ACSP-4115-00

IBM Advanced Cryptographic Service Provider

ACSP Client Installation and Configuration Guide

Version 1 Release 5.2

Notices

Trademarks

This edition applies to IBM Advanced Cryptographic Service Provider, Version 1 Release 5.2 solution and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this Publication

This publication contains information about Client installation and configuration of the **IBM Advanced Cryptographic Service Provider.**

The IBM Advanced Cryptographic Service Provider components are called ACSP. ACSP includes Clients, Server, and Monitor. The ACSP Server is a service that provides IBM hardware based IBM Common Cryptographic Architecture (CCA) and PKCS#11 Cryptographic Services for z/OS.

The ACSP Clients are a Java based CCA client, a C based CCA client and a C based PKCS#11 client. They interact with the ACSP Server to access hardware based cryptographic services remotely.

For more information about PKCS#11 and Cryptoki see “PKCS #11 v2.20: Cryptographic Token Interface Standard RSA” below.

Related Publications

IBM Advanced Cryptographic Service Provider

ACSP-4102 IBM Advanced Cryptographic Service Provider - z/OS Program Directory

ACSP-1100 IBM Advanced Cryptographic Service Provider - General Information and Planning Guide

ACSP-4105 IBM Advanced Cryptographic Service Provider - ACSP z/OS Server Installation, Configuration and Operation Guide

ACSP-4106 IBM Advanced Cryptographic Service Provider - ACSP AIX/Linux Server Installation, Configuration and Operation Guide

ACSP-4115 IBM Advanced Cryptographic Service Provider - ACSP Client Installation Guide

ACSP-2105 IBM Advanced Cryptographic Service Provider - ACSP Java Programmers Guide

ACSP-2115 IBM Advanced Cryptographic Service Provider - ACSP CCA and PKCS#11 Programmers Guide

ACSP-4200 IBM Advanced Cryptographic Service Provider - ACSP Server Messages

ACSP-4210 IBM Advanced Cryptographic Service Provider - ACSP Client Messages

IBM z/OS Cryptographic Services Integrated Cryptographic Service Facility (ICSF)

SC14-7506 z/OS ICSF System Administrators Guide

SC14-7507 z/OS ICSF System Programmers Guide

SC14-7508 z/OS ICSF Application Programmers Guide

IBM z/OS Security Server Resource Access Control Facility (RACF)

SA23-2287 z/OS RACF System Programmers Guide

SA23-2289 z/OS RACF Security Administrators Guide

SA23-2292 z/OS RACF Command Language Reference

IBM Java API to CCA (jCCA)

See: http://www.ibm.com/security/cccc/tools/jcca.shtml

IBM Cryptographic Coprocessor

For a list of IBM Cryptographic Coprocessor Publications see:

http://www.ibm.com/security/cryptocards/

PKCS #11 v2.20: Cryptographic Token Interface Standard RSA

See: https://www.cryptsoft.com/pkcs11doc/v220

Apache Log4J

http://logging.apache.org/log4j/1.2/manual.html

Summary of Changes

New document number and structure. Replaces old ACSP-4015 manual.

1. Overview

IBM Advanced Cryptographic Service Provider provides remote access to hardware based cryptographic services for your applications. This makes it possible to achieve a more cost effective, scalable and more manageable cryptographic infrastructure. The hardware based cryptographic services can be a server farm with cryptographic hardware, or an existing IBM mainframe infrastructure with CryptoExpress3 (CEX3), CryptoExpress4 (CEX4), or CryptoExpress5 (CEX5).

For a more detailed description of the IBM Advanced Cryptographic Service Provider product itself, please refer to the ACSP-1100 IBM Advanced Cryptographic Service Provider - General Information and Planning Guide.

One of the key benefits of the IBM Advanced Cryptographic Service Provider is that it makes it possible to utilize IBM 4765 (CEX3 and CEX4) and IBM 4767 (CEX5) cryptographic coprocessors, not only from multiple client machines but also via multiple programming interfaces. A Java CCA client and CCA and PKCS#11 C client interfaces are provided by the IBM Advanced Cryptographic Service Provider product.

A description of the ACSP programming interfaces can be found in the following manuals:

- ACSP-2105 IBM Advanced Cryptographic Service Provider - ACSP Java Programmers Guide
- ACSP-2115 IBM Advanced Cryptographic Service Provider - ACSP CCA and PKCS#11 Programmers Guide

This publication explains how to install the ACSP Clients on the supported operating system platforms.

The ACSP Clients enables:

- Existing Java programs that use IBM Java API to CCA (jCCA)
- Existing C CCA programs for IBM 4764, IBM 4765, IBM 4767, or ICSF
- Existing PKCS#11 C programs

to use the remote Cryptographic Services provided by ACSP Server, in many cases without modifying the actual program code.

ACSP Clients are provided for the following platforms:

- AIX V6.1 and V7.1
- I5/OS V5, V6 and V7
- Linux
- Linux on z
- Windows
- Generic Java Platforms

2. ACSP Java Client Installation

2.1 Supported Platforms

The ACSP Java client is supported on the following operating systems platforms:

- IBM AIX V5.3, V6.1, V7.1 (32-bit and 64-bit)
- Linux Ubuntu, RedHat and SUSE (32-bit and 64-bit) and Linux on z (64-bit)
- Microsoft Windows (32-bit and 64-bit)
- Generic Java (JRE following Java Language Specification 1.5+)

2.2 Limitations

The current version of the ACSP Java client supports CCA using IBM Java API to CCA (jCCA) which supports the CCA standard up to and including version 5. However, the available functionality is ultimately limited by which functions are implemented on the ACSP Server. If an unsupported function is called on the server, a return code 20/20 will be returned to the application.

2.3 Prerequisites

The ACSP Java Client Version 1 Release 5.2 requires the following IBM Advanced Cryptographic Service Provider components:

- ACSP Server Version 1 Release 5.2+.

It is expected that you have transferred the installation binary to the target client machine before starting the installation. For Linux and AIX systems also make sure that the **bin** file executable, with the command:

```
chmod +x installation-file-name
```

2.4 Installation

To install the ACSP Java client on AIX 32-bit use the following command:

```
sudo ./acsp-java-client-aix-1.5-2.5.ppc.bin
```

To install the ACSP Java client on 64-bit use the following command:

```
sudo ./acsp-java-client-aix-1.5-2.5.ppc64.bin
```

To install the ACSP Java client on Linux 32-bit use the following command:

```
sudo ./acsp-java-client-linux-1.5-2.5.i386.bin
```

On Linux 64-bit use the following command:

```
sudo ./acsp-java-client-linux-1.5-2.5.X86_64.bin
```

On Linux for z 64-bit use the following command:

```
sudo ./acsp-java-client-zlinux-1.5-2.5.s390x.bin
```

If you agree with the license agreements, the installation will be done using the RPM manager.

The installation creates an ***/opt/ibm/acsp-java-client*** directory with the client test procedures.

Note: The ./lib subdirectory contains the Java libraries needed to run the client.

On Windows 32-bit and Windows 64-bit use the following command:

```
acsp-java-client-win.exe
```

Follow the installation wizard instructions to install the ACSP Java Client.

The installation creates a **c:\Program Files\IBM\IBM ACSP Java Client** directory with the client test procedures. Note that the windows installer has no JRE included, it will detect and use a previously installed IBM or Oracle JRE.

2.4.1 Installed Files

The following files are installed in the installation directory:

config	The subdirectory that contains the configuration files for the ACSP Java client
jvm	The subdirectory that contains the IBM Java Runtime Environment (JRE) 1.8 for running the 32-bit version of the ACSP Java Client. *The directory is only present in the 32-bit version for AIX or Linux.*
jvm64	The subdirectory that contains the IBM Java Runtime Environment (JRE) 1.8 for running the 64-bit version of the ACSP Java Client. *the directory is only present in the 64-bit version for AIX or Linux.*
lib	The subdirectory that contains the java binary jar files necessary to run the ACSP Java client
libmon	The subdirectory that contains the Java binary jar file for the ACSP Monitor.
license	This subdirectory contains license files
sample	A subdirectory that contains Java sample code.
allverbs	A test script for a call test of all CCA verbs on the server.
monitor	A script to start the monitor
sunshinetest	The script to run a sunshine verb test against the local server
threadtest	A test script for running performance tests.

All the command scripts expect that the current working directory is the installation directory.

The ACSP Java Client can be used to run connection and performance tests from the supported platforms client machines.

2.5 Uninstallation

The ACSP Java Client is installed using the RPM manager on AIX or Linux. To uninstall use the following command:

```
rpm -e acsp-java-client
```

On Windows use the operating systems normal uninstall procedure for applications.

2.6 Client Configuration

The **config** subdirectory contains the configuration files for the ACSP Java client.

The **acsp.properties** file should be configured to reflect the ACSP Server setup, which the ACSP Java client should use. Use the provided acsp.sample.properties as the basis of the acsp.properties file.

Generally the parameters *host.<name>, port.<name>.xxxx,* should be reviewed and updated to reflect your ACSP server. The *hostgroup*, *client.transport* and *client.connect* parameters are used by the ACSP Java client to select hosts and services.

See “Configuration Reference” on page 44 for a complete reference.

2.7 Running the ACSP Java Client in Application Server Context

If you want to use the ACSP Client in application servers like the WebSphere Application Server, GlassFish Application Server, or similar products, there are certain things that you must be aware of.

The ACSP Java Client uses the JVM for establishing SSL/TLS connections to the ACSP Server, and therefore the SSL/TLS configuration used for the ACSP Java Client may conflict with the configuration of the Java application server itself.

