ACMSDI Gateway Version 5.5-0S

Release Notes

Publication Date: February 2026

Operating System: VSI OpenVMS IA-64 Version 8.4-2L1 or higher

Kit Name: VSI-I64VMS-ACMSDI-V0505-0S-1.PCSI

VMS Software, Inc. (VSI)
Boston, Massachusetts, USA

ACMSDI Gateway Version 5.5-0S Release Notes

VMS Software

Legal Notice

Table of Contents

1. Introduction

ACMS (Application Control and Management System) is a powerful and sophisticated transaction processing monitor software system for OpenVMS that was originally developed by Digital Equipment Corporation in the early 1980s and remains popular with many large OpenVMS users to this day, most of whom rely on it to run key business-critical applications. Traditional methods of interacting with ACMS-based applications are via green-screen user interfaces (typically TDMS or DECforms) or using the TPware family of products to facilitate remote access to ACMS applications in a client-server fashion. However, TPware has to date only been available for the Windows platform. TPware has also not been updated for many years, and it cannot be used safely by modern multi-threaded Windows applications. For many ACMS users this lack of good integration options can be a serious issue as it becomes increasingly necessary for these business-critical ACMS applications to interact with non-OpenVMS application environments across the enterprise in an efficient manner.

To help ACMS customers address these challenges and to allow customers to get the most out of their legacy ACMS-based applications, VMS Software Inc. (VSI) has recently updated the original TPware and ACMSDI gateway products to provide new and improved facilities for interacting with ACMS applications in a secure and reliable fashion from multiple client operating systems using modern programming languages such as C, Java, Python, C#, PHP, and Lua. This document briefly describes these updates and provides simple examples of how the new and updated features can be used. The updated software is freely available to all ACMS customers, and VMS Software Inc. can provide services to help customers make the best possible use of the updated ACMSDI software package[1].

It is assumed that the reader is familiar with OpenVMS, ACMS, and TPware. It is also assumed that the reader is familiar with developing applications in C/C++ on OpenVMS, Microsoft Windows, or Linux or has some familiarity with the other programming languages mentioned above.

2. What's New in This Release

The current release of ACMSDI gateway introduces the following new features, enhancements, and bug fixes.

Client Sign-in Returns Correct Login Status

The ACMSDI gateway now returns correct status messages for passwords that have expired or are in the pre-expired state.

Previous releases of the gateway could in some situations report that authentication had succeeded even if the password was expired or pre-expired.

HP-UX Client No Longer Supported

The HP-UX ACMSDI client is no longer supported, and the HP-UX client kit is no longer included with the OpenVMS PCSI kit.

Changes to Client OpenSSL Support

The Windows client kit has been updated to use OpenSSL 3.0.16, and OpenSSL support is now statically linked into the ACMSDI client DLLs (both debug and non-debug). It is therefore no longer necessary to deploy the OpenSSL DLLs along with the ACMSDI client DLL. The OpenSSL DLLs are no longer included in the ACMSDI Windows client kit.

[1]The updated VSI ACMSDI package includes the ACMSDI gateway for VSI OpenVMS 8.4-2L1 I64 or higher and TPware client software components for Windows and Linux.

64-bit Windows DLLs

The ACMSDI V5.5-0S kit for Windows includes *only* the 64-bit versions of the client DLLs (debug and non-debug versions). Windows client applications must be linked with the appropriate ACMSDI client DLL and the relevant client DLL must be installed on all Windows systems that will run client applications and must be found in the application PATH.

3. Modifications and Enhancements (cumulative)

The list below summarises the new and updated features that were introduced in the previous versions of the new VSI ACMSDI package, starting with version 5.5-0A.

Corrected Password Expiration Status Message

When connecting to the ACMSDI gateway and using the option ACMSDI_OPT_PWD_EXPIRING to determine the number of hours remaining until the users' password expires, previous versions of the ACMSDI gateway would incorrectly return an expiration status of ACMSDI_PWDEXPIRING even if the number of hours remaining was in fact less than or equal to 0. The gateway now correctly returns an expiration status of ACMSDI_PWDEXPIRED in this situation.

Removal of DECnet and AppleTalk Protocols

The TPware and ACMSDI gateway code historically supported multiple network communications protocols, including TCP/IP, DECnet, and AppleTalk. The AppleTalk protocol was discontinued over a decade ago, and DECnet, even though it is still used by many OpenVMS users for communications between OpenVMS systems, is no longer used for interactions between OpenVMS and other operating systems. For these reasons, the support for both protocols has been removed from TPware and ACMSDI gateway, and the only supported network protocol now is TCP/IP.

Changes to ACMSDI Gateway Logging

The ACMSDI gateway process historically logged operational and error messages to the ACMS SWLUP facility via the SWLUP mailbox. However, due to the manner in which this logging interface was implemented, there was a definite risk that under a heavy operational load, the gateway process would hang in a RWAST state after attempting to log a large number of messages within a very short time period. To avoid this situation, the logging mechanism has been changed to log all operational and error messages to a gateway log file. Logged messages are timestamped to allow them to be correlated with other events that may have occurred on the OpenVMS system or on client systems during the same period. Client connection and identification details (such as usernames and client host details) are also included in logged messages where possible. When starting gateway instances, it is possible (via either the ACMSDI CLI utility or using the ACMSDI$LOGDIR logical name) to specify the location of these log files.

Support for SSL/TLS

The updated ACMSDI software includes optional support for SSL/TLS encryption of all network traffic. Previous versions of the software provided only a very basic algorithm for encrypting messages, which is not considered secure by modern standards. Support for SSL/TLS is provided via OpenSSL and allows the software to take full advantage of the latest ciphers and encryption standards. Both 32-bit and 64-bit versions of the OpenSSL libraries are included with the ACMSDI for Windows kit. Note that these libraries must be installed along with the appropriate ACMSDI DLL (32-bit or 64-bit) on all Windows client systems that will use the ACMSDI interface.

