAR** units. So replace code that looks like this:

```
char *cp;

for (cp = string; *cp != '\0'; ++cp)

{

    …

}
```

with code that looks like this:

```
TCHAR const *cp;

for (cp = string; *cp != _T('\0'); cp = _tcsinc(cp))
```

```
{
    …
}
```

- Avoid using a variable to hold the value of a logical character. Instead, use a pointer to a character in the string. In particular, avoid the **_tcsnextc()** macro, because the value it returns appears to be incompatible with some of the C runtime library functions.
- Use the functions **_tccpy()** and **_tccmp()** and string pointers instead of the = and == operators on characters.
- Use **GetStringTypeEx()** instead of the character classification macros such as **_istalpha()** .
- Use **CharUpper()** and **CharLower()** instead of **_toupper()** and **_tolower()** .

Recommended Reading

Microsoft has several good papers on writing international code on its Web site. To find documentation on the web site, go to http://msdn.microsoft.com and search for MBCS.

The following book is also available:

Schmitt, David A., Internationalization Programming for Microsoft® Windows®, ISBN 1-57231-956-9

Device Communications Enabler Design

Device Communications Enabler Design

In CIMPLICITY software, a device communications enabler uses a specific protocol to communicate through a single port to one or more devices.

A port typically corresponds to a hardware interface such as a serial port or network adapter. Multiple copies of the same enabler may run concurrently to support several ports.

A port can connect to one or more devices. Usually these devices are addressed by CPU ID, network address, or both.

A device contains one or more domains which are addressed by domain index.

A domain is a contiguous region of homogeneous data elements that are selected by domain offset. Examples of domain elements are bits, bytes, and words. A domain that is bit-addressable is called a "bit domain" or a "digital domain".

A domain that is made up of multi-bit elements (such as bytes or words), is called an "analog domain" because its elements can hold analog values.

A point is an ID and other attributes associated with one or more contiguous elements in a domain.

Multiple points of the same or different type may be configured for the same address in a domain.

Point types need not match domain types. For example, analog values can be made up of multiple bits in a bit domain, and bits can be extracted from elements in analog domains.

Within the data buffers used by CIMPLICITY software, all data is handled in bytes.

- Bit data is stored packed, 8 bits per byte with the most significant bit to the left.
- Multi-byte data is stored little-endian (least significant bytes first). Floating point values are in 4-byte IEEE format, unless you set **use_dp_flag** in the SUPPORT structure to TOOLKIT_YES. In that case, floating point values are in 8-byte format.

Multiple points in the same domain may be grouped by the toolkit into a cache for more efficient polling. A cache is a range of memory in a domain that covers one or more points. All points in the cache have the same update criteria and scan rate multiplier. When a point I/O routine is called, the Point ID in the ADDR_DATA structure is for the "top" (or first) point in the cache.

Decisions to be Made Prior to Implementing an Enabler

Decisions to be Made Prior to Implementing an Enabler

Below is a list of decisions to make before implementing an enabler.

	Determine the device models the enabler will support. • Determine if a customized number of domains will be supported.
	Determine if a customized number of domains will be supported.
	Determine the domains to be used. • Determine if a customized number of domains will be supported.
	Determine if point and alarm identifiers will be identified by their display or internal representations.
	Determine the form of addressing used to configure points for the enabler.
	Determine the list of supported features.
	Design the specialized functions.

- Determine the name of the port(s) the enabler will support.
- Define the name of the protocol(s) the enabler will support.
- Define the name of the enabler.
- Update the configuration files for the enabler.

Determine the Device Models the Enabler Will Support

This information is used in two places:

- It is defined in a configuration file (the configuration file defines the choices for device model for any CIMPLICITY software device configured for the protocol).
- The model information from configuration is available to the user-customizable functions (the enabler can perform model verification of devices).

Determine if a Customized Number of Domains will be Supported

By default, the number of device domains that any device may have is defined by the constant `TOOLKIT_MAX_DEVICE_DOMAINS`, which is defined in

`%BSM_ROOT%\api\include\inc_path\toolkit.h`.

This default may be made larger or smaller based on the requirements of the interface.

No more than 16384 domains may be defined although practical limitations based on the size of the device communication interface itself may dictate a lower limit.

Determine the Domains to be Used

All configured points must translate into a domain and offset. Memory locations that share the same characteristics and that can be read contiguously should be grouped to form a domain.

Determine if Point and Alarm Identifiers will be Identified by their Display or Internal Representations

Point and alarm ids can be represented either by an internal identifier or by using a display name configured in the application.

Application designers can choose the format in which the data is to be provided by the device communication interface.

Note: Using the internal identifier is more efficient and is recommended.

Functions and macros are provided to convert the internal name to the display name, where required, such as storing the information in an external system or in logging error messages for an end user.

Determine the Form of Addressing Used to Configure Points for the Enabler

Valid forms of addressing are as follows:

Standard Addressing	For standardized addressing, the number and name for each supported model is predefined in the configuration file, **domain.cfg** . The point address is of the form DOMAIN_NAME and an offset.
Custom Addressing	For customized addressing, a free-format field of ASCII characters is used to specify a point address. Do not use *, - or | in the address field as they have a special meaning in the configuration files.
	No checks are performed at configuration time to determine whether the format of the address is correct. It is the enabler's responsibility to validate the point address and translate the address into a domain and offset.
	This function is accomplished in a user-provided routine. When customized addressing is used, **domain.cfg** is not used by the enabler or application configuration, but a **domain.cfg** should still exist. The file may be empty.

Determine the List of Supported Features

Features are defined on a protocol basis, but may be further limited on a device basis. Supported features are listed below:

Perform verification based on model

Supporting this functions allows the enabler to attempt model-specific verification for each device.

Support reading data from device memory

Supporting this function allows the enabler to attempt to read data from the device's memory.

Support writing data to device memory

This function must be supported if setpoints are to work. When writing bit-data, the data must first be read before the data will be written.

Support read of unsolicited data from the device

Support of this function permits the enabler to support unsolicited requests from the device.

User-defined determination of device status

If this feature is not selected, the enabler determines device status based on the success of reading the device's memory. If, after the configured number of retries, the read is unsuccessful, the device is considered to be down and a $DEVICE_DOWN alarm is generated.

Design the specialized functions

- If a function is listed as optional, the original template does not require modification.

1. Determine the name of the port(s) on which the enabler will run
2. Define the name of the protocol(s) the enabler will support
3. Define the name of the enabler
   - The build procedure produces an **.exe** and a **.dll** file for the enabler.

   The default names in the build procedure are **tlkitusr.exe** and **tlkituser.dll** .
4. Change the names to more meaningful ones.
5. Update the configuration files for the enabler
   - The configuration files of interest are as follows:

   domain.cfg (optional)

   < product >**.** **proto**

   < product >**.** **model**

   Where

   < product > is a protocol product name that you define. You must also use the product name when you add entries to the Registry.
6. Define a directory location, outside of the CIMPLICITY directory structure, to store master copies of these configuration files.

   This will ensure that the configuration files are still available after you perform updates on your CIMPLICITY software.
7. (After you make changes in the master copies of these files) do the following.
   a. Merge or copy **domain.cfg** to **%SITE_ROOT%\master** and **%SITE_ROOT%\data** . See Merging the Domain Configuration Data into a Project *(page 288)* for more information.
   b. Copy < product >**.** **proto** to **%BSM_ROOT%\bsm_data** .
   c. Copy < product >**.** **model** to **% BSM_ROOT%\bsm_data** .

Note: You can find sample files in the **%BSM_ROOT%\api\dc_api** directory.

User Customizable Functions

User Customizable Functions

The Device Communications Toolkit gives you a set of functions that you can customize for each enabler you implement.

The following functions are available for customization:

	user_accept_unsolicited_data()
	user_cpu_model()
	user_cvt_data_from_device()
	user_cvt_data_to_device()
	user_device_info()
	user_device_okay()
	user_device_set_max_device_domain_count()
	user_heartbeat_device()
	user_init()
	user_on_demand_response()
	user_open_port()
	user_proc_event_1()
	user_proc_event_2()
	user_proc_event_3()
	user_proc_event_4()
	user_proc_event_5()
	user_proc_event_6()
	user_proc_event_7()
	user_proc_event_8()
	user_proc_event_9()
	user_proc_event_10()
	user_process_unsolicited_data()
	user_process_unsolicited_data_stamp()

	user_protocol_info()
	user_read_data()
	user_read_diag_data()
	user_remove_point()
	user_term()
	user_valid_point()
	user_valid_diag_point()
	user_write_data()
	user_write_point_quality() or user_write_point_quality2()

user_accept_unsolicited_data()

Filename: **usrtm_accept.c**

Indicates whether unsolicited data, received when the function is called, should be processed.

This function is called only when unsolicited data is received between polls.

user_cpu_model()

Filename: **usrtm_cpumdl.c**

Performs model verification.

user_cvt_data_from_device()

Filename: **usrtm_cvtfrm.c**

Converts from device internal format to CIMPLICITY defined format.

user_cvt_data_to_device()

Filename: **usrtm_cvtto.c**

Converts from CIMPLICITY defined format to device internal format.

user_device_info()

Filename: **usrtm_dvin.c**

Determines device specific information (such as memory locations) and device-specific supported features (read/write support, etc.)

user_device_okay()

Filename: **usrtm_dev_ok.c**

Indicates whether device is able to communicate with the enabler.

user_device_set_max_device_domain_count())

Filename: usrtm_maxdom.c

Defines the maximum number of domains for the device when custom domain size is defined.

user_heartbeat_device()

Filename: **usrtm_hrtbt.c**

Verifies communications with a device running on the acting secondary in a host redundant configuration..

user_init()

Filename: **usrtm_init.c**

Performs initialization functions specific to the interface being created. This is the first user routine called by the Toolkit. It should do all the setup needed for the other user routines.

user_on_demand_response()

Filename: **usrtm_dmdres.c**

Defines the protocol-specific processing performed when the demand state of a point configured with an update criteria of `ondemand` transitions into or out of demand.

user_open_port()

Filename: **usrtm_opport.c**

Initiates communications with the port.

user_proc_event_1()

Filename: **usrtm_evt1.c**

Performs processing for user-defined event 1.

user_proc_event_2()

Filename: **usrtm_evt2.c**

Performs processing for user-defined event 2.

user_proc_event_3()

Filename: **usrtm_evt3.c**

Performs processing for user-defined event 3.

user_proc_event_4()

Filename: **usrtm_evt4.c**

Performs processing for user-defined event 4.

user_proc_event_5()

Filename: **usrtm_evt5.c**

Performs processing for user-defined event 5.

user_proc_event_6()

Filename: **usrtm_evt6.c**

Performs processing for user-defined event 6.

user_proc_event_7()

Filename: **usrtm_evt7.c**

Performs processing for user-defined event 7.

user_proc_event_8()

Filename: **usrtm_evt8.c**

Performs processing for user-defined event 8.

user_proc_event_9()

Filename: **usrtm_evt9.c**

Performs processing for user-defined event 9.

user_proc_event_10()

Filename: **usrtm_evt10.c**

Performs processing for user-defined event 10.

user_process_unsolicited_data()

Filename: **usrtm_unso.c**

Process the received unsolicited data.

user_process_unsolicited_data_stamp()

Filename: **usrtm_unsost.c**

Process the received unsolicited data with a timestamp.

user_protocol_info()

Filename: **usrtm_protin.c**

Defines features supported by the device and the protocol used to communicate with the device.

user_read_data()

Filename: **usrtm_readda.c**

Gets the requested data from the designated portion of the device's memory.

A non-blocking read must be implemented.

user_read_diag_data()

Filename: **usrtm_readdiag.c**

Returns diagnostic data from specific locations within the enabler.

user_remove_point()

Filename: `usrtm_rm_pt.c`

Performs user specific processing when a previously validated point is to be removed from processing within the current configuration.

user_term()

Filename: **usrtm_term.c**

Performs user termination functions.

user_valid_point()

Filename: **usrtm_valpt.c**

Performs point validation and in some cases translates the point address to domain and offset.

user_valid_diag_point()

Filename: **usrtm_valdiagpt.c**

Determines the validity of a diagnostic point.

user_write_data()

Filename: **usrtm_wrda.c**

Writes the provided data to the requested portion of the device's memory.

A non-blocking write must be implemented.

user_write_point_quality() or user_write_point_quality2()

Filename: **usrtm_wrtpqual.c**

Writes the point quality data to the device.

A non-blocking write must be implemented.

Subroutine Guidelines

- Required subroutines.
- Optional subroutines.
- Subroutines that must be customized.
- Read and write request note.

Required Subroutines

You must design the following subroutines irrespective of the supported options:

user_device_info()

user_protocol_info()

Optional Subroutines

Below is a list of subroutines that are optional regardless of implemented functions. These functions may be called within the enabler, but the templates for these subroutines should suffice when no customization is required.

- user_init()
- user_on_demand_response()
- user_open_port()
- user_proc_event_1()
- user_proc_event_2()
- user_proc_event_3()
- user_proc_event_4()
- user_proc_event_5()
- user_proc_event_6()
- user_proc_event_7()
- user_proc_event_8()
- user_proc_event_9()
- user_proc_event_10()
- user_remove_point()
- user_term()

Subroutines That Must Be Customized

Below is a list of subroutines that must be customized based on the list of supported features:

For	Subroutine

Custom addressing	• user_valid_point()
Model verification	• user_cpu_model()
Domain definition	• user_device_set_max_device_domain_count())
Reading device data	• user_read_data()
Writing device data	• user_write_data()
Writing point quality data	• user_write_point_quality() and/or • user_write_point_quality2()
Receiving unsolicited data	• user_accept_unsolicited_data() • user_process_unsolicited_data() • user_process_unsolicited_data_stamp()
Device status	• user_device_okay()
Reading diagnostic data	• user_valid_diag_point() • user_read_diag_data()
Host redundancy	• user_heartbeat_device()

Read and Write Requests Note

Read and Write requests to devices are often grouped to improve efficiency. A grouped request fails if any point in the group is invalid. For this reason, it is **strongly** recommended that regardless of addressing-mode, you should use **user_valid_point()** to verify that data at the configured memory location is readable if read requests are supported.

Implementation Checklist

Implementation Checklist

The following files are all located in **%BSM_ROOT%\api\dc_api** .

Important: The files in this directory will be overwritten when you un-install, reinstall, or upgrade your CIMPLICITY software. For each enabler you develop

1. Create a separate directory and copy files from %BSM_ROOT%\api\dc_api to that directory.
2. Develop your software in the separate directory.

   guide: Guidelines

   Follow the guidelines in Decisions to be Made Prior to Implementing an Enabler and generate the list of supported memory locations, models and features. Based on this list, design each of the needed subroutines.

   Do the following to create your driver:

3. Implement each of the designed subroutines.
4. Install the enabler, once built, into the CIMPLICITY environment.
5. Create the configuration data necessary to run the executable.

   Files

   The following gives you a brief list of the subroutines to be updated and items that you need to take into consideration when updating them.

	user_init() and user_term()
	user_protocol_info()
	user_device_info()
	user_open_port()
	user_cpu_model()
	user_device_set_max_device_domain_count()
	user_valid_point()
	user_read_data()

	user_remove_point()
	user_write_data()
	user_write_point_quality() or user_write_point_quality2()
	user_process_unsolicited_data(), user_accept_unsolicited_data(), and user_process_unsolicited_data_stamp()
	user_on_demand_response()

user_init() and user_term()

If enabler-specific initialization and termination steps are required, implement **user_init()** in **user_init.c** and **user_term()** in **usrtm_term.c** .

user_protocol_info()

Determine the supported features for the protocol and implement **user_protocol_info()** in **usrtm_protin.c** . Supported features may be further limited on a device by device basis.

user_device_info()

1. Determine the list of supported features for each device. The list of supported features must be a subset of those supported by the protocol. Also, determine the list of supported memory locations.
2. Memory locations sharing the same characteristics and which can be read contiguously should be grouped to form a domain.
3. Assign a domain index to each domain. If using standard addressing, each domain should also be assigned a name of up to 16 characters. The domain index must be between '0' and '254', and should be sequential. Each domain must have a unique index (and where standard addressing is used, a unique name).
4. Determine the value used to identify the first memory location in each domain.
5. Determine the number of elements in each domain.
6. Determine the form of addressing used to reference an element in the domain. The available forms are:

   TOOLKIT_BIT (1 bit long)

   TOOLKIT_BYTE (1 byte long)

TOOLKIT_WORD (2 bytes long)

TOOLKIT_4BYTE (4 bytes long)

TOOLKIT_8BYTE (8 bytes long)

Incorporate the above information in **user_device_info()** in **usrtm_dvin.c** .

user_open_port()

If a port must be opened to communicate with the device, implement **user_open_port()** in **usrtm_opport.c** to perform this function.

Any initialization of the port that must be done through the operating system should be performed in this routine. In addition, the baud rate and parity passed to this routine may be overridden by the routine.

user_cpu_model()

Implement **user_cpu_model if model_req** is 'TOOLKIT_YES'.

This function should perform device-specific verification such as model verification. Communications with the device to perform this function may be performed by the subroutine. The template is located in **usrtm_cpu_mdl.c** .

user_device_set_max_device_domain_count()

Implement `user_device_set_max_device_domain_count()` in `usrtm_maxdom.c if use_default_domain_count` is `TOOLKIT_NO`

Defines the maximum number of domains for the device.

The maximum number of defined domains that the software will allow is 16384.

Practical considerations for performance and size may impose a smaller limit.

user_valid_point()

Implement **user_valid_point** in **usrtm_valpt.c** .

If custom addressing is used, this function must translate the string form of the address to its corresponding domain and offset. If the address is invalid or the address is out of the range, the point should be returned as invalid.

Regardless of addressing mode, if the device has configurable memory, this function should determine whether the point address is a valid one. One method for accomplishing this is to attempt to read the data value from the configured location in the device's memory. A second method is to read the memory configuration from the device and use this information to determine if the address is valid.

user_read_data()

Implement **user_read_data in usrtm_readda.c** if **read_req** is 'TOOLKIT_YES'.

This function translates the given address into a physical memory location. It then queries the device to request the data starting at the given location for the given number of bytes. The information is then returned via the data buffer along with the status indicating the status of the read.

user_remove_point()

Implement `user_remove_point in usrtm_rm_pt.c` if there is specific processing to be done within the interface when a previously valid point is removed from the configuration.

user_write_data()

Implement **user_write_data** in **usrtm_wrda.c** if **write_req** is 'TOOLKIT_YES'.

This function translates the given address to a physical memory location and requests that the device change a range of memory starting from this address to the values contained in the data buffer.

user_write_point_quality() or user_write_point_quality2()

Implement `user_write_point_quality` or `user_write_point_quality2`.

These functions write point quality data to the device.

When the write request is made,

- If `extended_user_bits` is `TOOLKIT_YES`, then `user_write_point_quality2` is called to perform the requested operation.

- Otherwise `user_write_point_quality` is called.

user_process_unsolicited_data(), user_accept_unsolicited_data(), and user_process_unsolicited_data_stamp()

Implement **user_process_unsolicited_data()** in **usrtm_unso.c** and **user_accept_unsolicited_data()** in **usrtm_accept.c** if unsolicited data is to be supported.

If you want to timestamp the unsolicited data, implement **user_process_unsolicited_data_timestamp()** in **usrtm_unsost.c** instead of **user_process_unsolicited_data()** .

user_accept_unsolicited_data() should return TRUE if the processing for received unsolicited data is performed asynchronously.

If **unsolicited_quality_data** is 'TOOLKIT_YES', the data buffer in **user_process_unsolicited_data_timestamp** and **user_process_unsolicited_data** must be able to handle the receipt of unsolicited quality data.

user_on_demand_response()

Implement **user_on_demand_response()** in **usertm_dmdres.c** if you need to perform device-specific actions when points with an `ONDEMAND` update criteria are placed in or out of demand.

Create the Executable Image

Create the Executable Image

Creation of an executable customized communication enabler is specific to the operating system on which the executable is built and executed.

To build and link a customized communication enabler for your CIMPLICITY system, you must install the Device Communications Toolkit API and the Microsoft Visual C++ compiler.

The build procedure for the Device Communications Toolkit produces two files:

- The executable file (< name > **.exe**)
- A dynamically linked library file (< name > **.dll**)

Note: Depending on how you installed Visual C++, the INCLUDE, LIB and PATH environment variables may not be automatically set when you install MSDEV. If they are not set automatically,

you will have to set them manually or run the following to set them before building any user programs.

```
 for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft Visual
 Studio\Installer\vswhere.exe" -property installationPath`) do set VSPATH=
%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
```

Build a Communication Enabler

1. Select Tools>Command Prompt on the Workbench menu bar.
2. In the Command Prompt... window, issue the following commands:

   < drive >

   cd %BSM_ROOT%\api

   where < drive > is the disk where your CIMPLICITY software is installed (for example, **C:**).

3. If environment variables are not set automatically, issue the following command to set them.

   ```
   for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
    Visual Studio\Installer\vswhere.exe" -property installationPath`) do
    set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
   ```

4. Launch Visual Studio.
   `devenv CimplicityAPI.sln`
5. Modify the toolkit project to include the compilation and linking of any additional files created for your customized communication enabler.
6. Right-click the project in Solution Explorer.
7. Select **Build** on the Popup menu.
   The customized communication enabler is built.

   Note: The executable and .dll file will be located automatically in %BSM_ROOT%\exe.

Add a new Driver

Add a new Driver

1 *(page 277)*	Identify the Protocol and model information.
2 *(page 278)*	Add new entries to the Registry.
3 *(page 279)*	Define the Protocol.
4 *(page 286)*	Define the model.
5 *(page 288)*	Merge user files into CIMPLICITY software.
6 *(page 288)*	Merge the Domain configuration file into a project.

1. Identify the Protocol and Model Information

- Process overview.
- Special characters In configuration files.
- Definition of terms.

Process Overview

When you add a new driver, you will need to:

1. Add entries to the Registry for the new devcom.
2. Define the protocol in < product >**.** **proto** .
3. Define the device models in < product >**.** **model** .
4. Copy < product >**.** **proto** and < product >**.** **model** to **%BSM_ROOT%\bsm_data** .
5. Create **domain.cfg** .
6. Merge the devcom's **domain.cfg** file with the current **domain.cfg** file in the project's **%SITE_ROOT%\master** and **%SITE_ROOT%\data** directories.

   Special Characters In Configuration Files

Some characters have special significance in CIMPLICITY configuration files and should only be used in the following ways:

Vertical Bar	**|**	The vertical bar is interpreted as a field delimiter. This character should be avoided.
Dash	**-**	The dash is interpreted as a continuation character when placed at the end of a line of an ASCII configuration file. Users should avoid placing a dash at the end of a field of configuration data.
Asterisk	*****	The asterisk is interpreted as a comment character when positioned in the first position of a record. Users should avoid placing an asterisk as the first character of identifiers.

The first line in the < product >**. proto** , < product >**. model** , and **domain.cfg** files must consist solely of these characters in this order:

```
| - *
```

Do not use any of these characters in the data fields of those configuration files.

Definition of Terms

You should understand the following terms:

Port Prefix	Three to six character prefix used to identify ports of the same type.
	Note: Port Prefix and Port Type are used interchangeably.
Protocol	Common name used to refer to the protocol supported by the enabler.
Master Directory	**%SITE_ROOT%\master**
Data Directory	**%SITE_ROOT%\data**

2. Add new Entries to the Registry

First, you will need to add some entries to the Registry to allow the CIMPLICITY Configuration program to recognize your new device communication interface (devcom).

Important: Making changes to the Registry is very dangerous and should be done with care.

Do the following to add the new devcom to the Registry:

1. Open an MS-DOS window.
2. Type **regedt32** at the command prompt.

The Registry Editor opens.

3. Select the following.

   32-bit machines

   ```
   HKEY_LOCAL_MACHINE>SOFTWARE>GE Fanuc>CIMPLICITY>HMI>CIMPLICITY
   Version>Products
   ```

   64-bit machines

   ```
   HKEY_LOCAL_MACHINE>SOFTWARE>Wow6432Node>GE Fanuc>CIMPLICITY>HMI>CIMPLICITY
   Version>Products
   ```

4. Click Edit>Add Key on the Registry Editor menu bar to add a key for your new devcom.

   Important: The **Key Name** must be in upper-case and match the <product> prefix you gave to your **. proto** and **. model** files.

5. Select the new key.

6. From the Edit menu, select Add Value..., enter **Name** in the **Value Name** field and REG_SZ in the **Data Type** field, then click **OK**. In the **String** field, enter the name that you want to be displayed for the protocol when you create a project, then click **OK** again.

7. From the Edit menu, select Add Value..., enter **Serial Number** in the **Value Name** field, then click **OK**. Leave the **String** field blank, then click **OK** again.

8. From the Edit menu, select Add Value..., enter **Type** in the **Value Name** field and "REG_SZ" in the **Data Type** field, then click **OK**. In the **String** field, enter **Protocol**, then click **OK** again.

9. Exit the Registry.

3. Define the Protocol

3. Define the Protocol

- <product>.proto
- Sample configuration file.
- Protocol field definitions.

<product>.proto

The <product>.proto file contains the information needed to install the protocol as part of your CIMPLICITY system.

The format of the .proto file is as follows:

Sample Configuration File

```
|-*
NETW|501|BSM_ROOT:[exe]netw_rp.exe|20|1|NET|1|20|10|2|54|1|-
10|1000|1000|1000|50|1|3|1|1|1|19200|2|-
N|30|N|3|900|N|N
```

Protocol Field Definitions

The fields in the record are ordered as follows:

- protocol_id
- protocol_number
- image_name
- Priority
- PM_flags
- port_prefix
- low_index
- high_index
- Base
- address_type
- allow_unsolicited
- min_points_to_poll
- forced_poll_count
- load_buffer_size
- cache_buffer_size
- min_scan
- scan_units
- dead_count
- read_all
- poll_state
- max_associations
- baud_rate
- parity
- check_health
- response_period
- restart
- threshold

- retry_period
- kill_process
- fail_process

protocol_id

ASCII string of up to 16 characters identifying the protocol.

protocol_number

Unique number required for each user-protocol (must be unique to the CIMPLICITY system).

Valid values range from **500 - 999** (values outside this range are reserved for use by GE Intelligent Platforms).

image_name

Fully qualified name of the executable (that is, the full path name from the top of the Base (or Root) directory.

Priority

Process priority.

Always set to 20.

PM_flags

Process Management startup and shutdown options for this process. Set the flag according to the following:

1	Cause Process Management to wait for a check-in message from this process when it is initializing. This value must always be used.
2	Cause Process Management to not generate an alarm when the process terminates
8	The Process Manager can reset the Process Down alarm that gets generated when this process terminates.

You can sum values to combine options. For example, if you want to have Process Management wait for a check-in message and reset the Process Down alarm, enter a "9" in this field.

port_prefix

3- to 6-character prefix defining the base name used for the port or group of ports being defined.

Example

‘`COM`’ or ‘`TCP_IP`’.

Note: Port prefixes should always be upper-case. Valid port IDs are defined based on the concatenation of the port prefix with an index (except for noted special case).

low_index

Lowest valid index value for the port; specified in decimal.

Valid range is '0' to '999' and '-1'

Note: If the port being defined is a single port that does not end in a digit, both the Low and High values should be set to '-1' to indicate that no range is allowed for this port prefix.

high_index

Highest valid index value for the port; specified in decimal.

Valid range is '0' to '999' and '-1'.

Note: If the port being defined is a single port that does not end in a digit, both the Low and High values should be set to '-1' to indicate that no range is allowed for this port prefix.

Base

Indicates the number base used when specifying port IDs.

Current valid values are:

10	Decimal

address_type

1. Standard addressing
2. Custom addressing

allow_unsolicited

Defines the update criteria available for points using this protocol.

Values are defined as follows:

1	Support Unsolicited point updates
2	Support On Demand point updates
4	Support Poll Once point updates
8	Support Ethernet Global Data
16	Support On Change point updates
32	Support On Scan point updates

You can sum values to combine options. For example, if you want to support Unsolicited and On Demand unsolicited data, enter a **3** in this field.

The recommended default for this field is **48** (On Change and On scan point updates available).

min_points_to_poll

When the enabler determines it is time to query a device for point values, it groups the points within like domains into caches. By definition, a cache starts at a given memory location and spans a given number of bytes.

Polling of points may be interrupted by certain events such as a request to set a point value. The **min_points_to_poll** variable defines how many caches must be processed before the query of point values may be interrupted.

forced_poll_count

Reserved for GE Intelligent Platforms use (set this value to **10**).

load_buffer_size

Set this value to the **max_buffer_size** defined in **user_protocol_info** .

If the values differ, than the lesser of the values will take precedence

poll_buffer_size

Set this value to the **max_buffer_size** defined in **user_protocol_info** .

cache_buffer_size

For Toolkit-derived enablers, set this value to the **max_buffer_size** defined in **user_protocol_info** .

If the values differ, than the lesser of the values will take precedence.

min_scan

This number along with the units defined in **scan_units** defines the minimum scan rate to be used for this driver. All point scan rates are multiples of the minimum scan rate.

scan_units

The units associated with **min_scan** . Select one of the following:

1	Ticks (100 ticks = 1 second)
2	Seconds
3	Minutes
4	Hours

dead_count

The length of time the driver will wait before declaring a device dead. This number is a multiple of the minimum scan rate.

read_all

Reserved for GE Intelligent Platforms use.

Set this value to **1**.

poll_state

The initial state of the point polling function. Select one of the following:

0	Disabled
1	Enabled

max_associations

The maximum number of associations that may be performed by the driver.

Set this value to **1**.

baud_rate

Number stored for the baud rate in the port parameter settings

parity

Number stored for the parity in the port parameter settings.

check_health

Process health value for Check Health.

Y	Enables the feature
N	Disables the feature
Default	N (recommended).

When enabled, messages are sent to the process to determine if it is running correctly.

Example

A process may appear to be running when, in fact, it is hung.

response_period

When Process Health is enabled, the number of seconds that elapse between polls for health checks.

Recommended default value is 30 seconds.

restart

When process health is enabled, this enables or disables the restart of a process as a result of the detection of a problem through process health.

Y	Restarts the project.
N	Does not restart the project.
Default	N (recommended).

threshold

When Process Health is enabled, the number of times the process can be restarted, within the number of seconds specified by the response period, before it is failed in Process Health.. Recommended default value is 3.

retry_period

When process health is enabled, this is the number of seconds in which the restart threshold is operational.

Recommended default value is 900.

kill_process

When process health is enabled, specifies whether the process should be terminated if it does not respond when it is polled.

Y	Terminates the project.
N	Does not terminate the project
Default	N (recommended).

fail_process

When a project is running on a cluster and process health is enabled, specifies whether the project should be failed when the selected process fails.

Y	Fails the project.
N	Does not fail the project
Default	N (recommended).

4. Define the Model

4. Define the Model

- <product> model file.
- Sample configuration file.
- Model field definitions.

<product> model file

The < product >. **model** file defines the device models supported by a protocol. Each combination of protocol and model must be defined in a separate record.

⚠ **Important:** Model numbers must be unique across CIMPLICITY. If you have more than one protocol, you cannot re-use the same model number in different < product >. **model** files.

Application configuration uses this information during device configuration to generate the list of models that can be configured for a device executing on a port running the given protocol.

Sample Configuration File

To define a pair of models, Model A1 and Model A1-Plus that are valid for the SLP protocol, the entries would be as follows:

```
|-*
SLP|Model A1|500
SLP|Model A1-Plus|501
```

Model Field Definitions

The records in this file contain the following fields.

- protocol_id
- model
- model_number

protocol_id

ASCII string of up to 35 characters identifying the protocol (must be defined in the **protocol** file).

Model

ASCII string of up to 35 characters used to represent the device model.

model_number

Unique numeric identifier for the model.

Value should be between **500** and **999** (values outside this range are reserved by GE Intelligent Platforms).

5. Merge User Files into CIMPLICITY Software

After you have modified < product >. **proto** and < product >. **model** files, copy the files to the **%BSM_ROOT%\bsm_data** directory.

6. Merge the Domain Configuration File into a Project

6. Merge the Domain Configuration File into a Project

After you have defined the **domain.cfg** file for the enabler, you need to merge the information in this file into the **domain.cfg** file for each project where you want to use the enabler. To do this:

1. From the CIMPLICITY Workbench for the project, select Command Prompt... from the Tools menu.

   This will ensure that you environment variables (in particular **%BSM_ROOT%** and **%SITE_ROOT%** are set correctly.

2. To update or create the **domain.cfg** file in **%SITE_ROOT%\master** , you need to do one of the following:
   a. If you currently have a %SITE_ROOT%\master\domain.cfg file, issue the following commands to merge the domain information for your new enabler:

   ```
   cd %SITE_ROOT%\master
   ```

   ```
   notepad domain.cfg
   ```

   a. In the Notepad window, copy the information from your domain.cfg into the opened file.
   b. Close and save the modified **domain.cfg** file.
   c. If you do not have a %SITE_ROOT%\master\domain.cfg file, copy your domain.cfg file to the %SITE_ROOT%\master directory.

3. Issue the following command to copy the updated domain.cfg file to %SITE_ROOT%\data:

   copy domain.cfg %SITE_ROOT%\data

Domain Configuration - domain.cfg

- Memory to form a domain.
- Standard-Address Mode Enablers.
- Sample domain.cfg file.

- Custom address mode enablers.

Memory to form a domain

Accessible memory that can be read contiguously and that shares the same characteristics, are typically grouped together to form a domain.

Elements within a domain must be readable/writable by a single request to read or write via **user_read_data** and **user_write_data** .

Standard-Address Mode Enablers

An additional configuration file is required if an enabler uses the standard addressing scheme. The file, **domain.cfg** , defines the different types of memory supported for each device model. The file must exist in the **%SITE_ROOT%\master** and **%SITE_ROOT%\data** directories.

Use the domain names and domain indexes defined in this file when defining the starting address, size, and address type of standard domains in **user_device_info** .

The record format for this file is:

```
model|domain_name|domain_index
```

where the fields are defined as follows:

model	ASCII string of up to 35 characters used to represent the device model.
domain_name	ASCII representation of a type of device memory.
domain_index	Unique number identifying domain.

Note: The maximum number of domains is defined by the symbol TOOLKIT_MAX_NUM_DOMAINS. The current value of the symbol is found in **%BSM_ROOT%\api\include\inc_path\toolkit.h** . This value should not be changed.

Sample domain.cfg File

The following is the domain.cfg file used by the Device Communications Toolkit API:

```
|-*
MODELA|REG_PLC|0
MODELA|INP_PLC|1
MODELA|OUT_PLC|2
MODELA|INOVR_PLC|3
MODELA|OUTOVR_PLC|4
MODELA|SP_PLC|5
MODELA|UL_PLC|6
```

Custom-Address Mode Enablers

- If the enabler uses a custom mode of addressing, domain.cfg must still exist, but the file may be empty.
- domain.cfg must exist in the %SITE_ROOT%\master and %SITE_ROOT%\data directories.
- When creating an enabler, care must be taken to avoid conflicts with other existing custom enablers.

Note: Site and Application configuration must be run before the enabler can be used within CIMPLICITY software.

Device Communications API Sample Program

Device Communications API Demonstration Application

A demonstration application is distributed with the Device Communications Toolkit API. It is an example of a very simple driver enabler. The demonstration is an application where device memory is simulated. No actual communication with a device is involved, hence no protocol is needed. In this example the device name is TOOLKIT_DEVICE, and there are 7 different types of memory (domains), as follows:

Name	Size
REG_PLC	300 words
INP_PLC	128 bytes
OUT_PLC	128 bytes
INOVR_PLC	128 bytes
OUTOVR_PLC	128 bytes
SP_PLC	32 bytes
UL_PLC	300 words

Where 1 word = 2 bytes

The demo may be built and installed as a CIMPLICITY software application as discussed below.

Note: The hardware and software requirements for building the demo program are the same as those for building any driver enabler.

Build the Device Communications API Sample Program

A sample Microsoft C++ project, `tlkittst_dll.vcxproj`, is provided to build the demonstration program.

Use this project as a basis for constructing projects for your own application.

Perform the following steps to build the executable:

1. Select Tools>Command Prompt on your project's Workbench menu bar.

   This will ensure that your environment variables (in particular **%BSM_ROOT%** and **%SITE_ROOT%**) are set correctly.

2. Issue the following commands in the Command Prompt… window, :

   <drive>

   cd %BSM_ROOT%\api

   where <drive> is the disk where your CIMPLICITY software is installed (for example, C:).

3. If the environment variables are not set automatically, issue the following command to set them:

   ```
   for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
    Visual Studio\Installer\vswhere.exe" -property installationPath`) do
    set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
   ```

4. Start Visual Studio:

   devenv CimplicityAPI.sln

5. Right click tlkittst_dll in the Solution Explorer.

6. Select **Build** on the Popup menu.

7. Issue the following commands to copy files to the correct directories:

   ```
   copy dc_api\TLKITTST.PROTO %BSM_ROOT%\bsm_data
   copy dc_api\TLKITTST.MODEL %BSM_ROOT%\bsm_data
   ```

8. Now update or create the **domain.cfg** file in **%SITE_ROOT%\master** . You need to do one of the following:
   - If you currently have a %SITE_ROOT%\master\domain.cfg file, issue the following commands to merge the domain information for the demo:
     - `cd %SITE_ROOT%\master`
     - `notepad domain.cfg`

- From Notepad, copy the information from %BSM_ROOT%\api\dc_api\domain.cfg into the opened file.
- Close and save the modified **domain.cfg** file. If you do not have a %SITE_ROOT%\master\domain.cfg file, then:
  - `copy domain.cfg %SITE_ROOT%\master`

9. Issue the following command to copy the updated domain.cfg file to %SITE_ROOT%\data:

   ```
   copy domain.cfg %SITE_ROOT%\data
   ```

10. Create the TLKITTST key in the Registry *(page 278)* .

11. Under the TLKITTST key, create a **Value Name** of **Name** with the string **TLKITTST Devcom** and the **Value Name** of **Type** with the string **Protocol**
    The demonstration communication enabler should now be able to run.