Ensure that the JRE used by the application server is supported by the ACSP Java Client:

1. Please look carefully at all the configuration parameters, which start with *java.* or *javax.* These parameters are global for the JVM, so if your application server and the ACSP Java Client share the JVM process, this may cause conflicts with the application server and the ACSP Client.
2. The definition of the *ssl.trustStore* parameter in the *acsp.properties* configuration file may conflict with the application server setup. If the application server and the ACSP Java Client share the JVM, this configuration parameter may already be in use by the application server and therefore cannot be reused by the ACSP Java Client.
   The configuration parameter "*ssl.use.local.context=true*" can be added to make the ACSP Java Client create a local instance of the truststore, instead of sharing it with the JVM. In this way conflicts with other certificates can be avoided.
   If *ssl.use.local.context* is not defined or set to *false, the ACSP* Client will use the JVM parameter *javax.net.ssl.trustStore* definition for establishing SSL connections.
3. The definition of the *ssl.keyStore* and the *ssl.keyStorePassword* parameters in the *acsp.properties* ACSP Java Client configuration file may also conflict with the application server setup as described above.
   The new configuration parameter "*ssl.use.local.context=true*" also adheres to the keystore definition parameters, and makes the ACSP Java client create a local instance of the keystore instead of sharing it with the JVM. In this way conflicts with other certificates can be avoided. If *ssl.use.local.context* is not defined or set to false, the ACSP Java Client will use the JVM parameters *javax.net.ssl.keyStore* and *javax.net.ssl.keyStorePassword* definitions to establish SSL connections.

NOTE: To do a seamless migration of the configuration parameters prefixed with "*javax.net.ssl*" to the new "*ssl*" prefixed parameters, these configuration parameters will be mapped by ACSP Client, and the mapping will be written to the ACSP Client log.

2.8 ACSP Java Client Test Scripts

The test scripts provided in the ACSP Java Client installation directory, can be used to test the ACSP Client and ACSP Server setup, connectivity and capabilities.

To run the ACSP Java Client test scripts, open a shell or command prompt window and change the current directory to the ACSP Java Client installation directory using one of the following commands:

For AIX or Linux:	cd /opt/ibm/acsp-java-client
For Windows:	cd c:\Program Files\IBM\IBM ACSP Java Client

2.8.1 Allverbs Script

Use the **allverbs** script to test the availability of verbs on the ACSP Server. A report of the test is displayed on the screen. The **allverbs** script has no parameters.
Return codes >= 20 indicate no availability on the server platform.
This script also tests the mapping capability in server zOS 64-bit environments.

2.8.2 Monitor Script

Use the **monitor** script to start the ACSP Monitor which is located in the ***libmon*** subdirectory. You will be prompted for the ACSP Server host name you want to monitor.

NOTE: You will need a graphical user interface to run the monitor.

Please refer to the manual “” for more information about monitoring and JMX.

2.8.3 Sunshinetest Script

Use the **sunshinetest** script to run a sunshine test of CCA verbs or services against your target server. The ***sunshinetest*** is different depending on the ACSP Server target environment – either the CCA (IBM 476x) or the ICSF (z/OS) environment.

The following command invokes the sunshine test:

```
./sunshinetest
```

2.8.4 Threadtest Script

Use the **threadtest** script to measure ACSP Java Client to ACSP Server response times and throughput, using different test scenarios.

To see the parameters for thread test use the following:

```
./threadtest or ./threadtest ?
```

To see the thread test sample report for a command use the following:

threadtest 2 2 6 5

The following is an example of the report generated by the thread test:

```
Result of test 2 variant 2: 'MAC generate single key using token' (1 trans) with 6 threads ... sampling time 5 seconds
==============================================================================================================================
Thread  no:  Count  Trans/s     Min.Time   cnt<1ms   Avg.Time       Max.Time    cnt>100ms     Init.trans      1st.trans Failed
------------------------------------------------------------------------------------------------------------------------------
Thread   1:   6527   1305       0.418 ms   5816      0.762 ms      35.963 ms       0            1.771 ms       3.017 ms      0
Thread   2:   6605   1321       0.413 ms   5934      0.744 ms      34.326 ms       0           26.883 ms      60.466 ms      0
Thread   3:   6539   1307       0.422 ms   5893      0.759 ms      35.907 ms       0           20.097 ms      10.578 ms      0
Thread   4:   6565   1313       0.428 ms   5870      0.756 ms      35.771 ms       0           42.376 ms      11.541 ms      0
Thread   5:   6455   1291       0.423 ms   5772      0.769 ms      36.161 ms       0            1.603 ms      10.509 ms      0
Thread   6:   6549   1309       0.423 ms   5839      0.760 ms      55.650 ms       0           10.348 ms       3.014 ms      0
------------------------------------------------------------------------------------------------------------------------------
All Total:   39240   7848       0.413 ms  35124      0.758 ms      55.650 ms       0    - Failed = 0 - Avg./trans = 0.758 ms

***INFO*** **** End of ThreadTest *****
```

3. ACSP CCA Client Installation

3.1 Supported Platforms

The ACSP CCA Client is supported on the following operating system platforms:

- IBM AIX V5.3, V6.1, V7.1 (32-bit32-bit and 64-bit)
- IBM I i5/OS V6.1, V7.1
- Linux RedHat (32-bit and 64-bit),
  Linux SUSE (32-bit and 64-bit)
  Linux on z (64-bit)
- Microsoft Windows 7(32-bit and 64-bit),
  Windows 8.1 (32-bit and 64-bit),
  Microsoft Windows 10 (32-bit and 64-bit),
  Windows Server 2008 R2 (64-bit only), and
  Windows Server 2012 R2 (64-bit only)

3.2 Limitations

The current version of the ACSP CCA Client interface for C is compatible with the CCA standard up to and including version 5. However, the available functionality is ultimately limited by which functions are implemented on the ACSP Server. If an unsupported function is called on the server, a return code 20/20 will be returned to the application.

3.3 Prerequisites

The ACSP CCA Client Version 1 Release 5.2 requires the following components:

- IBM Advanced Cryptographic Service Provider Server Version 1 Release 5.2.

It is expected that you have transferred the installation binary to the target client machine before starting the installation. For Linux and AIX systems also make sure that the **bin** file executable, with the command:

```
chmod +x installation-file-name
```

3.4 AIX Installation

3.4.1 Program Requirements

The program requirements for the ACSP CCA Client are the following

- The ACSP CCA Client requires a network connection to establish connection to the ACSP Server.
- No existing IBM 476x CCA driver installation must be present. The ACSP CCA Client installation may conflict with the regular CCA installation.
- Root or sudo access is required for installation.

3.4.2 Packages

The ACSP CCA Client software is delivered with one or more of the following packages:

acsp-cca-client-aix-1.5-2.5.ppc64.bin	Client installation file for 32-bit and 64-bit systems – This will replace the library that provides access to the physical hardware
acsp-cca-client-aix-1.5-2.5-coexist.ppc64.bin	Client installation file for 32-bit and 64-bit systems – This can coexist with physical hardware.

3.4.3 Client Installation

To install the client package on AIX 32-bit and 64-bit editions use the following command:

```
  ./acsp-cca-client-aix-1.5-2.5.ppc64.bin
or
  ./acsp-cca-client-aix-1.5-2.5-coexist.ppc64.bin
```

If you agree with the license agreements, the installation is done using the RPM manager.

The installation creates an ***/opt/ibm/acsp-cca-client*** directory.

3.4.4 Installed Files

The files in the following table are installed in the installation directory:

Directory	File	Description
bin	ivp32.1.5.2.5 Ivp32 ivp64.1.5.2.5 ivp64	The Installation Verification Program for 32-bit and 64-bit packages
config	acsp.properties acsp.sample.properties	ACSP configuration file: A Sample ACSP configuration file
	ivp.client_1.p12 ivp.client_2.p12 ivp.client_3.p12 ivp.ca.pem	PKCS#12 keystores with client key and certificate The PEM encoded CA certificate.
doc	readme.txt	Read me and changes to the cca client
include	acspincl.h csufincl.h	The Include file for CCA verbs or services.

Directory	File	Description
lib	libacsp-csufcca.a.1.5.2.5 libacsp-csufcca.a libcsufcca.a libcsufsapi.a	A 32-bit and 64-bit link library for applications
license	acsp_ilan.1.5.2.txt acsp_license_information.1.5.2.txt acsp_notices.1.5.2.txt	License information files
sample	mac.c	A MAC sample in C
	udfcall.c	A UDFCALL sample in C
	hostreserve.c	A sample program, showing how to use the host reservation API.
	readme.txt	The sample program compile instructions
The files below are not included in the coexist package		
/usr/lib	libacsp-csufcca.a libcsufcca.a libcsufsapi.a	Soft links pointing to 32-bit libraries
/usr/include	acspincl.h csufincl.h	Soft links to acspincl.h

3.4.5 AIX Uninstallation

The ACSP CCA Client is installed using the RPM manager. To uninstall use the following command:

```
rpm -e acsp-cca-client
```

3.5 i5/OS Installation

3.5.1 Program Requirements

The program requirements for the ACSP CCA Client are the following:

- The ACSP CCA Client requires a network connection in order to establish a connection to the ACSP server.
- No existing CCA driver (IBM 476x) installation must be present. The ACSP CCA Client installation will conflict with the regular CCA installation.
- Administrator access.