Restrictions:

- ACMSDI gateway cannot simultaneously support both encrypted and non-encrypted traffic.
- SSL/TLS support uses the TLS_AES_256_GCM_SHA384 TLS1.3 cipher suite, which implements the Advanced Encryption Standard with a 256-bit key in Galois/Counter mode (AES 256 GCM) and Secure Hash Algorithm 384 (SHA384). Other cipher suites or TLS versions are currently not supported.

Support for Multiple Connections

Previous versions of the ACMSDI client API did not permit the creation of multiple connections to the same ACMSDI gateway but would instead multiplex a single connection to support multiple sessions. Thus, each session would equate to a separate connection by the gateway to ACMS, but all TCP/IP network traffic for every such session would utilize the same common TCP/IP connection. This approach has some merit and was presumably taken to help conserve network resources when the software was originally developed; however, this is much less of a concern today, and having the ability to create and use multiple connections to the gateway can provide significant advantages in certain situations (particularly for multi-threaded applications in which it can often be desirable for each thread to have its own distinct connection).

The maximum number of concurrent connections that an instance of the ACMSDI gateway will allow can be specified by defining the logical name ACMSDI$MAX_CONN to the desired value or by using the **-c** command line option[2] for ACMSDI$GATEWAY.EXE. ACMSDI$MAX_CONN can be defined at any level visible to the ACMSDI gateway process instance but is typically defined at either the system or the process level. The supplied value for the maximum number of concurrent connections must be between 200 and 400. If the specified value is outside this range, it will be adjusted as appropriate to one or the other of these two values.

Important

By default, the maximum number of concurrent connections allowed is 400. While it should be possible on most systems for an ACMSDI gateway instance to support more than 400 concurrent connections, it can become problematical to set process quotas appropriately, and better management of resources (and better availability) can be achieved by running multiple instances of the gateway.

Support for Linux

In addition to providing 32-bit and 64-bit client support for Windows, a client kit is also provided for Linux. That client can be used to develop Linux-based client applications in a variety of programming languages that can call into ACMS applications (C/C++, Java, PHP, and Python).

Alternative Language Bindings

In addition to C/C++, the updated software also provides the facilities for the development of client applications in Java, Python, PHP, Lua, and C#. Support for these languages is provided via SWIG (http://www.swig.org/) and a new tool (Twiddle) that generates a SWIG-compliant interface definition for the ACMS application from a supplied STDL file for the programming language of choice[3]. The

[2] In order to use this and other command line options mentioned in this document, it will be necessary to edit the file SYS$STARTUP:ACMSDI$RUN.COM to define a foreign command for ACMSDI$GATEWAY.EXE and to run the gateway using the foreign command with any desired command line options.

[3] Although no C# examples are currently provided with the client kits, client applications can be implemented using C# or any other language that is supported by SWIG. This includes scripting languages such as JavaScript, Perl, PHP, Python, Tcl, and Ruby and non-scripting languages such as C#, Go, Java, and others.

generated interface is linked with ACMSDI client library to facilitate communication with the ACMSDI gateway and exchange data with the target ACMS application.

Interface Code Generation Facilities

As mentioned above, the updated ACMSDI client kit includes code generation facilities to generate client interfaces for a variety of programming languages based on a supplied STDL input file, which completely defines the ACMS application in terms of the tasks and workspaces. The supplied STDL file is parsed by the Twiddle tool to generate a SWIG-compliant interface that can then be compiled and linked with the ACMSDI client library and client language runtime.

It is not uncommon for ACMS applications to implement large numbers of tasks and use many different workspaces. If you do not wish to generate interface code for all ACMS tasks and workspaces, you can edit the STDL file to include only those tasks and workspaces that are specifically required.

PCSI Installation Kit

The ACMSDI gateway software is now provided as an easily installed PCSI kit, bringing it in line with most other OpenVMS software kits provided by VMS Software, Inc. The kit includes the ACMSDI gateway software, client API, example client code for various programming languages, and a simple example ACMS application that can be used in conjunction with the example client programs.

Generic Microsoft .NET Wrapper

The ACMSDI client kit also includes a generic .NET wrapper that provides direct access from .NET application code to the core ACMSDI client API (`acmsdi-x64.dll`). The wrapper (provided by the two 64-bit DLLs `TpwareClient.DLL` and `TPware.DLL`) provides a .NET class library that can be called from any C# or Visual Basic .NET application.

Support for External Authentication (via ACME LDAP)

The ACMSDI gateway can optionally authenticate users via ACME LDAP, using the supplied LDAP credentials for user authentication at the OpenVMS level, and the local username determined from the ACME LDAP mapping file for at ACMS level.

Management CLI

The ACMSDI OpenVMS kit includes a simple command line utility (ACMSDI.EXE) that can be used to start, stop, and list ACMSDI gateway instances (started on different TCP/IP ports), as well as to view various metrics (user connection details, number of concurrent connections, number of successful and failed logins, and so on). The command line utility can be found in the directory ACMS$ROOT:[BIN] and can be run only by suitably privileged accounts, such as SYSTEM. A detailed description of the utility and the functions it provides can be found in the Help section of the utility (which can be accessed via the **`HELP`** command).

Aside from being useful when performing general gateway administrative and monitoring functions, a common use of the CLI utility is to implement customised startup and shutdown procedures to start and stop multiple ACMSDI gateway instances, as described in *Section 8, "Customised Start-up and Shutdown Procedures"*.

Detection and Reporting of Expired Passwords

Previous versions of the ACMSDI gateway did not detect and report expired OpenVMS user passwords correctly in certain circumstances, such as when the user password was pre-expired. This issue has been resolved.

SSL/TLS Bug Fixes and Enhancements

- The ACMSDI.EXE CLI utility can now be used when SSL support is enabled. On startup, if the utility determines that SSL is enabled for the gateway (the ACMSDI$SSL_ENABLE logical name is defined), the utility will use secure connections when connecting to the gateway to retrieve data.