Programming Notes

Programming Notes

The following restrictions apply when customizing the Device Communications Toolkit to create an enabler:

- On each new release of CIMPLICITY software, Toolkit derived applications should be recompiled and re-linked to ensure compatibility with the new release.
- Whenever upgrading, copy all user-written functions and header files to another directory prior to performing the upgrade. Failure to save these files will result in their replacement by the template files distributed with the product.
- Use **cor_sleep** instead of **Sleep** or **SleepEx** within the application.
- User functions that return both **comm_status** and **status** should set **status** to TOOLKIT_FAILURE when **comm_status** is not TOOLKIT_SUCCESS.
- The function **user_read_data** may be asked to read up to 7 bits beyond the end of a bit domain. This will occur when a bit within the last byte of the domain is configured. The bits outside of the domain should be set to zero (0) by the **user_read_data** function.
- When unsolicited data is received and processed by **user_process_unsolicited_data** , only those configured points whose address exactly matches the start address will reflect the changed value because of processing by **user_process-unsolicited_data** . If a point has an update criteria other than UNSOLICITED, the point value will be updated during normal polling based on the update criteria.
- Array points that extend beyond the end of memory are not checked. The communications interface checks that the first element of the array is within bounds, but does not check if any

of the following elements is out of bounds. Points configured beyond the end of memory will cause reads of the device to fail, resulting in the unavailability of point data.

Handle Event Flags

Use the Device Communications Toolkit functions to manipulate event flags.

A CIMPLICITY software event flag is a bit within a predefined data structure. Applications can reserve some event flags to indicate some condition has occurred. Routines are provided to SET and CLEAR the event flag.

The enabler calls predefined (user-defined) functions when an event flag is SET. Ten (10) event flags are available to the enabler.

The following functions manipulate these flags:

- dcrp_call_on_time
- dcrp_clear_ef
- dcrp_get_any_ef
- dcrp_get_ef
- dcrp_release_ef
- dcrp_set_ef

Note: Some operating systems provide primitives for defining and manipulating event flags. It is recommended that the CIMPLICITY software-provided routines be used instead to avoid conflict with CIMPLICITY software routines within the enabler.

The Device Communications Toolkit reserves 10 event flags for use with user-customized subroutines and provides functions for reserving and manipulating the event flags.

Event flags should be cleared within the user-defined function for processing the event flag.

Device Communications Toolkit Subroutines

Device Communications Toolkit Subroutines

	user_accept_unsolicited_data
	user_cpu_model
	user_cvt_data_from_device
	user_cvt_data_to_device

	user_device_info
	user_device_okay
	user_device_set_max_device_domain_count()
	user_heartbeat_device
	user_init
	user_on_demand_response
	user_open_port
	user_proc_event_1
	user_proc_event_2
	user_proc_event_3
	user_proc_event_4
	user_proc_event_5
	user_proc_event_6
	user_proc_event_7
	user_proc_event_8
	user_proc_event_9
	user_proc_event_10
	user_process_unsolicited_data
	user_process_unsolicited_data_stamp
	user_protocol_info
	user_read_data
	user_read_diag_data
	user_remove_point
	user_term
	user_valid_diag_point
	user_valid_point
	user_write_data
	user_write_point_quality

user_accept_unsolicited_data

Determines whether unsolicited data between polls may be processed.

You can find the template for this subroutine in:

```
usrtm_accept.c
```

Syntax

```
int user_accept_unsolicited_data()
```

Input Arguments

None.

Output Arguments

None.

Return Value

This subroutines returns one of the following:

- TRUE
- FALSE

If TRUE, process the unsolicited data received now.

If FALSE, do not process the unsolicited data received at this time. No attempt is made to read the unsolicited data unless one of the following occurs:

- **dcrp_notify_unsolicited_data()** is called to initiate another attempt at unsolicited data.
- Unsolicited data is processed transparent to the Device Communications Toolkit. The function **dcrp_rcv_unsolicited_data()** or **dcrp_rcv_unsolicited_data_stamp()** may be called to force the processing of the received unsolicited data by the Toolkit.

user_cpu_model

Is called during initialization and verifies the configuration of the device model.

You can find the template for this subroutine in:

```
usrtm_cpumdl.c
```

Syntax

```
void user_cpu_model(DEVICE_DATA *device_struct,
```

```
                    int *comm_status,
                    int *status)
```

Input Arguments

```
device_struct
```

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

Output Arguments

```
comm_status
```

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_cvt_data_from_device

Converts data from the general toolkit format to a format compatible with the given point type.

The Device Communications Toolkit calls **user_read_data()** to read data for multiple points at the same address. The data returned by **user_read_data()** should be in a format consistent with the device domain and the host data format.

- For example, data from a word domain should be returned in little endian format (least significant byte first). If data in the device is in big endian format (most significant byte first), you should byte swap the data within **user_read_data()** .

The Device Communications Toolkit calls **user_cvt_data_from_device()** for individual points to perform data conversions specific to the point data type.

- For example, if the point data type is INT (16 bit integer), it may be necessary to byte swap the little endian data returned **from user_read_data()** .

You can find the template for this subroutine in:

```
usrtm_cvtfrom.c
```

Syntax

```
void user_cvt_data_from_device ( DEVICE_DATA *device_struct,
                                 ADDR_DATA *address_struct,
                                 int element_cnt,
                                 int element_size,
                                 int pnt_data_type,
                                 char *data_buffer)
```

Input Arguments

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit >**.**

element_cnt

Is the number of point elements to convert.

element_size

Is the number of bytes per point element.

pnt_data_type

Is the CIMPLICITY point type. The standard CIMPLICITY point types supported by the Device Communications Toolkit are:

- TOOLKIT_BOOLEAN
- TOOLKIT_BITSTRING
- TOOLKIT_OCTETSTRING

- TOOLKIT_TEXTPOINT
- TOOLKIT_UNSIGNED_ANALOG8
- TOOLKIT_UNSIGNED_ANALOG16
- TOOLKIT_UNSIGNED_ANALOG32
- TOOLKIT_ANALOG8
- TOOLKIT_ANALOG16
- TOOLKIT_ANALOG32
- TOOLKIT_FLOATINGPOINT

In addition, a device communication interface may optionally support:

- TOOLKIT_UNSIGNED_ANALOG64
- TOOLKIT_ANALOG64

Output Parameters

data_buffer

Is the buffer of data to be converted.

Return Value

None.

user_cvt_data_to_device

Converts data from the format for the given point type to the general toolkit format.

The Device Communications Toolkit calls **user_write_data()** to write to the same device domain from various points of different data types. The **user_write_data()** subroutine assumes data is in little endian format (least significant byte first) and is otherwise consistent with the data type of the device domain.

- For example, an SINT (8 bit integer) point can be configured in a bit domain and used to set eight (8) bits at a time, or the individual bits can be set with BOOL setpoints.

The Device Communications Toolkit calls **user_cvt_data_to_device()** for individual points to perform data conversion specific to the point data type.

- For example, if the point data type is not SINT (8 bit integer), it may be necessary to swap bytes or swap words in the point data to make it little endian for **user_write_data().**

You can find the template for this subroutine in:

```
usrtm_cvtto.c
```

Syntax

```
void user_cvt_data_to_device ( DEVICE_DATA *device_struct,
                               ADDR_DATA *address_struct,
                               int element_cnt,
                               int element_size,
                               int pnt_data_type,
                               char *data_buffer)
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

element_cnt

Is the number of point elements to convert.

element_size

Is the number of bytes per point element.

pnt_data_type

Is the CIMPLICITY point type. The standard CIMPLICITY point types supported by the Device Communications Toolkit are:

- TOOLKIT_BOOLEAN
- TOOLKIT_BITSTRING
- TOOLKIT_OCTETSTRING
- TOOLKIT_TEXTPOINT
- TOOLKIT_UNSIGNED_ANALOG8
- TOOLKIT_UNSIGNED_ANALOG16
- TOOLKIT_UNSIGNED_ANALOG32
- TOOLKIT_ANALOG8
- TOOLKIT_ANALOG16
- TOOLKIT_ANALOG32

- TOOLKIT_FLOATINGPOINT

In addition, a device communication interface may optionally support:

- TOOLKIT_UNSIGNED_ANALOG64
- TOOLKIT_ANALOG64

Output Parameters

data_buffer

Is the buffer of data to be converted.

Return Value

None.

user_device_info

Defines the device-specific characteristics for the accessible memory on the device.

Accessible memory sharing the same characteristics, and which can be read contiguously, is typically grouped together to form a domain. Elements within a domain must be readable/writable by a single request to read or write via **user_read_data()** and **user_write_data().**

You can find the template for this subroutine in:

```
usrtm_dvin.c
```

Syntax

```
void user_device_info(DEVICE_DATA *device_struct,
                      int *num_domains,
                      DOMAIN_ARRAY *domain,
                      SUPPORT *supported,
                      int *comm_status,
                      int *status)
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

Output Parameters

num_domains

Is the number of domains defined for the device.

domains

Is a pointer to an array of domain structures which define the characteristics of each group of memory locations.

The number of elements in the structure is either `TOOLKIT_MAX_DEVICE_DOMAINS` or the value returned by `user_device_set_max_device_domain_count()` if `use_default_domain_count` is set in the supported structure for the protocol.

The first domain element is **domain[0]** , and all elements through **domain[*num_domains - 1]** should contain valid data.

DOMAIN_ARRAY *(page 356)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

supported

Is a pointer to a structure defining the supported options for the device. SUPPORT *(page 357)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

Programming Note

The default value for each option is the value set in **user_protocol_info()** . A field whose value is TOOLKIT_NO cannot be reset to TOOLKIT_YES in this function. If the value for a return by **user_protocol_info()** is TOOLKIT_NO and it is reset to TOOLKIT_YES in this function, the new value is ignored.

user_device_okay

Defines whether the given device is able to communicate with the enabler at the time the function was invoked. This function may be called during initialization.

You can find the template for this subroutine in:

```
usrtm_dev_ok.c
```

Syntax

```
int user_device_okay(DEVICE_DATA *device_struct)
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

Output Parameters

None

Return Value

This subroutine returns one of the following:

- TRUE
- FALSE

If TRUE, the device is able to communicate with the communication enabler.

If FALSE, the device is not able to communicate with the communication enabler.

user_device_set_max_device_domain_count()

Defines the maximum number of domains for a device.

You can find the template for this subroutine in:

`usrtm_maxdom.c`

Syntax

void user_device_set_max_device_domain(DEVICE_DATA *device_struct,

int * max_domains_allowed,

int *comm_status,

int *status);

Input Parameters

`device_struct`

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a `typedef` to a structure defined in <inc_path/toolkit.h>.

Output Parameters

`max_num_domains_allowed`

Defines the maximum number of device domains that can be configured. Valid values are between 1 and 16384 although practical limitations including due to size and/or performance may impose a lessor upper limit.

`comm_status`

Indicates whether a status of `TOOLKIT_FAILURE` occurred as a result of a communication failure.

Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

`status`

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check `comm_status` to see if the failure was the result of a communication failure.

Return Value

None.

user_heartbeat_device

Verifies communication with a device for host redundancy support.

The **device_struct** and **address_struct** are initialized by the Toolkit to refer to a valid device and address in that device. Depending on your protocol requirements, you may be able to read the data to verify communications.

You can find the template for this subroutine in:

```
usrtm_hrtbt.c
```

Syntax

```
void user_heartbeat_device (DEVICE_DATA *device_struct,
                            ADDR_DATA *address_struct,
                            int length,
                            int *comm_status,
                            int *status);
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to a valid point address for the device. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

length

Contains the number of bytes of data which should be readable at the specified address.

Output Parameters

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_init

Performs enabler-specific initialization tasks. It is called when the enabler starts, and its uses might include setting up shared memory regions, signal handlers and exit handlers.

You can find the template for this subroutine in:

```
usrtm_init.c
```

Syntax

```
void user_init(int *status)
```

Input Parameters

None

Output Parameters

status

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

Return Value

None.

user_on_demand_response

This routine is only applicable for points with an update criteria of either **On Demand On Change** or **On Demand On Scan**. This routine is called whenever the demand status for one of these points is changed. This lets you perform device specific actions for the specified point when its demand status changes.

You can find the template for this subroutine in:

```
usrtm_dmdres.c
```

Syntax

```
void user_on_demand_response (ADDR_DATA *address_struct,
                              int ptState,
                              int *status)
```

Input Parameters

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

ptState

Contains the new Demand State of the point.

- A value of TRUE indicates that the point is going into demand.

- A value of FALSE indicates that the point is no longer in demand.

Output Parameters

status

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

Return Value

None.

user_open_port

Opens the port for communications.

You can find the template for this subroutine in:

```
usrtm_opport.c
```

Syntax

```
void user_open_port(char *port_id,
                    int *baud_rate,
                    int *parity,
                    int *status)
```

Input Parameters

port_id

Is a pointer to a character string containing port used for communications.

The name provided is in upper-case characters and identifies the port name on the computer.

baud_rate

Is the pointer to the baud rate selected for the given **port_id** .

parity

Is the pointer to the parity selected for the given **port_id** .

Valid values for serial ports are:

- TOOLKIT_NO_PARITY
- TOOLKIT_EVEN_PARITY
- TOOLKIT_ODD_PARITY

Output Parameters

status

Indicates whether the function completed successfully. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

Return Value

None.

user_proc_event_1

Defines actions to perform when TOOLKIT_USER_EVENT_1 is set.

You can find the template for this subroutine in:

```
usrtm_evt1.c
```

Syntax

```
int user_proc_event_1()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.

TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_2

Defines actions to perform when TOOLKIT_USER_EVENT_2 is set.

You can find the template for this subroutine in:

```
usrtm_evt2.c
```

Syntax

```
int user_proc_event_2()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_3

Defines actions to perform when TOOLKIT_USER_EVENT_3 is set.

You can find the template for this subroutine in:

```
usrtm_evt3.c
```

Syntax

```
int user_proc_event_3()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_4

Defines actions to perform when TOOLKIT_USER_EVENT_4 is set.

You can find the template for this subroutine in:

```
usrtm_evt4.c
```

Syntax

```
int user_proc_event_4()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_5

Defines actions to perform when TOOLKIT_USER_EVENT_5 is set.

You can find the template for this subroutine in:

```
usrtm_evt5.c
```

Syntax

```
int user_proc_event_5()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_6

Defines actions to perform when TOOLKIT_USER_EVENT_6 is set.

You can find the template for this subroutine in:

```
usrtm_evt6.c
```

Syntax

```
int user_proc_event_6()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_7

Defines actions to perform when TOOLKIT_USER_EVENT_7 is set.

You can find the template for this subroutine in:

```
usrtm_evt7.c
```

Syntax

```
int user_proc_event_7()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_8

Defines actions to perform when TOOLKIT_USER_EVENT_8 is set.

You can find the template for this subroutine in:

```
usrtm_evt8.c
```

Syntax

```
int user_proc_event_8()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_9

Defines actions to perform when TOOLKIT_USER_EVENT_9 is set.

You can find the template for this subroutine in:

```
usrtm_evt9.c
```

Syntax

```
int user_proc_event_9()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_proc_event_10

Defines actions to perform when TOOLKIT_USER_EVENT_10 is set.

You can find the template for this subroutine in:

```
usrtm_evt10.c
```

Syntax

```
int user_proc_event_10()
```

Input Parameters

None

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

user_process_unsolicited_data

Retrieves unsolicited data from a device and returns the data.

You can find the template for this subroutine in:

```
usrtm_unso.c
```

Syntax

```
void user_process_unsolicited_data(DEVICE_DATA *device_struct,
                                   char *data,
                                   ADDR_DATA *start_address,
                                   int *sizeof_data,
                                   int *more,
                                   int *comm_status,
                                   int *status)
```

Input Parameters

None

Output Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

The **device_id** field in this structure must be set.

data

Is the pointer to a buffer containing the data received.

If the devcomm has indicated support for quality data by setting **support.unsolicited_quality_data** to `TOOLKIT_YES` and `support.extended_user_bits` to `TOOLKIT_NO`, then the following code illustrates how the buffer should be handled:

```
TOOLKIT_QUALDATA *pqual_data = (TOOLKIT_QUALDATA *)data;
pqual_data->sys_flags = 0;          //bits to be retained
pqual_data->sys_changed_mask = 0;   //indicate bits to be retained
pqual_data->user_flags = 0;         //bits to be retained
pqual_data->user_changed_mask = 0;  //indicate bits to be retained
data += sizeof (TOOLKIT_QUALDATA);
```

If the devcomm has indicated support for quality data by setting `support.unsolicited_quality_data` to `TOOLKIT_YES` and `support.extended_user_bits` to `TOOLKIT_YES`, then the following code illustrates how the buffer should be handled:

```
TOOLKIT_QUALDATA2 *pqual_data = (TOOLKIT_QUALDATA2 *)data;
pqual_data->sys_flags = 0;          //bits to be retained
pqual_data->sys_changed_mask = 0;   //indicate bits to be retained
pqual_data->user_flags = 0;         //bits to be retained
pqual_data->user_changed_mask = 0;  //indicate bits to be retained
data += sizeof (TOOLKIT_QUALDATA2);
```

The **data** pointer now points to the area containing the point values. Any quality data changes indicated by the masks and values will be applied to all points serviced by the return buffer.

TOOLKIT_QUALDATA *(page 364)* and TOOLKIT_QUALDATA2 are **typedefs** to a structure defined in < inc_path/toolkit.h > .

start_address

Is a pointer to a structure that defines domain starting addresses in the device memory. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

If standard addressing is used, the **domain_index** and **domain_offset** should be correctly set.

For custom addressing, the **address** string should be set.

sizeof_data

Contains the number of bytes of data (must be less than TOOLKIT_MAX_INTERNAL_BUFFER bytes).

more

Indicates whether there is more unsolicited data to be processed. Valid values are:

TRUE	More data needs processing
FALSE	All data has been sent

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_process_unsolicited_data_stamp

Retrieves unsolicited data from a device and returns the data and timestamp.

You can find the template for this subroutine in:

```
usrtm_unsost.c
```

Syntax

```
void user_process_unsolicited_data_stamp(DEVICE_DATA *device_struct,
                                         char *data,
                                         ADDR_DATA *start_address,
                                         int *sizeof_data,
                                         int *more,
                                         COR_STAMP *timestamp,
                                         int *comm_status,
                                         int *status)
```

Input Parameters

None

Output Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

The **device_id** field in this structure must be set.

data

Is the pointer to a buffer containing the data received.

If the devcomm has indicated support for quality data by setting **support.unsolicited_quality_data** to `TOOLKIT_YES` and `support.extended_user_bits` to `TOOLKIT_NO`, then the following code illustrates how the buffer should be handled:

```
TOOLKIT_QUALDATA *pqual_data = (TOOLKIT_QUALDATA *)data;
pqual_data->sys_flags = 0;          //bits to be retained
```

```
pqual_data->sys_changed_mask = 0;   //indicate bits to be retained
pqual_data->user_flags = 0;         //bits to be retained
pqual_data->user_changed_mask = 0;  //indicate bits to be retained
data += sizeof (TOOLKIT_QUALDATA);
```

If the devcomm has indicated support for quality data by setting `support.unsolicited_quality_data` to `TOOLKIT_YES` and `support.extended_user_bits` to `TOOLKIT_YES`, then the following code illustrates how the buffer should be handled:

TOOLKIT_QUALDATA2 *pqual_data = (TOOLKIT_QUALDATA2 *)data;

pqual_data->sys_flags = 0; //bits to be retained

pqual_data->sys_changed_mask = 0; //indicate bits to be retained

pqual_data->user_flags = 0; //bits to be retained

pqual_data->user_changed_mask = 0; //indicate bits to be retained

data += sizeof (TOOLKIT_QUALDATA2);

The **data** pointer now points to the area containing the point values. Any quality data changes indicated by the masks and values will be applied to all points serviced by the return buffer.

TOOLKIT_QUALDATA *(page 364)* and TOOLKIT_QUALDATA2 are `typedefs` to a structure defined in < inc_path/toolkit.h > .

start_address

Is a pointer to a structure that defines domain starting addresses in the device memory. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

If standard addressing is used, the **domain_index** and **domain_offset** should be correctly set.

For custom addressing, the **address** string should be set.

sizeof_data

Contains the number of bytes of data (must be less than TOOLKIT_MAX_INTERNAL_BUFFER bytes).

more

Indicates whether there is more unsolicited data to be processed. Valid values are:

TRUE	More data needs processing
FALSE	All data has been sent

timestamp

Is a pointer to a structure that defines the timestamp to be used to record the time at which the data is reported. COR_STAMP is a **typedef** to a structure defined in < inc_path/cor.h > .

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function successfully obtained all of the requested information. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

Programming Note

The following is an example of how to use the **user_process_unsolicited_data_stamp** subroutine:

```
void user_process_unsolicited_data_stamp (DEVICE_DATA *device_struct,
                                          char *data,
                                          ADDR_DATA *start_address,
                                          int *sizeof_data,
                                          int *more,
                                          COR_STAMP *timestamp,
                                          int *comm_status,
                                          int *status)
{
int I;
strcpy (device_struct->device_id, "TOOLKIT_DEVICE");
reg_plc_data[3]++
start_address->domain_index = 0;
start_address->domain_offset = 3;
*sizeof_data=2;
memcpy (data, &reg_plc_data[3], *sizeof_data);
*more = FALSE;
*comm_status = TOOLKIT_SUCCESS;
*status = TOOLKIT_SUCCESS
/* set the timestamp to September 12, 1995 at 16:12:03:00 */
timestamp->yyyymmdd = 19950912;
```

```
timestamp->hhmmsstt = 16120300;
return;
}
```

user_protocol_info

Defines the Device Communications Toolkit features that are supported in this customized communication enabler.

You can find the template for this subroutine in:

```
usrtm_protin.c
```

Syntax

```
void user_protocol_info(int *max_buffer_size,
                        SUPPORT *supported)
```

Input Parameters

None.

Output Parameters

max_buffer_size

Defines the amount of data (in bytes) that can be transferred between the device and the enabler in one operation.

user_read_data(), user_write_data() and **user_process_unsolicited()** cannot process more than the **max_buffer_size** bytes of data.

max_buffer_size may not exceed the size of TOOLKIT_MAX_INTERNAL_BUFFER.

supported

Is a pointer to a structure defining the supported options for the device. SUPPORT is a **typedef** to a structure defined in < inc_path/toolkit.h > .

Return Value

None.

user_read_data

Reads data from specific memory locations from the device's memory.

You can find the template for this subroutine in:

```
usrtm_reada.c
```

Syntax

```
void user_read_data(DEVICE_DATA *device_struct,
                    ADDR_DATA *address_struct,
                    int length,
                    char *data,
                    int *comm_status,
                    int *status);
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

length

Is the number of bytes of data requested.

Output Parameters

data

Is a pointer to an array that contains the data that was read from the device.

For data from bit (TOOLKIT_BIT) domains, bit data should be packed into bytes so that the leftmost bit is the most significant.

For domains whose element size is greater than one byte, the bytes should be ordered in the same way as they are ordered by the underlying operating system on which the enabler is to run.

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_WRITE_FAILED	Could not send command or data to device.
TOOLKIT_TIMEOUT	Sent command or data to device, timed-out waiting for a response.
TOOLKIT_BAD_DATA	Received response from device but it contained invalid data such as an incorrect checksum.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function read all the data. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_read_diag_data

Returns diagnostic data from specific locations within the enabler.

You can find the template for this subroutine in:

```
usrtm_readdiag.c
```

Syntax

```
void user_read_diag_data(DEVICE_DATA *device_struct,
                         ADDR_DATA *address_struct,
                         int length,
                         char *data,
                         int *comm_status,
                         int *status);
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

length

Is the number of bytes of data requested.

Output Parameters

data

Is a pointer to an array that contains the data that was read from the enabler.

- For diagnostic bits, data should be packed into bytes so that the leftmost bit is the most significant.
- For domains whose element size is greater than one byte, the bytes should be ordered in the same way as they are ordered by the underlying operating system on which the enabler is to run.

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function read all the data. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_remove_point

Performs enabler-specific processing when a previously valid point is removed from the configuration.

You can find the template for this subroutine in `usrtm_rm_pt.c`

`usrtm_rm_pt.c`

Syntax

```
void user_remove_point(DEVICE_DATA *status,
                       ADDR_DATA *address_struct)
```

Input Parameters

`device_struct`

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a `typedef` to a structure defined in <inc_path/toolkit.h>.

`address_struct`

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a `typedef` to a structure defined in <inc_path/toolkit.h>.

Output Parameters

None.

Return Value

None.

user_term

Performs enabler-specific termination tasks. This function is called following a request from CIMPLICITY software for the enabler to terminate. Its uses include cleaning up shared memory regions and disassociating from the device.

You can find the template for this subroutine in:

```
usrtm_term.c
```

Syntax

```
void user_term(int *status)
```

Input Parameters

None

Output Parameters

status

Indicates whether the function successfully completed. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully.

Return Value

None.

user_valid_diag_point

Determines the validity of a diagnostic point.

You can find the template for this subroutine in:

```
usrtm_valdiagpt.c
```

Syntax

```
void user_valid_diag_point(DEVICE_DATA*device_struct,
                           ADDR_DATA*address_struct,
                           int *valid_pt,
                           int *comm_status,
                           int *status)
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

Output Parameters

valid_pt

Defines whether the point is valid. Valid values are:

TOOLKIT_SUCCESS	The point is valid
TOOLKIT_FAILURE	The point is not valid

valid_pt is not a Boolean value.

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function read all the data. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_valid_point

Defines whether the point is valid. Where custom addressing is used, both the **domain_index** and **offset** for the point must be determined.

You can find the template for this subroutine in:

```
usrtm_valpt.c
```

Syntax

```
void user_valid_point(DEVICE_DATA*device_struct,
                      ADDR_DATA *address_struct,
                      int *valid_pt,
                      int *comm_status,
                      int *status)
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

Output Parameters

valid_pt

Defines whether the point is valid. Valid values are:

TOOLKIT_SUCCESS	The point is valid
TOOLKIT_FAILURE	The point is not valid

valid_pt is not a Boolean value.

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function read all the data. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_write_data

Writes data to specific memory locations within the device's memory.

You can find the template for this subroutine in:

```
usrtm_wrda.c
```

Syntax

```
void user_write_data(DEVICE_DATA *device_struct,
                     ADDR_DATA *address_struct,
                     int length,
                     char *data,
                     int *download_req,
                     int download_comp,
                     int *comm_status,
                     int *status)
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

length

Is the number of bytes of data to be written.

data

Is a pointer to the actual data to be written.

For data from bit (TOOLKIT_BIT) domains, bit data is packed into bytes so that the leftmost bit is the most significant.

Bit data is always accessed in multiples of 8 bits, so write requests near the end of a bit domain must ignore extra bits that would be beyond the end of the domain.

For domains whose element size is greater than one byte, the bytes are ordered in the same way as they are ordered by the underlying operating system where the enabler is running.

download_req

Reserved for future use.

download_comp

Reserved for future use.

Output Parameters

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_WRITE_FAILED	Could not send command or data to device.
TOOLKIT_TIMEOUT	Sent command or data to device, timed-out waiting for a response.
TOOLKIT_BAD_DATA	Received response from device but it contained invalid data such as an incorrect checksum.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function read all the data. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_REPLY_LATER	Write request queued for future processing.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

user_write_point_quality

Writes point quality data to the specified device.

You can find the template for this subroutine in:

```
usrtm_wrtpqual.c
```

Syntax

```
void user_write_point_quality(DEVICE_DATA *device_struct,
```

```
                                ADDR_DATA *address_struct,
                                TOOLKIT_QUALDATA *pqualdata,
                               int *comm_status,
                               int *status)
```

Input Parameters

device_struct

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

address_struct

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

pqualdata

Is the pointer to the quality data to be written. TOOLKIT_QUALDATA *(page 364)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

Output Parameters

comm_status

Indicates whether a status of TOOLKIT_FAILURE occurred as a result of a communication failure. Valid values are:

TOOLKIT_SUCCESS	Failure is not due to communications failure.
TOOLKIT_WRITE_FAILED	Could not send command or data to device.
TOOLKIT_TIMEOUT	Sent command or data to device, timed-out waiting for a response.
TOOLKIT_FAILURE	Failure is due to communications failure.

status

Indicates whether the function read all the data. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_REPLY_LATER	Write request queued for future processing.
TOOLKIT_FAILURE	Function did not complete successfully. Check **comm_status** to see if the failure was the result of a communication failure.

Return Value

None.

Device Communications Toolkit Other Subroutines

Device Communications Toolkit Other Subroutines

	cor_sleep
	dcrp_align_read
	dcrp_call_on_time
	dcrp_clear_ef
	dcrp_get_any_ef
	dcrp_get_ef
	dcrp_get_port_parameters
	dcrp_get_serial_settings
	dcrp_log_status
	dcrp_notify_unsolicited_data
	dcrp_rcv_unsolicited_data
	dcrp_rcv_unsolicited_data_stamp
	dcrp_release_ef
	dcrp_set_device_down
	dcrp_set_device_up
	dcrp_set_ef
	dcrp_stat_process
	dcrp_to_long_point_id
	dcrp_user_alarm

cor_sleep

Waits (at least) the given number of seconds.

Syntax

```
int cor_sleep(int time_to_sleep)
```

Input Parameters

time_to_sleep

Contains the time to wait in **seconds**.

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

dcrp_align_read

Selects the method the toolkit uses when reading or writing data in digital domains. Call this subroutine once in **user_init()** .

Syntax

```
int dcrp_align_read (int flag)
```

Input Parameters

flag

Indicates how bits in a digital domain are to be read or written. Set the parameter to one of the following:

TRUE	Calculate the byte the requested bit belongs in and read that byte.
FALSE	Read one byte starting at the point's address. This may cause the user read function to read data beyond the valid address if the bit requested is at the end of the domain.

dcrp_call_on_time

Asynchronously calls the given function after at least the specified time period has passed.

Syntax

```
int dcrp_call_on_time (int time,
                       int time_unit,
                       char *function,
                       int timer_id)
```

Input Parameters

`time`

Contains the amount of time to wait before calling given function.

`time_unit`

Defines the units for the amount of time. Valid values are:

- TOOLKIT_TIME_TICKS
- TOOLKIT_TIME_SECONDS
- TOOLKIT_TIME_MINUTES
- TOOLKIT_TIME_HOURS

`function`

Is the pointer to function to be called when specified time has passed.

`timer_id`

Is a unique identifier for timer being set to track previously defined time interval. Value should be between 1 and 200.

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

On **TOOLKIT_FAILURE** , the function specified in the argument list will not be called.

dcrp_clear_ef

Clears the specified USER event flag.

Syntax

```
int dcrp_clear_ef(int *ef)
```

Input Parameters

```
ef
```

Is a pointer to the event flag to be cleared. Valid values for the event flag are:

TOOLKIT_USER_EVENT_1

TOOLKIT_USER_EVENT_2

TOOLKIT_USER_EVENT_3

TOOLKIT_USER_EVENT_4

TOOLKIT_USER_EVENT_5

TOOLKIT_USER_EVENT_6

TOOLKIT_USER_EVENT_7

TOOLKIT_USER_EVENT_8

TOOLKIT_USER_EVENT_9

TOOLKIT_USER_EVENT_10

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.

TOOLKIT_FAILURE	Function not completed successfully.

dcrp_get_any_ef

Reserves the next available USER event flag for Enabler use.

Syntax

```
int dcrp_get_any_ef(int *ef)
```

Input Parameters

None

Output Parameters

```
ef
```

Pointer to the reserved event flag. Valid values are:

TOOLKIT_USER_EVENT_1

TOOLKIT_USER_EVENT_2

TOOLKIT_USER_EVENT_3

TOOLKIT_USER_EVENT_4

TOOLKIT_USER_EVENT_5

TOOLKIT_USER_EVENT_6

TOOLKIT_USER_EVENT_7

TOOLKIT_USER_EVENT_8

TOOLKIT_USER_EVENT_9

TOOLKIT_USER_EVENT_10

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.

TOOLKIT_FAILURE	Function not completed successfully.

dcrp_get_ef

Reserves the specified USER event flag for Enabler use.

Syntax

```
int dcrp_get_ef(int *ef)
```

Input Parameters

```
ef
```

Is a pointer to the event flag to be reserved. Valid values for the event flag are:

- TOOLKIT_USER_EVENT_1
- TOOLKIT_USER_EVENT_2
- TOOLKIT_USER_EVENT_3
- TOOLKIT_USER_EVENT_4
- TOOLKIT_USER_EVENT_5
- TOOLKIT_USER_EVENT_6
- TOOLKIT_USER_EVENT_7
- TOOLKIT_USER_EVENT_8
- TOOLKIT_USER_EVENT_9
- TOOLKIT_USER_EVENT_10

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

dcrp_get_port_parameters

Gets the parameter string for non-serial ports. The **user_open_port()** routine is passed only baud rate and parity which may not apply to non-serial ports. Use this routine to access the parameter string for the port.

Syntax

```
void dcrp_get_port_parameters(char *port_id, char *parameters)
```

Input Parameters

```
port_id
```

Is the identifier of the port whose settings you want to get.

Output Parameters

```
parameters
```

Is the parameter string for the port.

dcrp_get_serial_settings

Gets the serial port parameters. The **user_open_port()** routine is passed only baud rate and parity. Use this routine to access data bits, stop bits, and flow control settings.

Syntax

```
void dcrp_get_serial_settings(char *port_id,
                              COR_I2 data_bits,
                              COR_I2 stop_bits,
                              COR_BOOLEAN *rts_cts_control,
                              COR_BOOLEAN *xon_xoff_control)
```

Input Parameters

```
            port_id
```

Is the identifier of the port whose settings you want to get.

Output Parameters

```
            data_bits
```

```
```

Is the number of data bits supported by the protocol. Valid value is 6, 7, or 8.

```
stop_bits
```

Is the number of stop bits supported by the protocol value. Valid value is 1 or 2.

```
rts_cts_control
```

Indicates whether the RTS (Ready To Send) and CTS (Clear To Send) lines are used for hardware flow control. This field contains one of the following:

TRUE	Raise RTS and check CTS before transmitting data. This usually suggests that your serial port Data Control Block (DCB) setup includes the following:

```
dcb.fOutxCtsFlow = TRUE;
```

```
dcb.fRtsControl = RTS_CONTROL_TOGGLE;
```

FALSE	The protocol does not use hardware flow control.

`xon_xoff_control`

Indicates whether the XON/XOFF software flow control is to be used. This field contains one of the following:

TRUE	XON/XOFF flow control is used. This usually suggests that your serial port Data Control Block (DCB) setup includes the following:

```
dcb.fOutX = TRUE;
```

`dcb.fInX = TRUE;`

FALSE	The protocol does not use software flow control.

dcrp_log_status

Logs a message to your CIMPLICITY project's Status Log.

Syntax

```
void dcrp_log_status(int status,
                     char *module,
                     int reference,
                     int log,
                     char *message)
```

Input Parameters

`status`

Contains the status of the action to be logged. Valid values are:

- TOOLKIT_SUCCESS
- TOOLKIT_WARNING
- TOOLKIT_FAILURE

`module`

Contains the name of module logging the information (only first 16 characters are printed).

`reference`

Is the internal reference number/value for the logged information.

`log`

Defines the disposition of the message. Valid values are:

TRUE	Causes the information to be logged to the Status Log and the **.err** file for the process.
FALSE	Causes the information to be logged to the Status Log.
FATAL	Indicates a fatal error. The information is logged to the Status Log and the **.err** file for the process, then the process exits.

`message`

Contains the message to be logged. The message should be no longer than 79 characters and should not contain special characters such as control characters or new lines.

Output Parameters

None

Return Value

None

dcrp_notify_unsolicited_data

Notifies the Toolkit that unsolicited data has been received (must be called before unsolicited data between polls can be processed).

Syntax

```
void dcrp_notify_unsolicited_data(int *status)
```

Input Parameters

None

Output Parameters

`status`

Contains the function completion status. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed.

Return Value

None

dcrp_rcv_unsolicited_data

Processes the received unsolicited data.

Syntax

```
int dcrp_rcv_unsolicited_data(char *device_id,
                              ADDR_DATA *addr_data,
                              int sizeof_data,
                              char *data_buffer,
                              int *status)
```

Input Parameters

`device_id`

Contains the Device ID for the device originating the unsolicited data.

`addr_data`

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

`sizeof_data`

Is the number of bytes of data being received.

`data_buffer`

Is the pointer to the buffer containing the unsolicited data.