3.5.2 Packages

The i5/OS installation of the ACSP CCA Client is provided as a SAVF-file.

csuccca.savf	Client installation file

3.5.3 Client Installation

Transfer the *csuccca.savf* to the target machine. The transferred file must have the type SAVF. The recommended procedure for transferring *csuccca.savf* to the target is to open a 5250 terminal emulator session for the target i5/OS machine, and create an empty SAVF-file using the following command:

```
CRTSAVF FILE(QGPL/CSUCCCA)
```

This step ensures that the uploaded file will have the correct type.

Use FTP in binary mode to upload the *csuccca.savf* to QGPL/CSUCCCA.

When the SAVF file has been transferred to the target machine, use the following RSTLIB command to install the ACSP CCA Client.

```
RSTLIB SAVLIB(QACSP) DEV(*SAVF) SAVF(QGPL/CSUCCCA)
```

3.5.4 Installed Files

The installation script will install the following library files:

QACSP/CSUCCCA	The ACSP CCA Client service program
QACSP/ACSPCMN	The Support service program used by the ACSP CCA Client
QACSP/IVP	The Installation Verification Program
IVPCLT1P12 IVPCLT2P12 IVPCLT3P12	PKCS#12 keystores with client key and certificate. They can be imported into the Digital Certificate Manager to set up a client application. The keystore password is: zaqwsx
QACSP/H	The Source physical file containing the members: ACSPINCL: C include file
QACSP/README	The Source physical file containing the members: README: Read me information
QACSP/LICENSE	The Source physical file that contains the members: • ILAN – International License Agreement for Non-Warranted Programs • LICINFO – License information. • NOTICES – Notices and Information

QACSP/SAMPLE	The Source physical file containing the members: • MAC – C sample source code • UDFCALL – C sample source code that shows how to use the UDFCALL ACSP Server extension to cca. • HOSTRESERV - C sample source code that shows how to use the host-reserve feature. • PINSAMPLE – RPG sample source code • PROPERTIES – sample ACSP Client configuration. • IVPCAPEM – IVP CA certificate in PEM format. This can be imported into the Digital Certificate Manager as a trusted CA to set up a client application's trusted certificates.

3.6 Linux Installation

3.6.1 Program Requirements

The program requirements for the ACSP CCA Client are the following:

- The ACSP CCA Client requires a network connection in order to establish a connection to the ACSP Server.
- No existing CCA driver installation must be present. The ACSP CCA Client installation will conflict with the regular CCA installation.
- Root or sudo access is required for installation

3.6.2 Packages

The ACSP CCA Client software is delivered with one of the following packages:

acsp-cca-client-linux-1.5-2.5.i386.bin	Client installation file for x 32-bit systems
acsp-cca-client-linux-1.5-2.5.x86_64.bin	Client installation file for x 64-bit systems
acsp-cca-client-zlinux-1.5-2.5.s390x.bin	Client installation file for Linux on z 64-bit systems

3.6.3 Client Installation

To install the client package on Linux 32-bit use the following command:

```
./acsp-cca-client-linux-1.5-2.5.i386.bin
```

To install the client package on Linux 64-bit use the following command:

```
./acsp-cca-client-linux-1.5-2.5.x86_64.bin
```

To install the client package on Linux on z 64-bit use the following command:

```
./acsp-cca-client-zlinux-1.5-2.5.s390x.bin
```

If you agree with the license agreements, the installation is done using the RPM manager.

The installation creates an **/opt/ibm/acsp-cca-client** directory with the client test procedures.

3.6.4 Installed Files

The following files are installed in the ***/opt/ibm/acsp-cca-client*** installation directory:

Directory	File	Description
bin	ivp.1.5.2.5 ivp	The 32-bit binaries and soft links
bin64	ivp.1.5.2.5 ivp	The 64-bit binaries and soft links
config	acsp.properties acsp.sample.properties	The ACSP Client sample configuration files
	ivp.client_1.p12 ivp.client_2.p12 ivp.client_3.p12 ivp.ca.pem	The PKCS#12 keystores with client key and certificate The PEM encoded CA certificate.
doc	readme.txt	Read me and changes to cca client
include	acspincl.h csulincl.h	The Include file for CCA verbs/services Soft link to acspincl.h

Directory	File	Description
lib	libacsp-csulcca.so.1.5.2.5 libacsp-csulcca.so libcsulcca.so libcsulcca.so.4	The 32-bit link library for applications Soft links pointing to 32-bit libraries
lib64	libacsp-csulcca.so.1.5.2.5 libacsp-csulcca.so libcsulcca.so libcsulcca.so.4	The 64-bit link library for applications Soft links pointing to 64-bit libraries
license	acsp_ilan.1.5.2.txt acsp_license_information.1.5.2.txt acsp_notices.1.5.2.txt	License information files
sample	mac.c	A MAC sample in C
	udfcall.c	A UDFCALL sample in C
	hostreserve.c	A sample program that shows how to use the host reservation API.
	readme.txt	Sample program compile instructions
/usr/lib	libacsp-csulcca.so libcsulcca.so libcsulcca.so.4 libcsulsapi.so	Soft links pointing to 32-bit libraries
/usr/lib64	libacsp-csulcca.so libcsulcca.so libcsulcca.so.4 libcsulsapi.so	Soft links pointing to 64-bit libraries
/usr/include	acspincl.h csulincl.h	Soft links for include files

3.6.5 Linux Uninstallation

The ACSP CCA Client is installed using the RPM manager. To uninstall use the following command:

```
rpm -e acsp-cca-client
```

3.7 Windows™ Installation

3.7.1 Program Requirements

The program requirements for the ACSP CCA Client are the following:

- The ACSP CCA Client requires a network connection to establish connection to the ACSP Server.
- The installation program must be run with administrative privileges.

3.7.2 Packages

The ACSP CCA Client software is delivered with the following package:

acsp-cca-client-win-1.5-2.5.exe	Client installation file for 32-bit and 64-bit systems

3.7.3 Client Installation

To install the client package on Windows 32-bit and 64-bit systems do:

```
acsp-cca-client-win-1.5-2.5.exe
```

During the installation you can specify where to install the files. The program will also prompt you for the ACSP Server to use. The server name is inserted into the ACSP configuration file. You can edit this file manually after installation to specify the server to use.

The installation program will automatically create the ACSP environment variable which points to the ACSP configuration file. However, this change does not take place for programs which are already running, and in some cases not in programs started by already running programs. For this reason we suggest logging off and logging in the current user after the installation has been completed.

3.7.4 Installed Files

The following files are installed in the ***C:\Program Files\IBM\IBM ACSP CCA Client*** installation directory:

Directory	**File**	**Description**
bin	ivp.exe csuncca32.dll csunsapi.dll	32-bit binaries and DLLs
bin64	ivp.exe csuncca.dll	64-bit binaries and DLLs
config	acsp.properties	Sample ACSP Client configuration file
	ivp.client_1.p12 ivp.client_2.p12 ivp.client_3.p12 ivp.ca.pem	PKCS#12 keystores with client key and certificate PEM encoded CA certificate.
doc	readme.txt	Read me and changes to cca client
include	acspincl.h csunincl.h	Include files for CCA verbs/services
lib	csuncca32.lib	Link library for 32-bit applications
lib64	csuncca.lib	Link library for 64-bit applications
license	acsp_ilan.1.5.2.txt acsp_license_information.1.5.2.txt acsp_notices.1.5.2.txt	License information files
sample	mac.c	Sample MAC sample in C

Directory	File	Description
	udfcall.c	UDFCALL sample in C
	hostreserve.c	Sample program, showing how to use the host reservation API.
	readme.txt	Sample program compile instructions

Note: For some components, separate files are used for the 32-bit and the 64-bit version. The 64-bit operating system is capable of using both the 32-bit and the 64-bit version. However, in all cases the application which will be using the library must use the same addressing environment (32-bit or 64-bit) as the DLL.

3.8 Installation Verification Program

The ACSP CCA Client includes a binary version of the provided sample program source “mac.c”, which is called Installation Verification Program (IVP) and is located in the **bin** directory shown above.

The IVP program can be used to test the ACSP CCA Client installation and configuration by using an existing ACSP Server.

With the the current ACSP CCA client configuration, the IVP program will perform a few CCA calls to the ACSP server to verify the ACSP CCA Client's ability to operate with the ACSP Server.

To run the installation verification program, the ACSP CCA Client must be able to find its configuration file. The ACSP CCA Client uses the value of the environment variable 'ACSP' to find it.

On Windows, the ACSP environment variable is set by the installer and points to the *acsp.properties* file generated by the installer. On AIX and Linux this variable must be set and exported before running the sample program:

Set the variable using the following command:

```
export ACSP=config/acsp.properties
```

See also “Setting the Environment Variables” on page 28.

For AIX and Linux, execute the program by using the following commands for 32-bit and 64-bit:

```
./bin/ivp32
./bin/ivp64
```

For Windows execute the program using the following commands for32-bit and 64-bit:

```
bin\ivp.exe
bin64\ivp.exe
```

Note: If the SSL keystore password is to be provided via the API, use the parameter “-p” to the ivp program.