- There was a bug in the gateway code that could cause the gateway to fail when cleaning up and closing SSL client connections (the bug did not impact non-SSL client connections). It was found that memory associated with one of the SSL-related structures could potentially be freed twice during client disconnection, resulting in an ACCVIO. This bug has now been fixed.

Enhanced Logging of Various Events

- On gateway startup, the startup message written to the log file now includes the name of the OpenVMS node and the gateway version. This information can be useful when using the gateway in a clustered environment where multiple gateway instances written on different cluster nodes share a common location for log files. For gateway instances that were started using the management CLI, log file names include the OpenVMS node name and the port number used by the gateway instance.

- When a client disconnects from the gateway unexpectedly (i.e., without performing a sign-out and orderly disconnection), additional details regarding the event are logged, including the associated username and submitter ID.

- When a client connects to the gateway, usernames and client host details (if available) will be included in the message written to the gateway log file.

- Error checking and logging associated with client disconnections and certain network-related errors has been improved. Confusing error messages caused by unexpected client disconnects have been eliminated; incorrectly reported error status codes have been corrected.

New Ping Function

The client API now includes the new ping function `acmsdi_ping()` that can be used by client applications to verify gateway connectivity and as a heartbeat to prevent network components from dropping client connections during extended periods of client inactivity. This function accepts two arguments – the first one is a pointer to a submitter ID returned by `acmsdi_sign_in()`, and the second one may be either NULL or a pointer to a byte array that is at least 4 bytes in length. If the second parameter is not NULL, `acmsdi_ping()` will copy the 4-byte response message received from the ACMSDI gateway (that should contain the letters PONG) into the supply array. If the return status of `acmsdi_ping()` is any value other than 0, it should be assumed that an error has occurred and there is a problem with the gateway connection. The example programs "ping01.c" and "ping02.c" included in the kits illustrate using the new ping function in conjunction with an OpenVMS timer (ping02.c) and with UNIX-style alarms (ping01.c).

It is also possible to call the function from a separate thread, however care should be taken to ensure that calls to the ping function do not overlap with task calls being executed on another thread. Note that the ping function can only be used in blocking mode.

New Logical Name to Specify Location of Log Files

The logical name ACMSDI$LOGDIR can be used to specify the location (directory) where gateway log files will be written. This logical name may be defined at any level that is visible when starting the gateway using ACMSDI$STARTUP.COM or when using the management CLI, however it would typically be defined at the system level. The logical name must translate to a valid directory specification

and gateway processes must have write access to the directory in question. It is also possible to specify the location of log files using the **/LOGDIR** qualifier when starting gateway instances using the management CLI (see example below) and specifying the log file location in this way takes precedence over a location specified by the ACMSDI$LOGDIR logical name. As noted previously, for gateway instances started using the management CLI, log file names include the OpenVMS node name and the port number used by the gateway instance.

Enhanced Handling of TCP/IP Keepalive

Handling of TCP/IP keepalive functionality has been enhanced in both gateway and client to ensure prompt detection of failed connections and to reduce the chances of connections being dropped due to inactivity.

Support for SHOW VERSION Command

The ACMSDI.EXE CLI utility now supports a **SHOW VERSION** command that can be used to show the version number and build date/time of the gateway.

4. Installation

This section briefly describes the installation of the updated ACMSDI gateway and client API software on OpenVMS, Microsoft Windows, and Ubuntu Linux. As noted previously, the OpenVMS installation has been updated to be a PCSI kit. The Windows kit is provided as a standard Windows MSI installer package. The Linux kit is provided as a compressed tar file that can be unpacked and installed locally (single user) or system-wide.

The ACMSDI client kits for Linux and Windows are bundled with the OpenVMS ACMSDI kit and can be found after the installation in the ACMSDI$ROOT:[KITS] directory. These kits can be copied via SFTP or otherwise to Windows and Linux systems and installed as described below.

4.1. Installing on OpenVMS

Requirements

The ACMSDI gateway and client API software for OpenVMS can be installed on a system that meets the following requirements:

- VSI IA-64 OpenVMS Version 8.4-2L1 or higher
- VSI TCP/IP Services, HPE TCP/IP Services for OpenVMS, or MultiNet TCP/IP
- VSI ACMS V5.3-2 or an equivalent
- It is recommended that ACMSDI is installed on an ODS-5 file system.
- VSI strongly recommends that you uninstall any old HP versions of the ACMSDI gateway software before installing the VSI version[4].

If you intend to develop client applications using the ACMSDI client API on VSI OpenVMS, some or all the following software products are also required:

[4]While the network protocol used by the VSI version is compatible with that of old HP versions, the SSL/TLS transport is not supported by the HP versions; other aspects of the software (such as the locations of some installed files and log files) are also not compatible with old HP versions of the software.

- VSI C V7.4
- SWIG V3.0-5
- OpenJDK V8.0-222B[5]
- Python 3.8[5]
- PHP V5.6-10J[5]
- Lua V5.3-5A[5]

Note that the ACMSDI gateway server is statically linked with OpenSSL libraries and therefore does not require any particular version of OpenSSL.

The ACMSDI kit for OpenVMS also includes forms manager components that can be used (with some restrictions) in conjunction with ACMS applications that use TDMS or DECforms; however, it is not mandatory for either of these products to be installed to be able to install and use ACMSDI gateway server.