- For data from bit (TOOLKIT_BIT) domains, bit data should be packed into bytes so that the left most bit is the most significant.
- For domains whose element size is greater than one byte, the bytes should be ordered in the same way as ordered by the underlying operating system on which the enabler is to run.

The caller is responsible for managing the data buffer (this might include allocating and de-allocating the space or using a static data structure to maintain the data).

The data buffer should not exceed `TOOLKIT_MAX_INTERNAL_BUFFER` bytes.

Output Parameters

`status`

Contains the function completion status. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed.

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

dcrp_rcv_unsolicited_data_stamp

Processes the received unsolicited data.

Syntax

```
int dcrp_rcv_unsolicited_data_stamp(device_id, addr_data,
        sizeof_data, data_buffer, pStamp, status)
char *device_id;
ADDR_DATA *addr_data;
int sizeof_data
char *data_buffer;
COR_STAMP *pStamp
int *status;
```

Input Parameters

```
device_id
```

Is the Device ID for the device originating the unsolicited data.

```
addr_data
```

Is a pointer to the address from which the data was read. ADDR_DATA *(page 353)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

sizeof_data

Is the number of bytes of data being received.

data_buffer

Is the pointer to the buffer containing the unsolicited data.

- For data from bit (TOOLKIT_BIT) domains, bit data should be packed into bytes so that the leftmost bit is the most significant.
- For domains whose element size is greater than one byte, the bytes should be ordered in the same way as ordered by the underlying operating system on which the enabler is to run.

The caller is responsible for managing the data buffer (this might include allocating and deallocating the space or using a static data structure to maintain the data).

The data buffer should not exceed TOOLKIT_MAX_INTERNAL_BUFFER bytes.

```
pStamp
```

Is a pointer to the structure defining the timestamp to be used to record the time at which the data is reported. COR_STAMP is a **typedef** to a structure defined in **<inc_path/cor.h>** .

Output Parameters

```
status
```

Is the function completion status. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed.

```
Return Value
```

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

dcrp_release_ef

Unreserves the specified USER event flag.

Syntax

```
int dcrp_release_ef(int *ef)
```

Input Parameters

```
ef
```

Pointer to the event flag to be released. Valid values are:

TOOLKIT_USER_EVENT_1

TOOLKIT_USER_EVENT_2

TOOLKIT_USER_EVENT_3

TOOLKIT_USER_EVENT_4

TOOLKIT_USER_EVENT_5

TOOLKIT_USER_EVENT_6

TOOLKIT_USER_EVENT_7

TOOLKIT_USER_EVENT_8

TOOLKIT_USER_EVENT_9

TOOLKIT_USER_EVENT_10

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

dcrp_set_device_down

Sets device to a "DEVICE DOWN" status.

Syntax

```
int dcrp_set_device_down(char *device_id,
                         int  *status)
```

Input Parameters

`device_id`

Contains the Device ID for the device whose status is to be set DOWN (UNAVAILABLE).

Output Parameters

`status`

Is the function completion status. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed.

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

Important: `DCRP_SET_DEVICE_DOWN` cannot be called until `USER_READ_DATA (page 320)` or `USER_READ_DIAG_DATA (page 322)` is called first.

dcrp_set_device_up

Sets device to a "DEVICE UP" status.

Syntax

```
int dcrp_set_device_up(char *device_id,
                       int *status)
```

Input Parameters

`device_id`

Contains the Device ID for the device whose status is to be set UP (AVAILABLE).

Output Parameters

`status`

Is the function completion status. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed.

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

dcrp_set_ef

Sets the specified USER event flag.

Syntax

```
int dcrp_set_ef(int *ef)
```

Input Parameters

```
ef
```

Pointer to the event flag to be set. Valid values are:

TOOLKIT_USER_EVENT_1

TOOLKIT_USER_EVENT_2

TOOLKIT_USER_EVENT_3

TOOLKIT_USER_EVENT_4

TOOLKIT_USER_EVENT_5

TOOLKIT_USER_EVENT_6

TOOLKIT_USER_EVENT_7

TOOLKIT_USER_EVENT_8

TOOLKIT_USER_EVENT_9

TOOLKIT_USER_EVENT_10

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed successfully.

dcrp_stat_process

dcrp_stat_process

Updates device communication statistics kept inside the enabler based on communication status and completion status and returns the normalized communication status.

The common code calls **dcrp_stat_process()** after each call to **user_read_data()** and **user_write_data()** . If those routines make multiple requests of the device, or if other routines, such as **user_valid_point()** or **user_device_info()** access the device, they should call **dcrp_stat_process()** to process the status values and update the device communication status.

Syntax

```
TOOLKIT_STATUS dcrp_stat_process (DEVICE_DATA *device,
                                  TOOLKIT_STATUS comm_status,
                                  TOOLKIT_STATUS status)
```

Input Parameters

`device`

Is a pointer to the structure defining device data. DEVICE_DATA *(page 355)* is a **typedef** to a structure defined in < inc_path/toolkit.h > .

`comm_status`

Is the communication status where TOOLKIT_STATUS *(page 365)* is a **typedef** to an ENUM defined in <inc_path/toolkit.h > .

Valid values are:

- TOOLKIT_WRITE_FAILED
- TOOLKIT_TIMEOUT
- TOOLKIT_BAD_DATA
- TOOLKIT_SUCCESS
- TOOLKIT_FAILURE

`status`

Is the function completion status. Valid values are:

- TOOLKIT_SUCCESS
- TOOLKIT_FAILURE

- TOOLKIT_REPLY_LATER

Output Parameters

None

Return Value

This subroutine returns one of the following:

TOOLKIT_SUCCESS	Communication completed successfully.
TOOLKIT_FAILURE	Communication not completed successfully.

dcrp_stat_process Algorithm

The algorithm used to update the device statistics is:

```
SWITCH (comm_status) {
   CASE TOOLKIT_WRITE_FAILED:
   CASE TOOLKIT_READ_TIMEOUT:
      Increment device transmissions
      break;
   CASE TOOLKIT_BAD_DATA:
      Increment device transmissions
      Increment device responses
      break;
   CASE_TOOLKIT_SUCCESS:
      Increment device transmissions
      IF status != TOOLKIT_REPLY_LATER
         Increment device responses
      ENDIF
      break;
   CASE TOOLKIT_UNSOLICITIED:
      // Don't increment transmissions
      Increment device responses
      comm_status = TOOLKIT_SUCCESS;
      break;
   CASE TOOLKIT_FAILURE:
      Increment device transmissions
      break:
ENDSWITCH
IF comm_status != TOOLKIT_SUCCESS
   IF last device status == TOOLKIT_SUCCESS
      Increment device failures
   ELSE IF comm_status != TOOLKIT_UNSOLICITED
      Increment device retries
   ENDIF
ELSE
   IF last device status != TOOLKIT_SUCCESS
   AND comm_status != TOOLKIT_UNSOLICITED
```

```
      Increment device retries
   ENDIF
ENDIR
last device status = comm_status
SWITCH (comm_status) {
   CASE TOOLKIT_WRITE_FAILED:
   CASE TOOLKIT_READ_TIMEOUT:
   CASE TOOLKIT_BAD_DATA:
      comm_status = TOOKIT_FAILURE;
      break;
   CASE TOOLKIT_UNSOLICITED:
      comm_status = TOOLKIT_SUCCESS;
      break;
ENDSWITCH
return comm_status;
```

dcrp_to_long_point_id

Converts from an internal representation of a point to its display name.

Syntax

```
TCHAR *dcrp_to_long_point_id (TCHAR *short_name)
```

Input Parameters

`short_name`

Internal representation of the point id.

Output Parameters

None.

Return Parameters

Point to the display name representation. If there is no match found, a pointer to a NULL string is returned.

dcrp_user_alarm

Generates or clears an alarm.

Syntax

```
void dcrp_user_alarm (char *alarm_msg,
```

```
                        char *gen_or_clr,
                        char *device_id,
                        int *status)
```

Input Parameters

`alarm_msg`

Is the alarm message to be displayed.

`gen_or_clear`

Defines whether the alarm is to generated or cleared. Valid values are:

- TOOLKIT_GENERATE_ALARM
- TOOLKIT_CLEAR_ALARM

`device_id`

Identifies the CIMPLICITY software device for which the alarm will be generated or cleared.

Output Parameters

`status`

Is the function completion status. Valid values are:

TOOLKIT_SUCCESS	Function completed successfully.
TOOLKIT_FAILURE	Function not completed.

Device Communications Toolkit File List

Device Communications Toolkit File List

This appendix lists the files that comprise the Device Communication Driver Toolkit. The installation may be verified by checking to see that all files were provided.

- Simulated user device routines.
- Toolkit object libraries.
- Toolkit include files.
- Command files used for building toolkit executables.
- Template source files for user device-specific protocol development.

- Configuration data files.

Simulated User Device Routines

The subroutines described in this manual are found in the following files:

```
%BSM_ROOT%\api\dc_api\user_accept.c
%BSM_ROOT%\api\dc_api\user_cpu_mdl.c
%BSM_ROOT%\api\dc_api\user_cvtfrm.c
%BSM_ROOT%\api\dc_api\user_cvtto.c
%BSM_ROOT%\api\dc_api\user_dev_ok.c
%BSM_ROOT%\api\dc_api\user_dmdres.c
%BSM_ROOT%\api\dc_api\user_dvin.c
%BSM_ROOT%\api\dc_api\user_evt1.c
%BSM_ROOT%\api\dc_api\user_evt10.c
%BSM_ROOT%\api\dc_api\user_evt2.c
%BSM_ROOT%\api\dc_api\user_evt3.c
%BSM_ROOT%\api\dc_api\user_evt4.c
%BSM_ROOT%\api\dc_api\user_evt5.c
%BSM_ROOT%\api\dc_api\user_evt6.c
%BSM_ROOT%\api\dc_api\user_evt7.c
%BSM_ROOT%\api\dc_api\user_evt8.c
%BSM_ROOT%\api\dc_api\user_evt9.c
%BSM_ROOT%\api\dc_api\user_hrtbt.c
%BSM_ROOT%\api\dc_api\user_init.c
%BSM_ROOT%\api\dc_api\user_maxdom.c
%BSM_ROOT%\api\dc_api\user_protin.c
%BSM_ROOT%\api\dc_api\user_read_da.c
%BSM_ROOT%\api\dc_api\user_read_diag.c
%BSM_ROOT%\api\dc_api\user_term.c
%BSM_ROOT%\api\dc_api\user_unso.c
%BSM_ROOT%\api\dc_api\user_unsost.c
%BSM_ROOT%\api\dc_api\user_valdiagpt.c
%BSM_ROOT%\api\dc_api\user_valpt.c
%BSM_ROOT%\api\dc_api\user_wrtdata.c
%BSM_ROOT%\api\dc_api\user_wrtpqual.c
%BSM_ROOT%\api\dc_api\usropen_port.c
```

Toolkit Object Libraries

The object libraries needed for the Device Communications Toolkit API are:

```
%BSM_ROOT%\api\lib\amaru.lib
%BSM_ROOT%\api\lib\corutil.lib
%BSM_ROOT%\api\lib\dc_common.lib
%BSM_ROOT%\api\lib\dd_common.lib
%BSM_ROOT%\api\lib\ddl.lib
%BSM_ROOT%\api\lib\fasrtl.lib
%BSM_ROOT%\api\lib\ipc.lib
```

```
%BSM_ROOT%\api\lib\cim_mf.lib
%BSM_ROOT%\api\lib\pm.lib
%BSM_ROOT%\api\lib\cim_sc.lib
%BSM_ROOT%\api\lib\toolkit.lib
%BSM_ROOT%\api\lib\tools.lib
```

Toolkit Include Files

The include files described in this document are found in the following locations:

```
%BSM_ROOT%\api\dc_api\inc_path\globals.h
%BSM_ROOT%\api\include\inc_path\toolkit.h
```

Command Files Used for Building Toolkit Executables

The command file described in this document is found in the following location:

```
%BSM_ROOT%\api\dc_api\makefile
tlkittst_dll.vcxproj
tlkitusr_dll.vcxproj (simulated exe)
```

Template Source Files for User Device-Specific Protocol Development

The template source files discussed in this document are found in the following locations:

```
%BSM_ROOT%\api\dc_api\usrtm_accept.c
%BSM_ROOT%\api\dc_api\usrtm_cpumdl.c
%BSM_ROOT%\api\dc_api\usrtm_cvtfrm.c
%BSM_ROOT%\api\dc_api\usrtm_cvtto.c
%BSM_ROOT%\api\dc_api\usrtm_dev_ok.c
%BSM_ROOT%\api\dc_api\usrtm_dmdres.c
%BSM_ROOT%\api\dc_api\usrtm_dvin.c
%BSM_ROOT%\api\dc_api\usrtm_evt1.c
%BSM_ROOT%\api\dc_api\usrtm_evt10.c
%BSM_ROOT%\api\dc_api\usrtm_evt2.c
%BSM_ROOT%\api\dc_api\usrtm_evt3.c
%BSM_ROOT%\api\dc_api\usrtm_evt4.c
%BSM_ROOT%\api\dc_api\usrtm_evt5.c
%BSM_ROOT%\api\dc_api\usrtm_evt6.c
%BSM_ROOT%\api\dc_api\usrtm_evt7.c
%BSM_ROOT%\api\dc_api\usrtm_evt8.c
%BSM_ROOT%\api\dc_api\usrtm_evt9.c
%BSM_ROOT%\api\dc_api\usrtm_hrtbt.c
%BSM_ROOT%\api\dc_api\usrtm_init.c
%BSM_ROOT%\api\dc_api\usrtm_maxdom.c
%BSM_ROOT%\api\dc_api\usrtm_opport.c
```

```
%BSM_ROOT%\api\dc_api\usrtm_protin.c
%BSM_ROOT%\api\dc_api\usrtm_readda.c
%BSM_ROOT%\api\dc_api\usrtm_rm_pt.c
%BSM_ROOT%\api\dc_api\usrtm_term.c
%BSM_ROOT%\api\dc_api\usrtm_unso.c
%BSM_ROOT%\api\dc_api\usrtm_unsost.c
%BSM_ROOT%\api\dc_api\usrtm_valpt.c
%BSM_ROOT%\api\dc_api\usrtm_wrda.c
%BSM_ROOT%\api\dc_api\usrtm_wrtpqual.c
```

Configuration Data Files

The template configuration files discussed in this document are found in the following locations:

```
%BSM_ROOT%\api\dc_api\domain.cfg
%BSM_ROOT%\api\dc_api\TLKITTST.MODEL
%BSM_ROOT%\api\dc_api\TLKITTST.PROTO
%BSM_ROOT%\api\dc_api\TLKITUSR.MODEL
%BSM_ROOT%\api\dc_api\TLKITUSR.PROTO
```

Device Communications Toolkit Structures

Device Communications Toolkit Structures

Device Communications Toolkit Structures

- ADDR_DATA
- COR_STAMP
- DEVICE_DATA
- DOMAIN_ARRAY
- SUPPORT
- TOOLKIT_QUALDATA
- TOOLKIT_QUALDATA2
- TOOLKIT_STATUS

ADDR_DATA

The ADDR_DATA structure is defined in **<inc_path/toolkit.h>** as:

```
typedef struct addr_data
{
   char  address[TOOLKIT_ADDR_LENGTH + 1];
   int   type;
   int   domain_index;
   int   domain_offset;
```

```
   int   addr_offset;                          /* RESERVED FOR GE
 INTELLIGENT PLATFORMS USE */
   int   point_type;
   int   elements;
   int   point_address_type;                   /* RESERVED FOR GE
 INTELLIGENT PLATFORMS USE */
   char  point_id[TOOLKIT_LONG_ID_LEN + 1 ];   /* RESERVED FOR GE
 INTELLIGENT PLATFORMS USE */
   char  scan_type;                            /* RESERVED FOR GE
 INTELLIGENT PLATFORMS USE */
   char  scan_rate;                            /* RESERVED FOR GE
 INTELLIGENT PLATFORMS USE */
   char  fullAddress[ADDR_LEN + 1];
   char  short_point_id[TOOLKIT_SHORT_POINT_ID_LEN + 1];
} ADDR_DATA;
```

Where:

address is the ASCII representation of the address.

type is the type of addressing used. Valid values are:

TOOLKIT_STD_ADDR (standard addressing)

TOOLKIT_USER_DEFINED_ADDR (custom addressing)

domain_index is the domain index of the start address.

For diagnostic data, valid values for **domain_index** are:

100	Protocol diagnostic bits
101	Protocol diagnostic bytes
102	Protocol diagnostic words
104	Protocol diagnostic double words
108	Protocol diagnostic 8-bytes

domain_offset is the domain offset of the start address.

addr_offset is reserved for GE Intelligent Platforms use.

point_type is the point data type. The standard CIMPLICITY point types supported by the Device Communications Toolkit are:

```
TOOLKIT_BOOLEAN           0       /* POINT TYPE IS DIGITAL         */
TOOLKIT_BITSTRING         1       /* POINT TYPE IS BITSTRING       */
TOOLKIT_OCTETSTRING       2       /* POINT TYPE OCTETSTRING        */
TOOLKIT_TEXTPOINT         3       /* POINT TYPE TEXTSTRING         */
TOOLKIT_UNSIGNED_ANALOG8  4       /* POINT TYPE UNSIGNED ANALOG 8  */
TOOLKIT_UNSIGNED_ANALOG16 5       /* POINT TYPE UNSIGNED ANALOG 16 */
```

```
TOOLKIT_UNSIGNED_ANALOG32 6       /* POINT TYPE UNSIGNED ANALOG 32 */
TOOLKIT_ANALOG8           7       /* POINT TYPE SIGNED   ANALOG 8  */
TOOLKIT_ANALOG16          8       /* POINT TYPE SIGNED   ANALOG 16 */
TOOLKIT_ANALOG32          9       /* POINT TYPE SIGNED   ANALOG 32 */
TOOLKIT_FLOATINGPOINT     10      /* POINT TYPE FLOATING POINT     */
TOOLKIT_UNSIGNED_ANALOG64 13      /* POINT TYPE UNSIGNED ANALOG 64 */
TOOLKIT_ANALOG64          14      /* POINT TYPE ANALOG 64 */

short_point_id – is reserved for GE Intelligent Platforms use.
```

elements is the number of point elements.

point_address_type is reserved for GE Intelligent Platforms use.

point_id is reserved for GE Intelligent Platforms use.

scan_type is reserved for GE Intelligent Platforms use.

scan_rate is reserved for GE Intelligent Platforms use.

full_address is the ASCII representation of the address to be used if the device supports addresses up to 256 characters, and **support.use_long_addresses** is set to TOOLKIT_YES.

COR_STAMP

The COR_STAMP structure is defined in < inc_path/cor.h > as:

```
typedef struct cor_time_stamp {
 COR_U4 dummy1;
 COR_U4 dummy2;
} COR_STAMP;
```

Where:

yyyymmdd is an integer that represents the date.

hhmmsstt is an integer that represents the time.

Note: Use `cor_stamp_get_componentsHR` *(page 51)* to decompose the time into its corresponding components .

DEVICE_DATA

The DEVICE_DATA structure is defined in < inc_path/toolkit.h > as:

```
typedef struct device_data
{
```

```
     int   model;
     int   cpu_id;
     char  device_id[TOOLKIT_DEVICE_LENGTH + 1];
     char  network_addr[TOOLKIT_NETWORK_ADDR_LEN + 1];
     char  primary_network_addr[TOOLKIT_NETWORK_ADDR_LEN + 1];
     char  secondary_network_addr[TOOLKIT_NETWORK_ADDR_LEN + 1];
     int   in_use ;
} DEVICE_DATA;
```

Where

model is the configured model number of the device. This number is defined in the < product >**.** **model** file.

cpu_id is the configured CPU ID of the device. his information is defined when you configure the device in your CIMPLICITY project.

device_id is the device identifier. This information is defined when you configure the device in your CIMPLICITY project.

network_addr is the network address of the device. This information is defined when you configure the device in your CIMPLICITY project.

primary_network_addr is reserved for GE Intelligent Platforms use.

secondary_network_addr is reserved for GE Intelligent Platforms use.

in_use is reserved for GE Intelligent Platforms use.

DOMAIN_ARRAY

The DOMAIN_ARRAY structure is defined in < inc_path/toolkit.h > as:

```
typedef struct domain_array
{
     int   domain_index;
     char  domain_name[TOOLKIT_DOMAIN_NAM_LEN + 1];
     int   start_addr;
     int   domain_size;
     int   addr_type;
     char  caching;
} DOMAIN_ARRAY;
```

Where:

domain_index is the internal reference used to access the information about the given domain. If you use standard addressing, this field must match the domain index assigned to the domain name in **domain.cfg** .

For diagnostic data, valid values for **domain_index** are:

100	Protocol diagnostic bits
101	Protocol diagnostic bytes
102	Protocol diagnostic words
104	Protocol diagnostic double words
108	Protocol diagnostic 8-bytes

domain_name is the ASCII name used to reference the domain. If you use standard addressing, this name must match a domain name in **domain.cfg** .

start_addr is the numeric value corresponding to the first memory location within the domain.

domain_size is the number of bytes in the domain.

addr_type is the type of addressing used in the domain. Valid values are:

- TOOLKIT_BIT
- TOOLKIT_BYTE
- TOOLKIT_WORD
- TOOLKIT_4BYTE
- TOOLKIT_8BYTE

caching is reserved for GE Intelligent Platforms use.

SUPPORT

The SUPPORT structure is defined in < inc_path/toolkit.h > as:

```
typedef struct support
{
     char read_req;
     char write_req;
     char upload_req;
     char dnload_req;
     char ondemand_req;
     char start_req;
     char stop_req;
     char model_req;
     char unsolic_req;
     char det_dev_status;
     char host_redundancy;
     char read_addr_req;
     char write_addr_req;
     char timestamp_unso_pt;
```

```
    char adhoc_req;
    char use_dp_fp;
    char network_redundancy;
    char set_array_element;
    char vlist_addressing;
    char use_long_addresses;
    char unsolicited_quality_data;
    char saved_device_req;
    char dyn_cfg_req;
    char extended_user_bits;
    char allow_64_bit_ints;
    char use_custom_cache_blocks;
    char use_long_point_id;
    char use_custom_domain_count;
    char state_unso_pt;
    char use_custom_persisted;
    char feature1;
    char feature2;
    char feature3;
    char feature4;
    char feature5;
    char feature6;
    char feature7;
} SUPPORT;
```

For all the above, valid values are:

TOOLKIT_YES	The option is supported.
TOOLKIT_NO	The option is not supported.

Where:

read_req

indicates whether or not it is possible to read from the device's memory using the implemented protocol.

Set to:

TOOLKIT_YES	If the protocol can read from the device's memory.
TOOLKIT_NO	If the protocol cannot read from the device's memory.

write_req

Indicates whether or not it is possible to write to the device's memory using the implemented protocol.

Set to:

TOOLKIT_YES	If the protocol can write to the device's memory.

TOOLKIT_NO	If the protocol cannot write to the device's memory.

upload_req

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

dnload_req

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

ondemand_req

Indicates whether or not **user_device_ok** is used to determine the device's status.

Set to:

TOOLKIT_YES	If **user_device_ok** is used.
TOOLKIT_NO	If the device status is determined from communication status.

start_req

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

stop_req

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

model_req

Indicates whether or not verification that communication with a device with the correct CPU model is occurring.

Set to:

TOOLKIT_YES	If verification is occurring.
TOOLKIT_NO	If verification is not occurring.

unsolic_req

Indicates whether or not unsolicited data is supported.

Set to:

TOOLKIT_YES	If unsolicited data is supported.
TOOLKIT_NO	If unsolicited data is not supported

det_dev_status

Indicates whether or not **user_device_ok** is used to determine device status.

Set to:

TOOLKIT_YES	If **user_device_ok** is to be used.
TOOLKIT_NO	If device status is determined from communication status.

host_redundancy indicates whether or not Host Redundancy is supported for this device.

TOOLKIT_YES	If Host Redundancy is supported for this device.
TOOLKIT_NO	If Host Redundancy is not supported for this device.

read_addr_req

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

write_addr_req

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

timestamp_unso_pt

Indicates whether or not a user-provided timestamp is sent with unsolicited data. This field is meaningful only if **unsolic_req** is set to TOOLKIT_YES.

Set to:

TOOLKIT_YES	To invoke **user_process_unsolicited_data_stamp** .
TOOLKIT_NO	To invoke **user_process_unsolicited_data** .

adhoc_req

Indicates whether or not the enabler supports Point by Address requests.

Set to:

TOOLKIT_YES	If the enabler supports Point by Address requests.
TOOLKIT_NO	If the enabler does not support Point by Address requests.

use_dp_fp

Indicates whether or not double-precision floating point numbers are supported.

Set to:

TOOLKIT_YES	If 8 byte floating point data is supported.
TOOLKIT_NO	If 4 byte floating point data is supported.

network_redundancy is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

set_array_element

Indicates whether a set array element request should be processed by the devcom.

Set to:

TOOLKIT_YES	If set array elements is supported.
TOOKIT_NO	If set array elements are not supported

vlist_addressing

Is reserved for for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

use_long_addresses

Indicates whether or not the device communications supports point addresses up to 256 characters.

Set to:

TOOLKIT_YES	If long addresses are supported.
TOOLKIT_NO	If long addresses are not supported.

unsolicited_quality_data

Indicates whether or not the receipt of unsolicited quality data is supported.

Set to:

TOOLKIT_YES	If the devices will be sending unsolicited quality data.

TOOLKIT_NO	If the devices will not be sending unsolicited quality data.

`saved_dev_req`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`dyn_cfg_req`

Indicates whether or not the dynamic configuration is supported.

Set to:

TOOLKIT_YES	If dynamic configuration is supported.
TOOLKIT_NO	If dynamic configuration is not supported

`extended_user_bits`

Indicates whether or not extended user bits will be used to represent the quality data..

Set to:

TOOLKIT_YES	If extended bits are supported. This means TOOLKIT_QUALDATA2 will be used to represent the quality data.
TOOLKIT_NO	If extended bits are not supported. This means TOOLKIT_QUALDATA will be used to represent the quality data.

`allow_64_bit_ints`

Indicates whether or not the device communications supports 64 bit signed or unsigned integer point types..

Set to:

TOOLKIT_YES	If signed and unsigned 64 bit integers are supported.
TOOLKIT_NO	If signed and unsigned 64 bit integers are not supported.

`use_custom_cache_blocks`

Is reserved for for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`use_long_point_id`

Indicates whether or not the device communications will receive point ids using their display format or internal representation.

Set to:

TOOLKIT_YES	If point IDs are represented as they are displayed.
TOOLKIT_NO	If point IDs are represented using their internal representation.

`use_default_domain_count`

Indicates whether or not the device communications will define the maximum number of domains per device based on user defined value or TOOLKIT_MAX_DEVICE_DOMAINS.

Set to:

TOOLKIT_YES	If the maximum number of possible device domains on the device is user-defined
TOOLKIT_NO	If the maximum number of possible device domains is TOOLKIT_MAX_DEVICE_DOMAINS.

`state_unso_points`

Indicates whether or not point state is supported.

Set to:

TOOLKIT_YES	If point state is supported.
TOOLKIT_NO	If point state is not supported.

`use_custom_persisted`

Is reserved for for GE Intelligent Platforms use.

Set to TOOLKIT_NO unless specifically told by GE Intelligent Platforms to set it to TOOLKIT_YES.

`Feature1`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`Feature2`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`Feature3`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`Feature4`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`Feature5`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`Feature6`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

`Feature7`

Is reserved for GE Intelligent Platforms use.

Set to TOOLKIT_NO.

TOOLKIT_QUALDATA

The TOOLKIT_QUALDATA structure is defined in < inc_path/toolkit.h > as:

```
typedef struct toolkit_qualdata2
{
    unsigned __int64   sys_flags ;
    unsigned __int64   sys_changed_mask ; /* indicates items of interest in
 sys_qual_flags */
    unsigned __int64   user_flags ;
    unsigned __int64   user_changed_mask ; /* indicates items of interest
 in user_qual_flags */
} TOOLKIT_QUALDATA2 ;
```

Where:

sys_flags is reserved for future use. Set the value to zero (0).

sys_changed_mask is reserved for future use. Set the value to zero (0).

user_flags correlates to the USER_FLAGS attributes in Point Configuration.

The value is set by the Toolkit devcom as determined by the developer. Bit definitions are assigned by the developer. It is anticipated that the bit definitions match an attribute set definition in the project.

user_changed_mask indicates which bits of **user_flags** are part of the update. Bits in **user_flags** will be ignored unless their corresponding bits in **user_changed_mask** are set to **1**.

TOOLKIT_QUALDATA2

The TOOLKIT_QUALDATA2 structure is defined in < inc_path/toolkit.h > as:

```
typedef struct toolkit_qualdata2
{
    unsigned short   sys_flags ;
    unsigned short   sys_changed_mask ; /* indicates items
                                           of interest in
                                           sys_qual_flags */
    unsigned short   user_flags ;
    unsigned short user_changed_mask ; /* indicates items
                                           of interest in
                                           user_qual_flags */
} TOOLKIT_QUALDATA2 ;
```

Where:

sys_flags is reserved for future use. Set the value to zero (0).

sys_changed_mask is reserved for future use. Set the value to zero (0).

user_flags correlates to the USER_FLAGS attributes in Point Configuration.

The value is set by the Toolkit devcom as determined by the developer. Bit definitions are assigned by the developer. It is anticipated that the bit definitions match an attribute set definition in the project.

user_changed_mask indicates which bits of **user_flags** are part of the update. Bits in **user_flags** will be ignored unless their corresponding bits in **user_changed_mask** are set to **1**.

TOOLKIT_STATUS

The TOOLKIT_STATUS ENUM structure is defined in **<inc_path/cor.h>** as:

```
typedef enum {
   TOOLKIT_SUCCESS = 1,
   TOOLKIT_FAILURE = 2,
   TOOLKIT_WARNING = 0,
```

```
    TOOLKIT_UNSUPPORTED = 3,
    TOOLKIT_REPLY_LATER = 4,
    TOOLKIT_RESPONSE_REQ = 5,
    TOOLKIT_RESPONSE_NO_REQ = 6,
    TOOLKIT_GMR_PARTIAL_WRITE_FAIL = 7,
    TOOLKIT_WRITE_FAILED,
    TOOLKIT_READ_TIMEOUT,
    TOOLKIT_BAD_DATA,
    TOOLKIT_UNSOLICITED
    TOOLKIT_SUCCESS_NO_DATA
    TOOLKIT_NO_CHANGE_APPROVAL_SUPPORT,
    TOOLKIT_CHANGEAPPROVAL_FAIL
} TOOLKIT_STATUS;
```

Chapter 8. Point Management API

About Point Management API

Point Management API provides an interface between application programs and CIMPLICITY software's ability to monitor data point values.

Point Management is a product option for GE Intelligent Platforms' CIMPLICITY software. This Application Program Interface (API) is fully integrated with CIMPLICITY software's Base System functionality to enhance its already powerful monitoring capability in a full range of computer integrated manufacturing environments.

Point Management API Overview

Point Management API Overview

Point Management is a set of processes and functions that manages CIMPLICITY points. Mechanisms are provided to define points, to distribute point data across networked systems, and to generate alarms based on pre-configured conditions. Each CIMPLICITY point is identified by a Point ID and may be either a device point or a derived point.

A device point is one whose value is associated directly with a data source such as a PLC device.

A derived point (also known as a virtual point) is a data point whose value is calculated by an arithmetic or logical expression.

System Overview

Point Management consists of the following modules:

- Point Management Resident Process (PTMRP)
- Point Management Application Library (PTMAP)
- Point Translation Process (PTX)
- Derived Point Process (PTMDP)
- Point Configuration Data

The function of each module is discussed in the following sections.

Point Management Resident Process (PTMRP)

The Point Management Resident Process (PTMRP) receives point data from other processes, responds to application requests for point data, and generates alarms when point data is outside configured limits.

Applications that need access to point data send requests to a PTMRP. These messages are sent using the Point Management Application Library (PTMAP). It is the job of PTMAP to determine which PTMRP is responsible for a specific point. Thus, applications do not need to be aware of the location of point data in a system with multiple PTMRPs.

Each PTMRP generates alarms for the points it handles.

Point Translation Process (PTX)

The Point Translation Process (PTX) manages point configuration data for applications. PTX accesses configuration data when Point Management starts up, and sends that configuration data to applications on request. Applications do not communicate directly with PTX. Application requests to PTMAP result in communications between PTX and the application.

Derived Point Process (PTMDP)

The Derived Point Process (PTMDP) provides a mechanism for summarizing information about the system, or identifying conditions that can only be recognized by evaluating several pieces of data. The Derived Point Process collects point data from the Point Management Resident Processes and uses that data to determine the value of derived points. Once the value of a derived point is established, the PTMRP that handles the point is sent the new value. The point data may then be accessed by applications.

Derived points are configured by specifying a point and an expression that is used to calculate the point's value. The expression may contain arithmetic, logical, and bitwise operators. Several Derived Point Processes may exist in a system, each handling a subset of derived point data.

Point Management Application Library (PTMAP)

The Point Management Application Library (PTMAP) is a function library through which applications access Point Management data. PTMAP accepts requests from the application, accesses configuration data through communications with PTX, and relays those requests on to a PTMRP. PTMAP receives responses back from PTMRP and provides mechanisms for sending those responses to the application.

Point Configuration Data

Defining an application that works with Point Management requires that points be defined through configuration data. Points that are device points must have Device Communications configuration as well.

External Interfaces

Device Communication

PTMRPs interface with Device Communications subsystems in order to receive point data values. Each PTMRP is capable of receiving data from multiple Device Communication processes.

The relationship between a PTMRP and Device Communications processes is defined in the configuration file, DEVCOM_PROC. The **ptmgmt_process_id** field in DEVCOM_PROC specifies the PTMRP to which the Device Communication process sends data.

Alarm Management

The PTMRPs interface with Alarm Management in order to notify Alarm Management of alarm conditions. The PTMRPs recognize alarm conditions by comparing point data values provided by Device Communications with alarm limits configured in Point Management configuration files. When a point data value exceeds its alarm limits, Point Management assembles an alarm message and sends that message to Alarm Management.

Application Processes (Shopping List Requests)

PTMRPs interface with application processes that need to access point data. Application processes communicate with the PTMRPs through an application library (PTMAP) that manages the exchange of data with the PTMRPs.

Application processes make requests through PTMAP using shopping lists. Requests for points are added to shopping lists by specifying the point address that was returned by the **PTMAP_add_point** function. The application library accesses the Point Translation Process to determine which PTMRP handles that point.

Notes on Internationalization for the Point Management API

- Work with strings.
- Recommended reading.

Work with strings

This API is written for the international environment. In an international environment, strings in CIMPLICITY software can be multi-byte strings. If you want your code to conform to international standards, GE Intelligent Platforms recommends that you do the following when working with strings:

- Use the **TCHAR** macros found in **TCHAR.H** .
- Declare string buffers as **TCHAR[]** . Declare string pointers as **TCHAR*** or **LPTSTR** .
- Wrap string and character constants with the **_ T()** macro.
- Use the **_tcs...()** functions in place of the **str...()** functions. For example, use **_tcslen()** in place of **strlen()** .
- Be careful when incrementing a pointer through a string. Remember that a logical character may occupy one or two **TCHAR** units. So replace code that looks like this:

```
char *cp;

for (cp = string; *cp != '\0'; ++cp)

{

    …

}
```

with code that looks like this:

```
TCHAR const *cp;

for (cp = string; *cp != _T('\0'); cp = _tcsinc(cp))

{

    …

}
```

- Avoid using a variable to hold the value of a logical character. Instead, use a pointer to a character in the string. In particular, avoid the **_tcsnextc()** macro, because the value it § Use the functions **_tccpy()** and **_tccmp()** and string pointers instead of the = and == operators on characters.
- Use **GetStringTypeEx()** instead of the character classification macros such as **_istalpha()** .
- Use **CharUpper()** and **CharLower()** instead of **_toupper()** and **_tolower()** .

Recommended Reading

Microsoft has several good papers on writing international code on its Developer Network CD and its web site. To find documentation on the web site, go to http://msdn.microsoft.com/default.asp and search for MBBCS.

For documentation on globalization, go to http://www.microsoft.com/globaldev/.

The following book is also available:

Schmitt, David A., Internationalization Programming for Microsoft ® Windows ®, ISBN 1-57231-956-9

For more information about this book, go to **http://mspress.microsoft.com/books/2323.htm** .

Point Management API Getting Started

Point Management API Getting Started

The Point Management Application Program Interface provides a C language interface to the CIMPLICITY software Point database. Using the C functions, you may create C language applications that can receive current point data from standard CIMPLICITY software devices. Once developed, these applications will run on any CIMPLICITY computer in your enterprise.

The Point Management API consists of a set of object libraries and include files you use to customize your application. It also includes seven example programs to help you design and validate your own applications. These programs are **ptq_snap.c, ptq_onchange.c** , **ptq_onchgstru.c** , **ptq_setpoint.c** , **ptq_setpt._eu.c** , **ptm_monitor.c** , and **ptm_script.c**

Using the API requires that you do the following:

- Understand the general and Point Management specific subroutine interfaces provided by CIMPLICITY software's Point Management capability.
- Understand the Point Configuration requirements described in the CIMPLICITY Base System User Document.
- Code appropriate application programs.
- Compile and link the programs as explained in this chapter.
- Be familiar with the Point Configuration file structure.