On success, the output from the IVP will be similar to the following example:

```
Cryptographic Coprocessor Support Program example program.
2017.01.26 07:20:55.805 (24534) AZH70000I Log-level was set to INFO
2017.01.26 07:20:55.811 (24534) AZH30223I Connecting to cccc30.bal.dk.ibm.com:9994 using
connection(0)
Key_Generate successful.
MAC_Generate successful.
MAC_value = 56B9 2CFD
MAC_Verify successful.
2017.01.26 07:21:01.000 (24534) AZH30228I Closing connection(0)
```

In case of failure, the output will look like the following example:

```
Cryptographic Coprocessor Support Program example program.
2017.01.26 09:00:08.443 (25012) AZH30504F Could not determine the location of the configuration
file. Checked environment variables ACSP_CCA and ACSP
2017.01.26 09:00:08.443 (25012) ACSP30051W An error occurred during processing of call to verb
CSNBKGN: 24/28002 - The ACSP environment variable was not set. Please consult "Use of the library"
in the documentation.
2017.01.26 09:00:08.443 (25012) AZH30050E An error occurred during processing of call to verb
CSNBKGN. Reporting 24/28002 to the application - The ACSP environment variable was not set. Please
consult "Use of the library" in the documentation.
Key Generate Failed
Return_code = 24
Reason_code = 28002
```

Use the return/reason codes and the ACSP Client log to determine the cause of the error.

In case of runtime errors, verify that the configuration file has been configured correctly.

Check the following options when using auto-configuration:

```
client.autoconf.host
client.autoconf.port
client.transport
ssl.keystore
ssl.truststore
ssl.private.key
```

Or you may check the following options when using a direct connection:

```
client.connect
client.connect.port
client.transport
ssl.keystore
ssl.truststore
ssl.private.key
```

It is recommended that only absolute file system paths are entered in the configuration file.

Please refer to the ACSP-2115 IBM Advanced Cryptographic Service Provider - ACSP CCA and PKCS#11 Programmers Guide for further details on ACSP Client programming.

4. ACSP PKCS #11 CCA and EP11 Client Installation

4.1 Description

The ACSP PKCS #11 CCA Client and ACSP PKCS #11 EP11 Client consist of two independent PKCS #11 libraries which both use the ACSP CCA Client to access the ACSP Server. See ACSP CCA Client Installation above.

The ACSP PKCS #11 CCA Client will use CCA verbs for cryptographic operations.

The ACSP PKCS #11 EP11 Client will use the PKCS #11 verbs on ICSF/zOS.

These clients operate without a configuration file but need a configured ACSP CCA Client.

4.2 Program Requirements

The program requirement for the ACSP PKCS#11 CCA + EP11 clients is the following:

- The ACSP CCA client

4.3 Limitations

See ACSP-2115 IBM Advanced Cryptographic Service Provider - ACSP CCA and PKCS#11 Programmers Guide .

4.4 Supported Platforms

The ACSP PKCS#11 CCA + EP11 Clients are supported on the following operating system platforms:

- IBM AIX V5.3, V6.1, V7.1 (32-bit and 64-bit)
- Linux RedHat (32-bit and 64-bit), Linux SUSE (32-bit and 64-bit), and Linux on z (64-bit)
- Microsoft Windows 7 SP1 (32-bit and 64-bit), Windows 8.1 Update (32-bit and 64-bit), Windows Server 2008 R2 (64-bit only), and Windows Server 2012 R2 (64-bit only).

4.5 Installation

The installation is done the same way on all platforms, by running the installation program. There are two separate installation programs, one for each client.

4.5.1 Packages

The ACSP PKCS#11 CCA and EP11 Client software is delivered with one or more of the following packages:

acsp-pkcs11-cca-client-aix-1.5-2.5.ppc64.bin	The ACSP PKCS #11 CCA Client for AIX installation executable for 32-bit or 64-bit systems
acsp-pkcs11-cca-client-linux-1.5-2.5.i386.bin	The ACSP PKCS #11 CCA Client for Linux32-bit systems
acsp-pkcs11-cca-client-linux-1.5-2.5.x86_64.bin	The ACSP PKCS #11 CCA Client for Linux 64 bit systems

acsp-pkcs11-cca-client-linux-1.5-2.5.s390x.bin	The ACSP PKCS #11 CCA Client for Linux on z 64 bit systems
acsp-pkcs11-cca-client-win-1.5-2.5.exe	The ACSP PKCS #11 CCA Client for Windows 32/64.
acsp-pkcs11-ep11-client-aix-1.5-2.5.ppc64.bin	The ACSP PKCS #11 EP11 Client for AIX installation file for 32-bit or 64-bit systems
acsp-pkcs11-ep11-client-linux-1.5-2.5.i386.bin	The ACSP PKCS #11 EP11 Client for Linux 32-bit systems
acsp-pkcs11-ep11-client-linux-1.5-2.5.x86_64.bin	The ACSP PKCS #11 EP11 Client for Linux 64-bit systems
acsp-pkcs11-ep11-client-linux-1.5-2.5.s390x.bin	The ACSP PKCS #11 EP11 Client for Linux on z 64-bit systems
acsp-pkcs11-ep11-client-win-1.5-2.5.exe	The ACSP PKCS #11 EP11Client for Windows 32-bit or 64-bit.

4.5.2 Client Installation

To install the client package on AIX and Linux run the following executable:

```
./<name of package>
```

To install the client package on Windows run the following executable:

```
<name of package>.exe
```

If you agree with the license agreements, the installation proceeds as follows:

For **Linux and AIX** the installation is done using the RPM manager and the installation creates a **/opt/ibm/acsp-pkcs11-cca-client** or **/opt/ibm/acsp-pkcs11-ep11-client** directory with the client modules.

For **Windows** the modules are installed in ***C:\Program Files\IBM\IBM ACSP PKCS#11 CCA Client*** or ***C:\Program Files\IBM\IBM ACSP PKCS#11 EP11 Client*** respectively and the **bin** directory is put on **PATH**.

4.5.3 AIX and Linux Installed Files

The following files are installed in the installation directory:

Directory	File	Description
bin	apitest.1.5.2.5 apitest threadtest.1.5.2.5 threadtest	32-bit binaries and soft links
bin64	ivp.1.5.2.5 ivp apitest.1.5.2.5 apitest threadtest.1.5.2.5 threadtest	64-bit binaries and soft links
doc	readme.txt	Read me and changes to client
include	cryptoki.h pkcs11f.h pkcs11.h pkcs11t.h	The include file for PKCS#11 types and functions.
lib	libacsp-pkcs11-cca.so.1.5.2.5 libacsp-pkcs11-cca.so libacsp-pkcs11-ep11.so.1.5.2.5 libacsp-pkcs11-ep11.so	The 32-bit shared load library and soft link

lib64	libacsp-pkcs11-cca.so.1.5.2.5 libacsp-pkcs11-cca.so libacsp-pkcs11-ep11.so.1.5.2.5 libacsp-pkcs11-ep11.so	The 64-bit shared load library and soft link
license	acsp_ilan.1.5.2.txt acsp_license_information.1.5.2.txt acsp_notices.1.5.2.txt	License information files
sample	sample.c readme.txt	A sample PKCS#11 programs in C and its description.
/usr/lib	libacsp-pkcs11-cca.so libacsp-pkcs11-cca64.so (aix only) libacsp-pkcs11-ep11.so libacsp-pkcs11-ep1164.so (aix only)	Soft links pointing to the PKCS#11 client.

4.5.4 Windows Installed Files

The following files are installed in the installation directory:

Directory	**File**	**Description**
bin	apitest.exe apitest32.exe apitest64.exe acsp-pkcs11-cca.dll (cca only) acsp-pkcs11-cca32.dll (cca only) acsp-pkcs11-ep11.dll (ep11 only) acsp-pkcs11-ep1132.dll (ep11 only) pkcs11cca-win32.1.5.2.5.dll	32-bit binaries and DLL
bin64	apitest.exe apitest-win64-1.5.2.5.exe (cca only) ivp-p11-win64-1.5.2.5.exe (ep11 only) pkcs11cca-win64.1.5.2.5..dll (cca only) pkcs11ep11-win64.1.5.2.5..dll (ep11 only)	64-bit binaries and DLL's
doc	readme.txt	Read me and changes to the pkcs#11 client
include	cryptoki.h pkcs11f.h pkcs11.h pkcs11t.h	The Include file for PKCS#11 types and functions.
license	acsp_ilan.1.5.2.txt acsp_license_information.1.5.2.txt acsp_notices.1.5.2.txt	License information files
sample	sample.c readme.txt	A sample PKCS#11 program in C and its description
.	Uninstall.exe	The program to uninstall the Client

4.6 Installation Verification Program

Use the *apitest* program to verify the installation, but before that ensure that the ACSP CCA Client is configured and working by using the IVP program.

From a command line issue the following command:

```
apitest <path to pkcs#11 dll>
```

By default *apitest* will create a file called latest-testreport.html. This will show mechanisms and key types supported.

5. Using the ACSP CCA Client

The ACSP CCA Client uses a configuration file for determining the runtime parameters, for example which server to connect to and how to authenticate with the server. This file is known as the configuration file.

A sample configuration file is delivered as part of the installation. See “ACSP CCA Client Installation” on page 14.

5.1 Setting the Environment Variables

The ACSP CCA Client library finds the configuration file by using the system environment variable

ACSP_CCA

The alternative is to use the environment variable, as follows:

ACSP

The environment variable that is used must contain the absolute path to the configuration file. This must be set so that the value can be accessed by the process, which is to call the ACSP CCA Client library. The library will check for this environment variable when it is loaded by a process. It will then load and parse the file and use this configuration to execute the CCA call or PKCS#11 function for the 1st and subsequent calls.

The ACSP CCA client library also reads the environment variable

LOGLEVEL

to define the logging level at startup. When this environment variable is used, the given logging level will take effect earlier than when it is read from the configuration file. See the **loglevel** configuration parameter below for valid values.