Installation

The OpenVMS kit is provided as an OpenVMS PCSI kit that can be installed by a suitably privileged user using the following command:

```
$ PRODUCT INSTALL ACMSDI
```

The installation will then proceed as follows (output may differ slightly from that shown below depending on various factors):

```
Performing product kit validation of signed kits ...

The following product has been selected:
    VSI I64VMS ACMSDI V5.5-0S              Layered Product

Do you want to continue? [YES]

Configuration phase starting ...

You will be asked to choose options, if any, for each selected product and
for any products that may be installed to satisfy software dependency
requirements.

Configuring VSI I64VMS ACMSDI V5.5-0S: ACMSDI gateway for OpenVMS

    © Copyright 2025 VMS Software Inc.

    VSI Software Inc.

* This product does not have any configuration options.

Execution phase starting ...

The following product will be installed to destination:
    VSI I64VMS ACMSDI V5.5-0S              DISK$I64SYS:[VMS$COMMON.]

Portion done: 0%...10%...50%...90%...100%

The following product has been installed:
    VSI I64VMS ACMSDI V5.5-0S              Layered Product
```

[5]Only required if you intend to develop client applications using this language.

```
VSI I64VMS ACMSDI V5.5-0S: ACMSDI gateway for OpenVMS

    Post-installation tasks are required.

    To start the ACMSDI gateway at system boot time, add the
    following lines to SYS$MANAGER:SYSTARTUP_VMS.COM:

        $ file := SYS$STARTUP:ACMSDI$STARTUP.COM
        $ if f$search("'''file'") .nes. "" then @'file'

    To shutdown the ACMSDI gateway at system shutdown, add the
    following lines to SYS$MANAGER:SYSHUTDWN.COM:

        $ file := SYS$STARTUP:ACMSDI$SHUTDOWN.COM
        $ if f$search("'''file'") .nes. "" then @'file'
```

Post-Installation Configuration

After the installation has successfully completed, VSI recommends that you perform the following actions:

1. Include the commands displayed at the end of the installation procedure into the files SYSTARTUP_VMS.COM and SYSHUTDWN.COM. This ensures that the ACMSDI gateway server is started and stopped when OpenVMS is booted and shut down.

   Startup and Shutdown Order

   The ACMSDI gateway server must be started *after* ACMS and TCP/IP, and should be shut down *before* ACMS.

   It is not necessary for any ACMS applications to be running when the gateway is started, as the gateway will only attempt to connect to an ACMS application upon receiving an explicit request from a client to do so; however, the ACMS system must be started for the gateway to start correctly.

   ---

2. Manually start the ACMSDI gateway and confirm that it is operating correctly. To do so, follow these steps:

   a. Run the command **`SHOW SYSTEM`** and make sure that the displayed output does not contain the process named ACMSDI$GATEWAY.

   b. Run the gateway startup procedure.

   c. Verify that the gateway is running by issuing the command **`SHOW SYSTEM`** and checking the displayed output for the process named ACMSDI$GATEWAY.

      Note

      By default, the ACMS gateway process uses port 1023. If another process is already using this port, the ACMSDI gateway process will fail to start. To instruct the gateway to use another port, you can define the desired port number using the logical name ACMSDI$TCPIP_PORT. This logical name may be defined at the system level or it can be defined in SYS$STARTUP:ACMSDI$RUN.COM at the process level.

      ---

   d. Verify that the logical name ACMSDI$ROOT is defined at system level and points to the root of the ACMSDI installation tree.

e. Examine the log file ACMSDI$ROOT:[LOGS]ACMSDI.LOG[6] and verify that it includes no errors or warnings.

Note

A new version of this log file will be created whenever the ACMSDI gateway is restarted; VSI recommends that appropriate processes are put in place to backup and purge old log files.

Required Process Privileges and Quotas

The privileges TMPMBX, NETMBX, BYPASS, SYSPRV, and DETACH are currently required to run the ACMSDI startup and shutdown scripts. Note that the ACMSDI gateway process (run as a detached process) will inherit the default privileges for the username under which it is started.

The following process quotas should be sufficient for the ACMSDI gateway in most cases:

```
Maxjobs:          0  Fillm:       256  Bytlm:       128000
Maxacctjobs:      0  Shrfillm:      0  Pbytlm:           0
Maxdetach:        0  BIOlm:       150  JTquota:       4096
Prclm:           50  DIOlm:       150  WSdef:         4096
Prio:             4  ASTlm:       300  WSquo:         8192
Queprio:          4  TQElm:       100  WSextent:     16384
CPU:         (none)  Enqlm:      4000  Pgflquo:     256000
```

If the ACMSDI gateway is expected to support large numbers of simultaneous connections, then it may also be necessary to increase the CHANNELCNT system parameter – although remember that an individual gateway instance will not allow more than 400 concurrent connections.

Note that by default, the ACMSDI gateway does not permit connections for users whose accounts have the DISUSER flag set. This behaviour can be overridden either by defining the logical name ACMSDI$ALLOW_DISUSER (to anything) or by using the **-a** command line option for ACMSDI$GATEWAY.EXE. The logical ACMSDI$ALLOW_DISUSER can be defined at any level that is visible to the ACMSDI gateway process.

If you are developing clients for OpenVMS and intend to run those clients on the same OpenVMS system (or in the same OpenVMS cluster) as your ACMS application(s), it is not necessary to start the ACMSDI gateway, as OpenVMS-based clients use the ACMS SI API to communicate directly with ACMS applications as opposed to communicating via the ACMSDI gateway. However, from a development perspective, the logical name ACMSDI$ROOT still needs to be defined in order for developers to be able to build applications (generated code references C header files in uses C header files in ACMSDI$ROOT:[INCLUDE] and the client build process links with the object library ACMSDI$ROOT:[LIB]LIBAGENT.OLB).

4.2. Installing on Linux

Requirements

Regardless of the language that you intend to use for developing client applications, the following products must be installed:

- OpenSSL development libraries (OpenSSL 3.0.16 or higher)

[6] If you defined the logical name ACMSDI$LOGDIR prior to starting the gateway, the log file will be created in the directory to which the logical name is pointing.

- gcc 9.3.0 or higher (alternatively the Clang compiler may be used)
- SWIG 4.0.1 or higher (see http://www.swig.org)

If you are intending only to build client applications using C/C++, then installing the above products is sufficient. However, if you wish to build applications using Java, Lua, Python, PHP, or any other language supported by SWIG, then you must also install development kits for those languages.