Point Management API Contents

The following is a list of all files distributed with the Point Management API. The files are loaded into the directory indicated. The environment variable **%BSM_ROOT%** is the directory where CIMPLICITY software was installed.

Include files in **%BSM_ROOT%\api\include\inc_path** are:

```
adhoc_defs.h
cimpoint.hpp
cor.h
cor_event.h
cor_mutex.h
cor_os.h
cor_stat.h
cor_strver.h
cor_time.h
ddl.h
gfclass.hpp
ipcerr.h
netcom.h
noshare_ext.h
ptexp_defs.h
ptm_defs.h
ptm_errors.h
ptm_ms.h
ptmap_defs.h
ptmap_proto.h
rtr_bcst.h
sc_recs.h
ssdef.h
```

Libraries for the PTMAP program in **%BSM_ROOT%\api\lib** are:

```
ptmap.lib
ipc.lib
cim_sc.lib
cim_mf.lib
ddl.lib
fasrtl.lib
corutil.lib
tools.lib
```

Sample program source files in **%BSM_ROOT%\api\ptm_api** are:

```
makefile
ptq_snap.c
ptq_onchange.c
ptq_onchgstru.c
ptq_setpoint.c
ptq_setpt_eu.c
ptm_monitor.c
ptm_script.c
```

```
monitor.input
script.input
```

Point Management API Sample Programs

Point Management API Sample Programs

Overview

A sample Microsoft Visual C++ `makefile`, is provided to build the sample programs. Use this `makefile` as a basis for constructing makefiles for your own applications.

Depending on how you installed Visual C++, the INCLUDE, LIB, and PATH environment variables may not be automatically set when you install MSDEV. If they are not set automatically, you will have to set them manually or run the following to set them before building any user programs.

```
for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft Visual
 Studio\Installer\vswhere.exe" -property installationPath`) do set VSPATH=
%F call "%VSPATH%\Common7\Tools\VsDevp.bat"
```

Build a Sample Program

From the CIMPLICITY Configuration cabinet for your project, select Command Prompt from the Tools menu.

1. This will ensure that your environment variables (in particular **%BSM_ROOT%** and **%SITE_ROOT%**) are set correctly.
2. In the Command Prompt window, issue the following commands (where drive is the disk where your CIMPLICITY software is installed):

   ```
   <drive>:
   cd %BSM_ROOT%\api\ptm_api
   ```

3. If the environment variables are not set automatically, issue the following command to set them:

   ```
   for /F "tokens=* USEBACKQ" %F in (`"%PROGRAMFILES(x86)%\Microsoft
    Visual Studio\Installer\vswhere.exe" -property installationPath`) do
    set VSPATH=%F call "%VSPATH%\Common7\Tools\VsDevCmd.bat"
   ```

4. Now build the executables: `nmake`

Run a Sample Program

The API process name must be stored in the PRCNAM environment variable for a sample program to run. The name is an arbitrary character string of up to 10 characters. To create PRCNAM, enter the following command in the Command Prompt window:

`set PRCNAM=<name>`

Where < name > is the API process name.

To run a sample program, following the instructions below.

Review Sample Programs

The following sample programs demonstrate how point information is collected on the system.

Program	Illustrates collecting:
ptq_snap.c	Data via SNAPSHOT requests. The application receives only the point data value when the request is processed.
ptq_onchange.c	Data via ONCHANGE requests. The application receives the current point data value, and then receives an updated point data value whenever the data value changes.
ptq_onchgstru.c	Data from structure points via ONCHANGE requests
ptm_monitor.c	Multiple points via ONCHANGE requests

The following sample programs demonstrate how point values can be modified.

Program	Illustrates changing:
ptq_setpoint.c	Point values using raw data values
ptq_setpt_eu.c	Point values using data values expressed in engineering units
ptm_script.c	Multiple point values using raw data values

When you run any of the PTQ programs, you will be prompted to enter a Point ID. The point ID that you specify must be:

- For a point that has been configured through the application configuration functions.
- Defined in the current running project.

If the point is not in the current running project, an error message such as "Null Point Address" is displayed. The system must be updated with that configuration data.

Before you run any of the test programs (or one of your own applications), you must always be sure that the CIMPLICITY processes have completed their startup.

Note: When you run the **ptm_monitor** and **ptm_script** programs, you will be prompted for an input file to be used. Sample **monitor.input** and **script.input** files are provided for references.

Run ptq_snap

To run **ptq_snap** , enter the command:

```
ptq_snap
```

The **PTQ >** prompt appears. You can display point values or point attribute information.

- To display a point value, enter the Point ID for the point you want to display then press **Enter**.
- To display attribute information for a point, enter the Point ID for the point followed by a period (.) and the attribute name you want to display, then press **Enter**. You can use any of the attributes documented in Point Attribute Descriptions.

Always enter the Point ID in upper-case characters since the point data base is case sensitive. The sample program does not allow you to view Point IDs that contain embedded spaces.

After you press **Enter**, the current value for the point in the point management data base is displayed. Remember that this is the current value in the data base. The frequency with which the point is collected by the system is determined by the point's configured scan rate. If the configured rate is 5 seconds, there may be as much as a five second delay between changes at the controller and in the point management data base.

The point value is displayed with engineering units conversion if so configured. After the value is shown, the **PTQ>** appears again, and you may enter another Point ID.

Run ptq_onchange

To run **ptq_onchange** , enter the command:

```
ptq_onchange
```

The **PTQ >** prompt appears. You can display point values or point attribute information.

- To display a point value, enter the Point ID for the point you want to display then press **Enter**.
- To display attribute information for a point, enter the Point ID for the point followed by a period (.) and the attribute name you want to display, then press **Enter**. You can use any of the attributes documented in Point Attribute Descriptions.

Always enter the Point ID in upper-case characters since the point database is case sensitive. You will not be able to view Point IDs that contain embedded spaces.

After you press **Enter**, the current value for the point in the point management database is displayed. Remember that this is the current value in the database. The frequency with which the point is collected by the system is determined by the point's configured scan rate. If the configured rate is 5 seconds, there may be as much as a five second delay between changes at the controller and in the point management data base.

The point value is displayed with engineering units conversion if so configured. After the value is shown, the program waits for a new value to be received from point management. Whenever the value in the point management data base changes, that value is sent to the program, and the value is displayed on the terminal. Press **Ctrl+C** to terminate the program.

Run ptq_onchgstru

If you have a communications enabler (like Siemens H1-TF) that supports Structure points, you can use this test program to display them.

To run **ptq_onchgstru** , enter the command:

```
ptq_onchgstru
```

The **PTQ** > prompt appears. Enter the Point ID for the Structure point you want to display and press **Enter .**

After you press Enter, the current value for the Structure point in the point management database is displayed. Remember that this is the current value in the database. The frequency with which the Structure point is collected by the system is determined by the Structure point's configured scan rate. If the configured rate is 5 seconds, there may be as much as a five second delay between changes at the controller and in the point management data base.

The Structure point field values are displayed with engineering units conversion if so configured. After the values are shown, the program waits for the Structure point update to be received from point management. Whenever one of the fields changes value, the Structure point is sent to the program, and the updated field values are displayed on the terminal. Press **Ctrl+C** to terminate the program.

Run ptq_setpoint or ptq_setpt_eq

To run **ptq_setpoint** , enter the command:

```
ptq_setpoint
```

To run **ptq_setpt_eu** , enter the command:

```
ptq_setpt_eu
```

For either of these programs, the **PTQ_SETPOINT** > prompt appears. Enter the Point ID for the point you want to display and press **Enter**. The Point ID should be entered in upper-case characters since the point data base is case sensitive. You are not able to modify point values for tag names that contain embedded spaces.

Note: The point you select must have read/write access. Also, for ptq_setpt_eu, the point must have linear or custom conversion defined

After you press **Enter**, the **Value >** prompt appears if you are running **ptq_setpoint** , or the **Converted Value>** prompt appears if you are running **ptq_setpt_eu** . Enter the new point value in raw data or engineering units, as indicated by the prompt, and press **Enter**.

After you press **Enter**, the program attempts to change the value of the point. The results of this attempt are displayed and the program terminates.

Run ptm_monitor

The program **ptm_monitor** requires an input file that contains a list of Point IDs for points to be monitored. A sample **monitor.input** file is included with the API. The last word of the input file must be the word **EXIT**, and each Point ID must be on a separate line.

To run **ptm_monitor** , enter the command:

```
ptm_monitor
```

The **Input File :** prompt appears. Enter the name of the file that contains the list of points to be monitored.

Every time one of the points in the input file changes value, the new value will be displayed.

To stop the program, press **Ctrl+C**.

Note: The point management API does not support referencing the following points in the same request:

- Points by address (adhoc points).
- Points by point ID.

Run ptm_script

The program **ptm_script** requires an input file that contains a list of Point IDs and values to be set. A sample **script.input** file is included with the API. The following commands are also valid in this file:

WAIT	The program will wait <n> seconds before processing the next record.
PAUSE	The program will wait until the user presses **Enter**.
EXIT	The program will stop reading the input file and terminate.

To run **ptm_script** , enter the command:

```
ptm_script
<filename>
```

Where < filename > is the name of the input file.

Every time one of the points in the input file changes value, the new value is displayed.

Point Management Application Interface Overview

Point Management Application Interface Overview

PTMAP provides the application interface to point data. Its function library provides the following services:

Manage Local Point Data	Each point to be accessed by an application must be added to a local store of point data.
Specify Requests	Once a point is added to the local store, the application may make requests for that point data. For example, the application may request receipt of a point's data value every time the value changes.
Group Requests	Requests may be organized into Shopping Lists. A Shopping List represents a group of requests that the application makes at one time. The application can organize its requests in any number of Shopping Lists.
Send Requests	Once requests have been specified and grouped together, the application transmits that information to a PTMRP. The actual sending of the requests is handled by PTMAP.
Wait for Responses	After requests have been made, the application may wait for responses to be returned. The application may choose to wait for single responses to be returned or for complete sets of responses.
Receive Responses	Once responses have been received by PTMAP, the application is notified (either by explicitly waiting or by testing an event flag) that responses are available. The application may then access each response and the data that has been returned.

Static Efficiency of Point Management Requests

To develop an efficient application you will usually design the application so the least number of messages are passed between the application and Point Management.

In general a message is sent from:

- The application to Point Management when you send a request to point management,
- Point Management to the application when your application receives a data value (multiple requests and data values do get packed into the messages when possible).

From this you can assume the following about requests.

Requests	Comments
SNAPSHOT	Most efficient way to get a value required only once. However, when a value must be retrieved multiple times, SNAPSHOT requests will be staticly inefficient. This is because two messages are required to collect a single data value: one to send the request; and another when the data value is sent to your application.
TIMED	more efficient when a value is required periodically, because once your application makes the request of Point Management, it never has to do so again. On the other hand, if the time interval specified in the TIMED request is short (a few seconds or less), your application will consume substantial CPU resources because it will be receiving data values frequently.
ONCHANGE	(Like timed requests) are more efficient when a value must be monitored in an ongoing fashion, because once your application makes the request of Point Management, it never has to do so again. Since your process will be informed whenever the point value changes, you receive the data only when necessary. Depending on the frequency of changes in point value, the polling rate for the point, and the requirements of your application, ONCHANGE requests may be more or less efficient than TIMED requests.

Consider the following for point types.

Point Type	Comments
Digital	There should be no reason (other than programming convenience) for requesting points on a TIMED basis. Your application should always request that the point be requested ONCHANGE.
Analog	Consider the frequency at which you really need to track the data. If your analog values tend to change slowly with small fluctuations, it may be better to make timed requests.
	Example You have configured an analog point that it is sampled every 5 seconds, i.e. the base scan rate for the device is 5 seconds and the point is configured to be scanned at 1 times the base scan rate. The dynamics of the point are such that it changes every 5 seconds, but within a range that is insignificant to your application. In this case it may be better to request the values on a TIMED basis at 1 minute intervals if small fluctuations in value matter little to your application. Note that although your application receives the point every minute, status monitoring screens will continue to receive the point whenever it changes. If you think that under no circumstance is it necessary for the point to be sampled at 5 second intervals, you should consider changing the scan rate for the point.

On-Alarm Requests

On-Alarm requests provide the ability to request notification from PTMRP, when the alarm state of a point changes. Each point can have four alarm limits defined, each of which defines an alarm state. These states are referred to as **Alarm High**, **Warning High**, **Warning Low**, and **Alarm Low**. The following diagram illustrates these states:

PTMAP Error Handling

Errors that occur during calls to PTMAP are recorded in the COR_STATUS structure and returned to the application. Unless noted otherwise in the function descriptions, all PTMAP functions return

either COR_SUCCESS, COR_WARNING, or COR_FAILURE. If either COR_WARNING or COR_FAILURE is returned, additional information is returned in the status structure: the **err_msg** field is filled with an error message and the **err_code** field contains the PTMAP error code.

The COR_STATUS structure definition:

```
typedef struct cor_status
{
  COR_I4      status;       /* success, failure, or warning */
  COR_I4      err_source;   /* what detected the error */
  COR_I4      err_code;     /* what the error code is */
  COR_I4      err_ref;      /* where the error occurred */
  COR_BOOLEAN err_reported; /* has it been logged yet? */
  char        err_msg[80]   /* any text message */
}
  COR_STATUS;
```

A list of error codes for all Point Management processes is included in Appendix A.

Initialize and Terminate PTMAP Services

PTMAP_initiate *(page 425)*	Initiate PTMAP Services
PTMAP_terminate *(page 438)*	Terminate PTMAP Services

Manage Local Point Data

PTMAP maintains a local data store of point information for each application. The application must declare the points that it accesses before using that point in a Point Management request.

Applications declare the points with the function **PTMAP_add_point** . When that function is called, PTMAP communicates with PTX (the Point Translation Process) to validate that the point is configured in the system and to access the point's configuration data.

Once a point has been added to the local data store, it can be referenced in multiple requests. PTMAP maintains the point in its local store until the application removes it by calling the function **PTMAP_remove_point .**

The subroutines are:

PTMAP_add_point *(page 395)*	Add Point To Local Data Storage
PTMAP_add_pt_list *(page 396)*	Add Points To The Local Data Store From A List

PTMAP_remove_point *(page 427)*	Remove A Point From The Local Data Store

Manage Shopping Lists

Applications create Shopping Lists and add requests to the Shopping Lists in order to access Point Management services. Point Management provides the following services through requests:

Requests that cause Point Management to continually supply point values:

Request Type	Description
On-Change	Point Management returns the current value of a specified point to the application and then returns the point value every time that the point changes. To stop receiving point values, the application must cancel the request.
On-Alarm	Point Management returns the value of a specified point when it enters into an alarm state. Point Management will continue to send the value whenever the point changes state. To stop receiving point values, the application must cancel the request.
On-Acknowledge	Point Management returns the value of a specified point when the alarm acknowledge status of the point changes. Point Management will continue to send the value whenever the alarm acknowledge status changes. To stop receiving point values, the application must cancel the request.
Timed	Point Management returns the value of a specified point after the expiration of a designated time interval. Point Management reinitiates the interval when it sends point data and sends the point data again after the interval next expires. To stop receiving point values, the application must cancel the request.

Requests that must be submitted each time a point value is required:

Request Type	Description
Snapshot	Point Management returns the current value of a specified point to the application. The application must send the request a second time in order to receive another value.

Requests to change a point's value or Alarm Parameters.

Request Type	Description
Setpoint	The application sends a data value to Point Management to be downloaded to a device.

Requests cannot be added to null Shopping Lists. That is, before any add requests are issued for a particular Shopping List, a `PTMAP_add_sl` must have been issued to define the Shopping List. After requests have been added to a Shopping List, the Shopping List must be sent to Point Management to register the request with a PTMRP.

The subroutines are:

PTMAP_add_sl *(page 399)*	Create A Shopping List

PTMAP_remove_sl *(page 428)*	Remove A Shopping List
PTMAP_add_onchange *(page 394)*	Add "On-Change" Request To Shopping List
PTMAP_add_onalarm *(page 392)*	Add "On-Alarm" Request To Shopping List
PTMAP_add_alarm_ack_state *(page 391)*	Receive Change of Alarm Acknowledge State Message
PTMAP_add_timedpt *(page 400)*	Add Timed Point Request To Shopping List
PTMAP_add_snapshot *(page 400)*	Add "Snapshot" Request To Shopping List
PTMAP_add_setpoint *(page 397)*	Add "Setpoint" Request To A Shopping List
PTMAP_set_req *(page 435)*	Disable/Enable An Existing Request

Modify Requests

Once a request has been added to a Shopping List, it remains on that Shopping List until it is removed. PTMAP provides a function to modify "Setpoint" requests existing on a Shopping List. Other Shopping List requests cannot be modified using this function. Instead, they are canceled (using one of the **PTMAP_cancel** functions) and then added in the modified state.

The subroutine is:

PTMAP_modify_setpoint *(page 425)*	Modify "Setpoint" Request

Suspend and Resume Receipt of Responses

PTMAP provides the ability to suspend and resume the receipt of responses from Point Management. These functions should be used whenever the application will be unable to accept responses from the PTMRP. For example, if the application must synchronously accept user input for an undetermined length of time, it should suspend requests before allowing user input, and resume requests when interaction with the user has completed.

During the time that responses are suspended, the application does not receive "On-Change", "On-Alarm", "On-Acknowledge", and Timed requests. When responses are resumed, the application is sent all point values for which "On-Change", "On-Alarm", "On-Acknowledge", and Timed requests were made.

The subroutines are:

PTMAP_suspend *(page 437)*	Suspend Receipt Of Responses
PTMAP_resume *(page 429)*	Resume Receipt Of Responses

Enable/Disable Requests

PTMAP provides the ability to selectively enable or disable requests that have been added to Shopping Lists. Once the request has been added to the Shopping List, the application must set its state as either enabled or disabled. If the request is disabled, it will not be sent to the PTMRP.

The subroutines are:

PTMAP_set_sl *(page 435)*	Enable/Disable All Requests In A Shopping List
PTMAP_set_point *(page 434)*	Enable/Disable Requests For Points
PTMAP_set_sl_point *(page 436)*	Enable/Disable Requests For A Shopping List Point
PTMAP_set_all *(page 433)*	Enable/Disable All Requests

Cancel Requests

PTMAP provides functions to cancel outstanding requests. When a request has been sent to the PTMRP, the application is considered to have an outstanding request until PTMRP responds. "On-Alarm", "On-Acknowledge", or "On-Change" requests cause the application to have outstanding requests until the request has been canceled.

Applications must cancel requests to stop the PTMRP from responding to the request. Responses received prior to the cancellation of the request must be processed before deleting that request using **PTMAP_get** functions.

The subroutines are:

PTMAP_cancel_req *(page 404)*	Cancel A Single Request
PTMAP_cancel_sl *(page 405)*	Cancel Requests For A Shopping List
PTMAP_cancel_all *(page 403)*	Cancel All Outstanding Requests
PTMAP_cancel_point *(page 403)*	Cancel Requests For A Point

Send Requests to Point Management

In order for an application to send requests to Point Management, it must use the **PTMAP_send** functions described in this section. (Responses to requests can be accessed using **PTMAP_get** functions)

The subroutines are:

PTMAP_send_req *(page 431)*	Send A Request
PTMAP_send_point *(page 431)*	Send All Requests For A Point
PTMAP_send_sl *(page 432)*	Send A Shopping List
PTMAP_send_sl_point *(page 433)*	Send Shopping List Requests For A Point
PTMAP_send_all *(page 430)*	Send All Shopping List Requests

Wait for Point Management Responses

Once an application has sent requests to Point Management, it must wait for responses to be returned. The application can use one of two strategies for recognizing that responses have been received. The application can check the event flag that was passed to PTMAP in the **PTMAP_initiate** function; the event flag is set high when a response is received. Or the application can call a **PTMAP_wait** function to wait for the receipt of one or more responses.

Once responses have been received, the application may call one of the **PTMAP_get** functions to access the responses.

The subroutines are:

PTMAP_wait_req *(page 440)*	Wait For A Response To A Request
PTMAP_wait_point *(page 439)*	Wait For A Response For A Point
PTMAP_wait_sl *(page 440)*	Wait For Response To Shopping List Request
PTMAP_wait_sl_point *(page 441)*	Wait For Response To Shopping List Point Request
PTMAP_wait_all *(page 438)*	Wait For Responses To Any Request

Get Point Management Responses

Once Point Management returns responses to the application process, the application may call one of the **PTMAP_get** functions to access the responses. The **PTMAP_get** functions allow the application to specify the responses that it is interested in. Each **PTMAP_get** function returns a single response and must be called iteratively to get all responses. An error is returned when there are no more responses to be returned. For example, after sending a Shopping List containing four "Snapshot" requests and waiting for the responses, the application should call **PTMAP_get_sl** four times to get the responses. If the application calls **PTMAP_get_sl** a fifth time, COR_WARNING is returned.

In addition to checking the return status on the **PTMAP_get** function (returned in retstat argument) the application must check the status of the response which is returned in the **rsp_stat** argument. Depending on the **rsp_stat -> status** value, **rsp_ptr** points to a **ptm_rsp** record containing the response data as follows:

rsp_stat -> **status**	Remainder of **rsp_stat** -> **status**	**rsp_ptr**	**ptm_rsp**
COR_FAILURE	Set	null	----
COR_WARNING	set *	valid	Point ID Point state; no data, no stamp, data present if last known value is available
COR_SUCCESS	Undefined	valid	each field set

* If a point is unavailable, **rsp_stat -> err_code** is set to PTM_POINT_UNAVAILABLE. Additional information is also available in this case in **rsp_stat -> err_ref** . If **rsp_stat -> err_ref** is set to PTMAP_RP_UNREACHABLE, the Point Management Resident Process is unreachable; otherwise, the point is unavailable for some other reason.

After accessing required information following a **PTMAP_get** function, applications must deallocate this structure by calling the **PTMAP_free_ptm_rsp** function. See the next section for additional information on accessing point data.

The subroutines are:

PTMAP_get_req *(page 418)*	Get Point Management Response By Request
PTMAP_get_point *(page 413)*	Point Management Response By Point
PTMAP_get_sl *(page 420)*	Get Point Management Response By Shopping List
PTMAP_get_sl_point *(page 421)*	Get Point Management Response By Shopping List Point

PTMAP_get_all *(page 410)*	Get All PTMAP Responses

Access Point Data

Access Point Data

Once the application has responses in the form of PTM_RSP structure, it is possible to access the point value and point state information. The application has access to the response structure (PTM_RSP) and to the data structure (PTM_DATA) that is part of the response structure. The data value returned to the application is the raw value. The application may choose to convert that data to a real number and also retrieve the engineering-units label for the point.

The structures discussed below are in two files:

- BSM_ROOT%\api\include\inc_path\ptm_defs.h
- %BSM_ROOT%\api\include\inc_path\ptmap_defs.h

PTMAP Response Structure

The response structure is:

```
typedef struct ptm_rsp
{
    PTM_POINT_STATE state;
    TCHAR        point_id[EXT_ADHOC_POINT_ID_LEN + 1];
    UCHAR        _fill1[2];
    COR_U1       rsp_complete;
    COR_STAMP    timestamp;
    UCHAR        default_data;
    UCHAR        alarm_enabled;
    UCHAR        warning_enabled;
    UCHAR        ack_occurred;
    COR_I2       array_index;
    PTM_DATA     *data;
} PTM_RSP;
```

Applications may access the fields for the following information:

State	The point state. The state of the point indicates if it is in a normal state, an alarm state, or if the point is unavailable. See the next section for complete documentation on the alarm state.
point_id	Configured point identifier.
timestamp	The time that the point was collected.
data	A pointer to the point data.

The other fields listed in the structure are reserved for use by GE Intelligent Platforms.

PTMAP Data Structure

The PTMAP data structure is:

```
typedef struct ptm_data_rec
{
    PTM_DATA_TYPE    type;
    PTM_DATA_LENGTH  len;
    PTM_ELEMENTS     elem;
    UCHAR            fill[PTM_DATA_FILL];
    COR_I1           value[PTM_MAXSIZE];
} PTM_DATA;
```

Applications can access the fields for the following information:

`type`	Configured point data type, such as BOOL, SINT, UINT.

See the POINT_TYPE.DAT description for further details.

`len`	Length of point type.
`elem`	Number of elements in the point.
`value`	Point value.

Note: Never declare a variable as PTM_DATA since PTM_MAXSIZE can equal 64K.

The macros described below are used to access these fields.

Memory for PTMAP data must be allocated and deallocated using the following functions:

- PTMAP_alloc_ptm_data
- PTMAP_alloc_eu_conv
- PTM_free_ptm_data
- Data structures can be copied from one record to another already allocated record using

The subroutines are:

PTMAP_fold_ack_state *(page 407)*	Determine the Alarm State of a Point
PTMAP_free_ptm_rsp *(page 409)*	Deallocate Memory For PTM_rsp
PTMAP_alloc_ptm_data *(page 402)*	Allocate Memory For PTM_DATA
PTMAP_alloc_eu_data *(page 402)*	Allocate Memory For EU Conversion Result

PTMAP_free_ptm_data *(page 409)*	Deallocate Memory For PTM_DATA
PTMAP_copy_point *(page 405)*	Copy A PTM_DATA Record
PTMAP_eu_conv *(page 406)*	Convert Raw Value To Real Number With Engineering Units
PTMAP_rev_eu_conv *(page 429)*	Convert Point Value From Engineering Units To Raw Value

Access Point Configuration Data

PTMAP provides access to point configuration data for applications. PTMAP acquires the configuration data from the Point Management Translation Process (PTX) when the point is added to the local data store. The application has access to the points data size, type and length.

The subroutines are:

PTMAP_get_point_info *(page 416)*	Get Point Information
PTMAP_get_eu_info *(page 411)*	Get Configured Engineering Units
PTMAP_get_eu_label *(page 412)*	Get Label For Engineering Units
PTMAP_get_init_state *(page 412)*	Get The Initial State Of A Point
PTMAP_get_type *(page 424)*	Get The Data Type For A Point
PTMAP_get_struct_components *(page 423)*	Get Structure Point Components
PTMAP_get_point_type *(page 418)*	Get The Point Type ID For A Given Point ID

Point Management API Subroutines

Point Management API Subroutines

A *(page 388)*	C *(page 389)*	E *(page 389)*	F *(page 389)*	G *(page 390)*	I *(page 390)*	M *(page 390)*	R *(page 390)*	S *(page 391)*	T *(page 391)*	W *(page 391)*

A

- PTMAP_add_alarm_ack_state
- PTMAP_add_setpoint_chgapproval
- PTMAP_add_onalarm
- PTMAP_add_sl
- PTMAP_add_onchange
- PTMAP_add_snapshot
- PTMAP_add_point
- PTMAP_add_timedpt
- PTMAP_add_pt_list
- PTMAP_alloc_eu_data
- PTMAP_add_setpoint
- PTMAP_alloc_ptm_data

C

	PTMAP_cancel_all		
	PTMAP_cancel_point		
	PTMAP_cancel_req		
	PTMAP_cancel_sl		
	PTMAP_copy_point		

E

- PTMAP_eu_conv
-
-

F

- PTMAP_fold_ack_state
-
-
- PTMAP_free_point_list
-
-
- PTMAP_free_ptm_data
-
-
- PTMAP_free_ptm_rsp
-
-

G

- PTMAP_get_all
- PTMAP_get_point_type
- PTMAP_get_eu_info
- PTMAP_get__req
- PTMAP_get_eu_label
- PTMAP_get_req_point_id
- PTMAP_get_init_state
- PTMAP_get_sl
- PTMAP_get_point
- PTMAP_get_sl_point
- PTMAP_get_point_ChangeApprovalInfo
- PTMAP_get_struct_components
- PTMAP_get_point_info
- PTMAP_get_type
- PTMAP_get_point_List
-
-

I

- PTMAP_initiate
-
-

M

- PTMAP_modify_setpoint
- PTMAP_modifysetpoint_chgapproval

R

- PTMAP_remove_point
-
-
- PTMAP_remove_sl
-
-
- PTMAP_resume
-
-
- PTMAP_rev_eu_conv

-
-

S

	PTMAP_send_all		PTMAP_set_point
	PTMAP_send_point		PTMAP_set_req
	PTMAP_send_req		PTMAP_set_sl
	PTMAP_send_sl		PTMAP_set_sl_point
	PTMAP_send_sl_point		PTMAP_suspend
	PTMAP_set_all		

T

- PTMAP_terminate
-
-

W

	PTMAP_wait_all		PTMAP_wait_sl
	PTMAP_wait_point		PTMAP_wait_sl_point
	PTMAP_wait_req		

PTMAP_add_alarm_ack_state

This subroutine adds a request to receive point information when the point's alarm acknowledges state changes.

Point Management sends the information whenever the acknowledge state of the point changes. Use **PTMAP_fold_ack_state** to determine the new state of the point.

Syntax

```
int PTMAP_add_alam_ack_state ( sl_adr, point_adr,
                               req_adr, retstat )
PTMAP_ADDR      *sl_adr;
PTMAP_ADDR      *point_adr;
PTMAP_ADDR      *req_adr;
COR_STATUS          *retstat;
```

Input Arguments

sl_adr	A Shopping List ID created by a call to PTMAP_add_sl.
point_adr	A Point ID created by a call to PTMAP_add_point.

Output Arguments

req_adr	Identifier used to reference this request.
retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
PTMAP_ADDR_PTR_NULL PTMAP_SL_ADR_NULL PTMAP_SL_ADDR_NOTF PTMAP_SEQ_NUM_MISMATCH PTMAP_INVAL_DATA_TYPE PTMAP_INVAL_ELEMENTS	

Return Value

The contents of **retstat.status** .

PTMAP_add_onalarm

This subroutine adds a request to receive a point value when an alarm condition exists. Point Management sends the point value to the requester when the point is in an alarm condition.

The alarm condition is specified as a combination of **Alarm** or **Warning** and **High** or **Low**. For example, it is possible to request to be notified about an **Alarm High** condition or a **Warning Low** condition, or any other combination. When the status of the point satisfies the specified condition, Point Management sends the point value. Point Management continues to send the value whenever a point value enters the state specified by the alarm condition until the request is canceled.

The following table shows the relationship between the setting of the parameters **aw_state** and **high_low** , and the point state when your application receives notification. For example, the table shows that if **aw_state** = PTM_AW_ALARM and **high_low** = PTM_LOW, your application will receive the point data when the point state goes into **Alarm Low** state.

aw_state	high_low	point state
PTM_AW_ALARM	PTM_LOW	Alarm Low
PTM_AW_WARNING	PTM_LOW	Warning Low
PTM_AW_WARNING + PTM_AW_ALARM	PTM_LOW	Warning Low or Alarm Low
PTM_AW_ALARM	PTM_AW_ALARM	Alarm High
PTM_AW_WARNING	PTM_HIGH	Warning High
PTM_AW_WARNING + PTM_AW_ALARM	PTM_HIGH	Warning High or Alarm High
PTM_AW_ALARM	PTM_HIGH + PTM_LOW	Alarm High or Alarm Low

PTM_AW_WARNING	PTM_HIGH + PTM_LOW	Warning High or Warning Low
PTM_AW_WARNING + PTM_AW_ALARM	PTM_HIGH + PTM_LOW	Any alarm state

An On-Alarm request may only be made for point types for which alarms may be defined. That is, an "On-Alarm" request may not be made for BITSTRING, OCTET_STRING, and CHARACTER_STRING types, or for points that are more than one element in length.

Syntax

```
int PTMAP_add_onalarm ( sl_adr, point_adr, aw_state,
                        high_low, immediate,
                        req_adr, retstat )
PTMAP_ADDR      *sl_adr;
PTMAP_ADDR      *point_adr;
PTM_AW          aw_state;
PTM_HIGH_LOW    high_low;
COR_BOOLEAN     immediate
PTMAP_ADDR      *req_adr;
COR_STATUS      *retstat;
```

Input Arguments

sl_adr	A Shopping List ID created by a call to **PTMAP_add_sl** .
point_adr	A point ID created by a call to **PTMAP_add_point** .
aw_state	The alarm or warning state. The possibilities are:
	PTM_AW_ALARM - send on alarm only
	PTM_AW_WARNING - send on warning only
	PTM_AW_ALARM + PTM_AW_WARNING - send on either alarm or warning
high_low	The variance direction for which the alarm state applies. The possibilities are:
	PTM_HIGH - send on high alarm
	PTM_LOW - send on low alarm
	PTM_HIGH + PTM_LOW - send on either high or low alarm
immediate	Set to TRUE if the application requires an immediate response from Point Management containing the point value whether or not the point is in alarm state. Following the immediate response, the point value is sent to the application only when the alarm state of the point changes.

Output Arguments

req_adr	Identifier used to reference this request.
retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL

	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	PTMAP_INVAL_ONALARM_LIMIT
	PTMAP_INVAL_ONALARM_STATE
	PTMAP_INVAL_DATA_TYPE
	PTMAP_INVAL_ELEMENTS

Return Value

The contents of **retstat.status** .

PTMAP_add_onchange

This subroutine adds a request to a shopping list to receive a point value each time the point value changes. The current value of the point is returned by Point Management, and then sent whenever the point value changes. The point value is collected on each change in its value until the request is canceled or requests are suspended.

Syntax

```
PTMAP_add_onchange ( sl_adr, point_adr, req_adr,
                     retstat )
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
PTMAP_ADDR  *req_adr;
COR_STATUS  *retstat;
```

Input Arguments

sl_addr	Pointer to Shopping List created by a call to **PTMAP_add_sl** .
point_adr	A point ID created by a call to **PTMAP_add_point** .

Output Arguments

req_adr	Identifier used to reference this request.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL

	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_add_point

This subroutine adds a point to the local data store of point information. All points that are referenced in other PTMAP functions must first be added via a call to **PTMAP_add_point** . When **PTMAP_add_point** is called, the point's configuration data is accessed. Designated configuration information is available.

Syntax

```
int PTMAP_add_point ( point_id, full, point_adr,
                      retstat )
char          *point_id;
COR_BOOLEAN   full;
PTMAP_ADDR    *point_adr;
COR_STATUS    *retstat;
```

Input Arguments

point_id	Unique identifier for the point. The point must be defined in configuration data. To request the point's value, just enter the Point ID. To request the value of an attribute for the point, enter the Point ID followed by a period (.) and the attribute name. See Point Attribute Descriptions for a list of value attributes.
Full	For GE Intelligent Platforms use. Always set this argument to FALSE.

Output Arguments

point_adr	A pointer to the point record that was created. This pointer can be saved by the application to access PTMAP data on a point basis.
Retstat	Pointer to status structure. The following error may be returned (see Appendix A for an explanation of this code):
	PTMAP_POINT_ALRDY_ADDED
	PTMAP_PT_ADR_NOTF

Return Value

The contents of **retstat.status** .

PTMAP_add_pt_list

This subroutine adds points to the local data store from a list of points. This function can be used in place successive calls to **PTMAP_add_point** when several points are to be added to the local data store.

In order to use this function, an array of PTMAP_PT_LIST structs must first be allocated. The array must contain one struct for each point plus one as a terminating record. Each element of the array must be populated and the last used record should be given a NULL Point ID and point address.

The PTMAP_PT_LIST struct is defined as follows:

```
        typedef struct
          {
            RECORD_PTR ptr;
            COR_I4 seq_num;
          } PTMAP_ADDR;
        typedef struct
          {
            char       *point_id;
          PTMAP_ADDR *addr;
          } PTMAP_PT_LIST;
```

PTMAP_ADDR contains values for each point added from the list.

The following C macro has been defined to load the point list:

```
#define        LOAD_PT_LIST(dest,pt_id,point_addr)\
    dest.point_id = pt_id;\
    dest.addr =  point_addr;\
```

When you load the point list, the **pt_id** field can contain a Point ID or a Point ID followed by a period (.) and an attribute name. For more information on point attributes, see Point Attribute Descriptions.

Syntax

```
int PTMAP_add_pt_list ( point_list, full, retstat )
PTMAP_PT_LIST *point_list;
COR_BOOLEAN   full;
COR_STATUS    *retstat;
```

Input Arguments

point_list	List of Point IDs to be added

full	For GE Intelligent Platforms use. Always set this argument to FALSE.

Output Arguments

retstat	Pointer to status structure. The following error may be returned (see Appendix A for an explanation of this code):
	PTMAP_POINT_ALRDY_ADDED
	PTMAP_POINT_ADR_NOTF

Return Value

The contents of **retstat.status** .

PTMAP_add_setpoint

This subroutine adds a "Setpoint" request to a Shopping List. Point Management downloads the specified value to the device address designated by the point. The application must have a pointer to a PTM_DATA struct containing the data value. This structure must be allocated by a call to **PTM_alloc_ptm_data** .