The ACSP CCA Client also accept the following environment variables to override the ssl.keystore and ssl.keystore.password parameters:

ACSP_CLIENT_KEYSTORE

ACSP_CLIENT_KEYSTORE_PW

The above environment variables allow the user to override the parameters in the sample or common configuration file with a local keystore and password.

ACSP_NO_CONNPOOL_MANAGEMENT

If this environment variable is defined the ACSP CCA Client will not start background threads for connection management. Connections are not closed unless they fail to communicate.

Environment variables are set up differently, both locally and globally depending on the platform used:

On AIX or Linux:

Set up the environment variable with the following command:

```
  export ACSP=path/filename
```

The command above can be specified in the */etc/.profile* for global reference or in the /home/<user>/.profile for local access.

On i5/OS:

Set up the environment variable with the following command:

```
  ADDENVVAR ENVVAR(ACSP)
VALUE('QACSP/ACSP(properties)') REPLACE(*YES)
```

The command above defines a reference to library 'QACSP' with file 'ACSP' and member name 'properties'. The properties file must be placed as a member of a source-physical file.

On Windows:

Set up the environment variable with the following command:

```
  SET ACSP=drive:\path\filename
```

The Windows GUI may also be used for this.

6. ACSP Client Configuration Guide

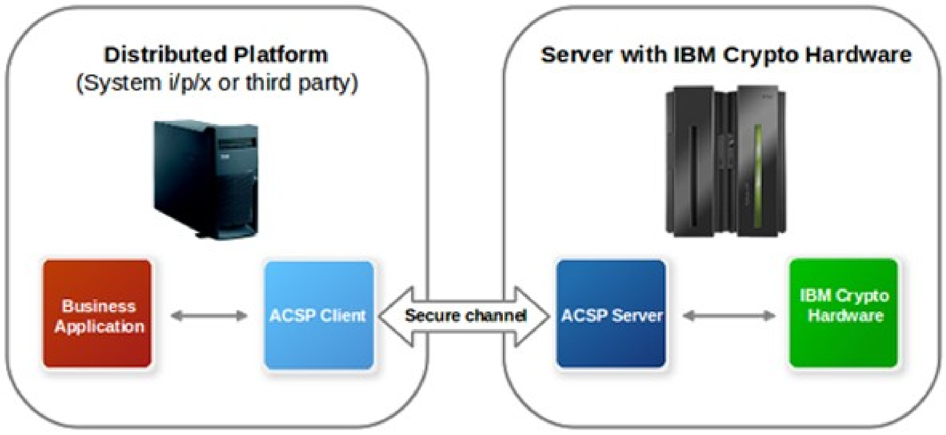
This chapter contains descriptions of various parts of ACSP Client, and guidelines for the configuration of the ACSP Client.

The “ACSP CCA Load Balancing Basics” and “Connection Pool Configurations” will focus on the load balancing between primary servers. The specifics of when secondary servers are enabled for use is described in “Active-Active Server Configuration” and derivations in ”Deviations From Normal Operations“. Once secondary servers are activated, these are load balanced on equal terms with the primary servers.

6.1 Overview

The IBM Advanced Cryptographic Service Provider is a product that provides remote access to existing CCA or ICSF based cryptographic environments.

IBM Advanced Cryptographic Service Provider consists of two major components. A Server component and a Client component.



On z/OS the ACSP Server provides the cryptographic services using ICSF, while on IBM AIX, and Linux it uses IBM 4765 or IBM 4767 cryptographic coprocessors through CCA.

The client/server nature of IBM Advanced Cryptographic Service Provider makes it possible to leverage the IBM mainframe cryptographic capabilities from a non-mainframe platform.

This cross-platform setup is possible because ICSF and CCA are the same for the typical cryptographic operations used by business applications. In the more atypical cases, for example key management, there are minor differences between ICSF and CCA that must be considered.

The basics of IBM Advanced Cryptographic Service Provider is that when a business application needs to do a cryptographic operation, it makes a CCA call. This call is passed to the ACSP CCA or Java Client which transfers the call to the ACSP Server. The server will use the available cryptographic service provider (ICSF or CCA) to perform the cryptographic operation and return the result to the client and subsequently to the business application.

6.1.1 Server Management

The ACSP CCA and Java Clients operate with two categories of servers: Primary servers and Secondary backup servers.

Primary servers are used by the client to process calls for normal operations.
Secondary servers are used only when one or more primary servers are offline.

Using these two categories, the ACSP CCA and Java Clients can be configured to support active with active servers and active with standby servers. This is described in “Active-Standby Server Configuration” on page 35.

6.2 ACSP CCA Load Balancing Basics

6.2.1 Introduction

The ACSP CCA and Java Clients has the ability to use multiple servers for processing CCA calls. This provides increased capacity and redundancy for processing calls.

The ACSP Clients have a list of the servers that are available to them. The clients are not visible to each other, nor are the clients visible to the servers. Load balancing is therefore the responsibility of the clients, and because each client balances its own load, the total load will also be balanced.

To achieve this, the design of the load balancing algorithm in the ACSP CCA and Java Client aims to balance the number of calls between servers at any given time and not as an average over time. This means that the server with the fewest outstanding calls for this client will get the next call.

Note that the algorithm does not take the complexity of the individual calls into account. A call is a call.

Specifically, this is achieved by managing the number of connections to each server.
The client can have active connections and idle connections. Idle connections can become active connections when the client needs a new connection to a server. For this reason, idle connections are kept for a time to prevent repeatedly increasing and decreasing the number of connections with varying load from the application.

This chapter discusses how to best manage the connections.

6.2.2 Load Balancing Situations

If all servers have the same Cryptographic capacity available for the clients, and the clients experience the same load from its application, it must be anticipated that a balanced load between servers is achieved.

Below are two less balanced situations for consideration:

Situation 1

One of the clients experiences a load from its applications that is significantly more costly than the other clients do. An example could be the generation of 4096 bit RSA signatures for one client, where the other clients perform symmetric encryption on short messages.

The client generating the RSA signatures will balance its costly calls evenly as will the other clients, and the result will be a balanced load on the servers

Situation 2

One of the servers has significantly less Cryptographic capacity available for the clients than the other servers. This can happen if the server has fewer cryptographic coprocessors, or if there are other processes using the cryptographic coprocessors that run on this server.

In this situation the response times for the calls from this server will increase. Calls to this server will therefore return less frequently to a client than calls to other servers. When a new call on the client needs to be routed to a server, it is less probable that the server with for example fewer cryptographic coprocessors is the server with the fewest outstanding calls.

Note that across all cryptographic coprocessors in the infrastructure the load may still be balanced.

6.2.3 Server Specification

The client contains a list of servers to use when load balancing calls. Under normal circumstances the client uses only its primary servers. See the parameter ***client.connect.host***.

Under normal operations only the primary servers are used. Specification of secondary servers to use in error situations is described in Section *Activating Secondary Servers on page 36*. The section also describes the conditions for activation.

6.2.4 Server Selection

The client maintains a connection pool, which allows the reuse of connections when processing calls.
When a call is made, a connection is requested from the connection pool. Once the call has completed, the connection is returned to the pool to be reused by another call.

When a call is made, three situations can occur:

- The connection pool contains one or more available connections.
- No existing connections are available. The client must create a new connection.
- No existing connection are available, but the client is not allowed to create a new connection.

In the first two cases the client must select a server to be used to process the call. However, the client uses different criteria. The last case happens only if the client has been limited in the number of connections, it is allowed to create.

Connection Selection Criteria

When the connection pool contains one or more available connections, the criterion for selecting the connection to use for a given call is to choose the server which has the lowest number of outstanding calls, or calls in progress.
The client can calculate this because it keeps track of the number of established connections for each server and the number of connections in use.
When choosing between multiple servers with an equal number of outstanding calls, the client selects the one less used.

Connection Creation Criteria

The criterion for selecting a server to create a connection to, is to create a connection to the server which has the fewest connections already established. This ensures that there is an equal number of connections to each server. The worst case scenario in steady state is that high and low differ by 1. In exceptional situations, such as during error recovery, the balance will be unpredictable, but will recover quickly.

Connection Pool Restrictions

The connection pool has a configurable maximum number of connections. See the parameter ***client.connections.max***.

If a request for a connection is made, and the connection pool has no available connections, and the total number of connections is equal to the maximum so that the request is not allowed to open a connection, then the requesting thread will be queued either until a connection is available, or the request times out. See the parameter ***client.connect.max.time***.

6.2.5 Background Tasks

At regular intervals, a background task is run. (See parameter the ***client.connections.balance.interval***).This background task is referred to as the **Balancer** and is responsible for the following tasks:

1. Closing unused connections, and

2. Sending keep-alive requests to the server, using the persistent connections.

Closing unused connections will reduce the resource requirements on both the client itself and the servers. Note that 1 unused connection on 10 clients translates to 10 unused connections on the servers. Unused connections are closed before the balancing is done. This process will respect the configured minimum number of connections.
Idle connections that are not closed because of the minimum requirement, are referred to as persistent connections.

Keep-alive messages sent on persistent connections prevent the server from closing the connections. If the server did close a persistent connection, the client would get an IO error when the connection was used in the future.

An unused connection is closed when the time since last use exceeds the idle time-out value, see the parameter ***client.connection.max.idle.time,*** and if the number of connections is above the minimum connection limit, see the parameter ***client.connections.min***.

Since the balance interval is likely to be higher than the idle time-out, an unused connection can live in the system for longer than the idle time-out value.

Effectively, the idle time-out parameter should be considered the minimum time before a connection is *allowed* to be closed by the load balancer, not the maximum time for how long a connection can be allowed to remain idle.