Note that, depending on the language in question, installing just the runtime for that language may not be sufficient, as SWIG-generated client interface code must typically be compiled and linked with the language runtime, which generally requires access to header files and sometimes shared or archive libraries that are provided only with the language development kits.

Installation

The kit is provided as a compressed tar file (`acmsdi-linux-5.5.0s.tar.gz`) that can be installed locally or system-wide. The usage of a particular installation mainly depends on the definition of the environment variable ACMSDI_HOME, which points to the top-level directory of the extracted `tar.gz` file.

To install the Linux client kit system-wide under `/usr/local`, follow these steps:

1. Extract the contents of the compressed tar file as shown below:

   ```
   $ cd /usr/local
   $ sudo tar xvf $HOME/acmsdi-linux-5.5.0s.tar.gz
   ```

   This will create the top-level directory `/usr/local/acmsdi` for the installation.

2. To start using the software, define the environment variable ACMSDI_HOME and include this definition in your .profile file (or a similar login script, depending on the Linux shell that you use):

   ```
   $ ACMSDI_HOME=/usr/local/acmsdi
   $ export ACMSDI_HOME
   ```

Working with Example Applications

Once the kit has been installed and the environment variable ACMSDI_HOME has been defined, the best way to become familiar with developing applications using the ACMSDI client kit is to try building and running one or more of the example programs provided with the kit under the directory `$ACMSDI_HOME/examples`; however before attempting to build any of the examples it is necessary to ensure that all prerequisite software products are installed, and before attempting to run any of the examples, the associated ACMS application must be running on the target OpenVMS system[7].

To build any of the example, copy the example to a local directory, **`cd`** into that directory and type **`make`** to compile and link the interface. Note that the makefiles may need to be modified to correctly specify paths to language-specific header files and libraries, and the example code will need to be modified to specify the host, username, and password details for the applicable to your OpenVMS and ACMS application environment. For some languages (such as PHP) it will also be necessary to copy the shared library created by the build procedure into a specific location, or to ensure that the directory in which the shared library resides is included in LDPATH. A detailed description of these steps for each language is beyond the scope of this document, and it is expected that the reader will be aware of any such requirements.

[7] As for the Windows kit discussed below, the examples provided with the Linux kit make use of two ACMS applications, namely the "employee" example application that is included in the ACMS layered product kit and the "multi" example, which is included in the OpenVMS ACMSDI kit and can be found in ACMSDI$ROOT:[EXAMPLES.MULTI]. In order to run the Linux examples, these ACMS applications must be running and accessible.

4.3. Installing on Windows

The Microsoft Windows kit is provided as a simple ZIP file named `Libacmsdi-5.5.0f.zip` that can be easily expanded and used by any Windows user. The following text assumes that the contents of the ZIP file have been extracted into the folder `Libacmsdi` under the users' home directory.

Several examples in various languages are included with the Windows kit, including examples for C/C++, Java, Python, PHP, and Lua. In order to build and run the sample applications, the following points should be noted (analogous considerations apply when building and running your own applications):

- Irrespective of the programming language being used, the Microsoft Windows Visual Studio C/C++ compiler must be installed. Microsoft Visual Studio 2019 or similar is recommended, and many of the examples include Visual Studio solution files and assume that the Visual Studio C/C++ compiler will be used.
- When using any of the other languages (Java, Python, PHP, or Lua), in addition to the Visual Studio C/C++ compiler and the programming language in question being installed[8], it is also necessary to install the SWIG interface generator (for details, see http://www.swig.org/), as SWIG is required to build the interface between these languages and the ADMSDI DLL. When working with Java it may also be desirable to use an IDE such as Eclipse to edit and debug Java code. With regard to Python, while it is possible to use Python 2.7, it is strongly recommended that Python 3.8 (or later) should be used.
- The Windows kit includes 64-bit debug and non-debug libraries that can be used when building client applications. Code must be linked with either `acmsdi-x64.lib` (for production release builds) or `acmsdi.lib` (for debug builds). These library files can be found in `Libacmsdi\lib`. The associated DLLs are `acmsdi-x64.dll` and `acmsdi.dll` respectively.
- The folder `Libacmsdi\bin` must be included in the user or system PATH in order for applications to find ACMSDI DLL's at runtime, or alternatively the DLLs may be copied to another location that is included in the user or system PATH. Note that the DLLs TpwareClient.DLL and TPware.DLL are required when using the generic .NET wrapper; they are no required for applications written in C/C++ or for applications that are created using the SWIG interface generator. In addition to DLL's, this folder also contains the file `twiddle.exe`, which is used as described previously to generate a SWIG interface from a supplied ACMS STDL interface definition. This program also needs to be in the user or system PATH in order to build applications.
- When compiling C and C++ code, the include path for the compiler should include the folder `Libacmsdi` in order for the compiler to find the header file `acmsdi.h`, or alternatively this header file can be copied to another folder than is already specified in the include path.
- The folder `Libacmsdi\examples` includes sample certificates that can be used in conjunction with sample applications when using SSL/TLS; however, it should be noted that these are self-signed certificates, which may not be appropriate for some environments, and additionally the certificates may be expired. It is therefore recommended that developers create their own self-signed certificates for development and testing purposes (if appropriate) or that official certificates are used. Self-signed certificates should not be used in a production environment.

In addition to these general comments about installing and using the Windows ACMSDI kit, the following language-specific specific points should be noted with regard to building and running the example applications for C, Java, Lua, and Python[9].

[8] Note that for in order to build Java applications using the ACMSDI kit for Windows it will be necessary to install the JDK; it will not be sufficient to install only the Java runtime.

[9] Note that these notes do not discuss the use of the generic .NET wrapper; information on using the generic wrapper and associated example code will be provided with future releases of the software.