Syntax:

```
int PTMAP_add_setpoint ( sl_adr, point_adr,
                         point_value,req_adr,
                         retstat )
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
PTM_DATA    *point_value;
PTMAP_ADDR  *req_adr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	A Shopping List ID created by a call to **PTMAP_add_sl** .
point_adr	A Point ID created by a call to **PTMAP_add_point** .
point_value	A pointer to a PTM_DATA structure containing the point value.

Output Arguments

req_adr	Identifier used to reference this request.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL

	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	PTMAP_DATA_NULLPTR
	PTMAP_DATA_TYPE_MISMATCH
	PTMAP_DATA_LEN_MISMATCH
	PTMAP_DATA_ELEM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_add_setpoint_chgapproval

This subroutine adds a **setpoint** request to a shopping list, including the change approval information. Point Management downloads the specified value to the device address designated by the point. The application must have a pointer to a `PTM_DATA struct` containing the data value.

This structure must be allocated by a call to `PTM_alloc_ptm_data`.

Syntax

```
int PTMAP_add_setpoint_chgapproval  ( sl_adr,
                                      point_adr,
                                      point_value,
                                      req_adr,
                                      changeapproval_obj,
                                      retstat )
PTMAP_ADDR          *sl_adr;
PTMAP_ADDR          *point_adr;
PTM_DATA            *point_value;
PTMAP_ADDR          *req_adr;
ChangeapprovalInfo *changeapproval_obj;
COR_STATUS  *retstat;
```

Input Arguments

`sl_adr`	A Shopping List ID created by a call to PTMAP_add_sl.
`point_adr`	A Point ID created by a call to PTMAP_add_point.
`point_value`	A pointer to a PTM_DATA structure containing the point value.
`changeapproval_obj`	A pointer to the change approval information for the setpoint operation. If there is no change approval data, this should be NULL.

Output Arguments

`req_adr`	Identifier used to reference this request.
`Retstat`	Pointer to status structure. The following warnings may be returned. PTMAP_ADDR_PTR_NULL PTM_BAD_PERFORMUSRID_PASSWORD PTM_BAD_VERIFYUSRID_PASSWORD PTMAP_CHANGEAPPROVAL_SAME_USERID
	PTMAP_DATA_ELEM_MISMATCH PTMAP_DATA_LEN_MISMATCH
	PTMAP_DATA_NULLPTR
	PTMAP_DATA_TYPE_MISMATCH PTMAP_SEQ_NUM_MISMATCH PTMAP_SL_ADDR_NOTF
	PTMAP_SL_ADR_NULL
	PTM_INVAL_CHANGEAPPROVAL_LEVEL PTM_INVAL_CHANGEAPPROVAL_RESOURCESETPOINT PTM_INVAL_PERFORMUSERID PTM_INVAL_VERIFYUSERID
	PTM_NO_PERFORMUSERID PTM_NO_PERFORM_PRIV PTM_NO_VERIFYUSERID PTM_NO_VERIFY_PRIV PTM_PERFORMER_PASSWORD_EXPIRED PTM_POINT_HASCHANGEAPPROVAL PTM_VERIFIER_PASSWORD_EXPIRED

Return Value

The contents of **retstat.status** .

PTMAP_add_sl

This subroutine creates a new Shopping List.

Once a Shopping List has been created, applications may add requests to the Shopping List and send those requests to Point Management. An application may create any number of Shopping Lists and add any number of points to that list. (These numbers are limited only by virtual memory use.) The application is returned a Shopping List identifier that uniquely identifies this Shopping List.

Syntax

```
int PTMAP_add_sl (sl_adr, retstat)
PTMAP_ADDR  *sl_adr;
COR_STATUS  *retstat;
```

Input Arguments

None.

Output Arguments

sl_adr	Shopping List identifier that may be used in calls to add/remove points to/from Shopping List or to send the Shopping List.

retstat	Pointer to status structure. No errors are defined.

Return Value

COR_SUCCESS.

PTMAP_add_snapshot

This subroutine adds a request for a "Snapshot" of a point to a Shopping List. The current point value is returned by Point Management after the Shopping List has been sent.

Syntax

```
PTMAP_add_snapshot ( sl_adr, point_adr, req_adr,
                     retstat )
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
PTMAP_ADDR  *req_adr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	A Shopping List ID created by a call to **PTMAP_add_sl** .
point_adr	A Point ID created by a call to **PTMAP_add_point** .

Output Arguments

req_adr	Identifier used to reference this request.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_add_timedpt

This subroutine adds a request to a shopping list for a point value at regular time intervals. The current value of the point will be returned by Point Management and then will be sent to the requesting process at regular time intervals. The length of the time interval is specified in the call to the function. The point values are sent until the request is canceled, or requests are suspended. If timed point requests are outstanding when requests are resumed, point values are sent immediately after resuming requests, and the delay intervals are timed from that time.

Syntax

```
PTMAP_add_timedpt ( sl_adr, point_adr, interval,
                    units, req_adr, retstat )
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
COR_I4      interval;
COR_I4      units;
PTMAP_ADDR  *req_adr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	A Shopping List ID created by a call to **PTMAP_add_sl** .
point_adr	A Point ID created by a call to **PTMAP_add_point** .
Interval	Integer specifying the length of time which Point Management is to wait between sending point values.
Units	Integer which specified the time units to which interval refers. It must be one of the following: PTM_HOURS, PTM_MINUTES, or PTM_SECONDS.

Output Arguments

req_adr	Identifier used to reference this request.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	PTMAP_INV_TIME_UNIT

Return Value

The contents of **retstat.status** .

PTMAP_alloc_eu_data

This subroutine allocates memory for PTM_DATA structure and sets the length, type, and element fields.

Syntax

```
PTM_DATA *PTMAP_alloc_eu_data ()
PTM_DATA  *result
```

Input Arguments

None

Output Arguments

Result	Pointer to data structure allocated for a result generated by **PTMAP_eu_conv** .

Return Value

Pointer to the PTM_DATA structure

PTMAP_alloc_ptm_data

This subroutine allocates memory for PTM_DATA structure and sets the length, type, and element fields.

Syntax

```
PTM_DATA *PTMAP_alloc_ptm_data (type, len, elem)
PTM_DATA_TYPE   type;
PTM_DATA_LENGTH len;
PTM_ELEMENTS elem;
```

Input Arguments

Type	Configured point data type (such as, BOOL, SINT, INT).
Len	Length of point type.
Elem	Number of elements in the point.

Output Arguments

None.

Return Value

Pointer to the PTM_DATA structure

Note: A null will be returned if virtual memory for the process is exhausted.

PTMAP_cancel_all

This subroutine cancels all outstanding requests that have been made by the application. Shopping Lists that have been created may still be referenced. Following a call to **PTMAP_cancel_all** , Point Management does not send any more responses.

Syntax

```
int PTMAP_cancel_all (retstat)
COR_STATUS  *retstat;
```

Input Arguments

None.

Output Arguments

Retstat	Pointer to status structure

Return Value

The contents of **retstat.status** .

PTMAP_cancel_point

This subroutine cancels all outstanding requests for a point. If multiple requests have been made for a point in different Shopping Lists, each of those requests are canceled. This function sends the cancellation request to Point Management and waits for a response before returning control to the application.

Syntax

```
int PTMAP_cancel_point (point_adr, retstat)
```

```
PTMAP_ADDR  *point_adr;
COR_STATUS  *retstat;
```

Input Arguments

point_adr	Point for which requests are to be canceled

Output Arguments

Retstat	Pointer to status structure. The following error may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL

Return Value

The contents of **retstat.status** .

PTMAP_cancel_req

This subroutine cancels a request that has been sent to Point Management. For example, if an "On-Change" request has been sent to Point Management, this function can be used to cancel that request. When this request is made, a message is sent to Point Management, and the application waits until a response is received from Point Management. The request is still part of the Shopping List, but it is no longer active. The request can be made active by sending the Shopping List a second time.

An error is returned if no outstanding response exists for the specified request.

Syntax

```
int PTMAP_cancel_req (can_req_adr, retstat)
PTMAP_ADDR  *can_req_adr;
COR_STATUS  *retstat;
```

Input Arguments

can_req_adr	The identifier that was returned when the request was made.

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_REQ_ADR_NULL
	PTMAP_REQ_ADDR_NOTF

	PTMAP_SEQ_NUM_MISMATCH
	PTMAP_REQ_NOT_OUT

Return Value

The contents of **retstat.status** .

PTMAP_cancel_sl

This subroutine cancels all requests made via a specified Shopping List. PTMAP cancels the outstanding requests in the specified Shopping List.

Syntax

```
int PTMAP_cancel_sl (sl_adr, retstat)
PTMAP_ADDR  *sl_adr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	A Shopping List ID created by a call to **PTMAP_add_sl** .

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_REQ_ADR_NULL
	PTMAP_REQ_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_copy_point

This subroutine copies the data value of a source point to a destination point.

Syntax

```
int PTMAP_copy_point (dest_point, src_point, retstat)
PTM_DATA   *dest_point;
PTM_DATA   *src_point;
COR_STATUS *retstat;
```

Input Arguments

Dest_point	Point to which data value is to be copied.
Src_point	Point from which data value is being copied.

Output Arguments

Retstat	Pointer to status structure.
	COR_SUCCESS will be returned on a successful copy.
	The following errors may be returned if either the source or destination point record is NULL or if the record definitions are note the same (see Appendix A for an explanation of this code):
	PTMAP_DATA_NULLPTR
	PTMAP_DATA_TYPE_MISMATCH
	PTMAP_DATA_LEN_MISMATCH
	PTMAP_DATA_ELEM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_eu_conv

This subroutine converts a raw data value to a real number with engineering units. The application passes the raw value and is returned a real value and an engineering units string. The conversion is defined by configuration data for the data point in the file, EU_CONV. If no conversion has been specified, an error is returned.

Syntax

```
int PTMAP_eu_conv (point_id, pt_val, result, eu_label,
                   retstat)
              *point_id;
PTM_DATA      *pt_val;
PTM_DATA      *result;
char          *eu_label;
```

```
COR_STATUS   *retstat;
```

Input Arguments

Point_id	Point identifier.
pt_val	Point value structure containing the raw value.

Output Arguments

Result	Point value structure containing the converted value (FLOATINGPOINT). Result is a PTM_DATA structure. Space for this structure must be allocated by a call to **PTMAP_alloc_eu_data.**
	For example, **result = PTMAP_alloc_eu_data** .
eu_label	Engineering units string from configuration data.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_INVAL_POINT_ID
	PTMAP_HAS_NOT_EU_CONV
	PTMAP_INVAL_EU_RES_TYPE

Return Value

The contents of **retstat.status** .

PTMAP_fold_ack_state

The standard point states are:

- PTM_NORMAL
- PTM_ALARM_HIGH
- PTM_ALARM_LOW
- PTM_WARNING_HIGH
- PTM_WARNING_LOW
- PTM_ALARM
- PTM_WARNING
- PTM_AVAILABLE
- PTM_OUT_OF_RANGE
- PTM_UNAVAILABLE

Whenever an alarm that was generated by Point Management is acknowledged by any source, the Alarm Manager sends notification of that fact to the Point Manager. This information is passed by Point Manager to all PTMAP clients in the PTM_RSP structure.

The additional point states that reflect point acknowledgment are:

- PTM_NORMAL_NOACK
- PTM_ALARM_HIGH_NOACK
- PTM_ALARM_LOW_NOACK
- PTM_WARNING_HIGH_NOACK
- PTM_WARNING_LOW_NOACK
- PTM_ALARM_NOACK
- PTM_WARNING_NOACK
- PTM_AVAILABLE_NOACK
- PTM_OUT_OF_RANGE_NOACK
- PTM_UNAVAILABLE_NOACK

This subroutine lets you fold the NOACK state of a point into the standard point state, and create the additional point states. These additional point states are recognized by the expression processor that is used for derived points and for graphic object annunciation.

Syntax

```
int PTMAP_fold_ack_state (state, ack_occurred)
PTM_POINT_STATE state;
int             ack_occurred;
```

Input Arguments

State	Current standard alarm state.
ack_occurred	Flag to determine acknowledge state.

Output Arguments

None.

Return Value

New alarm state. If the alarm has been acknowledged, the original alarm state is returned. If the alarm has not been acknowledged, one of the "NOACK" states will be returned.

PTMAP_free_point_list

This subroutine deletes the lists compiled by PTMAP_get_point_list *(page 417)* , thus freeing the used memory.

Syntax

```
void PTMAP_free_point_list(pszPoints)
WCHAR *pszPoints;
```

Input Arguments

`pszPoints`	Return value from PTMAP_get_point_list.

Output Arguments

Retstat	Pointer to status structure.

Return Value

None.

PTMAP_free_ptm_data

Memory for the PTM_DATA structure is not automatically deallocated. Applications must use the **PTMAP_free_ptm_data** function to deallocate memory for **ptm_data** . This function returns the value NULL in **ptm_data_ptr** , where **ptm_data_ptr** is a pointer to a PTM_DATA structure.

Syntax

```
PTM_DATA *PTMAP_free_ptm_data (ptm_data_ptr)
PTM_DATA *ptm_data_ptr
```

Input Arguments

ptm_data_ptr	Pointer created by **PTMAP_alloc_ptm_data** .

Output Arguments

ptm_data_ptr	Value is set to NULL

Return Value

None.

PTMAP_free_ptm_rsp

Memory for the PTM_RSP structure is not automatically deallocated. Applications must use the **PTMAP_free_ptm_rsp** function to deallocate memory for **ptm_rsp** . This function returns the value NULL in **ptm_rsp_ptr** , where **ptm_rsp_ptr** is a pointer to a PTM_RSP structure. This function also deallocates memory for the PTM_DATA structure **ptm_rsp_ptr -> data** .

Syntax

```
PTM_RSP *PTMAP_free_ptm_rsp (ptm_rsp_ptr)
PTM_RSP *ptm_rsp_ptr;
```

Input Arguments

ptm_rsp_ptr	Pointer to the PTMAP response structure.

Output Arguments

ptm_rsp_ptr	Value of pointer is set to NULL.

Return Value

None.

PTMAP_get_all

This subroutine gets the responses that have been returned by Point Management. This function should be called following one of the **PTMAP_wait** functions. **PTMAP_get_all** must be called once to access each response.

PTMAP_get_all will return with a success status if it has any response to be returned to the application. Otherwise, it returns a failure status.

Syntax

```
int PTMAP_get_all (req_adr, sl_adr, point_adr,
                   rsp_type, rsp_stat, rsp_ptr,
                   retstat)
PTMAP_ADDR  *req_adr;
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
int         *rsp_type;
COR_STATUS  *rsp_stat;
PTM_RSP     **rsp_ptr;
COR_STATUS  *retstat;
```

Input Arguments

None

Output Arguments

req_adr	Pointer to the request for which the response was generated..
sl_adr	Pointer to the Shopping List.
point_adr	A pointer to the point for the request
rsp_type	The type of response (such as, PTM_ONCHANGE or PTM_SNAPSHOT).
rsp_stat	Status of the response: COR_SUCCESS, COR_WARNING, or COR_FAILURE.
rsp_ptr	Pointer to a response structure.
Retstat	Pointer to status structure. The following warnings may be returned (see Appendix A for an explanation of this code):
	PTMAP_NO_RSP_RCV
	PTMAP_RCV_QUE_ERR

Return Value

The contents of **retstat.status** .

PTMAP_get_eu_info

This subroutine accesses the point data type, the engineering units label, and a pointer to the conversion expression.

Syntax

```
int PTMAP_get_eu_info (point_id, expr_ptr, eu_label,
                       eu_type, retstat)
char             *point_id;
char             **expr_ptr;
char             *eu_label;
PTM_DATA_TYPE    *eu_type;
COR_STATUS       *retstat;
```

Input Arguments

point_id	Point identifier.

Output Arguments

expr_ptr	Pointer to a translated expression in a database. (Space neither has to be allocated or deallocated.)
eu_label	The configured engineering units label.
eu_type	Data type of input value.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_INVAL_POINT_ID
	PTMAP_HAS_NOT_EU_CONV

Return Value

The contents of **retstat.status** .

PTMAP_get_eu_label

This subroutine returns the **eu_label** value from the point record. Note that it is not necessary to have defined conversion expressions in a configuration procedure in order to use this function.

Syntax

```
int PTMAP_get_eu_label (point_id, eu_label, retstat)
char            *point_id;
char            *eu_label;
COR_STATUS      *retstat;
```

Input Arguments

point_id	Point identifier.

OUTPUT ARGUMENTS:

eu_label	Pointer to a string of length EU_LABEL_LEN+1.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_INVAL_POINT_ID

Return Value

The contents of **retstat.status** .

PTMAP_get_init_state

This subroutine gets the configured state for the point from the POINT configuration file. Points can be configured and set to either ENABLED or DISABLED. If the point is disabled, no data is collected for that point.

Syntax

```
int PTMAP_get_init_state (point_addr, init_state,
                          retstat)
PTMAP_ADDR      *point_addr;
COR_BOOLEAN     *init_state;
COR_STATUS      *retstat;
```

Input Arguments

point_addr	Point address (pointer to the point).

Output Arguments

init_state	Either TRUE (the point was configured to be enabled) or FALSE (the point was configured to be disabled).
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADR_PTR_NULL
	PTMAP_PT_ADR_NULL
	PTMAP_PT_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_get_point

This subroutine gets the response to a request for a specific point. If more than a single request is outstanding for a point, this function should be called once for each request. If no response is available for the specified request, this function returns COR_WARNING.

Syntax

```
int PTMAP_get_point (point_adr, sl_adr, req_adr,
                     rsp_type, rsp_stat, rsp_ptr,
                     retstat)
PTMAP_ADDR  *point_adr;
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *req_adr;
```

```
int          *rsp_type;
COR_STATUS   *rsp_stat;
PTM_RSP      **rsp_ptr;
COR_STATUS   *retstat;
```

Input Arguments

point_adr	Pointer to the point.

Output Arguments

sl_adr	Pointer to the Shopping List associated with this request.
req_adr	Pointer to the request for which the response was generated..
rsp_type	Request type (such as, PTM_ONCHANGE or PTM_SNAPSHOT).
rsp_stat	Status of the response: COR_SUCCESS, COR_WARNING, or COR_FAILURE.
rsp_ptr	Pointer to a response structure.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADR_PTR_NULL
	PTMAP_PT_ADR_NULL
	PTMAP_PT_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warnings may be returned:
	PTMAP_NO_RSP_RCV
	PTMAP_RCV_QUE_ERR

Return Value

The contents of **retstat.status** .

PTMAP_get_point_ChangeApprovalinfo

This subroutine sends a request to Point Management for information about the point data type including change approval information.

Syntax

iint PTMAP_get_point_info_ChangeApprovalinfo (point_adr,

```
                                                data_type,
                                                data_length,
```

```
                                                elements,
                                                data_size,
                                                display_format,
                                                change_approval_mask,
                                                retstat)
PTMAP_ADDR      *point_adr;
PTM_DATA_TYPE   *data_type;
PTM_DATA_LENGTH *data_length;
PTM_ELEMENTS    *elements;
COR_I4          *data_size;
CHAR            *display_format;
int             *change_approval_mask;
COR_STATUS      *retstat;
```

Input Arguments

`point_adr`	Pointer to the point.

Output Arguments

`data_type`	Point data type.
`data_length`	The length of the data type.
`Elements`	Number of elements in point.
`data_size`	Total data size.
`display_format`	The configured display format from POINT.DAT.

`change_approval_mask`	Flag indicating what type of change approval is required for the point
	• To test whether the point is configured for unsigned writes:
	If (change_approval_mask & PTM_CHANGEAPPROVAL_UNSIGNEDWRITES)
	• To test whether the point is configured to require a performer authorization:
	If (change_approval_mask & PTM_CHANGEAPPROVAL_PERFORM)
	• To test whether the point is configured to require a performer and verifier authorization:
	If(ChangeApproval & PTM_CHANGEAPPROVAL_PERFORMVERIFY)
	• To test if unsigned writes is selected for the point :-
	If(ChangeApproval & PTM_CHANGEAPPROVAL_UNSIGNEDWRITES).

`Retstat`	Pointer to status structure. The following errors *(page 481)* may be returned. PTMAP_ADR_PTR_NULL PTMAP_PT_ADR_NULL PTMAP_PT_ADR_NOTF PTMAP_SEQ_NUM_MISMATCH PTM_POINT_HASCHANGEAPPROVAL: PTM_NO_PERFORMUSERID: PTM_NO_VERIFYUSERID: PTM_BAD_PERFORMUSRID_PASSWORD: PTM_BAD_VERIFYUSRID_PASSWORD: PTM_NO_PERFORM_PRIV: PTM_NO_VERIFY_PRIV: PTM_INVAL_CHANGEAPPROVAL_LEVEL: PTM_INVAL_CHANGEAPPROVAL_RESOURCESETPOINT: PTM_INVAL_PERFORMUSERID: PTM_INVAL_VERIFYUSERID: PTMAP_CHANGEAPPROVAL_SAME_USERID: PTM_VERIFIER_PASSWORD_EXPIRED: PTM_PERFORMER_PASSWORD_EXPIRED:

Return Value

The contents of retstat.status.

PTMAP_get_point_info

This subroutine sends a request to Point Management for information about the point data type.

Syntax

```
int PTMAP_get_point_info (point_adr, data_type,
                          data_length, elements,
                          data_size, display_format,
                          retstat)
PTMAP_ADDR      *point_adr;
PTM_DATA_TYPE   *data_type;
PTM_DATA_LENGTH *data_length;
PTM_ELEMENTS    *elements;
COR_I4          *data_size;
CHAR            *display_format
COR_STATUS      *retstat;
```

Input Arguments

point_adr	Pointer to the point.

Output Arguments

data_type	Point data type.
data_length	The length of the data type.
Elements	Number of elements in point.
data_size	Total data size.
display_format	The configured display format from POINT.DAT.

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADR_PTR_NULL
	PTMAP_PT_ADR_NULL
	PTMAP_PT_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_get_point_list

This subroutine can retrieve the list of points in any of the following points.

- In manual mode.
- With modified alarm limits.
- With disabled alarms.

Syntax

```
WCHAR*PTMAP_get_point_list (pszProject,
nListType,
retstat);
const WCHAR*pszProject;
int nListType;
COR_STATUS*retstat;
```

Input Arguments

`pszProject`	Project to get the point list from.	
`nListType`	One of the following.	
	`PTMAP_LT_MANUAL_MODE`	Get the list of points in manual mode
	`PTMAP_LT_ALM_DISABLED`	Get the list of points with disabled alarms.
	`PTMAP_LT_ALM_LIM_MODIFIED`	Get the list of points with alarm limits modified.
	`PTMAP_LT_ALM_MODIFIED`	Get the list of points with any alarm modifications.

Output Arguments

Retstat	Pointer to status structure.

Return Value

NULL terminated list of strings with each individual point id separated with a new line: `\n`.

Note: Memory used by this subroutine return can be freed using PTMAP_free_point_list *(page 408)*.

PTMAP_get_point_type

This function provides the user with the Point Type ID for the Point ID that was provided.

This routine can only return a valid Point Type ID if the point has already been added to the internal list of points through the **PTMAP_add_pt** function call.

Syntax

```
int PTMAP_get_point_type (point_id, point_type_id,
                          retstat)
char            *point_id;
char            *point_type_id;
COR_STATUS      *retstat;
```

Input Arguments

point_id	The Point ID of the point for which you want to know the Point Type ID.

Output Arguments

point_type_id	The Point Type ID of the point provided in **point_id** .
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_INVAL_POINT_ID

Return Value

The contents of **retstat.status** .

PTMAP_get_req

This subroutine gets the response to a specific request. If the call is successful, Point Management returns the following:

- Pointers to the Shopping List and point associated with the request;

- A status;
- A value.

If no response is available for the specified request, this function returns COR_WARNING.

Syntax

```
int PTMAP_get_req (req_adr, sl_adr, point_adr,
                   rsp_type, rsp_stat, rsp_ptr,
                   retstat)
PTMAP_ADDR  *req_adr;
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
int         *rsp_type;
COR_STATUS  *rsp_stat;
PTM_RSP     **rsp_ptr;
COR_STATUS  *retstat;
```

Input Arguments

req_adr	Pointer to the request.

Output Arguments

sl_adr	Pointer to the Shopping List associated with this request.
Point_adr	Pointer to point associated with this request.
Rsp_type	Request type (such as, PTM_ONCHANGE or PTM_SNAPSHOT).
Rsp_stat	Status of the response: COR_SUCCESS, COR_WARNING, or COR_FAILURE.
Rsp_ptr	Pointer to a response structure.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADR_PTR_NULL
	PTMAP_REQ_ADR_NULL
	PTMAP_REQ_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warnings may be returned:
	PTMAP_NO_RSP_RCV
	PTMAP_RCV_QUE_ERR

Return Value

The contents of **retstat.status** .

PTMAP_get_req_point_id

This subroutine retrieves the identifier of the point for into the buffer specified by the parameter point_id when

- There is a a valid PTMAP point request identified by `req_addr` and
- The request is outstanding.

If the buffer length identified by point_id_len is too short to hold the entire point identifier with null termination, the returned value will have been truncated.

Syntax

int PTMAP_get_req_point_id(PTMAP_ADDR *req_addr, TCHAR *point_id, int point_id_len, COR_STATUS *retstat) ;

Input Arguments

`req_addr`	Value returned from a valid PTMAP routine which added a request for a point.
`point_id_len`	Length of the buffer pointed to by point_id

Output Arguments:

`point_id`	Will receive the retrieved point ID.
`retstat`	Pointer to a COR_STATUS structure to receive any errors that might occur.

PTMAP_get_sl

This subroutine gets the responses associated with a Shopping List. It should be called following one of the **PTMAP_wait** functions. **PTMAP_get_sl** must be called once to access each response on the shopping list. For example, after sending a Shopping List containing four "Snapshot" requests and waiting for the responses, the application should call **PTMAP_get_sl** four times to get the responses. If the application calls **PTMAP_get_sl** a fifth time, COR_WARNING is returned.

Syntax

```
int PTMAP_get_sl (sl_adr, req_adr, point_adr,
                  rsp_type, rsp_stat, rsp_ptr,
```

```
                    retstat)
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *req_adr;
PTMAP_ADDR  *point_adr;
int         *rsp_type;
COR_STATUS  *rsp_stat;
PTM_RSP     **rsp_ptr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	The Shopping List used to send the requests that are to be processed.

Output Arguments

req_adr	Pointer to the request for which the response was generated..
point_adr	A pointer to the point for the request
rsp_type	The type of response (such as, PTM_ONCHANGE or PTM_SNAPSHOT).
rsp_stat	Status of the response: COR_SUCCESS, COR_WARNING, or COR_FAILURE.
rsp_ptr	Pointer to a response structure.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warnings may be returned:
	PTMAP_NO_RSP_RCV
	PTMAP_RCV_QUE_ERR

Return Value

The contents of **retstat.status** .

PTMAP_get_sl_point

This subroutine gets the response to a request for a specific point on a Shopping List. The point and Shopping List are specified by the application. This function must be called once for each outstanding response. If no response is available for the specified request, this function returns COR_WARNING.

Syntax

```
int PTMAP_get_sl_point (sl_adr, point_adr, req_adr,
                        rsp_type, rsp_stat, rsp_ptr,
                        retstat)
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
PTMAP_ADDR  *req_adr;
int         *rsp_type;
COR_STATUS  *rsp_stat;
PTM_RSP     **rsp_ptr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	Pointer to the Shopping List.
point_adr	A pointer to the point for the request

Output Arguments

req_adr	Pointer to the request for which the response was generated..
rsp_type	The type of response (such as, PTM_ONCHANGE or PTM_SNAPSHOT).
rsp_stat	Status of the response: COR_SUCCESS, COR_WARNING, or COR_FAILURE.
rsp_ptr	Pointer to a response structure.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warnings may be returned:
	PTMAP_NO_RSP_RCV
	PTMAP_RCV_QUE_ERR

Return Value

The contents of **retstat.status** .

PTMAP_get_struct_components

PTMAP_get_struct_components

This subroutine provides the ability to get the information regarding a structure point type that is necessary to decompose the raw point data into the various structure components. When a response for a structure point is received, the data for all of the components are packed into a single value in the response message. By calling **PTMAP_get_struct_components** , you can get information regarding the order, type, size, and offset of each component within the point value.

This function only provides the structure layout; it does not process the data in the point value.

Syntax

```
int PTMAP_get_struct_components (point_type_id,
                                 struct_details,
                                 length, comp_qty,
                                 retstat)
                     *point_type_id;
STRUCT_DETAILS_PTR   struct_details;
COR_U2               *length;
COR_U2               *comp_qty;
COR_STATUS           *retstat;
```

Input Arguments

point_type_id	The Point Type ID of a structure whose details are to be returned.
struct_details	A pointer to the record structure STRUCT_DETAILS. This will be populated with the list of components and their information. If you pass a NULL in this pointer, only the length and total number of components are returned.

Output Arguments

length	The overall length, in bytes, of the STRUCTURE, including any filler needed for data alignment.
comp_qty	The total number of distinct components that are defined for the STRUCTURE. The STRUCT_DETAILS record is made up of arrays, and this is the maximum number of valid entries that should be used.
retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_INV_PT_TYPE
	PTMAP_TYPE_NOT_STRUCT
	PTMAP_COMP_INVALID
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

Structure Details Data Structure (STRUCT_DETAILS)

The Structure Details data structure is:

```
typedef struct details
{
    TCHAR comp_name[COMP_NAME_LEN + 1];
    TCHAR comp_pt_type[POINT_TYPE_ID_LEN + 1];
    COR_I4 comp_offset;
    COR_U2 comp_elem;
    COR_U2 comp_data_type;
    COR_U2 comp_data_length;
} *STRUCT_DETAILS_PTR;
```

PTMAP_get_type

This subroutine gets the configured data type for the point from the POINT_TYPE configuration file.

Syntax

```
int PTMAP_get_type (point_adr, data_type, retstat)
PTMAP_ADDR      *point_adr;
PTM_DATA_TYPE   *data_type;
COR_STATUS      *retstat;
```

Input Arguments

Point_addr	A point ID created by a call to **PTMAP_add_point** .

Output Arguments

Data_type	The data type of the point ID.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADR_PTR_NULL
	PTMAP_PT_ADR_NULL
	PTMAP_PT_ADR_NOTF

	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_initiate

This subroutine initiates Point Management services. The application must supply an event flag that is used to communicate the completion of Point Management Requests.

Syntax

```
int PTMAP_initiate (event_flag, retstat)
COR_I4      event_flag;
COR_STATUS  *retstat;
```

Input Arguments

Event_flag	The event flag used to communicate completion of Point Management requests.

Output Arguments

Retstat	Pointer to status structure. Structure contains IPC errors on failure.

Return Value

The contents of **retstat.status** .

PTMAP_modify_setpoint

This subroutine modifies an existing "Setpoint" request. The application must specify the request and the new point value. Following a call to this function the application must call a **PTMAP_send** function to initiate the download to the device.

Syntax

```
PTMAP_modify_setpoint(point_value, req_adr, retstat)
PTM_DATA    *point_value;
PTMAP_ADDR  *req_adr;
COR_STATUS  *retstat;
```

Input Arguments

Point_value	A pointer to a PTM_DATA structure containing the point value.
req_adr	Identifier used to reference this request.

Output Arguments

Retstat	Pointer to status structure.

Return Value

The contents of **retstat.status** .

PTMAP_modifysetpoint_chgapproval

This subroutine modifies an existing **setpoint** request requiring change approval. The application must specify the request and the new point value. Following a call to this function, the application must call a `PTMAP_send` function to initiate the download to the device.

Syntax

```
int PTMAP_modify_setpoint_chgapproval  ( point_adr,
                                         point_value,
                                         req_adr,
                                         changeapproval_obj,
                                         retstat )
PTMAP_ADDR         *point_adr;
PTM_DATA           *point_value;
PTMAP_ADDR         *req_adr;
ChangeapprovalInfo *changeapproval_obj;
COR_STATUS  *retstat;
```

Input Arguments

`point_adr`	A Point ID created by a call to PTMAP_add_point.
`point_value`	A pointer to a PTM_DATA structure containing the point value.
`changeapproval_obj`	A pointer to the change approval information for the setpoint operation. If there is no change approval data, this should be NULL.

Output Arguments

`req_adr`	Identifier used to reference this request.
`Retstat`	Pointer to status structure. The following warnings may be returned. PTMAP_REQ_NOT_ENBL PTMAP_REQ_CUR_OUT PTMAP_REQ_NOT_FOUND

	The following errors may also be returned: PTMAP_CHANGEAPPROVAL_SAME_USERID PTM_BAD_PERFORMUSRID_PASSWORD PTM_BAD_VERIFYUSRID_PASSWORD PTM_INVAL_CHANGEAPPROVAL_LEVEL PTM_INVAL_CHANGEAPPROVAL_RESOURCESETPOINT PTM_INVAL_PERFORMUSERID PTM_INVAL_VERIFYUSERID PTM_NO_PERFORMUSERID PTM_NO_PERFORM_PRIV PTM_NO_VERIFYUSERID PTM_NO_VERIFY_PRIV PTM_PERFORMER_PASSWORD_EXPIRED PTM_POINT_HASCHANGEAPPROVAL PTM_VERIFIER_PASSWORD_EXPIRED

Return Value

The contents of **retstat.status** .

PTMAP_remove_point

Before calling this function, the application must ensure that no responses to outstanding requests exist for this point. For example, if the application used this point in an "On-Change" request, the application must cancel that request before removing the point since "On-Change" requests remain outstanding until they are canceled.

In the case of "Snapshot" and other one-time requests, the request is no longer outstanding when Point Management has sent a response to an application's request for a point value and the application has accessed the response. In addition, when the point is removed, all requests associated with that point are deleted. (Standing and one-time requests are described in the next section.)

Name**: PTMAP_remove_point**

Description

Remove a point from the local data store.

If requests are deleted from shopping lists, a warning is returned to the application.

Syntax

```
int PTMAP_remove_point (point_addr, retstat)
PTMAP_ADDR  *point_addr;
COR_STATUS  *retstat;
```

Input Arguments

Point_addr	Pointer to point to be removed.

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTM_REM_OUTST_REQ

	PTM_PTM_PT_NOTF
	The following warning may be returned:
	PTM_RSP_DELETED

Return Value

The contents of **retstat.status** .

PTMAP_remove_sl

This subroutine deletes a Shopping List from the application. If any responses are outstanding for requests on the Shopping List, PTMAP returns an error. If no responses are outstanding, PTMAP deletes each request that was added to the Shopping List and returns the number of requests that were deleted in the "err_ref" field of the status structure.

Syntax

```
int PTMAP_remove_sl (sl_addr, retstat)
PTMAP_ADDR  *sl_addr;
COR_STATUS  *retstat;
```

Input Arguments

sl_addr	Pointer to Shopping List.

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTM_REM_OUTST_REQ
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warning may be returned:
	PTMAP_RSP_DELETED

Return Value

The contents of **retstat.status** .

PTMAP_resume

This subroutine resumes receiving responses from the PTMRP after a call to **PTMAP_suspend** .

Syntax

```
int PTMAP_resume (retstat)
COR_STATUS  *retstat;
```

Input Arguments

None.

Output Arguments

Retstat	Pointer to status structure.

Return Value

The contents of **retstat.status** .

PTMAP_rev_eu_conv

This subroutine converts a point value from engineering units to the raw value. This function is necessary for those applications that accept (or compute) a point value in engineering units format and need to download that point to a PLC as a RAW value. This is the normal scenario for points that have engineering units conversion. Point values are sent to and received from point management as integers, and are converted to or from engineering units (floating point values) by the application.

If the point is configured with CUSTOM conversion, you must configure a Reverse expression for the point.

If the point is configured with LINEAR conversion, a Reverse expression is automatically created for the point.

Synopsis

```
int PTMAP_rev_eu_conv ( point_id, eu_value, result,
                        ret_stat )
char        *point_id;
double      eu_value;
PTM_DATA    **result;
COR_STATUS  *ret_stat;
```

Input Arguments

point_id	Point identifier for conversion.
eu_value	The floating-point value for conversion.

Output Arguments

Result	A pointer to the address of a PTM_DATA struct that will receive the result of the computation.
	For example, **result = PTMAP_alloc_eu_data** .
eu_label	Engineering units string from configuration data.
Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_INVAL_POINT_ID
	PTMAP_HAS_NOT_EU_CONV

Return Value

The contents of **retstat.status** .

PTMAP_send_all

This subroutine sends all requests on all existing Shopping Lists to the appropriate PTMRP. If a request is already outstanding or disabled, the request is ignored. New requests are now flagged as "outstanding." If no requests are sent, a warning is returned. PTMAP returns control to the application after the response or group of responses for the point have been received.

Syntax

```
int PTMAP_send_all (retstat)
COR_STATUS  *retstat;
```

Input Arguments

None.

Output Arguments

Retstat	Pointer to status structure. The following warnings may be returned (see Appendix A for an explanation of this code):
	PTMAP_NO_REQ_SENT

	PTMAP_REQ_NOT_FOUND

Return Value

The contents of **retstat.status** .

PTMAP_send_point

ThIS subroutines sends all requests for a specified point to the appropriate PTMRP. If a request is already outstanding or disabled, the request is ignored. New requests are now flagged as "outstanding." If no requests are sent, a warning is returned. PTMAP returns control to the application after the response or group of responses for the point have been received.

Syntax

```
int PTMAP_send_point (point_adr, retstat)
PTMAP_ADDR  *point_adr;
COR_STATUS  *retstat;
```

Input Arguments

point_adr	A Point ID created by a call to **PTMAP_add_point** .

Output Arguments

Retstat	Pointer to status structure. The following warnings may be returned (see Appendix A for an explanation of this code):
	PTMAP_NO_REQ_SENT
	PTMAP_REQ_NOT_FOUND

Return Value

The contents of **retstat.status** .