Once the idle connections have been closed, the remaining number of connections are the connections necessary to process the load generated by the business application.

6.3 Connection Pool Configurations

Using the parameters that control the connection pool, the management of the connections can be configured in several different ways. In the following, a number of situations will be discussed.

6.3.1 Using a Fixed Number of Connections

To avoid closing and recreating connections it may be preferable to have a fixed number of connections, regardless of the load.

To achieve this the minimum and maximum connection count must be set to the same number of desired connections.
Setting the minimum prevents the connection pool from reducing the number of connections to below the desired number.
Setting the maximum prevents the connection pool from increasing the number of connections to above the desired number.

This keeps the number of connections constant during normal operations. Only connection errors can affect the number of connections negatively. Such a situation will quickly be rectified by the connection pool, returning the number of connections to the desired number.

The benefit of this approach is that despite a varying load from the application, the application threads will not be taxed with the delay associated with opening a connection, except during the initialization phase and possibly during error recovery.

6.3.2 Dynamic Unlimited

Allowing the client to grow and shrink the connection pool as it sees fit, ensures that the client uses the smallest number of connections necessary, while still having enough to efficiently handle the load that is created by the application.

To achieve this the minimum and maximum connection count must be set to zero. **Setting the minimum to zero** allows the client to close all connections to all servers. **Setting the maximum to zero** indicates that the maximum is unlimited.

The benefit of this approach is that the number of connections to servers is reduced to a minimum, without constraining a client that experiences a high load from the application.

6.3.3 Dynamic Upper Bounded

Allowing the client to grow and shrink the connection pool as it sees fit, but only up to a set limit, ensures that the client uses the smallest number of connections necessary, while not overloading the servers with connections when the load is high.

To achieve this the minimum connection count should be set to zero, and the maximum connection count should be set to the desired upper limit.
Setting the minimum to zero allows the client to close all connections to all servers. **Setting the maximum** prevents the connection pool from increasing the number of connections beyond the desired upper limit.

The benefit of this approach is that the number of connections to servers is reduced to a minimum, while a known maximum connection count is set for the client. Using this knowledge, and the number of clients and servers, a maximum number of connections for each server can be calculated as follows:
*Max connections per server = (Max connections per client * number of clients) / number of servers.*

The result of the calculation changes if a server is lost. The maximum number of connections that a server can handle, should be sufficiently high to handle the maximum connections necessary in the case of server failure(s).

6.3.4 Dynamic Fully Bounded

Allowing the client to grow and shrink the connection pool as it sees fit between a lower and upper limit, ensures that a client with notable variations in load is always ready to handle a certain number of calls without the delay which is needed to open the necessary number of connections. At the same time the servers are not overloaded with connections when the load is high. In addition, the connection pool is allowed to reduce the number of connections to a set minimum, for example the number required to handle the typical load.

To achieve this, **the minimum connection count** should be set to the number of connections required to handle the usual load that is generated by the application.
The maximum connection count should be set to the number of connections that are needed to handle the peak load which is generated by the client.

The benefit of this approach is that the connections are kept open for a client which has a load that varies a lot, and which has long periods of low load. This removes the delay incurred by opening connections when the load normalizes.

This type of load could be generated by a service that faces the public. Such a service would be most busy during lunch hours and evening hours. In addition the number of connections to servers is reduced, while ensuring a defined maximum connection count for the client. Using this information and the number of clients and servers, a maximum number of connections for each server can be calculated:

*Max connections per server = (Max connections per client * number of clients) / number of servers.*

The result of the calculation changes if a server is lost. The maximum number of connections that a server can handle, should be sufficiently high to handle the maximum connections in the case of server failure.

6.4 Dynamic Lower Bounded

Allowing the client to grow and shrink the connection pool as it sees fit, while always keeping a minimum number of connections open, ensures that a client with notable variations in load is always ready to handle a certain amount of calls without the delay needed to open the necessary number of connections.

To achieve this, **the minimum connection count** should be set to the number of connections required to handle the usual load generated by the application.
The maximum connection count should be set to zero to indicate “unlimited”.

The benefit of this approach is that for a client which has a load that varies a lot, and which has long periods of low load, the connections are kept open. This removes the delay, incurred by opening connections, when the load normalizes. This type of load could be generated by a service which faces the public. Such a connection would be most busy during lunch hours and evening hours. In addition, the client is not constrained when the client experiences a high load from the application.

This is the default configuration, with the minimum set to 1.

6.5 Active-Active Server Configuration

Configuring the ACSP CCA and Java Client with backup servers that can be used in the event of a failure in the primary server, can be done in multiple ways. This section discusses the active-active configuration for providing backups.

In active-active all servers are running and processing calls from the clients. If one server fails, the other servers are ready to take over for the failed server.

This assumes that the remaining servers have enough capacity to process the extra load.

For an active-active configuration all servers are specified in the host list (***client.connect.host***). This causes the client to do load balancing between all the specified servers. When a server fails, the client simply stops sending calls to the failed server, and they will instead be directed to the remaining servers.

6.6 Active-Standby Server Configuration

The configuration of the ACSP CCA and Java Client with backup servers that can be used in the event of a failure in the primary server, can be done in multiple ways. This section discusses the active-standby configuration for providing backups.

In active-standby the active servers are running and processing calls from the clients while the standby servers are running, but doing little to no work. If the primary servers fail, the standby servers are ready to take over for the failed primary servers.

For an active-standby configuration the active servers are specified in the host list (***client.connect.host***). In addition, the standby servers are specified in the backup list (***client.backup.host***).

To configure a *text-book* active-standby, the client should also be configured to enable the standby servers, but only when the last primary server fails. This is done by specifying a minimum primary server count of 1. See the parameter ***client.connect.min***.

This causes the client to do load balancing between the specified primary servers.

When a primary server fails, the client stops sending calls to that server but keeps load balancing between the remaining primary servers. When the last primary server fails, and the primary server count falls below the configured limit, the client will begin to load balance between the standby servers.

The default minimum number of primary servers is 1. However, setting the minimum number of primary servers to a value higher than 1 will make it possible to enable the standby servers, before all primary servers have failed. This can be helpful when more than one server is required to process the load that is generated by the applications.

By setting the minimum primary server count higher than 1, the standby servers will be enabled before all primary servers have failed. As described above, when the number of remaining primary servers falls below the required number, the client enables the use of the standby servers. Since not all primary servers have failed, this causes the client to load balance between remaining primary servers and backup servers.

6.7 Deviations From Normal Operations

The purpose of the connection pool is to provide the client with working connections to facilitate the successful execution of CCA verbs on the working servers.
In effect, the connection pool is an abstraction level above the specific, physical servers which shields the rest of the clients from using the servers that are unusable.

In order to accomplish this, the connection pool maintains the state of the servers. This state is updated when the client either detects a problem with one of the servers or determines that all a server’s previous problems have been resolved.
An understanding of the connection pool as described above, is assumed in the following.

6.7.1 Activating Secondary Servers

In addition to the primary servers, as described in Section *Server Specification* on page 31, the client can activate the use of secondary servers to ensure capacity during error situations.

When a connection fails, and a server is marked “down” for whatever reason, the connection pool counts the number of active primary servers and compares it with the threshold for activating secondary servers. See the parameter ***client.connect.min.***

If the number of active primary servers is below the threshold, all servers specified as backup servers are added to the set of servers, used by the load balancer. See the parameter ***client.backup.host.*** This applies to the normal creation of connections executed in the application threads, and to the balancer running in the background.

6.7.2 Error Detection

There are two conditions for determining if a problem has occurred with the server:

- The client's ability to communicate with the server. This is a prerequisite for executing verbs.
- The return/reason codes returned by the server. These are returned when a verb is executed on the relevant server.

It is the return/reason codes that provide the client with the ability to detect errors with the cryptographic environment on the specific server.

To determine if a server's problems have been resolved, the client will establish a connection to the server, and if it is successful, send a call to the server. This call queries the status of the server's cryptographic environment. If the client is able to communicate with the server, and the server reports that the cryptographic environment is operational, the client will consider previous problems with the server as resolved. This operation is done in a background task. See section *The Reconnector* on page 38.

Communication errors are divided into:
- failure to connect to a server, and
- IO errors when using a connection. IO (input/output) errors are errors that occur when attempting to read or write data from or to a connection.

6.7.3 Connection Errors

If the client cannot connect to a server, the client assumes that this is an indication that the server is unreachable and therefore unusable.

There are many reasons why a client would fail to successfully establish a connection to a server. These may range from network issues and servers not running to SSL-handshake failures and time-outs. See the parameter ***client.connect.timeout.ms***. Regardless of the cause, the consequence is that the server is unusable. A failure to connect will result in the client marking the server as “down”. Subsequent attempts to create a connection will bypass any server marked as “down”.

6.7.4 IO Errors

The client will consider it an IO error if the client fails to read or write data before a specified time-out. See the parameter ***client.connection.io.timeout.ms***.

Reasons for this may be one of the following:

• If a server fails after a connection has been established, the failed server will be unable to service the connection. This will ultimately result in a failure to send or receive data on the connection.

• If the server is running very slowly, for example if it is overloaded. Although this could be a simple network glitch, the client follows a fail-fast approach and will interpret the error as an indication that the server or the network path to the server is having problems.

The client will mark the connection as broken and subsequently return the connection to the connection pool. With the broken attribute set, the occurrence of the IO error is evident to the connection pool. A broken connection is handled by marking the server to which it is or was connected, as “down”.

