What was old is new again

Resurrecting the ACMSDI gateway and adding some new bells and whistles

Brett Cameron (brett.cameron@vmssoftware.com), February 2021

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 1980's and remains popular with many large OpenVMS users to this day, most of whom rely upon it to run their 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 TCPware 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 other 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 assets, 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 system platforms using modern programming languages such as C, Java, Python, C#, PHP, and Lua. This paper 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. is able to 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 familiarity with one of the other programming languages mentioned above.

Modifications and enhancements

The following list summarises the new and updated features of the VSI ACMSDI package. As noted previously, the software has been significantly enhanced to include new functionality and to support a wider range of platforms and client programming languages, making the software more generally useful and applicable to a wider range of client development and integration scenarios.

- <u>Removal of DECnet and AppleTalk protocols</u>:

  The TPware and ACMSDI gateway code historically supported multiple network communications protocols, including TCP/IP, DECnet, and AppleTalk. While DECnet is still used by many OpenVMS users for communications between OpenVMS systems, it is longer used from other operating system platforms to interact with OpenVMS, and the AppleTalk protocol was discontinued over a decade ago. Accordingly, there is no need for TPware and the ACMSDI gateway to support

[1] The updated VSI ACMSDI package includes the ACMSDI gateway for VSI OpenVMS 8.4-1H1 I64 or higher and TPware client software components for Windows and Linux.

these protocols, and the code associated with these protocols has been removed from the software such that the only protocol now supported 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, the manner in which the ACMSDI gateway code implemented this logging interface was somewhat fragile such that there was a definite risk under heavy operational load that the gateway process could find itself hung in a RWAST state if logging large number of messages within a very short time period. To avoid this situation from occurring logging has been changed to instead log all operational and error messages to the 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 time period. Where possible client connection and identification details are also included in logged messages.

- 32-bit and 64-bit Windows DLLs:

  Previous versions of the ACMSDI Windows client DLL were 32-bit only (used 32-bit pointers) and could not be used with some modern 64-bit only programming language environments. The new ACMSDI kit for Windows includes both 32-bit and 64-bit versions of the client DLL, making it possible to implement fully 64-bit client applications and ensuring better compatibility with modern programming languages and Microsoft Windows operating system versions. Windows client applications must be linked with the appropriate version of the ACMSDI client DLL. The DLL must be installed on all systems that will run client applications and must be found in the application `PATH`.

- 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 and these 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.

  Note that the ACMSDI gateway cannot simultaneously support both encrypted and un-encrypted traffic; it is all one or the other. It is also not currently possible to specify specific cipher suites or TLS versions when enabling TLS/SSL support; it is hoped that such functionality will be provided in the future. SSL/TLS support uses the TLS_AES_256_GCM_SHA384 TLS1.3 cipher suite, which implements the Advanced Encryption Standard with 256bit key in Galois/Counter mode (AES 256 GCM) and Secure Hash Algorithm 384 (SHA384).

- 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 by supporting multiple sessions, whereby 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 when the software was originally developed to help conserve network resources; 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.

- 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 can be used to develop Linux-based client applications in a variety of programming languages that can call into ACMS applications. Currently supported languages on Linux include C/C++, Java, PHP, and Python.

- Alternative language bindings:

  In addition to C/C++ the updated software also provides facilities for the development of client applications in other programming languages including 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[2]. The 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 commented 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 new `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.

The following sections consider some of these items in more detail, with a particular emphasis on client development using SWIG and `twiddle`.

Installation

This section briefly describes 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, and 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 installation in the directory `ACMSDI$ROOT:[KITS]`. These kits can be copied via `sftp` or otherwise to Windows and Linux systems and installed as described below.

[2] Note that 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, which includes scripting languages such as JavaScript, Perl, PHP, Python, Tcl and Ruby and non-scripting languages such as C#, Go, Java, and others.

OpenVMS:

The ACMSDI gateway and client API software for VSI OpenVMS can be installed on VSI OpenVMS 8.4-1H1 I64 or higher and requires the following products to have been previously installed:

- VSI OpenVMS Version 8.4-1H1 or higher
- VSI TCP/IP, HPE TCP/IP Services for OpenVMS, or MultiNet TCP/IP
- VSI SSL111 V1.1-1g or later
- VSI ACMS V5.3-2 (or equivalent)
- It is recommended that the software is installed on an ODS-5-enabled file system

In addition to the above requirements, some or all of the following software products are required if you intend to develop client applications using ACMSDI client API on VSI OpenVMS. OpenJDK, Python, PHP, and Lua are only required if you intend to develop client applications using these languages; the C compiler and SWIG are required regardless.