Note that the examples provide with the Windows kit make use of two ACMS applications, namely the "employee" example application that is included in the ACMS kit and the "multi" example, which is included in the OpenVMS ACMSDI kit and can be found in ACMSDI$ROOT:[EXAMPLES.MULTI]. In order to run the Windows examples, these ACMS applications must be running and accessible. The "multi" example is a simple ACMS application that uses several numeric data types commonly used by COBOL programs on OpenVMS, such as packed decimal, and serves to test the conversion routines provided by the ACMSDI client API to handle such data types.

Before attempting to run any of the examples, ensure that the ACMSDI DLL's can be found in the user or system PATH, as discussed above. Note that this point is true for all of the examples, not only those written in C. For all examples it will also be necessary to change the OpenVMS login details and the address or host name for the ACMSI gateway. The supplied OpenVMS username must also be defined in the ACMS User Definition Utility (UDU) and have permissions to submit tasks from ACMS agent programs.

- To build the C examples (`Libacmsdi\examples\c`), open c.sln in Visual Studio, modify any project properties as required (such as the include path and the library path), and build the example application(s) of interest. Note that SWIG is not required in order to build the C examples.

- To build the Java example (`Libacmsdi\examples\java`), firstly modify the file `prepare-win.cmd` to correctly specify the paths to SWIG and `twiddle.exe` (if they are not otherwise in PATH), and run the script to generate the SWIG interfaces for each of the ACMS applications. Once the SWIG interfaces have been successfully generated, open `java.sln` with Visual Studio, modify the project properties as appropriate, and build the project. Note that in addition to specifying the correct include path for acmsdi.h and the correct library path for the ACMSDI link library, it will also be necessary to include in these path definitions paths to the JDK include directory and link library.

  Note that the Visual Studio project does not compile the Java code for the example (demo.java). This may be compiled manually using the `javac` command, or you may wish to use Eclipse or another such IDE to compile and run the Java client application. The Visual Studio project builds the two DLL's `emp.dll` and `multi.dll`, which are the interfaces for the "employee" and "multi" ACMS applications, respectively. In order to run the Java example, these DLL's must be able to be found in the system or user Windows PATH.

- To build the Lua example (`Libacmsdi\examples\lua`), as for Java, firstly modify `prepare-win.cmd` in the Lua example folder to correctly specify the paths to SWIG and `twiddle.exe`, and run the script to generate the SWIG interfaces for each of the ACMS applications. Once the interfaces have been successfully generated, open `lua.sln` with Visual Studio, modify the project properties as appropriate, and build the project. In addition to specifying the correct paths for the ACMSDI header file (`acmsdi.h`) and ACMSDI link library (`acmsdi.lib` or `acmsdi-x64.lib`), it will also be necessary to include in these path definitions paths to the header files and link library for Lua.

  As with the Java example discussed above, the Visual Studio project for the Lua example builds the two DLL's `emp.dll` and `multi.dll`, and in order to run the example, these DLL's must be able to be found in the system or user Windows PATH.

- Building the Python example follows much the same steps as for Lua. Firstly modify `prepare-win.cmd` in the Python example folder as required to correctly specify the paths to SWIG and `twiddle.exe`, and run the script to generate the SWIG interface for each of the two ACMS applications. Once the interfaces have been successfully generated, open `python.sln` with Visual Studio, modify the project properties as appropriate, and build the interfaces. In addition to specifying the correct paths for the ACMSDI header file (`acmsdi.h`) and ACMSDI link

library (`acmsdi.lib` or `acmsdi-x64.lib`), it will also be necessary to include in these path definitions paths to the Python C header files and link library.

Note that for the Python example, the Visual Studio project builds the two DLL's `_emp.pyd` and `_multi.pyd` as opposed to `emp.dll` and `multi.dll`. These files must reside in the system or user Windows PATH in order to run the Python example.

5. TCP/IP Keepalives

The ACMSDI gateway and client software enables TCP/IP keepalives for all connections. As of version ACMSDI V5.5-0P, it is possible to control the behaviour of these keepalive settings with regard to how quickly the gateway and client will initiate keepalive checking to detect stale connections and (perhaps more importantly) to help prevent inactivity from disconnecting the channel. For example, it can be a common issue for applications running behind a NAT proxy or a firewall to be disconnected without a reason. This behaviour is often caused by the connection tracking procedures implemented in proxies and firewalls, which keep track of all connections that pass through them. Because of the physical limits of these systems, they can only keep a finite number of connections in their memory. The most common and logical policy is to keep newest connections and to discard old and seemingly inactive connections first.

By default, the ACMSDI gateway and client will start to check connections after 300 seconds (5 minutes), however this threshold can be controlled via the logical name ACMSDI$TCP_KEEPIDLE on OpenVMS or the environment variable ACMSDI_KEEP_IDLE on Windows and Linux/UNIX, with the specified value being an integer in the range from 120 to 7200, representing the number of seconds before the first keepalive probe is issued. From an OpenVMS perspective, the logical name may be defined at any level, however it will typically need to be defined the system level.

6. Using SSL/TLS

Support for end-to-end SSL/TLS encryption of all network traffic between client applications and the OpenVMS-based ACMSDI gateway process is provided by OpenSSL. The use of SSL/TLS encryption is optional and is enabled on the gateway by the definition of the logical name ACMSDI$SSL_ENABLE and logical names that specify the location of key and certificate files and whether client certificates are to be verified or not. C client support for SSL/TLS encryption is provided via the new options that can be specified when calling `acmsdi_sign_in()` to connect to the ACMSDI gateway; for other client languages, the new methods are provided to specify key and certificate file details and to connect to the gateway via SSL/TLS. For details, see *the section called "Logical Names"*, *the section called "New C Client Options"*, and *the section called "Other Client Languages"* below.