PTMAP_send_req

This subroutine sends a specific request to PTMRP. If the request is already outstanding or disabled, a warning is returned. PTMAP returns control to the application after the response or group of responses for the request have been received.

Syntax

```
int PTMAP_send_req (req_adr, retstat)
PTMAP_ADDR  *req_adr;
COR_STATUS  *retstat;
```

Input Arguments

req_adr	Identifier used to reference this request.

Output Arguments

Retstat	Pointer to status structure. The following warnings may be returned (see Appendix A for an explanation of this code):
	PTMAP_REQ_NOT_ENBL
	PTMAP_REQ_CUR_OUT
	PTMAP_REQ_NOT_FOUND

Return Value

The contents of **retstat.status** .

PTMAP_send_sl

This subroutine sends all requests on a specified Shopping List request to the appropriate PTMRP. If a request is already outstanding or disabled, the request is ignored. New requests are now flagged as "outstanding." If no requests are sent, a warning is returned. PTMAP returns control to the application after the response or group of responses for the point have been received.

Syntax

```
int PTMAP_send_sl (sl_adr, retstat)
PTMAP_ADDR  *sl_adr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	Identifier of Shopping List.

Output Arguments

Retstat	Pointer to status structure. The following warnings may be returned (see Appendix A for an explanation of this code):

	PTMAP_NO_REQ_SENT
	PTMAP_REQ_NOT_FOUND

Return Value

The contents of **retstat.status** .

PTMAP_send_sl_point

This subroutine sends all requests on a Shopping List for a specific point to the appropriate PTMRP. If a request is already outstanding or disabled, the request is ignored. New requests are now flagged as "outstanding." If no requests are sent, a warning is returned. PTMAP returns control to the application after the response or group of responses for the point have been received.

Syntax

```
int PTMAP_send_sl_point (sl_adr, point_adr, retstat)
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	Identifier of Shopping List.
Point_adr	A point ID created by a call to PTMAP_add_point that identifies the point the requests are associated with.

Output Arguments

retstat	Pointer to status structure. The following warnings may be returned (see Appendix A for an explanation of this code):
	PTMAP_NO_REQ_SENT
	PTMAP_REQ_NOT_FOUND

Return Value

The contents of **retstat.status** .

PTMAP_set_all

This subroutine enables or disables all requests on all Shopping Lists.

Syntax

```
int PTMAP_set_all (enabled_state, retstat)
COR_BOOLEAN enabled_state;
COR_STATUS  *retstat;
```

Input Arguments

enabled_state	New state for the request.
	Either TRUE (enabled) or FALSE (DISABLED).

Output Arguments

retstat	Pointer to status structure.

Return Value

The contents of **retstat.status** .

PTMAP_set_point

This subroutine enables or disables a point that has been added to the local data store.

Syntax

```
int PTMAP_set_point (point_adr, enabled_state,
                     retstat)
PTMAP_ADDR  *point_adr;
COR_BOOLEAN enabled_state;
COR_STATUS  *retstat;
```

Input Arguments

point_adr	Pointer to point.
enabled_state	New state for the request.
	Either TRUE (enabled) or FALSE (DISABLED).

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL

	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_set_req

This subroutine disables or enables an existing request. When a request is disabled, it is ignored if it is part of a Shopping List that is sent to Point Management.

Syntax

```
int PTMAP_set_req (req_adr, enabled_state, retstat)
PTMAP_ADDR  *req_adr;
COR_BOOLEAN enabled state;
COR_STATUS  *retstat;
```

Input Arguments

req_adr	Identifier used to reference this request.
enabled_state	New state for the request.
	Either TRUE (enabled) or FALSE (DISABLED).

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_set_sl

This subroutine disables or enables all requests in a Shopping List. This function is useful when an application needs to disable all but a few points. All points can be disabled and then a few points can be selectively enabled using **PTMAP_set_req** .

Syntax

```
int PTMAP_set_sl (sl_adr, enabled_state,  retstat)
PTMAP_ADDR  *sl_adr;
COR_BOOLEAN enabled_state;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	Pointer to the Shopping List.
enabled_state	New state for the request.
	Either TRUE (enabled) or FALSE (DISABLED).

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_set_sl_point

This subroutine enables or disables all requests for a specified point on a Shopping List.

Syntax

```
int PTMAP_set_sl_point (sl_adr, point_adr,
                        enabled_state, retstat)
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
COR_BOOLEAN enabled_state;
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	Pointer to the Shopping List.
point_adr	Pointer to point created by a call to **PTMAP_add_point** ..
enabled_state	New state for the request.
	Either TRUE (enabled) or FALSE (DISABLED).

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_PT_ADR_NULL
	PTMAP_PT_ADDR_NOTF
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADDR_NOTF
	PTMAP_SEQ_NUM_MISMATCH

Return Value

The contents of **retstat.status** .

PTMAP_suspend

This subroutine suspends all requests that have been made to Point Management. While requests have been suspended, no responses are sent to the application by the PTMRP.

Syntax

```
int PTMAP_suspend (retstat)
COR_STATUS  *retstat;
```

Input Arguments

None.

Output Arguments

Retstat	Pointer to status structure.

Return Value

The contents of **retstat.status** .

PTMAP_terminate

This subroutine terminates Point Management services.

Syntax

```
int PTMAP_terminate (retstat)
COR_STATUS  *retstat;
```

Input Arguments

None.

Output Arguments

Retstat	Pointer to status structure. Structure contains IPC errors on failure.

Return Value

The contents of **retstat.status** .

PTMAP_wait_all

This subroutine waits for responses to any outstanding request. Using the **wait_flag** argument, the application specifies whether PTMAP should wait for either all or any responses. PTMAP returns control to the application after the response or group of responses have been received.

Syntax

```
int PTMAP_wait_all (wait_flag, retstat)
int         wait_flag;
COR_STATUS  *retstat;
```

Input Arguments

wait_flag	Either PTM_NEXT (wait for the next response) or PTM_ALL (wait until responses have been made for all outstanding requests for that Shopping List). A warning is returned if no outstanding responses exist.

Output Arguments

retstat	Pointer to status structure. The following warning may be returned (see Appendix A for an explanation of this code):
	PTMAP_REQ_NOT_OUT

Return Value

The contents of **retstat.status** .

PTMAP_wait_point

This subroutine waits for a response from Point Management for a specified point. The application may specify whether to wait for the next response or for all outstanding responses for the point. PTMAP returns control to the application after the response or group of responses for the point have been received.

Syntax

```
int PTMAP_wait_point (point_adr, wait_flag, retstat)
PTMAP_ADDR  *point_adr;
int         wait_flag;
COR_STATUS  *retstat;
```

Input Arguments

point_adr	A Point ID created by a call to **PTMAP_add_point** .
wait_flag	Either PTM_NEXT (wait for the next response) or PTM_ALL (wait until responses have been made for all outstanding requests for that point).

Output Arguments

retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_PT_ADR_NULL
	PTMAP_PT_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warning may be returned:
	PTMAP_REQ_NOT_OUT

Return Value

The contents of **retstat.status** .

PTMAP_wait_req

This subroutine waits for the Point Management response for a specified request. PTMAP maintains control until the response has been received, then returns control to the application.

Syntax

```
int PTMAP_wait_req (req_adr, retstat)
PTMAP_ADDR  *req_adr;
COR_STATUS  *retstat;
```

Input Arguments

req_adr	Identifier used to reference this request.

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_REQ_ADR_NULL
	PTMAP_REQ_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warning may be returned:
	PTMAP_REQ_NOT_OUT

Return Value

The contents of **retstat.status** .

PTMAP_wait_sl

This subroutine waits for a response to a Shopping List request. The application may specify whether to wait for all responses to the requests in the Shopping List, or any response to a request in the Shopping List. PTMAP returns control to the application after the response or group of responses for requests in the specified Shopping List have been returned.

Syntax

```
int PTMAP_wait_sl (sl_adr, wait_flag,  retstat)
PTMAP_ADDR  *sl_adr;
int         wait_flag;
COR_STATUS  *retstat;
```

Input Arguments

req_adr	Identifier used to reference this request.
wait_flag	Either PTM_NEXT (wait for the next response) or PTM_ALL (wait until responses have been made for all outstanding requests for that Shopping List).

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warning may be returned:
	PTMAP_REQ_NOT_OUT

Return Value

The contents of **retstat.status** .

PTMAP_wait_sl_point

This subroutine waits for a response to a request for a point in a Shopping List. The application may specify whether to wait for all or any responses to the requests for the point in the Shopping List. PTMAP returns control to the application after the response or group of responses for requests in the specified Shopping List have been returned.

Syntax

```
int PTMAP_wait_sl_point (sl_adr, point_adr,
                         wait_flag, retstat)
PTMAP_ADDR  *sl_adr;
PTMAP_ADDR  *point_adr;
int         wait_flag;
```

```
COR_STATUS  *retstat;
```

Input Arguments

sl_adr	Identifier used to reference this request.
point_adr	A Point ID created by a call to **PTMAP_add_point** .
wait_flag	Either PTM_NEXT (wait for the next response) or PTM_ALL (wait until responses have been made for all outstanding requests for that Shopping List).

Output Arguments

Retstat	Pointer to status structure. The following errors may be returned (see Appendix A for an explanation of this code):
	PTMAP_ADDR_PTR_NULL
	PTMAP_SL_ADR_NULL
	PTMAP_SL_ADR_NOTF
	PTMAP_SEQ_NUM_MISMATCH
	The following warning may be returned:
	PTMAP_REQ_NOT_OUT

Return Value

The contents of **retstat.status** .

PTMAP Data Macros

The following macros can be used to access point values. The first argument of each macro is a pointer to a PTM_DATA structure. The second argument specifies the element number.

BOOL_VAL(x,y)	Access a BOOLEAN value.
INT_1_VAL(x,y)	Access a 1 byte INTEGER value.
INT_2_VAL(x,y)	Access a 2 byte INTEGER value.
INT_4_VAL(x,y)	Access a 4 byte INTEGER value.
UINT_1_VAL(x,y)	Access a 1 byte UNSIGNED INTEGER value.
UINT_2_VAL(x,y)	Access a 2 byte UNSIGNED INTEGER value.
UINT_4_VAL(x,y)	Access a 4 byte UNSIGNED INTEGER value.
FLTPT_VAL(x,y)	Access a FLOATING POINT value.
BITSTR_PTR(x)	Access a BITSTRING value.

Macros are provided which may be used to access OCTET_STRING and CHAR_STRING data types. Each macro can be used to copy one element of the PTM_DATA structure to or from a memory location. The length of the element copied is determined from the PTM_DATA structure.

The following macros will copy an element of the PTM_DATA structure to a location in memory. In these macros, **x** is a pointer to memory, **y** is a pointer to a PTM_DATA structure, and **z** is the element in y which is to be copied.

- GET_OCTSTR(x,y,z)
- **GET_CHRSTR(x,y,z)**

The following macros will copy data from a location in memory to an element of the PTM_DATA structure. In these macros, **x** is a pointer to a PTM_DATA structure, **y** is the element in **x** to which data is to be copied, and **z** is a pointer to memory.

- **SET_OCTSTR(x,y,z)**
- SET_CHRSTR(x,y,z)

Macros are provided which may be used to access BITSTRING data types. One macro can be used to get the value of a single bit in BITSTRING data in a PTM_DATA structure and one macro can be used to set a single bit in BITSTRING data in a PTM_DATA structure.

The following macro will return the value of a bit in a BITSTRING. In this macro, **x** is a pointer to a PTM_DATA structure, and **y** is the number of the bit to be returned. The first bit in the bitstring is bit 0. This macro returns a value of 0 or 1.

- GETBITVAL(x,y)

The following macro will set a single bit in a BITSTRING data structure. In this macro, **x** is a pointer to a PTM_DATA structure, **y** is the number of the bit to be set, and **z** is the value to set the bit. The first bit in the bitstring is bit 0.

- SETBITVAL(x,y,z)

Point Management API General Subroutines

Point Management API General Subroutines

The general subroutines include:

- Cor_event_waitfr
- Wait For An Event Flag To Be Set

- cor_logstatus
- Write error status information to the next available record
- cor_setstatus
- Write values to the cor_status structure.
- Cor_sleep
- Suspend Process Temporarily
- Cor_vsetstatus
- Write values and a formatted message to the cor_status structure.
- Ipc_deactivate
- Deactivate Port
- Ipc_register
- Register With IPC
- Lib_get_ef
- Assign An Event Flag

cor_event_waitfr

Use this subroutine to wait for the specified event flag to be set. Return is made with COR_SUCCESS when the flag is set. If the wait fails, return is made with COR_FAILURE.

Syntax

```
COR_I4 cor_event_waitfr(event_flag, ret_stat)
unsigned int event_flag;
COR_STATUS *ret_stat;
```

Input Arguments

event_flag	The event flag to be tested.

Output Arguments

retstat	Pointer to status structure.

Return Value

Either COR_SUCCESS or COR_FAILURE.

If COR_FAILURE is returned, an error code is reported in **retstat.err_code** .

cor_logstatus

Use this function to write error status information to the next available record in the CIMPLICITY status log (COR_STATUS.LOG).

Syntax

```
void cor_logstatus( const TCHAR *proc,
    COR_BOOLEAN severe,
    COR_STATUS *status )
```

Input Arguments

`proc`	The name of the calling procedure where the error is detected.
`severe`	Is situation severe error (FALSE, TRUE, FATAL). A value of FATAL will cause your application to terminate after the error message has been logged.
`status`	Pointer to the status block (COR_STATUS structure) containing error info.

Output Arguments

None

Return Value

None

The value is sent to the status log, which is viewed through the Log Viewer application. The err_reported *(page 379)* field will be updated to indicate that this error has been logged.

cor_setstatus

Use this function to write values to the cor_status structure.

Syntax

```
COR_STATUS *cor_setstatus( int result, int source, int code, int ref,
      const TCHAR * msg, COR_STATUS *status)
```

Input Arguments

`result`	COR_WARNING, COR_SUCCESS, or COR_FAILURE: 0 - Error detected - non fatal. 1 - No error, all errors corrected. 2 - Fatal error detected.
`source`	Error source; the code that identifies the source subsystem.
`code`	Error code. See cor_stat.h.
`ref`	Error reference , used to distinguish the difference between two errors with the same error code.

`msg`	Error message.

Output Arguments

`status`	Pointer to the status block (COR_STATUS structure) that is set with input argument values.

Return Value

COR_STATUS

The value is sent to the status log, which is viewed through the Log Viewer application.

cor_sleep - Suspend Process Temporarily

Use this subroutine in place of the **Sleep** or **SleepEx** subroutines.

Syntax

```
int cor_sleep (secs)
int secs;
```

Input Arguments

secs	The number of seconds to sleep.

Output Arguments

None.

Return Value

None.

cor_vsetstatus

Use this function to write values and a formatted message to the cor_status structure.

Syntax

```
COR_STATUS* __cdecl cor_vsetstatus(int result, int source, int code, int
 ref,
                const TCHAR * msg, COR_STATUS* status)
```

Input Arguments

`result`	COR_WARNING, COR_SUCCESS, or COR_FAILURE: 0 - Error detected - non fatal. 1 - No error, all errors corrected. 2 - Fatal error detected.
`source`	Error source; the code that identifies the source subsystem.
`code`	Error code. See cor_stat.h.
`ref`	Error reference , used to distinguish the difference between two errors with the same error code.
`msg`	Error message. **Note:** The message can contain format arguments and format specifiers to control how the message will appear if output to the status log by cor_logstatus *(page 444)*.

Output Arguments

`status`	Pointer to the status block (COR_STATUS structure) that is set with input argument values.

Return Value

COR_STATUS

ipc_deactivate

Use this subroutine to terminate all activities associated with the IPC. If any RR messages are outstanding for the process that executes **ipc_deactivate** , a message is transmitted on its behalf. No further datagram messages can be sent to or from a process that executes this service.

Syntax

```
ipc_deactivate( retstat, port_index);
COR_STATUS *retstat;
int        port_index;
```

Input Arguments

point_index	The port index returned by **ipc_register** .

Output Arguments

Retstat	Pointer to status structure.

Return Value

Either COR_SUCCESS or COR_FAILURE.

If COR_FAILURE is returned, an error code is reported in **retstat.err_code** .

ipc_register

Use this subroutine to initialize IPC functions for datagram and logical link communications and register with the IPC router process.

ipc_register must be called prior to using any other communications functions besides **ipc_dg_alloc** or **ipc_dg_free** . Following successful execution of this function, an application can start sending and receiving datagram messages or establish logical link communications.

Syntax

```
int ipc_register ( retstat, port_index, object_name,
                   maxlnk, bufsiz)
COR_STATUS *retstat;
int        *port_index;
char       *object_name;
int        maxlnk;
int        bufsiz;
```

Input Arguments

object_name	The name under which the process registers. The **object_name** in combination with the node name defines the physical address of the process. All other processes address the process with this name. Maximum length is 10 characters.
Maxlnk	The maximum number of logical links this process can request. Each datagram port and each logical link connection counts as one link.
Bufsiz	The maximum message size used by this process. **ipc_dg_alloc** may be used to determine this size. The maximum is MAXMSGSIZ, which is defined in **netcom.h** .

Output Arguments

port_index	Identifier for port index to be used in datagram functions.
Retstat	Pointer to status structure. When the function is not successful, **retstat.err_code** contains one of the following values:

	103	IPC_ERR_BUFTOOBIG - requested buffer too big
	108	IPC_ERR_MAXLNKVIO 108 - max link violation
	109	IPC_ERR_MBXASGN - mailbox assignment failed
	110	IPC_ERR_RTRLNKFAI - router link attempt failed
	111	IPC_ERR_INTERNDAT - internal data failure

Return Value

Either COR_SUCCESS, COR_WARNING, or COR_FAILURE.

If the function returns anything other than COR_SUCCESS, additional error information can be found in **retstat.err_msg** and **retstat.err_code** .

lib_get_ef

Use this subroutine to allocate an event flag.

Syntax

```
COR_I4 lib_get_ef (&event_flag)
int event_flag;
```

Input Arguments

event_flag	The event flag to be allocated.

Output Arguments

None.

Return Value

COR_SUCCESS or COR_FAILURE.

If COR_FAILURE is returned, an error code is reported in **retstat.err_code** .

Point Management Configuration Files

Point Management Configuration Files

Point Management configuration data defines the data points for the system and the way in which Point Management handles individual points. Configuration data controls the following functions:

Alarm Generation	When to generate alarms, and the content of the alarm message.
Engineering Units Conversion	Conversion of raw data based on a configured expression.
Derived Point Values	Definition of derived point expressions.

Configuration data is entered on-line using configuration transactions covered in the CIMPLICITY Base System User's documentation. This is true except for the POINT_TYPE configuration file, which is supplied containing default data for the system; and the PTMGMT configuration file, which is set up by Site Configuration.

The following sections detail the Point Management configuration data and describe its use. In developing applications, you can refer to this section for definitions of the configuration data that Point Management has access to.

You can write applications that access any of the points defined through the CIMPLICITY System configuration functions. Keep in mind, however, that the system must always be updated with new or modified configuration data before your application will be able to access the new configuration. You must be sure that your Point Management application is stopped and restarted when the system is updated.

Point Management Parameters File (PTMGMT)

Point Management Parameters File (PTMGMT)

The Point Management Parameters file identifies the Point Management resident processes and the maximum number of application processes that can communication with them.

Record Type	PTMGMT
Filenames	**ptmgmt.idt, ptmgmt.dat, ptmgmt.idx**
Edit Location	**%SITE_ROOT%\master**

PTMGMT Field Definitions

PTMGMT Field Definitions

Records in this file contain the following fields:

- ptmgmt_process_id
- ptmgmt_ipc_que_siz

ptmgmt_process_id

Maximum Field Length	256 characters
Definition	An identifier for the PTMRP.

ptmgmt_ipc_que_siz

Maximum Field Length:	Double-precision integer
Definition	Maximum number of processes that can communicate simultaneously with the **ptmgmt_process_id** .

PTMGMT Sample Configuration File

An example of the PTMGMT configuration file is shown below:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: PTMGMT POINT MANAGEMENT PARAMETERS
*
*   0 PTMGMT_PROCESS_ID    Point Management Process ID
*   1 ptmgmt_ipc_que_siz   IPC DG Queue size
*
MASTER_PTM0_RP|20
```

Point Type File (POINT_TYPE)

Point Type File (POINT_TYPE)

The Point Type file defines the data types and lengths supported by CIMPLICITY software. All points identified must reference one of the types in this file. Point types are defined by Point Management and cannot be added by applications. The default point types shown below are supplied with the system.

Record Type	POINT_TYPE
Filenames	**point_type.idt, point_type.dat, point_type.idx**
Edit Locations	**%SITE_ROOT%\master**

POINT_TYPE Field Definitions

POINT_TYPE Field Definitions

Records in this file contain the following fields:

- data_length
- data_type
- point_type_id

data_length

Maximum Field Length	Integer

Definition	Maximum number of characters for the point type.

data_type

Maximum Field Length	Integer
Definition	Integer value associated with **point_type** . (See second column below.)

	Point Type (Internal)	Data Type	point_type_id
	BOOLEAN	**0**	BOOL
	BITSTRING	**1**	BYTE, DWORD, WORD
	OCTETSTRING	**2**	APPLICATION
	CHARSTRING	**3**	STRING
	UNSIGNED INTEGER*1	**4**	USINT
	UNSIGNED INTEGER*2	**5**	UINT
	UNSIGNED INTEGER*4	**6**	UDINT
	INTEGER*1	**7**	SINT
	INTEGER*2	**8**	INT
	INTEGER*2	**9**	DINT
	4-BYTE FLOATING POINT	**10**	REAL

Note: Point Management treats floating point values as double precision values.

point_type_id

Maximum Field Length	16 characters
Definition	Unique point type identifier. Default point types are shown in the first column of the sample **.idt** file below.

POINT_TYPE Sample Configuration File

The default point management point types (in their default lengths) are given in the following example of an ASCII file used to provide point type configuration data:

```
|-* IDT file generated by IDTPOP utility v1.0
* RECORD: POINT_TYPE POINT FORMAT DEFINITIONS
*
*   0 POINT_TYPE_ID     Key to Point Type Record
*   1 data_type         Point's Target Data Type
*   2 data_length       Number of data_type elements
```

```
*
3D_BCD|5|0
4D_BCD|5|0
4STATE|4|0
BOOL|0|0
BYTE|1|8
DINT|9|0
DWORD|1|32
INT|8|0
MMS_EVE_DIAG|2|6
OCTETSTRING_2|2|2
REAL|10|0
RECIPE|8|0
S6X_CLOCK|2|12
S6_CLOCK|2|6
SINT|7|0
SPC_ATT_1|2|2
SPC_ATT_2|2|4
SPC_ATT_4|2|8
SPC_DEF_1|2|5
SPC_DEF_2|2|6
SPC_DEF_4|2|8
STRING|3|1
STRING_20|3|20
STRING_8|3|8
STRING_80|3|80
UDINT|6|0
UINT|5|0
USINT|4|0
WORD|1|16
```

Point File (POINT)

Point File (POINT)

The Point file defines all data points.

A point can specify either a device point or a derived point. Each point has an identifier that is unique in the system.

Record Type	POINT
Filenames	**point.idt, point.dat, point.idx**
Edit Locations	**%SITE_ROOT%\master**

POINT Field Definitions

POINT Field Definitions

Records in this file contain the following fields:

- access_filter
- access_flag
- alarm_criteria
- alarm_high
- alarm_id
- alarm_low
- alarm_state
- alarm_str_id
- deadband
- description
- deviation_ptid
- display_format
- elements
- eu_exists
- eu_label
- first_index
- fr_id
- last_index
- point_id
- point_state
- point_type_id
- proc_vars
- pt_origin
- range_high
- range_low
- range_state
- rate_time_
- rate_time_interval
- setpt_check_ptid
- trigger_check_ptid
- warning_high
- warning_low
- warning_state

Note:

- Alarms may be specified only for points that are integers (SINT, INT, DINT, USINT, UINT, UDINT), Boolean (BOOL), or floating decimal point (REAL).
- Alarms may not be specified for points that contain more than a single element.
- To specify an alarm for a BOOLEAN point, specify a 1 or 0 as any of the alarm limits.

access_filter

Maximum Field Length	1 character
Definition	Determines whether or not the point can be accessed by the enterprise. E = can be accessed.

access_flag

Maximum Field Length	Byte	
Definition	Code indicating the READ/WRITE attribute of the point:	
	0	Read access
	1	Write access
	2	Read/Write access
	These access constants are defined in **%BSM_ROOT %\api\include\inc_path\ptm_defs.h**	

alarm_criteria

Maximum Field Length	Byte
Definition	The method to be used for evaluating alarm conditions. Valid entries are 1 for absolute alarming, 2 for deviation alarming, or 4 for rate of change alarming.

alarm_high

Maximum Field Length	10 characters
Definition	High alarm limit (see note below). If the point value equals or exceeds this limit an alarm is generated.

alarm_id

Maximum Field Length	256 characters
Definition	The alarm ID (from ALARM_DEF.DAT) that will be used when alarms are generated.

alarm_low

Maximum Field Length	10 characters
Definition	Low alarm limit (see note below). If the point value equals or falls below this limit an alarm is generated.

alarm_state

Maximum Field Length	Byte
Definition	Alarm checking: 1 = enabled; 0 = disabled.

alarm_str_id

Maximum Field Length	Integer
Definition	An integer keyed to a record in **point_alstr.dat** .

If not 0, this value indicates the related set of range/warning/alarm messages defined in that configuration file which Point Management will send to Alarm Management when the point state changes instead of the default messages.

Deadband

Maximum Field Length	10 characters
Definition	Value defining tolerance below the upper alarm limits or above the lower alarm limit. If no deadband is defined, the alarm state of the point changes each time the alarm limit is passed (in either direction).

A deadband is useful for points whose values may fluctuate near the alarm limit. The deadband value specifies that a point in an alarm state remains in the alarm state the value is equal to the alarm limit plus the deadband value.

For example, if a high alarm limit is 100 and the deadband is 5, an alarm is generated when the point value equals 100. However, the point is considered in an alarm state until it reports a value below 95 (the alarm limit plus the deadband). If a low alarm limit is 10 and the deadband is 5, an alarm is generated when the point value equals 10. However, the point is considered in an alarm state until it reports a value above 15 (the alarm limit plus the deadband).

Description

Maximum Field Length	40 characters
Definition	Description of point.

deviation_ptid

Maximum Field Length	256 characters

Definition	The Point ID that the current point will be compared with when checking for a deviation alarm.

display_format

Maximum Field Length	10 characters
Definition	A C format string (for example, %.2f) defining the way in which the point value is to be displayed (for example, with or without leading zeros, the number of decimal places, the total number of characters.)

Elements

Maximum Field Length	Integer
Definition	Number of elements of the point. Arrays of a **point_type** are created by setting this value greater than one.

eu_exists

Maximum Field Length	Byte	
Definition	Flag defining whether or not engineering unit conversion exists of this point. It will have one of the following values:	
	1	A record having the same ID as the point record is defined in **eu_conv.dat** to provide engineering conversion specifications.
	0	No record is defined.

Note: If this field is set to 1 and **eu_conv.dat** contains an **eu_rev_exp** (reverse-engineering-units expression), the values for **warning_high, warning_low, alarm_high, alarm_low**, **range_high**, **range_low**, and **deadband** are converted using that expression. Otherwise unconverted values are used for these parameters.

eu_label

Maximum Field Length	8 characters
Definition	Engineering unit label. The label returned with the floating point number by the function that performs the conversion.

first_index

Maximum Field Length	Integer

Definition	The first data element in the array to be accessed. (all arrays are assumed to be indexed starting at 1) Must be less than or equal to the elements argument.

fr_id

Maximum Field Length	16 characters
Definition	Factory resource ID from **fr.dat** . When Point Management generates an alarm, it passes this ID to Alarm Management to indicate the origin of the alarm.

last_index

Maximum Field Length	Integer
Definition	The last data element in the array to be accessed. Must be less than or equal to the elements argument and greater than or equal to **first_index** .

point_id

Maximum Field Length	256 characters
Definition	Unique point identifier. Embedded spaces and special characters (other than $) are allowed. However, the standard format requires that the first character be alphabetic and the remaining characters be alphanumeric or underscore (no embedded spaces).
	If a non-standard Point ID is used in an expression, (for example, for a derived point) the ID must be enclosed in quotation marks.

point_state

Maximum Field Length	Byte
Definition	1 = enabled; 0 = disabled.
	If disabled, the point is not polled by Device Communications.

point_type_id

Maximum Field Length	16 characters
Definition	Point type from **point_type.dat**.

proc_vars

Maximum Field Length	Integer
Definition	The number of process variables represented by this point. Must be set to one if elements is set to one. If elements is set to a value greater than one, this may be one (1) or a factor of elements.

pt_origin

Maximum Field Length	Byte	
Definition	Code indicating device or derived point	
	0	Derived point
	1	Device point
	2	Global point
	3	Device internal point
	4	Device always poll point
	Code 2 is used for derived points whose values may be modified by multiple processes. These origin constants are defined in **%BSM_ROOT%\api\include \inc_path\ptm_defs.h**	

range_high

Maximum Field Length	10 characters
Definition	High range limit (see note below). Point Management discards values that equal or exceed this limit.

range_low

Maximum Field Length	10 characters
Definition	Low range limit (see note below). Point Management discards values that equal or fall below this limit.

range_state

Maximum Field Length	Byte
Definition	Range checking: 1 = enabled; 0 = disabled.

*rate_time_*

Maximum Field Length	Integerunit
Definition	The units associated with **rate_time_interval** for a point using rate-of-change alarming. Valid entries are 1 for seconds, 60 for minutes, or 3600 for hours.

rate_time_interval

Maximum Field Length	Double-precision integer
Definition	The size of the sample interval for a point using rate-of-change alarming.

setpt_check_ptid

Maximum Field Length	256 characters
Definition	Also known as the safety point. If a safety point is defined for a point, and a setpoint request is made for the point, the current value of the safety point is evaluated. If the safety point evaluates to zero or the point is unavailable, the setpoint request will be denied. If the safety point evaluates to a non-zero value, the setpoint request will be permitted.

trigger_check_ptid

Maximum Field Length	256 characters
Definirtion	The Point ID used as the Availability Trigger for this point. This point is only available if its Availability Trigger is "True".

warning_high

Maximum Field Length	10 characters
Definition	High warning limit (see note below). If the point value equals or exceeds this limit a warning alarm is generated.

warning_low

Maximum Field Length	10 characters

Definition	Low warning limit (see note below). If the point value equals or falls below this limit a warning alarm is generated.

warning_state

Maximum Field Length	Byte
Definition	Warning alarm checking: 1 = enabled; 0 = disabled.

POINT Field Sample Configuration File

An example of an ASCII file that might be used to provide this configuration data is:

```
|-* point_id|point_type_id|warning_high|warning_low-
* |alarm_high|alarm_low|range_high|range_low|deadband-
* |FR_ID|alarm_id|alarm_str_id|eu_exists|range_state-
* |alarm_state|warning_state|point_state|elements-
* |first_index|last_index|pt_origin|access_flag-
* |eu_label|display_format|description
*
POINT_001|ANALOG_16|||||||SERIES_6||0|1|0|0|0|1|1|1-
|1|1|0|Furlongs|%d|Example point 1
POINT_002|ANALOG_16|||100|0||||SERIES_6|D1_RANGE|0|0-
|0|1|0|1|1|1|1|1|0||-%d|Example point 2
POINT_003|FLOATING|||||||||0|0|0|0|0|1|1|0|0|0|0|-
|%f| Example point 3
```

Device Point File (DEVICE_POINT)

Device Point File (DEVICE_POINT)

The Device Point file contains all device specific configuration data for device points.

Record type	DEVICE_POINT
Filenames	**device_point.idt, device_point.dat, device_point.idx**
Edit Locations	**%SITE_ROOT%\master**

DEVICE_POINT Field Definitions

DEVICE_POINT Field Definitions

The file contains the following fields:

- addr

- addr_offset
- addr_type
- analog_deadband
- device_id
- flags
- point_id
- raw_data_type
- scan_point
- scan_rate
- trigger_point
- trigger_type
- trigger_value

point_id

Maximum Field Length	256 characters
Definition	Point identifier from **point.dat**.

device_id

Maximum Field Length	16 characters
Definition	The device where the point originates from **device.dat**.

addr_type

Maximum Field Length	Integer	
Definition	The full address of the device. This field contains one of the following values:	
	1	VARIABLE NAME
	2	FULLY QUALIFIED
	3	LOGICAL ADDRESS
	4	UNCONSTRAINED

Addr

Maximum Field Length	256 characters
Definition	Device dependent address specification.

addr_offset

Maximum Field Length	Integer
Definition	Offset from specified address to start of point.

trigger_type

Maximum Field Length	Integer
Definition	Type of trigger used. Valid types are:

0	NO TRIGGER
1	ONCHANGE
2	EQ
3	LT
4	GT
5	LE
6	GE

trigger_value

Maximum Field Length	16 characters
Definition	For trigger_types other than DL_NO_TRIGGER and DL_ONCHANGE, the value to be compared with the **trigger_point** value.

trigger_point

Maximum Field Length	256 characters
Definition	Point ID of the point whose value is checked using the **trigger_type** and **trigger_value**.

Note: The trigger function allows points to be read conditionally based on the value of another (trigger) point.

scan_rate

Maximum Field Length	Integer
Definition	The frequency of the scan. This is an integral multiple of the scan rate for the device.

scan_point

Maximum Field Length	Integer	
Definition	Enable/disable scanning of point. This field contains one of the following values:	
	0	UNSOLICITED (disable scanning);
	1	ONCHANGE
	2	ONSCAN (that is, according to the scan rate defined in the previous parameter) format of data as stored on the device - for example, BCD, EBCDIC, etc. used to translate from device data to internal data.
	Enter 0 if not used.	

raw_data_type

Maximum Field Length	Integer
Definition	Reserved for GE Intelligent Platforms use.

analog_deadband

Maximum Field Length	10 characters
Definition	A value used to filter out changes in the raw value of the point. The raw value must change by at least this much before the point value is updated in CIMPLICITY software.

Flags

Maximum Field Length	Byte
Definition	Defines how the point will be scanned after a setpoint. Set to 0 for normal scanning or set to 1 to scan immediately after doing the setpoint.

DEVICE_POINT Sample Configuration File

An example of an ASCII file that might be used to provide this configuration data is:

```
|-* point_id|DEVICE_ID|addr_type|addr|addr_offset|
* trigger_type|trigger_value|trigger_point|scan_rate |
* scan_point|raw_data_type
*
POINT_001|gef1_resp|3|R65|0|0|||1|1|0
POINT_002|gef1_resp|3|R66|0|0|||1|1|0
```

Derived Point File (DERIVED_POINT)

Derived Point File (DERIVED_POINT)

Derived points are points whose values are derived from other points. Each derived point has an expression that references other device or derived points. The expression, which may contain arithmetic, logical, and bitwise operations as well as some Point Management defined functions, is defined in the Derived Point configuration file.

Record type	DERIVED_POINT
Filenames	**derived_point.idt, derived_point.dat, derived_point.idx**
Edit Locations	**%SITE_ROOT%\master**

DERIVED_POINT Field Definitions

DERIVED_POINT Field Definitions

The file contains the following fields:

- calculation_type
- description
- DP_flag
- init_value
- local
- output_units
- point_expression
- point_id
- point_set_interval
- point_set_time
- process_id
- ptmgmt_process_id
- reset_point_id
- rollover_val
- trigger_point_id
- variance_value

calculation_type

Maximum Field Length	Byte	
Definition	The type of point calculation to perform.	
	0	equation point

	1	delta accumulator point
	2	value accumulator point
	3	average point
	4	maximum capture
	5	minimum capture
	6	global
	7	transition high accumulator
	8	equation with override
	9	timer/counter
	10	histogram

All of these calculation types are described in detail in subsequent sections.

Description

Maximum Field Length	40 characters
Definition	Description of the derived point.

DP_flag

Maximum Field Length:	Byte	
Definition	True if an **init**-value is specified, and the value of the DP_flag defines the search sequence for the initial value of the point. The settings are as follows:	
	0	unavailable
	1	use the **init_value** found in **derived_point.dat**
	2	use the **save_value** found in **saved_point.dat**
	3	use the **save_value** found in **saved_point.dat** ; if there is no **saved_point.dat** data for this point, then use the **init_value** found in **derived_point.dat**

init_value

Maximum Field Length	256 characters
Definition	The initial value for the point.

Local

Maximum Field Length	Byte
Definition	When TRUE this indicates that the computed point value cannot be accessed outside of the derived point process and alarms cannot be generated.

output_units

Maximum Field Length	Byte
Definition	Reserved for GE Intelligent Platforms.

point_expression

Maximum Field Length	256 characters
Definition	An expression containing point identifiers, constants, and either arithmetic or logical operators. The expression may be up to 78 characters in length.

Note: If another Point ID is used in the expression, it must be in the recognized format, or else the Point ID must be enclosed in single quotation marks – for example, **'1 - this is a point_id'** . The recognized format is an alphabetic first character followed by alphanumeric characters and/or an underscore. If a Point ID contains embedded blanks or non-alphanumeric characters (other than underscore) or does not begin with a letter, its format is not standard.

point_id

Maximum Field Length	256 characters
Definition	Point identifier from **point.dat**

point_set_interval

Maximum Field Length	8 characters
Definition	Interval for point setting. Use **hh:mm:ss** format.

point_set_time

Maximum Field Length	8 characters
Definition	Start time for point setting. Use **hh:mm:ss** format.

process_id

Maximum Field Length	256 characters
Definition	Derived point Process identifier.

ptmgmt_process_id

Maximum Field Length	256 characters
Definition	PTMRP responsible for this point from **ptmgmt.dat** .

reset_point_id

Maximum Field Length	256 characters
Definition	Point identifier from **point.dat** for the reset point upon which this point is dependent. The reset point can be any type of point; however, a Boolean type is recommended. Whenever the reset point changes, all of its dependent points are reset.

rollover_val

Maximum Field Length	Double-precision integer
Definition	Accumulator rollover value.

trigger_point_id

Maximum Field Length	256 characters
Definition	Point identifier from **point.dat** for the trigger point upon which this point is dependent.

variance_value

Maximum Field Length	Double-precision integer
Definition	The variance threshold for accumulator points and has no meaning for all other types of points.

DERIVED_POINT Sample Configuration File

An example of an ASCII file that might be used to provide this configuration data is:

```
|-* point_id|PROCESS_ID|PTMGMT_PROCESS_ID|-
* point_expression|description|local|init_value|-
* dp|flag|calcultion_type|variance_value|-
* reset_point_id|trigger_point|rollover_val|-
* ouput_units|point_set_time|point_set_internal
*
POINT_003|PTM_DP|PTM_RP|POINT_001 * 2.0||0|17|1
```

Supported Operations for point_expression Field

The **point_expression** field supports the following operations:

Arithmetic	+, -, /, *,), (.	Include any point types except Boolean and floating point.
Logical	AND, OR, XOR, NOT.	Can include only Boolean points types.
Bitwise	BAND, BXOR, BNOT.	Can contain only Boolean and integer point types.
Relational	LT, GT, EQ, LE, NE, GE.	
Functions	ALARM(), A2(), WARNING(), A1(), ALARM_HI(), AH2(), ALARM_LO(), AL2() WARNING_HI(), AH1(), WARNING_LO(), AL1(), and AL()	Return a Boolean result depending on the alarm state of the point.
	ALARM_NOT_ACKED() and ANA()	Return a Boolean result depending on whether the alarm has been acknowledged or not.
	EUCONV()	Returns the engineering units value of the point.
	(expr1) ^ (expr2)	Raises expr1 to expr2 value
		(expr1) ? (expr2) : (expr3) tests if expr2 is TRUE then expr2 else expr3
	SQR (<point_id>)	Returns the square root of the point's raw value
	ABS(expr)	Returns the absolute value of expr
	ACOS(expr)	Returns the arc cosine of expr
	ASIN(expr)	Returns the arc sine of expr
	ATAN(expr)	Returns the arc tangent of expr
	CEIL(expr)	Returns the closest integer to expr that is larger
	COS(expr)	Returns the cosine of expr
	EXP(expr)	Returns the value of e raised to expr

	FLR(expr)	Returns the closest integer to expr that is smaller
	LOG(expr)	Returns the natural logarithm (base e) of expr
	LOG10(expr)	Returns the base 10 logarithm of expr
	(expr1) MAX (expr2)	Returns the maximum value of expr1 and expr2
	(expr1) MIN (expr2)	Returns the minimum value of expr1 and expr2
	(expr1) MOD (expr2)	Returns the mod value of expr1 based on expr2
	RND(expr)	Rounds expr to the nearest integer
	(expr1) SHL (expr2)	Shifts expr1 left expr2 bits
	(expr1) SHR (expr2)	Shifts expr1 right expr2 bits
	SIN(expr)	Returns the sine of expr
	TAN(expr)	Returns the tangent of expr
	TRUNC(expr)	Truncates expr to its integer value

Equation Points

Expression points are the typical derived points whose values are derived from the results of an arithmetic expression (the **point_expression** field in **derived_point.dat** .

Specifications: Expression points are indicated by a value of 0 in the **calculation_type** field.

Accumulator Points

The accumulator point keeps a running total of changes in the source point values. The accumulator can be either a signed integer, an unsigned integer or a floating point data type and can have one of two forms of operation:

Delta Accumulator Points

A Delta Accumulator point keeps a running total of the delta between the previous value of the results an arithmetic expression (the **point_expression** field in **derived_point.dat**) and the current value. The delta accumulator point may be a signed or unsigned integer or a floating point data type.

The value for a delta accumulator is initialized to zero, or if a Startup Condition is specified it is initialized according to the Startup Condition. The first update of the accumulator occurs after the second update of the **point_expression** value. Updates continue until a Reset Condition occurs.

A Reset Condition occurs when the value in the **reset_point_id** changes. At that point, the value of the Delta Accumulator is reset to the **init_value** or if the **init_value** is not present, it is reset to zero(0). The value in the **reset_point_id** can be changed by an operator or automatically by software.

You must set a **variance_value** for a Delta Accumulator. This is the maximum acceptable delta value that can be added to the accumulator. In the event of a variance condition, the system logs a message to the Status Log. Counting continues from the new value, but the delta is not added to the accumulator.

You may set a **rollover_val** for a Delta Accumulator. This is the maximum acceptable value for the point. If you do not specify a rollover value, the data type for the source point in the Equation is used as the default Rollover value.

When a Rollover condition occurs, the accumulator point is set to an adjusted value which accounts for the data overflow and a message is logged in the Status Log.

Value Accumulator Points

A Value Accumulator point keeps a running total of the changes in an expression by adding the current value of the expression to the Value Accumulator. The Value Accumulator may be a signed or unsigned integer or a floating point data type.

The value for a Value Accumulator is initialized to zero, or if a Startup Condition is specified, it is initialized according to the Startup Condition. Updates continue until a Reset Condition occurs.

A Reset Condition occurs when the value in the **reset_point_id** changes. At that point, the value of the Value Accumulator is reset to the **init_value** or if the **init_value** is not present, it is reset to zero(0). The value in the **reset_point_id** can be changed by an operator or automatically by software.

Delta Accumulator Points

A Delta Accumulator point keeps a running total of the delta between the previous value of the results an arithmetic expression (the **point_expression** field in **derived_point.dat**) and the current value. The delta accumulator point may be a signed or unsigned integer or a floating point data type.

The value for a delta accumulator is initialized to zero, or if a Startup Condition is specified it is initialized according to the Startup Condition. The first update of the accumulator occurs after the second update of the **point_expression** value. Updates continue until a Reset Condition occurs.

A Reset Condition occurs when the value in the **reset_point_id** changes. At that point, the value of the Delta Accumulator is reset to the **init_value** or if the **init_value** is not present, it is reset to zero(0). The value in the **reset_point_id** can be changed by an operator or automatically by software.

You must set a **variance_value** for a Delta Accumulator. This is the maximum acceptable delta value that can be added to the accumulator. In the event of a variance condition, the system logs a message to the Status Log. Counting continues from the new value, but the delta is not added to the accumulator.

You may set a **rollover_val** for a Delta Accumulator. This is the maximum acceptable value for the point. If you do not specify a rollover value, the data type for the source point in the Equation is used as the default Rollover value.

When a Rollover condition occurs, the accumulator point is set to an adjusted value which accounts for the data overflow and a message is logged in the Status Log.

Average Points

The average point maintains the average value for a source point. The average point can be either a signed integer, an unsigned integer or a floating point data type. The average is calculated as an eight byte floating point data type, and the result is cast into the resultant data type as defined by the **point_type_id** of the average point. The average value is calculated as the accumulation of the deviation from the average point data divided by the number of samples taken.

```
average = average + ((source - average)/sample_count )
```

The **sample_count** value is the number of source points in the average point calculation. In the event the **sample_count** value overflows the maximum unsigned integer value, then the average point will reset and log a message with the Status Log.

The average value is maintained in the average point until a Reset condition occurs.

A Reset condition occurs when the reset point is changed causing the average point to reset. The reset causes the average point to be reset to the configured **init_value** from **derived_point.dat** and, if the **init_value** is not present, then the average point has no value until a new data value is received. The reset point could be activated by an operator or automatically by the system.

Specifications: The average point is a derived point with the **calculation_type** set to **3**.

Note: For integer type points the resultant data will rounded to the nearest integer which may result in loss of accuracy. Therefore, it is suggested that average points be floating point type.

Maximum Capture Points

The maximum capture point maintains the maximum value encountered for a point. The maximum capture point can be either a signed integer, an unsigned integer or a floating point data type.
The maximum capture point is determined by comparing the current source point value with the maximum value, and if the current value is greater than the maximum value, the current value is then

stored as the maximum capture point. The maximum value is maintained in the maximum capture point until a Reset condition occurs.

A Reset condition occurs when the reset point is changed causing the maximum capture point to reset. The reset causes the maximum capture point to be reset to the configured init_value from **derived_point.dat** and, if the **init_value** is not present, then the maximum point has no value until a new data value is received. The reset point could be activated by an operator or automatically by the system.

Specifications: The maximum capture point is a derived point with the **calculation_type** set to 4.

Minimum Capture Points

The minimum capture point maintains the minimum encountered value for a point. The minimum capture point can be a signed integer, an unsigned integer or a floating point data type. The minimum capture point is determined by comparing the current source point value with the minimum value, and if the current value is less than the minimum value, the current value is then stored as the minimum capture point. The minimum value is maintained in the minimum capture point until a Reset condition occurs.

A Reset condition occurs when the reset point is changed causing the minimum capture point to reset. The reset causes the minimum capture point to be reset to the configured **init_value** from **derived_point.dat** and, if the **init_value** is not present, then the minimum point has no value until a new data value is received. The reset point could be activated by an operator or automatically by the system.

Specifications: The minimum capture point is a derived point with the **calculation_type** set to **5**.

Global Points

Global points are derived points whose values are updated by CIMPLICITY software applications, custom applications using the Point Management API, or through standard Setpoint functions.

Specifications: The global point is a derived point with the **calculation_type** set to 6.

Transition High Accumulator Points

Transition high accumulator points accumulate the number of times the value of the expression in the point_expression field transitions from a zero to a non-zero value. During the execution of a CIMPLICITY project, a Transition High Accumulator point remains in its latest state even if the points it depends on become unavailable.

A Reset condition occurs when the reset point is updated, causing the transition high accumulator point to reset. The reset point can be activated by an operator or automatically by the system.

Determining a transition takes into consideration the calculation type of the equation and the point type of the transition high accumulator point. For example:

- If the transition high accumulator point is DINT, and the equation uses floating point arithmetic, the result of the calculation is rounded to the nearest integer. Thus, a value of 0.1 is considered to be zero, and a value of 0.6 is considered to be non-zero.
- If the transition high accumulator point is FLOAT, and the equation uses floating-point arithmetic, then a transition from 0 to 0.1 is counted as a transition from a zero to a non-zero value.

Specifications

The transition high accumulator point is a derived point with the **calculation_type** set to **7**.

Equation With Override Points

Equation with override points are similar to equation points. They use the expression you specify in the **point_expression** field to update the point's value. The expression may contain one or more device or derived Point IDs, along with constant values, arithmetic operations, logical operations, bit operations or alarm functions.

In addition, the value of an equation with override point may be updated by CIMPLICITY software applications, custom applications using the Point Management API, or through standard Setpoint functions. The changed value remains in effect until one of the source points for the expression changes and the expression is recalculated.

A Reset condition occurs when the reset point is updated. This will cause the equation with override point to be reset to the current value of its expression.

Specifications

The equation with override point is a derived point with the **calculation_type** set to **8**.

Timer/Counter Points

Timer/counter points record the following data in three array elements:

- A count of the number of times the event has occurred.
- The accumulated duration of all HIGH event occurrences, stored in seconds.
- The last event time or zero. This field will always contain the time the HIGH event occurred while the point_expression evaluates to a HIGH (non-zero) state. While the equation in the **point_expression** field evaluates to a LOW (zero) state, this field contains zero. Time is stored as the number of seconds since time 00:00:00 on January 1, 1970 GMT.

Each time the equation evaluates to a value greater than zero, the event is considered to be in its HIGH state and:

- The count field, stored in the first element of the timer/counter array is incremented by one.
- The current system time is stored in the third element of the timer/counter array.

Each time the equation evaluates to a value equal to or less than zero, or one or more source points are unavailable, the event is considered to be in its LOW state and:

- The duration field, stored in the second element, is incremented by the duration of the last event. This duration is calculated by subtracting the start time of the last event from the current system time.
- The last event time, stored in the third array element is zeroed.

A Reset condition occurs when the reset point is updated. This will cause the timer/counter array to be reset to zero or **init_value** (if it is defined)

Specifications

- The timer/counter point is a derived point with the **calculation_type** set to **9**.
- A timer/counter point must be defined with a **point_type_id** of UDINT, and 3 **elements** .
- The event described in the **point_expression** field serves as an "edge trigger" for an event.

Histogram Points

Histogram points record the frequency at which the value of a point, identified in the point_expression field, occurs within specific range intervals. This information is typically displayed graphically as a histogram.

Each time a new point sample is received, the counter for the range that includes the value in the histogram point is updated.

A histogram point must be configured as an array point. You must specify the number of elements in the array as six (6) greater than the number of range intervals you desire. The system uses the six additional array elements to maintain the following data:

- The minimum of the point values received
- The maximum of the point values received
- The total number of samples received
- The sum of the values of all samples
- The number of sample values that were less than the lower limit of the point range (underflow bucket)

- The number of sample values that were greater that the upper limit of the point range (overflow bucket)

The **display_low_lim** and **display_high_lim** fields are used to specify the lower and upper range values within which the point values are expected to occur. The range intervals are calculated based on these limits and the number of array elements specified.

A Reset condition occurs when the reset point is updated. This will cause the histogram array to be reset to zero or **init_value** (if it is defined).

Specifications

- The histogram point is a derived point with the **calculation_type** set to **10**.
- The number of **elements** in a histogram point equals the number of ranges plus six.

Engineering Unit Conversion File (EU_CONV)

Engineering Unit Conversion File (EU_CONV)

Point Management provides the capability to convert data values collected from devices into real numbers. To use this function, the application accesses a conversion expression from Point Management's Engineering Unit Conversion file. PTMRP applies that expression to the collected data value and converts the point value to a real number for display in an alarm message. In addition, the application may call a PTMAP function that accesses the configuration data, performs the conversion, and returns a floating point value to the application.

Record type	EU_CONV
Filenames	**eu_conv.idt, eu_conv.dat, eu_conv.idx**
Edit Locations	**%SITE_ROOT%\master**

EU_CONV Field Definitions

EU_CONV Field Definitions

Records in this file contain the following fields:

- description
- eu_expression
- eu_rev_exp
- high_raw_limit
- lin_conv_flag
- low_raw_limit

- point_id

Note: The **eu_expression** should take into account the decimal precision you desire. for example, an **eu_expression** of **%P/10** converts a point value of **125** to **12.0** while the expression **%P/10.0** converts the same point value (**125**) to **12.5**.

Description

Maximum Field Length	256 characters
Definition	Description of the conversion

eu_expression

Maximum Field Length	256 characters
Definition	Expression used to convert the data value. The expression may contain the following:
	Arithmetic operators - +, -, /, *
	Place holders - the point in the expression is designated by the place holder %P.
	Left and right parentheses.
	An example of a conversion expression is:
	(%P-10)*3.5+7

If another Point ID is used in the expression, it must be in the recognized format, or else the Point ID must be enclosed in single quotation marks -- for example, **'1 - this is a point id '** . The recognized format is an alphabetic first character followed by alphanumeric characters and/or an underscore. If a Point ID contains embedded blanks or non-alphanumeric characters (other than underscore) or does not begin with a letter, its format is not standard.

eu_rev_exp

Maximum Field Length	256 characters
Definition	An expression that can be used to convert a real number to an unconverted value. If no expression is provided for this field, Point Management assumes that alarm, warning, range, and deadband values are in a format that does not need conversion.

high_raw_limit

Maximum Field Length	256 characters
Definition	Reserved for GE Intelligent Platforms internal use.

lin_conv_flag

Maximum Field Length	256 characters
Definition	Reserved for GE Intelligent Platforms internal use.

low_raw_limit

Maximum Field Length	256 characters
Definition	Reserved for GE Intelligent Platforms internal use.

point_id

Maximum Field Length	256 characters
Definition	Identifier of the point.

EU_CONV Sample Configuration File

An example of an ASCII file that might be used to provide this configuration data is:

```
|-* point_id|eu_expression|description|eu_rev_exp|-
* lin_conv_flag|low_raw_limit|high_raw_limit|
*
POINT_001|(%P - 5) / 100|||0||
```

Point Alarm String File (POINT_ALSTR)

Point Alarm String File (POINT_ALSTR)

This file is configured to provide user-defined strings that replace the default Point Management strings in alarm messages generated by Point Management. These strings are substituted in place of the default strings, "Hi-2", "Lo-2", "Hi-1", "Lo-1", in alarm messages when the alarm state is requested as part of the alarm message.

Record type	POINT_ALSTR
Filenames	**point_alstr.idt, point_alstr.dat, point_alstr.idx**
Edit Locations	**%SITE_ROOT%\master**

POINT_ALSTR Field Definitions

POINT_ALSTR Field Definitions

Records in this file contain the following fields:

- alarm_hi_str
- alarm_low_str
- alarm_str_id
- warning_hi_str
- warning_lo_str

alarm_hi_str

Maximum Field Length	16 characters
Definition	**Alarm High** substitute string.

alarm_low_str

Maximum Field Length	16 characters
Definition	**Alarm Low** substitute string.

alarm_str_id

Maximum Field Length	Double-precision integer
Definition	Integer identifying the record number of the set of strings. This key is referenced by the **alarm_str_id** field in the **point.dat** configuration file. The value must be less than 100 (that is. no more than 100 defined strings) for performance reasons.

warning_hi_str

Maximum Field Length	16 characters
Definition	**Warning High** substitute string.

warning_lo_str

Maximum Field Length	16 characters

Definition	**Warning Low** substitute string.

Point Display File (POINT_DISP)

Point Display File (POINT_DISP)

This file contains information pertaining to the display of point values. It provides display limits which must be configured for points that are used to control objects with a dynamic fill attribute on screens used by the CIMPLICITY System Base product, Status Monitoring, or objects that are used in trend charts.

Record type	POINT_DISP
Filenames	**point_disp.idt, point_disp.dat, point_disp.idx**
Edit Locations	**%SITE_ROOT%\master**

POINT_DISP Field Definitions

POINT_DISP Field Definitions

Records in this file contain the following fields:

- display_lim_high
- display_lim_low
- point_id
- screen_id

display_lim_high

Maximum Field Length	10 characters
Definition	Reserved for GE Intelligent Platforms use.

display_lim_low

Maximum Field Length	10 characters
Definition	Reserved for GE Intelligent Platforms use.

point_id

Maximum Field Length	256 characters

Definition	Identifier of the point.

screen_id

Maximum Field Length	16 characters
Definition	Valid only for CIMPLICITY software. Identifies a screen to access based on point type.

Saved Points File

Point Management allows for the saving of derived point data in the event the system is shutdown. The derived point data is stored in a disk file for retrieval during the next system startup.

Specifications:

- **%LOG_PATH%** is a logical name defined in the **%SITE_ROOT%\data\log_names.cfg** file that defines the path to save and find **saved_point.stg**
- **saved_point.stg** is the file containing saved derived point data

Note: The impact on performance of the system is related to the number of points that have been identified as save points. The save point process is I/O intensive such that the points are written to a static fixed record file as they are processed from the Point Management system; thus the more points defined as save points, the greater the impact on the entire system performance.

Point Management Error Messages

Point Management Error Messages

Error messages from:

- Point Management Expression processor.
- PTMDP.
- PTMRP.
- Point Management resident process.
- PTMAP.
- Point Translation process.

Error Messages from Point Management Expression Processor

The following codes are returned by the Point Management Expression Processor.

Number	Defined Constant	Description
23500	PTEXP_NORMAL	Normal successful completion
23501	NOT_PROPER_END	Expression not properly terminated
23502	LEX_ILLEGAL_CHAR	Character not in set for lexical analyzer
23503	LEX_POINT_NOT_AVAIL	Point information not available
23504	LEX_ILLEGAL_KEYWORD	The previous symbol is not a keyword
23505	OR_LIST_NOT_OP	Next symbol should be an operator
23506	OR_LEFT_FLOAT	Left operand of OR is of type FLOAT
23507	OR_RIGHT_FLOAT	Right operand of OR is of type FLOAT
23508	OR_LIST_ILLEGAL_OP	Next symbol should be EOL,) or OR
23509	XOR_LIST_NOT_OP	Next symbol should be an operator
23510	XOR_LEFT_FLOAT	Left operand of XOR is of type FLOAT
23511	XOR_RIGHT_FLOAT	Right operand of XOR is of type FLOAT
23512	XOR_LIST_ILLEGAL_OP	Next symbol should be EOL,), OR or XOR
23513	AND_LIST_NOT_OP	Next symbol should be an operator
23514	AND_LEFT_FLOAT	Left operand of AND is of type FLOAT
23515	AND_RIGHT_FLOAT	Right operand of AND is of type FLOAT
23516	AND_LIST_ILLEGAL_OP	Next symbol should be EOL,), OR, XOR or AND
23517	EQUAL_LIST_NOT_OP	Next symbol should be an operator
23518	EQUAL_LIST_ILLEGAL_OP	EOL,), OR, XOR, AND, EQ or NE expected
23519	RELOP_LIST_NOT_OP	Next symbol should be an operator
23520	RELOP_LIST_ILLEGAL_OP	EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE or GE expected
23521	BOR_LIST_NOT_OP	Next symbol should be an operator
23522	BOR_LEFT_FLOAT	Left operand of BOR is of type FLOAT
23523	BOR_RIGHT_FLOAT	Right operand of BOR is of type FLOAT
23524	BOR_LIST_ILLEGAL_OP	EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE or BOR expected
23525	BXOR_LIST_NOT_OP	Next symbol should be an operator
23526	BXOR_LEFT_FLOAT	Left operand of BXOR is of type FLOAT
23527	BXOR_RIGHT_FLOAT	Right operand of BXOR is of type FLOAT
23528	BXOR_LIST_ILLEGAL_OP	EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR or BXOR expected
23529	BAND_LIST_NOT_OP	Next symbol should be an operator

Number	Defined Constant	Description
23530	BAND_LEFT_FLOAT	Left operand of BAND is of type FLOAT
23531	BAND_RIGHT_FLOAT	Right operand of BAND is of type FLOAT
23532	BAND_LIST_ILLEGAL_OP	EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR, BXOR or BAND expected
23533	ADDOP_LIST_NOT_OP	Next symbol should be an operator
23534	ADDOP_LIST_ILLEGAL_OP	EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR, BXOR, BAND, + or - expected
23535	MULOP_LIST_NOT_OP	Next symbol should be an operator
23536	MULOP_LIST_ILLEGAL_OP	EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR, BXOR, BAND, +, -, * or / expected
23537	AROP_BOOL_BOOL	Both operands of arithmetic op are of type BOOLEAN
23538	AROP_BOOL_FLOAT	Left operand of arithmetic op is BOOLEAN, right operand is FLOAT
23539	AROP_FLOAT_BOOL	Left operand of arithmetic op is FLOAT, right operand is BOOLEAN
23540	UNOP_NOT_FLOAT	Operand of NOT is of type FLOAT
23541	UNOP_BNOT_FLOAT	Operand of BNOT is of type FLOAT
23542	UNOP_MINUS_BOOL	Operand of arithmetic NEGATION is of type BOOLEAN
23543	UNOP_ILLEGAL_OP	-, NOT or BNOT expected
23544	TERM_ILLEGAL_OP	(, Identifier or constant expected
23545	TERM_MISSING_RPAR	) expected
23546	TERM_MISSING_LPAR	(expected
23547	TERM_ID_EXPECT	Identifier expected
23548	CONV_FLOAT_TO_STR	FLOAT cannot be converted to a BIT/OCTET string
23549	CHAR_STR_NOT_IMPL	Character strings are not implemented
23550	BITSTR_TRUNCATED	Bitstring specified exceeds max size, max size used
23551	OCTETSTR_TRUNCATED	Octetstring specified exceeds max size, max size used
23552	EVAL_CORRUP_EXPR	Expr cannot be evaluated, memory may be corrupted
23553	RECURSION_LEVEL_TWO	EU_CONV was called by itself, memory may be corrupted
23554	DIVISION_BY_ZERO	Division by zero attempted
23555	PTEXP_EXPR_NOT_EOX	Expr_array not properly terminated
23556	SYM_TAB_NOT_EXIST	Symbol table does not exist
23557	PTEXP_ID_NOT_IN_TABLE	No table entry for this point_id

Number	Defined Constant	Description
23558	PTEXP_ALIEN	error was not discovered by PTEXP
23559	PTEXP_NULL_PTR	Pointer to expression string is NULL
23560	PTEXP_UNKNOWN_CODE	Unknown code/type in expression, memory may be corrupted
23561	PTEXP_ID_NOT_DECLARED	Point_id is not in symbol table and type_func is missing
23562	PTEXP_EMPTY_EU	Cannot translate empty string for eu_conversion
23563	PTEXP_ID_TOO_LONG	Identifier has too many characters
23564	PTEXP_REDECLARATION	Identifier is already in symbol table
23565	PTEXP_ILLEGAL_TYPE	This is not a valid code for a PTM_DATA_TYPE
23566	PTEXP_REDEFINITION	Function identifier is already in symbol table
23567	PTEXP_FPTR_NOT_SET	Pointer to user defined function was not set
23568	PTEXP_NEG_ARG_SQR	Negative argument passed to square root function
23569	TEXTSTR_TRUNCATED	Text string specified exceeds max size, max size used
23570	INVALID_TEXT_COMP	Text strings can only compare to text strings
23571	TEXT_EQ_ONLY	Text strings can only be used in EQ/NE comparisons
23572	INDEX_OUT_OF_RANGE	The index is out of range for the point
23573	BAD_SUBSCRIPT_SYNTAX	The subscript syntax is incorrect
23574	NO_SPACES_BEFORE_BRACKET	Subscript must immediately follow a point id
23575	LEFT_BRACKET_MISSING	Left bracket is missing
23576	NO_SUBSCRIPT	A subscript must be specified for an array point
23577	NOT_ARRAY_PT	A subscript is not valid for a non-array point
23578	NO_CLOSING_QUOTE	The closing quote for a string is missing
23579	INVALID_TEXT_OP	Illegal operation with a text string
23580	PTEXP_OUT_OF_RANGE	Result from expression is out of range
23581	TRIOP_TYPE_NOMATCH	Possible results from trinary expression must be same type
23582	TRIOP_LIST_ILLEGAL_OP	Expecting EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR, BXOR, BAND, +, -, *, /, SHL, SHR, MOD, ^, or ?
23583	TRIOP_LIST_NOT_OP	Next symbol should be an operator
23584	TRIOP_COND_BOOL_ONLY	Trinary expression condition must be numeric.
23585	TRIOP_COND_MISSING_TRIOR	Expecting trinary expression separator :
23586	POWOP_LIST_ILLEGAL_OP	Expecting EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR, BXOR, BAND, +, -, *, /, SHL, SHR, MOD or ^
23587	POWOP_LIST_NOT_OP	Next symbol should be an operator

Number	Defined Constant	Description
23588	SHFTOP_LIST_ILLEGAL_OP	Expecting EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR, BXOR, BAND, +, -, *, /, SHL or SHR
23589	SHFTOP_LIST_NOT_OP	Next symbol should be an operator
23590	MODOP_LIST_ILLEGAL_OP	Expecting EOL,), OR, XOR, AND, EQ, NE, LT, GT, LE, GE, BOR, BXOR, BAND, +, -, *, /, SHL, SHR or MOD
23591	MODOP_LIST_NOT_OP	Next symbol should be an operator
23592	PTEXP_UPG_OPEN_FILE	%s: can't open file %s
23593	TRIOP_COND_MISSING_THEN	Expecting trinary expression operator: then
23594	TRIOP_COND_MISSING_ELSE	Expecting trinary expression operator: else
23595	PTEXP_UPG_HEADER1	The following point_id(s) listed below conflict with a new
23596	PTEXP_UPG_HEADER2	point expression function name. These point_id(s) cannot
23597	PTEXP_UPG_HEADER3	be used in a point expression unless they are renamed.
23598	PTEXP_UPG_DATA DATA	DATA directory
23599	PTEXP_UPG_MASTER MASTER	MASTER directory
23601	PTEXP_EXPR_POINT_UNAVAIL	Point evaluated in expression is unavailable

Error Messages from PTMDP

The following error codes are returned by PTMDP:

Number	Defined Constant	Description
24000	PTMDP_NORMAL	Normal successful completion
24001	PTMDP_SELF_LOOP	Point depends only on itself, will cause an infinite loop
24002	PTMDP_TRANSL_FAILED	Translation of expression failed
24003	PTMDP_UNKNOWN_TYPE	Unknown code for type received
24004	PTMDP_ILLEGAL_SEG	Illegal message segment type (type in err_ref)
24005	PTMDP_RSP_POINT_ID	Point_id received in response record not found in hash table
24006	PTMDP_NOT_MY_REQUEST	The response received does not match any request sent
24007	PTMDP_OUT_OF_RANGE	: this derived point is out of range
24008	PTMDP_SRC_TYPE_ERROR	Source point type is NOT numeric :
24009	PTMDP_ACC_TYPE_ERROR	Accumulator point type is NOT numeric :
24010	PTMDP_AVE_TYPE_ERROR	Average point type is NOT numeric:

Number	Defined Constant	Description
24011	PTMDP_MAX_TYPE_ERROR	Maximum capture point type is NOT numeric :
24012	PTMDP_MIN_TYPE_ERROR	Minimum capture point type is NOT numeric :
24013	PTMDP_ACC_VARIANCE	Variance value exceeded for accumulator point :
24014	PTMDP_ACC_ROLLOVER	Rollover occurred for accumulator point :
24015	PTMDP_SAMPLE_OVERFLOW	Sample count overflow for point :
24016	PTMDP_SRC_PNT_NAVAIL	Source point unavailable for point :
24017	PTMDP_BAD_DELTA_ACCUM :	Delta accum point must have exactly one source
24018	PTMDP_POINT_USE_NOT_FOUND	DMS point use record not found - structure error
24019	PTMDP_PROC_TRIG_SL	Processing trigger shopping list
24020	PTMDP_PROC_INP_SL	Processing input shopping list
24021	PTMDP_CALL_PROC_TRIG_PT	Calling ptmdp_process_trigger_point for point %s
24022	PTMDP_DERIVED_PT	Derived point - %s
24023	PTMDP_NOT_ON_INP_SL	Point is not on input shopping list - %s
24024	PTMDP_ON_SNAP_SL	Point is on snapshot shopping list - %s
24025	PTMDP_PUT_SNAP_SL	Putting point on snapshot shopping list - %s
24026	PTMDP_SEND_SNAP_SL	Sending snapshot shopping list
24027	PTMDP_EMPTY_SNAP_SL	Snapshot shopping list is empty
24028	PTMDP_SNAP_DEP	Component point %s of on-demand %s received
24029	PTMDP_PT_AVAIL	On-demand group point %s available
24030	PTMDP_HISTOGRAM_TYPE_ERROR	Histogram point type is NOT numeric :
24031	PTMDP_TIMER_COUNTER_TYPE_ERROR	Timer/Counter point type is NOT numeric :
24032	PTMDP_CONFIG_ERROR	Configuration error for point
24033	PTMDP_PS_LOG_PATH	The LOG_PATH directory could not be found.
24034	PTMDP_PS_READ_ERROR	Error when reading from file SAVE_POINT.DAT.
24035	PTMDP_PS_WRITE_ERROR	Error when writing to file SAVE_POINT.DAT.
24036	PTMDP_PS_CLOSE_ERROR	Error when closing file SAVE_POINT.DAT.
24037	PTMDP_PS_RENAME_ERROR	Error when renaming file SAVE_POINT.DAT as SAVE_POINT.BAK.
24038	PTMDP_PS_OPEN_ERROR	Error when opening file SAVE_POINT.DAT.
24039	PTMDP_PS_FLUSH_ERROR	Error when flushing cached data to file SAVE_POINT.DAT.
24040	PTMDP_PS_GET_ERROR	Error when getting saved point data for point :

Number	Defined Constant	Description
24041	PTMDP_PS_SET_ERROR	Error when setting saved point data for point :
24042	PTMDP_PS_NFOUND_ERROR	Cannot find saved point data for point :
24043	PTMDP_POINT_LABEL	. Point :
24044	PTMDP_CONVERTING_TYPE	Converting calculation type to equation. Point type is not numeric :
24045	PTMDP_TRIGGER_ITSELF	Cannot use point as its own trigger point :
24046	PTMDP_RESET_ITSELF	Cannot use point as its own reset point :
24047	PTMDP_EXPRESSION_ITSELF	Cannot use point in its own equation :

Error Messages from PTMRP

Error codes that can originate from PTMRP, and be sent to Application Processes are:

Number	Defined Constant	Description
24500	PTM_INVAL_ACK	Invalid ACK message received
24501	PTM_INVAL_CTS	Invalid CTS message received
24502	PTM_INVAL_NAK	Invalid NAK message received
24503	PTM_INVAL_OPEN	Invalid OPEN message received
24504	PTM_INVAL_RTS	Invalid RTS message received
24505	PTM_INVAL_SND	Invalid SND message received
24506	PTM_NO_HDR_SEG	No header segment found in message
24507	PTM_MSG_OUT_SEQ	Message received out of sequence
24508	PTM_INVAL_PROC_TYPE	Invalid process type received
24509	PTM_UNKNOWN_SRC_ADDR	Message received from unknown source address
24510	PTM_NO_STAMP_AVAIL	No IPC stamp available for final response message
24511	PTM_POINT_UNAVAILABLE	Point is currently unavailable
24512	PTM_POINT_OUT_OF_RANGE	Current point value is out of range
24513	PTM_MSG_LOST	Message lost -
24514	PTM_UNEXPECTED_ACK	Unexpected message received (ACK) -
24515	PTM_UNEXPECTED_CTS	Unexpected message received (CTS) -
24516	PTM_UNEXPECTED_DTR	Unexpected message received (DTR) -
24517	PTM_UNEXPECTED_NAK	Unexpected message received (NAK) -

Number	Defined Constant	Description
24518	PTM_UNEXPECTED_RTS	Unexpected message received (RTS) -
24519	PTM_UNEXPECTED_SND	Unexpected message received (SND) -
24520	PTM_ACK_SEQ_MISMATCH	Message sequence mismatch (ACK) -
24521	PTM_NAK_RCVED	NAK message received -
24522	PTM_POINT_NOT_IN_VIEW	Resource for point not in user's view - setpoint disallowed
25423	PTM_NO_ALARMSTATE	No ALARMSTATE requests allowed on standby PTM
24524	PTM_NO_SETPTS	No SETPOINT requests allowed on standby PTM
25425	PTM_NOMODALARM	No MODIFIED ALARM LIMIT requests allowed on standby PTM
25426	PTM_PT_STRUCT_NOT_VALID	Structure point is not properly defined for type =
25427	PTM_NO_SETPOINT_PRIV	No setpoint privilege on system %.20s.
25428	PTM_BAD_DOWNLOAD_PASSWORD	Requested XASMgr action invalid

Error Messages from Point Management Resident Process

Error codes that can be set directly by the Point Management Resident Process are:

Number	Defined Constant	Description
24551	PTMRP_INVAL_ALRM_FLD	Invalid alarm field specified
24552	PTMRP_INVAL_SVC_REQ	Invalid service request
24553	PTMRP_INVAL_MSG	Invalid message received
24554	PTMRP_DUP_POINT_ID	Duplicate point id
24555	PTMRP_INVAL_POINT_STATE	Invalid point state
24556	PTMRP_INVAL_COMM_STATE	Invalid communication state
24557	PTMRP_INVAL_DEVICE_ID	Invalid device id
24558	PTMRP_LINK_DOWN	Logical link down
24559	PTMRP_INVAL_MSG_TYPE	Invalid message type
24560	PTMRP_NO_HEADER	No header segment found in message
24561	PTMRP_INVAL_MSG_LEN	Message length exceeds expected length
24562	PTMRP_INVAL_DEVICE_STATE	Invalid device state
24563	PTMRP_INVAL_POINT_ID	Invalid point id
24564	PTMRP_MSG_TOO_LARGE	Message exceeds available buffer space

Number	Defined Constant	Description
24565	PTMRP_LOST_AP_REQ	Lost application request DMS pointer
24566	PTMRP_NULL_PTR	Specified pointer has value NULL
24567	PTMRP_MUST_BE_DP	Only a derived point process may update a point
24568	PTMRP_NOT_OWNER_DP	Not the owner derived point process for specified point
24569	PTMRP_NO_WARNING	No warning limit configured for point
24570	PTMRP_NO_WARNING_LOW	No Alarm Lo-1 limit configured for point
24571	PTMRP_NO_WARNING_HIGH	No Alarm Hi-1 limit configured for point
24572	PTMRP_NO_ALARM	No alarm limit configured for point
24573	PTMRP_NO_ALARM_LOW	No Alarm Lo-2 limit configured for point
24574	PTMRP_NO_ALARM_HIGH	No Alarm Hi-2 limit configured for point
24575	PTMRP_MUST_BE_DEVP	Only a device point may be set
24576	PTMRP_NO_LIMITS_AVAIL	No ALARM/WARNING limits available for the point
24577	PTMRP_POINT_IN_REQ_STATE	Point is already in the requested ALARM/WARNING state
24578	PTMRP_NO_ALARM_CONF	Point is not configured with a valid alarm
24579	PTMRP_NO_RANGE_LOW	No range low limit configured for point
24580	PTMRP_NO_RANGE_HIGH	No range high limit configured for point
24581	PTMRP_NO_DEADBAND	No deadband configured for point
24582	PTMRP_NO_EU_CONV	No EU conversion record found in SC EU conv file for point =
24583	PTMRP_ALARM_ID_NOT_IN_SC	No alarm found in SC Alarm Def file for alarm id =
24584	PTMRP_FR_ID_NOT_IN_SC	No fact. rsrc. ID found in SC FR file for FR id =
24585	PTMRP_ALARM_MGR_NOT_IN_SC	No alarm mgr found in SC Alarm Mgr file for alarm mgr =
24586	PTMRP_PROC_ID_NOT_IN_SC	PTM proc ID not found in SC Proc Id file for proc id =
24587	PTMRP_POINT_NOT_IN_SC	No record found in SC point file for point =
24588	PTMRP_POINT_TYPE_NOT_IN_SC	Point type not found in SC point type file for point =
24589	PTMRP_NO_WRT_ACCESS	No write access for point
24590	PTMRP_POINT_DISABLED	Cannot set a disabled point
24591	PTMRP_INIT_OUT_OF_RANGE	Initial value is out of range for point =
24592	PTMRP_INVALID_ALARM_CRIT	Invalid Alarm Criteria for point =
24593	PTMRP_NULL_DEVPT_SPEC	Deviation point unspecified for point =
24594	PTMRP_INVALID_DEV_TYPE	Invalid Deviation point type for point =

24595	PTMRP_INVALID_SAMPLE_INIT	Invalid sample interval for point =
24596	PTMRP_INVALID_RATE_TYPE	Invalid Rate of Change point type for point =
24597	PTMRP_INV_SAFPT_TYPE	Safety point <%s> is invalid type for point <%s>
24598	PTMRP_INV_SAFPT_PT	Safety point <%s> not allowed for Derived point <%s>
24599	PTMRP_INV_SAFPT	Point <%s> cannot be its own Safety Point
24600	PTMRP_SAF_BAD_CONFIG	Safety point <%s> not configured for point <%s>
24601	PTMRP_INV_DEVPT_TYPE	Deviation point <%s> is invalid type for point <%s>
24602	PTMRP_DEV_BAD_CONFIG	Deviation point <%s> not configured for point <%s>
24603	PTMRP_SAFPT_NOT_AVAIL	Setpoint fail: <%s> - Safety point <%s> UNAVAILABLE
24604	PTMRP_SAFETY_FAIL	Setpoint fail: <%s> - Safety point <%s> SET TO FALSE
24605	PTMRP_NO_ALARM_LIMITS	No Alarm Limits specified for point =
24606	PTMRP_DEVPT_ABS_LIMIT	Using Alarm Limit absolute value for Deviation point =
24607	PTMRP_NO_ARRAY_PT	Array point not allowed for point =
24608	PTMRP_LOST_POINT_REQ	Lost point request DMS pointer
24609	PTMRP_BAD_DC_SVC_REQ	Unsupported DEVCOM service request
24610	PTMRP_NO_REDUND_SVC	Host redundancy specified but single PTM service configured
24611	PTMRP_REDUND_CFG_ERR	Host redundancy configuration error
24612	PTMRP_UNEXPECTED_TRANS	Unexpected TRANSITION message received - ignored
24613	PTMRP_UNEXPECTED_MAST	Unexpected server message received - ignored
24614	PTMRP_INVALID_MAST	Invalid server message received - ignored
24615	PTMRP_PARTNER_NOT_FOUND	Partner record not found in schema
24616	PTMRP_FORCED_ROLL_UP	Received request for forced roll on partner recovery
24617	PTMRP_FORCED_ROLL_DN	Received request for forced roll on partner failure
24618	PTMRP_LOGTRANS_FAILED	failed to translate logical name for archive
24619	PTMRP_UNEXPECTED_PART_UP	Partner recovery notice without previous partner failure
24610	PTMRP_NO_DL_ON_NODE	No DL process on this node - no forced rollover
24621	PTMRP_NO_PTDL_ON_NODE	No PTDL process on this node - no forced rollover
24622	PTMRP_SERVICE_NOT_OPEN	Failed to open service file - no forced rollover
24623	PTMRP_NO_BUF_VALUES	No buffered values for point =
24624	PTMRP_INVALID_BUF_VALUE	Invalid maximum point buffering and duration for point =
24625	PTMRP_ADHOC_REQ	
24626	PTMRP_ADHOC_CREATE	Point By Address Created: %.50s

24627	PTMRP_ADHOC_DELETE	Point by Address Deleted: %.50s
24628	PTMRP_TRIGGER_NOT_AVAIL	Point %.16s availability trigger %.16s is unavailable.
24629	PTMRP_TRIGGER_FAIL	Point %.16s availability trigger %.16s is not numeric.
24630	PTMRP_INV_TRIGGER_OWN	Point %.32s cannot be its own availability trigger.
24631	PTMRP_INV_TRIGGER_TYPE	Point %.16s availability trigger %.16s is not numeric.
24632	PTMRP_INV_TRIGGER_CFG	Point %.16s availability trigger %.16s is not configured.
24633	PTMRP_INV_TRIGGER_STATE	Point %.16s has an invalid availability trigger %.16s.
24634	PTMRP_CANNOT_HAVE_TRIGGER	A derived point %.16s cannot have an availability trigger %.16s.
24635	PTMRP_POINT_ES_ID_MISMATCH	Old/Invalid ES alias point ID -
24636	PTMRP_POINT_INV_ES_REQ	Invalid ES alias point ID request from

Error Messages from PTMAP

Error codes that can be returned by PTMAP are:

Number	Defined Constant	Description
25000	PTMAP_ADR_PTR_NULL	NULL Pointer to PTMAP_ADDR record specified
25001	PTMAP_SEQ_NUM_MISMATCH	Seq_num in PTMAP_ADDR differs from seq_num in database
25002	PTMAP_DATA_NULLPTR	Pointer to PTM_DATA structure is NULL
25003	PTMAP_DATA_TYPE_MISMATCH	The two PTM_DATA records have different type
25004	PTMAP_DATA_LEN_MISMATCH	Data lengths of PTM_DATA structures are different
25005	PTMAP_DATA_ELEM_MISMATCH	Number of elements of PTM_DATA records are different
25006	PTMAP_REQ_ADR_NULL	NULL request address specified
25007	PTMAP_REQ_ADR_NOTF	Request address not found
25008	PTMAP_PT_ADR_NULL	NULL Point address specified
25009	PTMAP_PT_ADR_NOTF	Point not found (%s)
25010	PTMAP_SL_ADR_NULL	Shopping list address specified is NULL
25011	PTMAP_SL_ADR_NOTF	Shopping list not found
25012	PTMAP_REQ_MISMATCH	Mismatch between function call and actual request type
25013	PTMAP_PTM_PT_NOTF	PTM_PROC Record not found in HAS_PTM_POINT set
25014	PTMAP_REM_OUTST_REQ	Tried to remove request record with state outstanding
25015	PTMAP_RSP_DELETED	Response records were deleted, number in COR_STATUS record

Number	Defined Constant	Description
25016	PTMAP_HAS_NOT_EU_CONV	There is no eu_conversion for this point
25017	PTMAP_INVAL_POINT_ID	Invalid Point Identifier specified
25018	PTMAP_NO_RSP_RCV	No response received
25019	PTMAP_REQ_NOT_ENBL	Request is currently disabled
25020	PTMAP_REQ_CUR_OUT	Request is currently outstanding
25021	PTMAP_NO_REQ_SENT	No request available to send
25022	PTMAP_RP_UNREACHABLE	Point Management resident process is unreachable
25023	PTMAP_UNKNOWN_SRC_ADDR	Message received from unknown source address
25024	PTMAP_INVAL_MSG	Invalid message received
25025	PTMAP_INVAL_IPC_FLAG	Invalid IPC message flags
25026	PTMAP_POINT_ALRDY_ADDED	Point already added
25027	PTMAP_REQ_NOT_OUT	No request is currently outstanding
25028	PTMAP_MSG_TOO_LARGE	Message exceeds available buffer space
25029	PTMAP_INVAL_REQ_TYPE	Invalid request type specified
25030	PTMAP_MSG_OUT_SEQ	Message received out of sequence
25031	PTMAP_INVAL_CTS	Invalid CTS message received
25032	PTMAP_INVAL_NAK	Invalid NAK message received
25033	PTMAP_RCV_QUE_ERR	Current receive queue not empty
25034	PTMAP_INVAL_RSP_ID	Invalid response ID received
25035	PTMAP_RSP_POINT_NOTF	Point not found for response message
25036	PTMAP_INVAL_EU_RES_TYPE	The type of the result is not FLOATINGPOINT
25037	PTMAP_NOT_FULL_INFO	Full information not available for this point
25038	PTMAP_INVAL_DATA_TYPE	Invalid point data type for request
25039	PTMAP_INVAL_ELEMENTS	Multiple element points not valid for request
25040	PTMAP_INVAL_MASK_POINT	Point data length too long for ONMASK request
25041	PTMAP_INVAL_ONALARM_STATE	Invalid alarm/warning state for ONALARM request
25042	PTMAP_INVAL_ONALARM_LIMIT	Invalid high/low limit for ONALARM request
25043	PTMAP_INVAL_RELATION	Invalid relation for ONEVENT request
25044	PTMAP_NOT_SUSPENDED	Process is not currently suspended
25045	PTMAP_SUSPENDED	Process is currently suspended
25046	PTMAP_INVAL_KEY	Invalid response key received from Point Translation

Number	Defined Constant	Description
25047	PTMAP_PTX_NOT_FOUND	There is no Point Translation Process on this node
25048	PTMAP_PTX_POINT_INFO	Point Translation error for point info
25049	PTMAP_PTX_EU_INFO	Point Translation error for eu_conv info
25050	PTMAP_INV_TIME_UNIT	Invalid time unit specified
25051	PTMAP_POINT_NOT_MON	The point specified is not currently being monitored
25052	PTMAP_INVALID_MODE	Invalid mode specified
25053	PTMAP_NO_POINTS_MON	No points are currently being monitored
25054	PTMAP_NO_LIMITS_MOD	No alarm limits have been modified
25055	PTMAP_NO_DISPLAY_LIMITS	No display limits exist for this point
25056	PTMAP_RESYNCHRONIZING	
25057	PTMAP_INV_TIME_INTERVAL	Invalid time interval specified
25058	PTMAP_RP_FAILOVER	Failover to Redundant Point Manager
25059	PTMAP_RP_REQUEUE	Requeued request to Redundant Point Manager
25060	PTMAP_NO_LOCAL_RP	No local Point Manager configured for redundant node
25061	PTMAP_UNEXPECTED_MS	Unexpected server or standby segment from Point Manager
25062	PTMAP_POINT_IN_USE_MODIFIED	
25063	PTMAP_MOD_POINT_NOT_FOUND	%s not found internally
25064	PTMAP_TYP_ID_INVAL	Point Type ID (%s) is invalid
25065	PTMAP_PT_NOT_STRUCT	Point Type ID (%s) is not a STRUCTURE
25066	PTMAP_COMP_TYPE_INVAL	Component type (%s) in structure (%s) is invalid
25067	PTMAP_SETPT_FILTER	Setpoint filtered out by standby application
25068	PTMAP_INV_XLATE	Invalid Remote xlate service request
25069	PTMAP_UNEXPECTED_SEG	Unexpected segment from PTX
25070	PTMAP_WAIT_TIMEOUT	Timeout waiting for response.
25071	PTMAP_AST_TABLE	Unable to find threadid in ast table.
25072	PTMAP_POINT_NOT_READ	%.55s has not been read.
25073	PTMAP_POINT_NOT_INIT	Point not initailized
25074	PTMAP_EXTERNAL_INT	External Interrupt Received.
25075	PTMAP_BAD_CONVERSION	%.32s Bad Conversion for Get
25076	PTMAP_NO_ADHOC_PRIV	Point By Address request denied - no privileges
25077	PTMAP_LP_TYPE_MISMATCH	Local point data type mismatch

Number	Defined Constant	Description
25078	PTMAP_LP_MSFT_EXCEPTION	Microsoft exception
25079	PTMAP_LP_UNKNOWN_ERROR	Unknown error
25080	PTMAP_POINT_NOT_FOUND	Point not found in local Point Manager
25081	PTMAP_REQUEST_NOT_FOUND	Request not found in local Point Manager
25082	PTMAP_REQUESTOR_NOT_FOUND	Requestor not found in local Point Manager
25083	PTMAP_POINT_NOT_LOCAL	Point %s is not local
25084	PTMAP_REQUEST_NOT_SUPPORTED	Request type is not supported in local Point Management
25085	PTMAP_LP_TYPE_UNSUPPORTED	Local point data type is not supported
25086	PTMAP_LP_INVALID_ACCESS_KEY	Local Point Management access key is invalid
25087	PTMAP_LP_POINT_ADDED	Point already added in local point manager
25088	PTMAP_LP_REQUEST_ADDED	Request already added in local point manager
25089	PTMAP_NO_ONALARMACK	OnAlarmAck is not active.

Error Messages from Point Translation Process

Error codes that can be sent from the Point Translation Process are:

Number	Defined Constant	Description
26000	PTX_NORMAL	Normal successful completion
26001	PTX_UNKNOWN_SEGMENT	Unknown Segment Sent - See err_ref for Segment Number
26002	PTX_MISSING_DEVCOM_PROC	Devcom process ID not found in SC files
26003	PTX_FR_NOT_FOUND	Factory resource ID not found in data base
26004	PTX_POINT_NOT_FOUND	Point ID not found in data base
26005	PTX_PTMGMT_OWNER_NOT_FOUND	Point Manager owner not found in data base
26006	PTX_HAS_NOT_EU_CONV	Specified Point has no Eu Conversion
26007	PTX_PT_TYPE_OWNER_NOT_FOUND	Point Type owner not found in data base
26009	PTX_DEVICE_NOT_IN_SC	No Device in SC device file for devcom proc =
26010	PTX_PTMGMT_DEVCOM_DB	No Point Manager in DB for devcom proc =
26011	PTX_DERIVE_PT_NOT_IN_SC	Point not found in SC derive file =
26012	PTX_PTMGMT_DERIVE_PT_DB	No point manager in DB for derive point =
26013	PTX_PTMGMT_OWN_DEVICE_DB	No point manager in DB for device point =
26014	PTX_DEVICE_NOT_IN_DB	No device in DB for device id =

Number	Defined Constant	Description
26015	PTX_DEVICE_PT_NOT_IN_SC	Point not found in SC device file =
26016	PTX_PT_TYPE_NOT_IN_DB	No point type in DB for point =
26017	DYN_CFG_SEG_ERR	Unrecognized segment in dyn cfg msg
26018	DYN_CFG_BAD_KEY_ERR	Key not valid for %s file
26019	DYN_CFG_NO_FILE_ERR	Could not write sc_path:dyn_cfg.cfg
26020	DYN_CFG_BAD_SEQ_ERR	Unexpected sequence number
26021	DYN_CFG_BAD_SENDER_ERR	Unexpected sender:
26022	PTX_PT_STRUCT_NOT_VALID	Structure point is not properly defined for type =
26030	PTX_ADHOC_NO_ADDRESS	Point By Address requires a Device Address
26030	PTX_ADHOC_NO_DEVICE	Point By Address requires a Device Specification
26030	PTX_ADHOC_DEV_NOTFOUND	Point By Address Device (%.32s) not found.
26030	PTX_ADHOC_PTM_NOTFOUND	Point By Address Device (%.32s) has not Point Manager.
26030	PTX_ADHOC_TYPE_NOTFOUND	Point By Address Type (%.32s) not found.
26030	PTX_ADHOC_BAD_SCAN	Invalid Point By Address Scan Rate (%.20s).
26030	PTX_ADHOC_BAD_OFFSET	Invalid Point By Address Bit Offset (%.20s).
26030	PTX_ADHOC_BAD_ELEM	Invalid Point By Address Array Elements (%.20s).
26030	PTX_ADHOC_BAD_ACCESS	Invalid Point By Address Access Mode (%.20s).
26030	PTX_ADHOC_BAD_ADDR_TYPE	Invalid Point By Address Address Type (%.32s).
26040	PTX_ADHOC_BAD_TYPE	Invalid Point By Address Data Type (%.32s).
26041	PTX_ERR_EXTREF_INV_REF	Invalid concentrated point reference (%s)

Chapter 9. CIMPLICITY to Windows Server

About the CIMPLICITY to Windows Server

The CIMPLICITY to Windows Server (CWSERV) lets you access CIMPLICITY point data from other Microsoft Windows products such as Microsoft Excel. CWSERV uses the Microsoft standard of Dynamic Data Exchange (DDE). With CWSERV, you can use your favorite software package that supports DDE to monitor, analyze, report, or modify CIMPLICITY point data.)

Note: Consult the Microsoft Web site for a list of Windows operating systems that are supported for DDE.

CWSERV with Microsoft Excel

CWSERV with Microsoft Excel

You can use CWSERV on a Microsoft Excel spreadsheet to display point information, and to update setpoints in the CIMPLICITY point database.

Important: Beginning with CIMPLICITY v9.0, CWSERV supports point names that are longer than 32 characters. However, it supports a maximum of 255 characters that includes both the point ID and point attribute.

Example

A CIMPLICITY project is running.

The following macro is used in an Excel spreadsheet to display the value of a point in that project.