- VSI C V7.4
- SWIG V3.0-5
- OpenJDK V8.0-222B
- Python 3.5.0
- PHP V5.6-10K
- Lua V5.3-5A

Note that the ACMSDI gateway server is dynamically linked with the OpenSSL 1.1.1 libraries and it is therefore essential that OpenSSL 1.1.1 is installed and started before attempting to install and use the gateway software. 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 in order to install and start the ACMSDI gateway server.

The OpenVMS kit is provided as an OpenVMS PCSI kit (`VSI-I64VMS-ACMSDI-V0505-0A-1.PCSI`) that can be installed by a suitably privileged user using the following command[3]:

```
$ PRODUCT INSTALL ACMSDI
```

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

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

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

Do you want to continue? [YES]

Configuration phase starting ...
```

[3] To avoid confusion and possible operational issues, it is recommended that any old HP versions of the ACMSDI gateway software be uninstalled before installing the new VSI version of the software. While the network protocol used by the new VSI version is compatible with that of old HP versions, SSL/TLS transport is not supported by the HP versions, and other aspects of the software (such as the locations of some installed files and log files) are not compatible old HP versions of the software.

```
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 V5.5-0A: ACMSDI gateway for OpenVMS

    © Copyright 2020 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 V5.5-0A              DISK$I64SYS:[VMS$COMMON.]