6.7.5 Server Errors

The client can also determine the health of the server based on the returned return/reason codes.

The set of return/reason codes which specify that a server is having problems, is the retry codes. These are configurable, see the parameter ***client.retry.codes.*** However, by default, they contain the most common hardware failure return/reason codes issued by CCA and ICSF.

When the return/reason code indicates that a call has failed, the client will check to see if the return/reason code is in the list of retry codes. If the returned return/reason code is on the retry list, the client marks the call as having an error in the call context. The client then proceeds to return the connection to the connection pool before attempting a retry of the call.

The client will retry a call, up to the retry limit, see the parameter ***client.connection.retry.max***, as long as the time used is below the time limit. See the parameter ***client.connect.max.time***.

The connection pool has access to the call context when the connection is returned to the connection pool.
When the connection pool detects that the call has an error, the connection is treated as if it is broken. Like with a broken connection, the server, that the connection goes to, is marked as being down.

The reason for marking the call with an error which results in the server being marked down, is to prevent a call from being retried on the same server.

6.7.6 Server Recovery

In order for the ACSP CCA and Java Client to operate, it needs available servers.
Recovering the use of a lost server is usually the job of the reconnector.
However, when the ACSP CCA and Java Client detects that all servers, both primary and backup, are down, it cannot operate.
In this case the client reactivates all servers to give the servers a second try. This is done to

avoid waiting up to several minutes for the reconnector background task to run. The servers that still have problems, will quickly fail again and be deactivated. However, the servers that have recovered, will continue to process calls from the client.
The process is not completely seamless, and some errors and delayed response times are to be expected during the recovery. However, a successful recovery produces far fewer errors than not being able to operate, while waiting for the reconnector to run.

6.7.7 The Reconnector

At regular intervals, see the parameter ***client.connections.poll.interval***, a background task is run.
This background task is referred to as the reconnector and is responsible for determining when servers return to working order. The reconnector will iterate through the list of servers, and for each server that is marked as being down, it will attempt to create a connection to that server. If the connection succeeds, the client verifies the server's cryptographic environment. If the client finds a working cryptographic environment, the server is reactivated, the connection is added to the pool of connections and made available for use.

When the reconnector reactivates a server it checks the number of active primary servers, and whether backup servers are in use. If backup servers are in use, and the number of primary servers are back above the threshold, see the parameter ***client.connect.min***, the reconnector takes the backup servers out of use.

6.8 Error Handling Configuration

Error handling can be configured in several different ways via the parameters that controls it.

6.8.1 Controlling the Time Needed to Detect a Failed Server on Connect

By default, the time it takes for a TCP connection request to fail may be anywhere from several seconds to several minutes. For a general purpose protocol this is acceptable, but for the IBM Advanced Cryptographic Service Provider product, where clients and servers are known to be in close proximity, a long time-out serves no purpose.
For systems set physically close together it is a good rule of thumb that if a connection has not completed within a few hundred milliseconds, the connection will not complete at all.

6.9 Summary of Parameters

The parameters referenced here are listed below. For a complete description of their use, see “Configuration Reference” on page 44.

6.9.1 Parameters Controlling Load Balancing

The following table lists the parameters that control load balancing:

Parameter	Description
client.connections.min	This controls the number of persistent connections in the client. These connections will not be closed, even when idle.
client.connections.max	This controls the upper limit of the number of connections that the client is allowed to have.
client.connect.max.time	This controls the maximum time allowed for retrieving a connection.
client.connect.host	The comma-separated list of server addresses. The client will load balance between these.
client.connections.balance.interval	The time interval between successive runs of the balancer background task.

client.connection.max.idle.time	The time a connection must be unused for the balancer task to consider it idle and eligible for closing.

6.9.2 Parameters Error Detection and Recovery

The following table lists the parameters that control error detection and recovery:

Parameter	Description
client.connect.timeout.ms	The time in milliseconds that a connection attempt is allowed to take before failing. This time includes the TLS handshake for secure connections.
client.connection.io.timeout.ms	The time in milliseconds that the client will wait for data from the server before failing.
client.retry.codes	The list of return/reason codes that will trigger the client to retry the call on a second server.
client.connection.retry.max	The number of times the client is allowed to retry a call before failing.
client.connections.poll.interval	The time interval between successive runs of the reconnector background task.
client.connection.retry.interval	The time to wait between successive retries of a call.
client.backup.host	The comma-separated list of server addresses to use as backups.
client.connect.min	The minimum number of primary servers. If the number of active primary servers decreases below this number, backup servers are activated.

7. Auto Configuration Feature

The ACSP Server auto-configuration feature is a facility which makes it much simpler to configure ACSP Clients with the ACSP Server defined ports. When not using the auto-configuration feature, the client must know exactly which ports are defined for either TCP or SSL/TLS service ports for the ACSP Server protocols (CCA, P11, and so on).

Using the auto configuration feature, the ACSP Client will ask the ACSP server which ports are active on the server and auto configure the ACSP Client to use those ports. This is implemented through a ACSP Server protocol called ACP (Auto-Configuration Protocol).

Here are the benefits using the auto-configuration feature:

• Simpler ACSP Client configuration

• All ACSP Clients are supported: CCA, PKCS#11, Java and monitor.

• Only one port, ACP, to synchronize with the ACSP Server

• The ACSP Server can change ports for other protocols, CCC, CCA, P11 and monitor, and transports, TCP and SSL/TLS, without affecting client configurations.

• Backup and fail-over hosts can be server defined and automatically defined in the client.

7.1 How It Works

Using the ACP protocol, the auto-configuration feature performs the following actions:

1. It asks the ACSP Server about the configured ports for the transport type

2. It configures the ports for CCC, CCA, and P11 protocols, hosts, and services

3. It creates a connection as requested

Simple but effective.

7.1.1 ACSP Client Auto-Configuration

To activate the auto-configuration, add the following statement to the configuration:

client.autoconf.host=hostname

To select the transport method use the following statement:

client.autoconf.tranport=ssl

Always use SSL/TLS transport in production environments for best security.

And optionally add the following statement:

client.autoconf.port=nnnnn

In this the host name is the server host name or IP address.

The transport selects which of the ACP ports to use. The default ports are 9900 for TCP transport and 9901 for SSL/TLS transport.

7.1.2 ACSP Server Auto-Configuration

The server must have been configured with the ACP protocols enabled. Please see the manuals or ACSP-4105 IBM Advanced Cryptographic Service Provider - ACSP z/OS Server Installation, Configuration and Operation Guide for more information.

8. Generation of Client Key and Certificate

As previously mentioned, the connection between the ACSP Clients and the ACSP Server are negotiated and protected using the SSL/TLS protocols.

In order to use SSL/TLS to encrypt sessions between the client and the server, the client needs to be configured with a private key and corresponding certificate which must be present in the server's truststore. See the server documentation for more information on this. These keys can be generated in a number of ways.

This section discusses how this can be accomplished by using the *openssl* command line tool. The tool must be downloaded separately from for example www.openssl.org, or installed using the operating system's package manager. On the i5/OS platform the OpenSSL command is available in the PASE environment.

8.1 Key Generation Using OpenSSL

The first step is to generate the client's private RSA key. This can be accomplished using the following command:

```
openssl genrsa -aes128 -f4 -out client.pem 2048
```

This will generate a 2048 bit RSA key with the commonly used “Fermat-4” public exponent (public exponent value equals to 65537). The key will be stored in the client.pem file. The program will prompt for the password to be used for encrypting the private key file. The encryption will be done using the AES-128 algorithm.

8.2 Generation of the Certificate Request Using OpenSSL

The next step is to generate a certificate request for the newly generated private key. To this end the following command can be used:

```
openssl req -new -key client.pem -out client.csr
```

The program will interactively prompt for information to be included in the certificate (subject name and so on.). Further, it will prompt for the password on the private key file (which was set in the previous step). The resulting certificate request will be put in the client.csr file.

8.3 Issuing the Certificate Using OpenSSL

As a final step a certificate needs to be issued. This can either be done by a certificate authority, or the certificate can be self-signed. This depends on the internal key management policy, as well as the trust policy configured on the ACSP Server. In the following we discuss how to self-sign the certificate using OpenSSL. The following command can be used:

```
openssl x509 -req -days 365 -in client.csr -signkey client.pem -out
client.crt
```

This will generate a self-signed certificate based on the certificate request and key just generated. The certificate is stored in the ***client.crt*** file.

This file must be installed in the ACSP server truststore. Please consult the ACSP Server documentation for more information on this.

8.3.1 Adding a Subject Alternative Name

Subject Alternative Name extensions are used with the authorization features of the ACSP Server.

Note! The SAN attribute is not used by the z/OS ACSP Server.

The extensions must be of type URI and be prefixed with 'acsp:'.

To include a Subject Alternative Name in the signed certificate, the signing process must be told to embed the SAN x509 extension. To indicate that the certificate is to be associated with the ACSP Client userID client1, create a file named **client1.ext** containing the following statement:

```
[san_ext]
subjectAltName=URI:acsp:client1
```

Then sign the certificate using the following command:

```
openssl x509 -req -days 365 -in client.csr -signkey client.pem -out
client.crt -extfile client1.ext -extensions san_ext
```

8.4 Adding a Client Application to i5/OS DCM

In the following, it is assumed that the Digital Certificate Manager has been installed and configured. Please refer to DCM documentation for information on how to start DCM.