```
=CWSERV|POINT!pointName.VALUE
```

Where

`pointName` is the point ID

`.VALUE` is the VALUE attribute.

The maximum length for the point ID is 255 - 6 characters = 249 characters.

1 *(page 497)*	Start CWSERV.
2 *(page 497)*	Display point data.
3 *(page 498)*	Modify point data.

1. Start CWSERV

The first time you enter point information into a spreadsheet, the following dialog box will be displayed:

This dialog box will also be displayed if you open a spreadsheet that contains CWSERV commands and the CWSERV server is not active.

Click **Yes** to start CWSERV.

This dialog box will appear every time you open the spreadsheet:

Click **Yes** to reconnect to CWSERV to access data. If CWSERV is not running, you will be asked if you want to start it. Click **Yes** to start CWSERV.

If you are connecting to the project for the first time, or your login timeout has expired, the CIMPLICITY Login dialog box for the project is displayed.

Enter your CIMPLICITY username and password and click **OK**. Your spreadsheet will now start to display the CIMPLICITY data you requested in the CWSERV commands.

CWSERV Icon

While the server is active, you will see this icon on your terminal screen:

2. Display Point Data

1. Select the cell where you want the point information to appear.
2. Enter the CWSERV formula in the cell, then press **Enter**.

   For example, to display the raw value for the point CWSERV_VIRT, you would type:

=cwserv|point!cwserv_virt.raw_value

To display an array in a Microsoft Excel spreadsheet:

3. Select a range of cells (horizontal or vertical) where you want the point information to appear.
4. Enter the CWSERV formula, and enclose the point information within single quotes, then press **Ctrl+Shift+Enter**.

   For example, to display the raw values for the ten-item array CWSERV_ARRAY in a column on your spreadsheet, you would type

   =cwserv|point!'cwserv_array.raw_value[0:9c]'

3. Modify Point Data

3. Modify Point Data

You can write Excel macros to update setpoints in CIMPLICITY software from a spreadsheet. These macros will use the DDE POKE request. You can use this request to change the **value** or **raw_value** attributes of a CIMPLICITY point.

Important: Setpoint security is not enforced. A user can set any CIMPLICITY point that has write access, regardless of the setpoint security restrictions configured within the CIMPLICITY system. You must establish appropriate operational practices and procedures to prevent undesired setpoint operations, and users must follow these practices and procedures carefully.

3.1. Visual Basic Format

1. Enter the Sub procedure name on the first line.
2. Enter channel= Application.DDEInitiate("cwserv","point") on the second line to open the CWSERV channel.

3. Enter Application.DDEPoke channel, "<point_id>.<attribute>", <sheet location> for each setpoint you want to perform.
4. Enter Application.DDETerminate channel to close the channel.
5. Enter End Sub to terminate the Sub procedure.

   Macro Example

```
Sub Poke()

channel = Application.DDEInitiate( app:="cwserv", topic:="point")

Set rangeToPoke = Worksheets("Sheet1").Range("A1")

Application.DDEPoke channel, "POINT1.value", rangeToPoke

Application.DDETerminate channel

End Sub
```

3.2. Performance Considerations

When you perform a large number of DDE POKE requests from an application such as Microsoft Excel, the DDE server application may fall behind. Under Excel, this will cause some requests to timeout and fail.

To avoid this condition, insert delays in your setpoint macros as follows:

```
ManyPoints
channel=INITIATE("cwserv","point")
=POKE(channel,"point001.value",Sheet1!D4)
=POKE(channel,"point002.value",Sheet1!D5)
=POKE(channel,"point003.value",Sheet1!D6)
=POKE(channel,"point004.value",Sheet1!D7)
=WAIT(NOW() + "00:00:01")
=POKE(channel,"point005.value",Sheet1!E4)
=POKE(channel,"point006.value",Sheet1!E5)
=POKE(channel,"point007.value",Sheet1!E6)
=POKE(channel,"point008.value",Sheet1!E7)
=WAIT(NOW() + "00:00:01")
=POKE(channel,"point009.value",Sheet1!F4)
=POKE(channel,"point010.value",Sheet1!F5)
=POKE(channel,"point011.value",Sheet1!F6)
=POKE(channel,"point012.value",Sheet1!F7)
=TERMINATE(channel)
=RETURN()
```

Depending on the performance and configuration of your computer, your delay requirements may vary.

3.3. Macro Implementation

1. Select a location on your spreadsheet where you want to enter the point's setpoint value. Make a note of the sheet name and cell location.
2. If you have not already done so, select Toolbars on the View menu and activate the Drawing toolbar.
3. Select Record Macro on the Tools menu on your spreadsheet.
4. Select Record New Macro on the Record Macro submenu. The Record New Macro dialog box will open.
5. Enter your new macro name in the **Macro Name** box.
6. Click **Options**.
7. Select **This Workbook** from the **Store In** input box.
8. Select **MS Excel 4.0 Macro** from the **Language** box.
9. Click **OK**.

   A new Macro sheet will be created and the macro name will be placed in the first cell (R1C1) of the sheet.

10. Select Record Macro on the Tools menu.
11. Select Stop Recording on the Record Macro submenu.
12. Enter your commands in the cells under the macro name on the Macro page.
13. Go back to your sheet.
14. Click **Create Button** on the Drawing toolbar.
15. Create a button on the spreadsheet. When you do this, the Assign Macro dialog box will open.
16. Select the setpoint macro from the Assign Macro dialog box.

    Whenever a user enters the setpoint in the cell referenced by the macro, then clicks the button, the setpoint value will be sent to CIMPLICITY software.

Sample Spreadsheets and Macros

Sample Spreadsheets and Macros

Two spreadsheet and macro sets are given here. The first spreadsheet and macro show you how to display all the attributes of a single point, and perform a setpoint on that point. The second spreadsheet and macro show you how to display the raw values for an array point and perform a setpoint on the entire array.

Single Point Example

The following spreadsheet was configured to display all available point attributes for CWSERV_VIRT, and to let the user perform a setpoint on that point.

The formulas that appear in cells R5C3 through R17C3 are:

R5C3	**=cwserv|point!cwserv_virt.value**
R6C3	**=cwserv|point!cwserv_virt.state**
R7C3	**=cwserv|point!cwserv_virt.disp_format**
R8C3	**=cwserv|point!cwserv_virt.eu_label**
R9C3	**=cwserv|point!cwserv_virt.alarm_high**
R10C3	**=cwserv|point!cwserv_virt.warn_high**
R11C3	**=cwserv|point!cwserv_virt.warn_low**
R12C3	**=cwserv|point!cwserv_virt.alarm_low**
R13C3	**=cwserv|point!cwserv_virt.disp_high**
R14C3	**=cwserv|point!cwserv_virt.disp_low**
R15C3	**=cwserv|point!cwserv_virt.alarm_enabled**
R16C3	**=cwserv|point!cwserv_virt.warn_enabled**
R17C3	**=cwserv|point!cwserv_virt.init_state**

The following macro is the one associated with the **Set** button above.

```
set_point
channel=INITIATE("cwserv","point")
=POKE(channel,"cwserv_virt.value",Sheet1!R19C3)
=TERMINATE(channel)
=RETURN()
```

To change the setpoint, enter the new point value in cell R19C3, then click **Set**.

Array Point Example

The following spreadsheet was configured to

- Display the raw values for CWSERV_ARRAY, an array point with ten (10) elements, and
- Let a user perform a setpoint on the array:

The procedure is as follows:

1. Highlight the number of rows or columns to match the number of elements of the row to be displayed. (Cells R5C3 through R14C3 were selected.)
2. Enter the SCSERV command. For example,

   =cwserv|point!'cwserv_array.raw_value[0:9c]'

3. Press **Ctrl+Shift+Enter** on the keyboard.

   Note: You must press **Ctrl+Shift+Enter** when entering a formula for a range of cells on the spreadsheet. If you press **Enter**, only the first cell of the range will contain data. In the case of array points, this means that only the first array element's value is displayed.

   The following macro is the one associated with the **Set** button above.

```
Array
channel=INITIATE("cwserv","point")
=POKE(channel,"cwserv_array.raw_value[0:9]",Sheet1!R5C4:R14C4)
=TERMINATE(channel)
=RETURN()
```

   To change the setpoint, enter the new values in R5C4 through R14C4, and click **Set**.

Command Syntax for CWSERV

Command Syntax for CWSERV

Generally, a DDE command consists of three elements - the name of the service being used, the topic, and the item.

For the CIMPLICITY to Windows Server:

- The service is CWSERV.
- The topics supported are POINT and SYSTEM. Use the Point topic to display CIMPLICITY point information. Use the System topic to display CWSERV system information.
- The items supported for the POINT topic correspond to various point attributes *(page 504)* such as value, alarm limits, and point state.
- The items supported for the SYSTEM topic are FORMATS, SYSITEMS, TOPICS, and HELP.

The format of a particular DDE command depends on how it has been implemented in the application where you are using it.

Microsoft Excel Example

Microsoft Excel Example

The syntax for a CWSERV command in a Microsoft Excel spreadsheet, is:

```
=cwserv|point!<point_id>.<attribute>[n:mD]
```

Where

< point_id **>**	is the CIMPLICITY Point ID whose data is being retrieved. You may enter an unqualified or fully qualified (by project or node name) Point ID.
<attribute>	is the point attribute *(page 504)* of interest. If you do not enter an attribute, CWSERV uses "VALUE" as the default.
`n:m`	is the range of array elements desired. If you do not enter a range, CWSERV uses "[0]" as the default. Enter "[n]" to specify a single element of an array.
`D`	If you have requested a range of elements, use this field to specify the display format. Enter "C" to display the elements in a column, or "R" to display the elements in a row. If you do not enter a display format, CWSERV uses "C" as the default. **Note:** Consult the CIMPLICITY Help Desk for updated examples.

Sample CWSERV Commands

This syntax will display the value of the point ANALOG_POINT:

```
=cwserv|point!analog_point
```

This syntax will display the current state of the point ANALOG_POINT:

```
=cwserv|point!analog_point.state
```

This syntax will display the sixth through tenth values of the array point ARRAY_POINT in a row:

```
=cwserv|point!'array_point.value[5:9R]'
```

This syntax will display the value of the point ANALOG_POINT in the TEST project:

```
=cwserv|point!\\test\analog_point.value
```

Point Topic Attributes

Point Topic Attributes

The following point attributes can be displayed:

- ALARM_ENABLED
- ALARM_HIGH
- ALARM_LOW
- DISP_FORMAT
- DISP_HIGH
- DISP_LOW
- ELEMENTS
- EU_LABEL
- INIT_STATE
- LENGTH
- RAW_VALUE
- SIZE
- STATE
- TYPE
- VALUE
- WARN_ENABLED
- WARNING_HIGH
- WARNING_LOW

ALARM_ENABLED

Indicates whether high/low alarms are enabled or disabled for this point. You will see one of the following values:

0	High/Low alarm messages are disabled for the point.
1	High/Low alarm messages are enabled for the point.

ALARM_HIGH

Displays the **Alarm High** value for the point. This field is only valid for the following point types:

- ANALOG
- APPL

If the point's value exceeds this number, the point is in ALARM HIGH state.

ALARM_LOW

Displays the **Alarm Low** value for the point. This field is only valid for the following point types:

- ANALOG
- APPL

If the point's value is less than this number, the point is in ALARM LOW state.

DISP_FORMAT

Displays the format used when displaying the point's value in Alarm Viewer, Status Log messages, or CimView.

DISP_HIGH

Displays the high limit for the display value for this point. This field is only valid for the following point types:

- ANALOG
- APPL

DISP_LOW

Displays the low limit for the display value for this point. This field is only valid for the following point types:

- ANALOG

- APPL

ELEMENTS

Displays the number of elements contained in the point.

EU_LABEL

Displays the engineering units label for the point. The label can be up to eight (8) characters long.

INIT_STATE

Indicates whether the point is enabled or disabled. You will see one of the following values:

0	The point is disabled
1	The point is enabled.

LENGTH

Displays the length of the point. This field is only meaningful for the following point types:

- BITSTRING
- OCTETSTRING

RAW_VALUE

Displays the raw value of the point.

SIZE

Displays the size of the data.

STATE

Displays the current state of the point.

The point's current state depends on its point class and alarm conditions.

For all point classes, the states that can be displayed are:

NORMAL	The point's value is within normal limits, and no alarms are outstanding.
UNAVAILABLE	If the point is a device point, communications with the device have failed, and the point can no longer be read. If the point is a virtual point, one or more of the source points that comprise this point is unavailable.

For Analog and APPL point classes, the additional states that can be displayed are:

ALARM HIGH	The point's value is greater than the high alarm limit.
ALARM LOW	The point's value is less than the low alarm limit.
WARNING HIGH	The point's value is greater than the warning high limit and less than the alarm high limit.
WARNING LOW	The point's value is less than the warning low limit and greater than the alarm low limit.
OUT OF RANGE	The point is an Analog or APPL device point with engineering units conversion, and its value exceeds one of its conversion limits.

For the Digital (Boolean) point class, the additional states that can be displayed are:

ALARM	The point's value is in the alarm state.
WARNING	You will only see this message if Enable Alarms has been reset, Enable Warning is set, and the point's value is in the alarm state.

TYPE

Displays the point's type. You will see one of the following:

- BOOLEAN
- BITSTRING
- OCTETSTRING
- CHARACTERSTRING
- UNSIGNED INTEGER 1
- UNSIGNED INTEGER 2
- UNSIGNED INTEGER 4
- INTEGER 1
- INTEGER 2
- INTEGER 4
- FLOATING POINT
- STRUCTURE

Note: OCTETSTRING points are not currently supported by CWSERV. If you try to display such a point's VALUE or RAW_VALUE attribute, "#NAME?" will be displayed on the Excel spreadsheet.

VALUE

Displays the converted (EU) value of the point. If there is no conversion, the raw value is displayed. If you do not enter an attribute in the **CWSERV** command, this is the default attribute that is `displayed.RAW_VALUE`.

Displays the raw value of the point.

WARN_ENABLED

Indicates whether high/low warnings are enabled or disabled for this point.

0	High/Low warning messages are disabled for the point
1	High/Low warning m essages are enabled for the point.

WARNING_HIGH

Displays the **Warning High** value for the point. This field is only valid for the following point types:

- ANALOG
- APPL

If the point's value is greater than this number, but less than the **Alarm High** number, the point is in WARNING HIGH state.

WARNING_LOW

Displays the **Warning Low** value for the point. This field is only valid for the following point types:

- ANALOG
- APPL

If the point's value is less than this number, but greater than the **Alarm Low** number, the point is in WARNING LOW state.

Command Syntax for System Topic

Command Syntax for System Topic

The System topic contains information about CWSERV.

You can display:

- Formats supported by CWSERV.
- System items supported by CWSERV.
- Topic supported by CWSERV.
- Help information about CWSERV.

Formats

1. Select one (1) cell in the spreadsheet.
2. Type the following formula in the cell and press **Enter**:

   ```
   =cwserv|system!formats
   ```

   The resulting display should look like this:

   ```
   CF_TEXT
   ```

System Items

1. Select four (4) rows in one column in the spreadsheet.
2. Type the following formula in the first cell and press **Ctrl+Shift+Enter**.

   ```
   =cwserv|system!sysitems
   ```

   The resulting display should look like this:

   ```
   Formats
   Help
   SysItems
   ```

```
Topics
```

Topics

1. Select two (2) rows in one column in the spreadsheet.
2. Type the following formula in the first cell and press **Ctrl**+**Shift**+**Enter**.

   ```
          =cwserv|system!topics
   ```

   The resulting display should look like this:

   ```
                       System
   POINT
   ```

Help

1. Select 38 rows in one column in the spreadsheet.
2. Type the following formula in the first cell and press **Ctrl**+**Shift**+**Enter**.

   ```
   =cwserv|system!help
   ```

   The resulting display should look like this:

   ```
   CIMPLICITY to Windows Server (CWSERV):
   0
   This server provides clients with access to current CIMPLICITY point
    data.
   The topics supported include SYSTEM and POINT.
   0
   The SYSTEM topic gives information about this server.
   Supported items under the SYSTEM topic include the following:
       Formats:  Lists all supported formats.
       SysItems: Lists all items supported under the SYSTEM topic.
       Topics:   Lists all supported Topics.
       Help:     Provides information about this server.
   0
   The items under the POINT topic are in the following format:
       point_id.attr[start:endR]
                point_id = tag name of the point [REQUIRED],
                attr = type of point data desired [default = VALUE],
                start and end = beginning and ending indices of an array
    point [default = 0], and
   ```

```
             R = Row formatting for array elements (tab deliminated)
 [default = Column (<CR><LF> deliminated)].
0
Supported items under the POINT topic include the following:
    VALUE:         Point data value. (converted).
    RAW_VALUE:     Point raw data value (non-converted).
    STATE:         Current state of the point.
    TYPE:          Data type.
    LENGTH:        Length of point data.
    ELEMENTS:      Number of elements in the point for an array point.
    SIZE:          Size of point data.
    DISP_FORMAT:   Display format.
    EU_LABEL:      Engineering units label.
    ALARM_HIGH:    High alarm limit.
    ALARM_LOW:     Low alarm limit.
    WARN_HIGH:     High warning limit.
    WARN_LOW:      Low warning limit.
    DISP_HIGH:     High display limit.
    DISP_LOW:      Low display limit.
    INIT_STATE:    Initial state of point.
    LARM_ENABLED:1 if the alarm is currently enabled.
    WARN_ENABLED: 1 if the warning limits are currently enabled.
```

Error Messages

If the VALUE or RAW_VALUE field displays "******", there is a problem reading the point data from the device.

If a field displays "#N/A", the attribute you requested does not contain a value.

If a field displays "#NAME?", you can have one of several problems; for example:

- You have entered an invalid Point ID or attribute name.
- The attribute is not supported by the point type.
- CWSERV does not support the point type for value displays.

When you see a "#NAME?" error in a field, check your project's Status Log file for detailed information on the cause of the problem.

Remote Access of CWSERV on Supported Operating Systems

Remote Access of CWSERV on Supported Operating Systems

You can access CWSERV from a networked supported operating system using Microsoft NetDDE. This requires that you first set up a DDE Share on the Server node where CWSERV is running. Then, when referencing CWSERV through DDE on the client, you must be sure to indicate that you are using NetDDE. The following are a few notes to help in this operation.

1 *(page 512)*	Create a DDE share on the Server node.
2 *(page 513)*	Reference CWSERV from a networked Windows client.

1. Create a DDE Share on the Server Node

1. Open a command prompt and type:

   start DDEShare

2. When the DDE Share window opens, select DDE Shares … on the Shares menu.
3. Click **Add a Share….**

   A DDE Share Properties dialog opens.

4. Enter **CWSERV$** as the **Share Name**.
5. In all three Application Name boxes, enter **CWSERV**.
6. In all three Topic Name boxes, enter **POINT**.
7. Check the **Allow Start Application** checkbox.
8. Check the **Is Service** checkbox.
9. Click **OK**.

2. Reference CWSERV from a Networked NT Client

Once you have setup the DDE Share on the server and CWSERV is running, you can access CWSERV from a client node. The syntax is slightly different than if CWSERV were running locally. The Service or Application name becomes the node name together with NDDE$. The topic name is the DDE Share name you just created. The Item name remains the same as it would locally.

Service/Application name	**\\nodeName\NDDE$**
Topic name	**CWSERV$**

Example in Microsoft Excel

When accessing CWSERV from EXCEL, you must put the Service/Application and Topic names in single quotes because of the '$' characters.

To access the value of the point, GEF_DEMO_ASETPT, from the node ALBP64, type the following:

```
'\\albp64\ndde$'|'CWSERV$'!GEF_DEMO_ASETPT.value
```