Logical Names

Logical name	Value(s)/description
`ACMSDI$SSL_ENABLE`	Defining this logical name as either 1 or TRUE enables SSL/TLS encryption on the ACMSDI gateway. Defining the logical name to any other value will have no effect.
`ACMSDI$SSL_CERT`	This logical name should be used to specify the certificate file path.
`ACMSDI$SSL_KEY`	This logical name should be used to specify the private key file path.

Logical name	Value(s)/description
ACMSDI$SSL_VERIFY_CLIENT	Defining this logical name to either 1 or TRUE will cause the ACMSDI gateway to verify the client certificate. Defining the logical name to any other value will have no effect.
ACMSDI$SSL_CA	This logical name should be used to specify the Certificate Authority (CA) bundle file path.

Warning

- If the logical name ACMSDI$SSL_ENABLE is not defined, then the definition of any of the other logical names listed above will have no effect.
- It should also be noted that the logical name ACMSDI$SSL_CA must be defined if the gateway has been instructed to verify client certificates by defining ACMSDI$SSL_VERIFY_CLIENT.

New C Client Options

The following table lists the new client option codes that can be used in C/C++ client code to enable SSL/TLS encryption for all communication with the ADMSDI gateway and to specify the locations of client certificate and key files. It should be noted that the specification of client certificate and key files is optional and is only required if the ADMSDI gateway has been configured to verify client certificates.

Option code	Value(s)/description
ACMSDI_OPT_SSL_ENABLE	Specifying this option code with a value of 1 instructs the client to use SSL/TLS encryption for all communication with the ACMSDI gateway.
ACMSDI_OPT_SSL_CERT	This option code can be used to (optionally) specify the client certificate file path.
ACMSDI_OPT_SSL_KEY	This option code can be used to (optionally) specify the client key file path.

The following code fragment illustrates the use of these new option codes to enable encryption and to specify values for the locations of the key and certificate files, using the new structures (`ssl`, `ssl_cert`, and `ssl_key`) that have been added to the ACMSDI_OPTION union to support the new SSL/TLS options. As noted previously, the specification of key and certificate files is only required if the ACMSDI gateway has been configured to verify client certificates.

```
ACMSDI_OPTION opts[4];

opts[0].ssl.option = ACMSDI_OPT_SSL_ENABLE;
opts[0].ssl.enable = 1;

opts[1].ssl_cert.option = ACMSDI_OPT_SSL_CERT;
opts[1].ssl_cert.path = "../cert.crt";
opts[2].ssl_key.option = ACMSDI_OPT_SSL_KEY;
opts[2].ssl_key.path = "../cert.key";

opts[3].option = ACMSDI_OPT_END_LIST;
```

Other Client Languages

For other programming languages, the methods `acmsSetSslCert()` and `acmsSignInSSL()` are provided in the client interface code generated by the Twiddle utility. The `acmsSetSslCert()` method allows developers to specify client key and certificate details, and the `acmsSignInSSL()` method can then subsequently be used to connect to the ACMSDI gateway using SSL/TLS encryption.

The following code fragment illustrates the use of these two methods with the Lua programming language:

```
emp = require("emp")

emp.acmsSetSslCert("../cert.crt", "../cert.key")
h = emp.acmsSignInSSL("BIGGLES", "BIGGLES1234", "10.10.100.11")

if h == -1 then
   print("Could not login to ACMS")
   os.exit()
end
```

7. Using ACMSDI With ACME LDAP

As noted in the introduction, the ACMSDI gateway can be configured to authenticate users via ACME LDAP, such that the supplied LDAP username/password credentials are used for authentication at the OpenVMS level, and access to ACMS uses the corresponding local OpenVMS username determined from the ACME LDAP username mapping file, which will be loaded by ACMSDI on startup if ACME LDAP authentication has been enabled. Assuming that ACME LDAP is configured and enabled on the OpenVMS system in question, configuring the ACMSDI gateway to use it involves the following steps:

1. Define the logical name ACMSDI$AUTH_USE_ACM at system level to anything. When this logical name is defined, the ACMSDI gateway will establish the necessary internal structures for ACME LDAP authentication and will attempt to load the local username mapping file on startup (see the next step).

2. Define the logical name ACMSDI$LDAP_USER_DB at system level to point to the local username mapping file. If ACME LDAP-based authentication is enabled and this logical name is not defined (or if the file pointed to by this logical name does not exist), the gateway will log an error and will fail to start.

Assuming that the above two logical names are appropriately defined, the ACMDI gateway will attempt to authenticate users via ACME LDAP and connect to ACMS using the corresponding local OpenVMS username read from the mapping file. If no username mapping exists for the supplied LDAP username, ACMSDI will attempt to connect to ACMS using the LDAP username (which in most cases will fail).

Notes

- The contents of the local username mapping file are loaded into memory on gateway startup, such that if any new mappings are added while the gateway is running, they will not be picked up until the gateway is restarted.

- While the logical name ACMSDI$LDAP_USER_DB will typically point to the ACME LDAP local username mapping file, this is not strictly a requirement, and it is possible to create a mapping file specifically for use by ACMSDI – but the file must conform to the format used by ACME LDAP.

The file access permissions on the mapping file must be such that the file can be opened for reading by the ACMSDI gateway process.

8. Customised Start-up and Shutdown Procedures

By default, the ACMSDI gateway startup procedure SYS$STARTUP:ACMSDI$STARTUP.COM will start a single instance of the gateway, and the process ACMSDI$GATEWAY will be listening on TCP/IP port 1023; on shutdown, the procedure ACMSDI$SHUTDOWN.COM will stop this process. It is possible to define an alternative port for the gateway using the logical name ACMSDI$TCPIP_PORT, but otherwise, there is little flexibility provided by the default startup and shutdown procedures.