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

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

VSI I64VMS ACMSDI V5.5-0A: 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 shut down 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'
```

After the installation has successfully completed, include the commands displayed at the end of the installation procedure into `SYSTARTUP_VMS.COM` and `SYSHUTDWN.COM` to ensure that the ACMSDI gateway server is started and stopped when OpenVMS is booted and shutdown. Note that the ACMSDI gateway server should be started after ACMS, TCP/IP, and SSL111 have been started, and the gateway should ideally be shut down before ACMS is shutdown. 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 in order for the gateway to start correctly.

Once the installation has completed you may also wish to manually start the ACMSDI gateway and confirm that it is operating correctly. This can be achieved by performing the following tasks after running the gateway start-up procedure:

- Verify that the logical name `ACMSDI$ROOT` is correctly defined. This system-level logical name points to the root of the ACMSDI installation tree.
- Verify that the gateway is running by issuing a "`SHOW SYSTEM`" command and checking the displayed output for the existence of the process name "`ACMSDI$GATEWAY`". Note that if there was already a process running on your system with this process name, the gateway will fail to start.

The ACMSDI gateway process will also fail to start if there is another process using the default port number (1023) used by the gateway. 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.

- Examine the log file `ACMSDI$ROOT:[LOGS]ACMSDI.LOG` and verify that there are no errors or warnings being reported. Note that a new version of this log file will be created whenever the ACMSDI gateway is restarted, and it is recommended that appropriate processes are put in place to backup and purge old log files.

Be aware that the privileges `TMPMBX`, `NETMBX`, `BYPASS`, `SYSPRV`, and `DETACH` are currently required in order to run the ACMSDI start-up and shutdown scripts, and the ACMSDI gateway process (run as a detached process) will inherit the default privileges for the username under which it is started. It is also important to note that the ACMSDI gateway can require considerable resources in order to operate efficiently, depending on workload requirements. The following quotas should be adequate for most purposes; however, resource usage should be carefully monitored, and quotas adjusted as necessary.

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 (this parameter can usually be safely set to its maximum value of 65535).

Note that 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`).

<u>Linux</u>:

The Linux client kit is provided as a compressed tar file (`acmsdi-linux-5.5.0.tar.gz`) that can easily be installed locally by individual users or system-wide, with usage of a particular installation depending only on the definition of a single environment variable (`ACMSDI_HOME`), which points to the top-level directory of the extracted tar file. Assuming that the kit is to be installed system-wide under `/usr/local`, installation is simply a matter of extracting the contents of the compressed tar file as shown below (assuming the kit file has been copied to the users' home directory and that the user in question has `sudo` permissions):

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

This will create the top-level directory `/usr/local/acmsdi` for the installation. In order to start using the software it is then simply a matter of defining the environment variable `ACMSDI_HOME` and including this definition in your `.profile` file (or similar such login script, depending on Linux shell being used):

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

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[4].

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

- OpenSSL 1.1.1 development libraries (OpenSSL 1.1.1f or later)
- gcc 9.3.0 or later (alternatively the Clang compiler may be used)
- SWIG 4.0.1 or later (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 the development kits for those languages must also be installed. Note that it is not necessarily sufficient to install only the language runtimes (depending on the language in question), 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. For example, for Ubuntu Linux the following packages (or similar) need to be installed in you wish to develop client applications using Lua, Python, or PHP, and for Java development OpenJDK 8 or similar should be installed:

- lua5.3-dev
- python3-dev
- php-dev

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 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 particular requirements.

[4] 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.

Windows:

The Microsoft Windows kit is provided as a single installable package named `libacmsdi-5.5.0.msi` that can be easily installed and used by Windows any user[5]. To install the package, simply double-click on `Libacmsdi.msi` from within Windows Explorer and click the "Next" prompts to step through the installation. Once the installation has successfully completed, the installed files can be found in under the users' home directory in the folder `AppData\Local\Libacmsdi`. 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[6], it is also necessary to install the SWIG interface generator (see http://www.swig.org/ for details), 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 both 32-bit and 64-bit libraries that can be used when building applications. When building 32-bit applications, code must be linked with `acmsdi.lib` (found in `AppData\Local\Libacmsdi\lib`), and when building 64-bit applications, application code must be linked with `acmsdi-x64.lib`.
- The folder `AppData\Local\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 DLL's may be copied to another location that is included in the user or system `PATH`. In addition to DLL's, this folder also contains `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 `AppData\Local\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 `AppData\Local\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 in a production environment.

[5] It should be noted that at this time the ACMSDI Windows kit has been testing only with Windows 10. It is unknown whether the software will work correctly with older versions of the Microsoft Windows operating system.

[6] 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.

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.

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 (`AppData\Local\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 (`AppData\Local\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 forthe 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 (`AppData\Local\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. First, modify the `prepare-win.cmd` file 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.

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 other whether client certificates are to be verified or not. C client support for SSL/TLS encryption is provided via new options that can be specified when calling `acmsdi_sign_in()` to connect to the ACMSDI gateway, and for other client languages new methods are provided to specify key and certificate file details and to connect to the gateway via SSL/TLS. The following sections describe these new features in more detail.

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$SSL_KEY`	This logical name should be used to specify the private key file path.
`ACMSDI$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.

Note that 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.

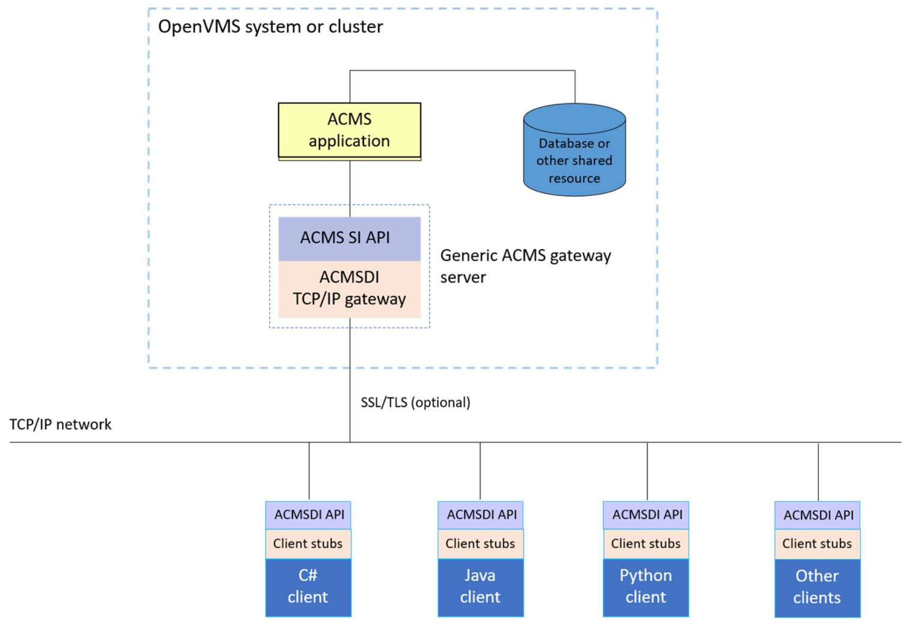
```
emp = require("emp")

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
```

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.