For various reasons, it is sometimes desirable to run multiple gateway instances, listening on different ports and with potentially different operational characteristics. For example, it may be desirable to use different port numbers for development and testing, or it may be necessary to run multiple instances of the gateway in production to support large numbers of users or to provide some level of separation between different groups of users. To facilitate these and other such requirements, the ACMSDI gateway software provides a simple means of customising the gateway startup and shutdown procedures.

On startup, the ACMSDI$STARTUP.COM procedure looks for the site-specific startup file SYS$STARTUP:ACMSDI$CUSTOM_STARTUP.COM and, if it is present, uses it instead of performing the default gateway startup. Likewise, on shutdown, the ACMSDI$SHUTDOWN.COM procedure looks for the site-specific script SYS$STARTUP:ACMSDI$CUSTOM_SHUTDOWN.COM and; if found, runs it instead of performing the default shutdown sequence. In this way, system administrators can implement their own site-specific startup and shutdown procedures without the need to modify the standard startup and shutdown procedures provided by the ACMSDI product.

The following site-specific startup and shutdown scripts illustrate the use of this customisation facility. The ACMSDI CLI tool is used to start and stop three gateway instances, listening on different ports, and writing logs to DKA100:[LOGS] instead of the default log directory[10].

```
$! acmsdi$custom_startup.com
$
$ run acmsdi$root:[bin]acmsdi.exe
start gateway/port=3320/logdir=dka100:[logs]
start gateway/port=3321/logdir=dka100:[logs]
start gateway/port=3322/logdir=dka100:[logs]
exit
$
$ exit
```

The above site-specific command procedure will start three instances of the ACMSDI, listening on ports 3320, 3321, and 3322. The following site-specific shutdown procedure will shut down all active gateway instances.

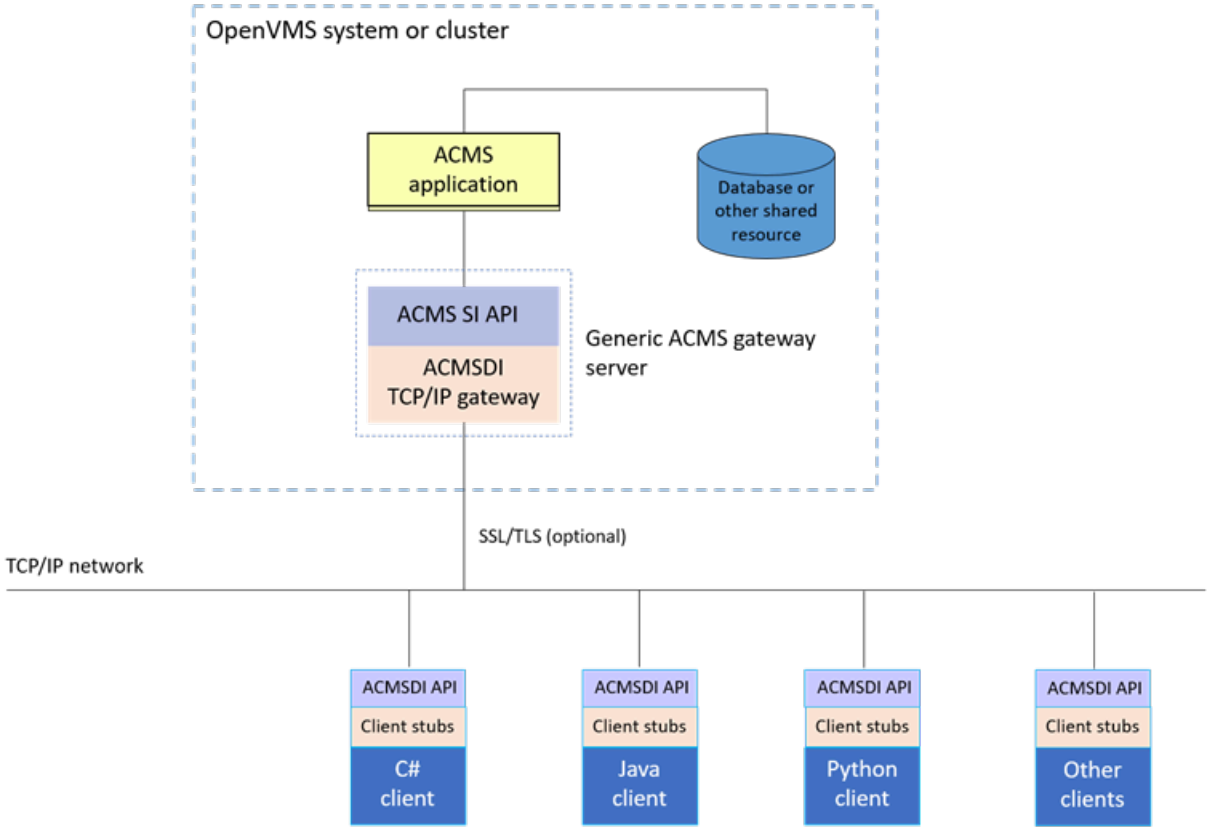
```
$! acmsdi$custom_startup.com
$
$ run acmsdi$root:[bin]acmsdi.exe
stop gateway/all
exit
```

[10]Be aware that specifying the location of log files using the **/LOGDIR** qualifier takes precedence over any directory location specified by the logical name ACMSDI$LOGDIR.

```
$
$ exit
```

9. Summary

The following diagram illustrates some of the key new features of the updated solution, including support for SSL/TLS encryption and the ability to implement client applications using a variety of programming languages on either Windows or Linux.



In summary, the ACMSDI gateway and TPware client components have been extensively updated to better address the integration of ACMS applications with non-OpenVMS application environments through the inclusion of support for multiple popular modern programming languages and the addition of various new features such as support for SSL/TLS encryption of all data transmitted over network links, better support for multi-threaded client application environments, and support for both Windows and Linux as client platforms, including support for both 32-bit and 64-bit Windows code. These enhancements provide users with considerably flexibility in terms of client development and integration options, and it is anticipated that additional functionality will be included in future releases of the product.