In a browser, navigate to open the IBM Systems Director Navigator, log in and do the following:

1. Under **IBM I Management** select: **Internet Configuration,**
2. then select **Digital Certificate Manager** and log in again.
3. In **DCM**, in the left pane, select a **Certificate Store**, for instance **SYSTEM*.
4. Click **continue**.
5. Enter the certificate store password.
6. Click **continue**.
7. Under **Fast path > Work with client applications**, click **add application**.
8. Enter an Application ID. This application ID will be used by the client as a handle to the private key and trusted certificates.
9. Fill in the remaining information:
10. **Exit** Program: ***NONE**
11. **Exit** Program Library: Thread safe: **NO**
12. Multi-threaded job action: run program and send message.
13. Application user profile: ***NONE**
14. Define the CA trust list: **YES**
15. Certificate revocation list CRL checking: **YES**
16. Application Description: **ACSP Client**

8.5 Creating a Client Private Key in i5/OS DCM

Under Fast path > Work with server and client certificates:

1. Click **create**.
2. Select **VeriSign or other Internet certificate authority (CA)**.
3. Click **continue**.
4. Fill in the key and certificate information and click **continue**.
5. You will be presented with an encoded certificate signing request. Copy the certificate signing request data into a file, and have the certificate signed by a trusted CA.
6. When the CA has processed the signing request, transfer the signed certificate to the AS/400 PASE environment.

    - The certificate should be PEM encoded, , or DER. It is recommended to have it ASCII-armored.
7. Under **Fast path > Work with server and client certificates**, click **import**.
8. Enter the fully qualified path to the signed certificate in the PASE environment. Click continue
    - If **DCM** asks for a password for importing a DER encoded certificate, check that the file was uploaded correctly, or convert it to PEM encoding.
9. In **DCM** under **Fast path > Work with client applications**. Select the application that needs to use the private key and click **work with applications**.
10. Click **Update Certificate Assignment**. Select the certificate and click **Assign new Certificate**.

8.6 Adding Trusted Certificates in i5/OS DCM

In order for the ACSP CCA client to establish a secure connection with the server, the server's self-signed certificate or its CA certificate must be added to the DCM as trusted.

Note: See details about how to extract a certificate from the Java keystore used by the ACSP server in the or the ACSP-4105 IBM Advanced Cryptographic Service Provider - ACSP z/OS Server Installation, Configuration and Operation Guide.

1. Transfer the self-signed certificate or the CA certificate to the i5/OS PASE environment.
    - The certificate should be PEM encoded or DER. It is recommended to have it ASCII-armored.
2. In **DCM** under **Fast path > Work with CA certificates**, click **import**.
3. Enter the fully qualified path to the certificate, click **continue**.
    - If **DCM** asks for a password for importing a DER encoded certificate, check that the file was uploaded correctly, or convert it to PEM encoding.
4. Enter a label for the certificate. Click **continue**.
5. In **DCM** under **Fast path > Work with client applications**. Select the application that should trust the certificate, and click **work with applications**.
6. Click **Define CA Trust List**.
7. Put a check mark in the check box for the imported certificate and click **OK**.

8.7 Importing a Key/Certificate Pair to i5/OS DCM

If the private key to be used has been generated on a different system, the key can be imported into the DCM system. The key should be packaged, with its signed certificate, in a PKCS#12 file.

1. Transfer the PKCS#12 file to the i5/OS PASE environment.
2. Click **Fast path > Work with server and client certificates**
3. Click **import**
4. Enter the fully qualified path to the PKCS#12 file, click **continue**.
5. Enter the password for the PKCS#12 file and the certificate label to use.
6. In **DCM** under **Fast path > Work with client applications,** select the application that should use the private key, and click **work with applications**.
7. Click **Update Certificate Assignment**. Select the certificate and click **Assign new Certificate**.

9. Configuration Reference

This chapter contains a reference for all configuration parameters in the ACSP Clients.

The configuration parameters are organized in alphabetical order, and each configuration parameter contains a header, which describes the relationship for the parameter.

The header contains:

- The software package to which the configuration parameter applies, for example the ACSP Java Client.
- The Operating System platform to which the parameter applies. Some parameters are specific to the platform they run on.
- The ACSP Server API for which the parameter is used

The following is an example of a header:

Applies to	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X			X		X	X	

The *Applies to* information lists the IBM Advanced Cryptographic Service Provider software package to which the configuration parameter adheres. Several packages can be listed here as shown in the example above.
The last columns indicate which API's are supported by the ACSP Server.

The columns below this indicate which *platforms or Operating Systems* this configuration are available for.

Each configuration parameter also contains:

- a description
- syntax and defaults definition
- examples, when relevant

The syntax notation format is as follows:

parameter.name = opt1 | <u>opt2</u> | opt3

The underscored option denotes the default.

Parameters can also be comma separated lists, and sometimes brackets are used to denote optional parameters, as in the following example:

client.retry = condition [,condition] […]

Some configuration parameters use notation with angle brackets to indicate customer insertions, as the following examples show:

monitor.agent.bind.address= <IP/hostname>

service.<name>= port-name [,port-name] […]

The configuration parameters in the configuration files are typically written in lower case and are in most situations not case-sensitive, otherwise this will be noted.

Each configuration parameter should be kept on a single line – multi line parameters are not possible.

Comments to the file can be specified by using a hash (#) sign in column 1.

9.1 Client Auto-Configuration Host

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	X

The ***client.autoconf.host*** property specifies the host name, or a list of host names, for the ACSP Servers which will be used with the ACSP Client auto-configuration feature.

The ***client.autoconf.host*** parameter must be defined to use the ACSP Server auto-configuration feature. If this parameter is not defined the auto-configuration feature is not used.

If a list of host names are specified, the next host will be tried in the specified sequence until a connection succeeds. If no alternate hosts are defined, auto-configuration fails.

Syntax

client.autoconf.host = <hostname> [,<hostname>][...]

Example

client.autoconf.host = acsp1.example.com, acsp2.example.com

9.2 Client Auto-Configuration Port

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	X

The ***client.autoconf.port*** parameter is used to redefine the default ports used for the auto-configuration feature. If this parameter is not defined, the following ports are used:

9900 for *tcp transports* and 9901 for *SSL*/TLS *transports*.

Note: Reassigning the port number also means that the transport has to be manually chosen. When for example the *client.autoconf.port* has been assigned to 9977, and the *client.autoconf.transport* has been assigned to ***ssl***, the ACP port on the ACSP Server must be assigned to use the scheme **ssl:acp**. The same applies to transport TCP which requires a scheme **tcp:acp**.

The configuration paramter for the ***client.transport,*** ACSP CCA and PKCS#11 Clients, or the ***client.autoconf.transport,*** ACSP Java Client, is used to determine the connection type.

Syntax

client.autoconf.port = <port number>

Example

client.autoconf.port = 9967

9.3 Client Auto-Configuration Transport

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	X

To apply the transport scheme to use for auto-configuration client connections to the server specify the ***client.autoconf.transport*** parameter. The transport scheme can be either **tcp** or **ssl**.

The auto-configuration transport definition is used to direct the ACP protocol to use either TCP or SSL/TLS when it is set up for communication with the ACSP Server.

If for example the ACP ports have been assigned to use SSL/TLS with the ***port.ssl-acp.9901=ssl:acp*** then the transport has to be ***client.autoconf.transport=ssl****,* otherwise the auto-configuration will fail. The default is ***ssl****.*

Note: If the ***client.autoconf.transport*** parameter is not defined, the definition is read from the ***client.transport*** parameter. See “Client Transport” at page 58.

Syntax and Defaults

client.autoconf.transport= tcp **|** <u>**ssl**</u>

9.4 Client Connect

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	

Note: This parameter should not be used when using the auto-configuration feature.

The ACSP Java Client has an advanced connection setup for managing transparent client connections. It allows configuration of the ACSP Client to use multiple threads and connection reuse for best performance.
It also allows the use of multiple defined hosts for load balancing and fail-over in case of host connection failure. In case of fail-over, the transaction is cached and sent to an alternate serving host.

Syntax

client.connect =<hostname> [: <service>] | <host-groupname> [: <service>] [, …]

Here <hostname> is the logical name for the defined host, <host-groupname> is the logical name of the grouped host and <service> is the logical name of a defined service.
If “: <service>” is not specified, the default service which contains all ports will be used.

Example

```
client.connect = olsen:prod,mvsf:prod
```

9.5 Client Backup

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	

Note: This parameter should not be used when using the auto-configuration feature.

The ACSP Java Client has an advanced connection setup for managing transparent client connections. It allows configuration of the ACSP Client to use multiple threads and connection reuse for best performance.
It also allows the use of multiple defined hosts for load balancing and fail-over in case of host connection failure. In case of fail-over, the transaction is cached and sent to an alternate serving host.

The ***client.backup*** parameter specifies a list of ACSP Servers and services the client should connect to when an insufficient number of primary servers are available. See Client Connect Minimum below. The entries are separated by commas. When activated, the ACSP Client will include these servers in the normal load balancing.

Note: Only available for the balanced load balancing scheme. See Client Load Balancing Algorithm below.

Syntax

client.backup = <hostname> **|** <host-groupname> : <service> [, …]

where <hostname> is a logical name for the defined host, <host-groupname> is the logical name of the grouped host and <service> is the logical name of a defined service.

Example

```
client.backup = olsen:prod,mvsf:prod
```

9.6 Client Connect Host

Related Software Package	ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X			X	X

Note: This parameter should not be used when using the auto-configuration feature.

The ***client.connect.host*** parameter specifies a list of ACSP Servers that the client should connect to. The server names are separated by commas. The ACSP Client will create an equal number of connections to each host and perform load balancing between them, so that the ACSP Client has an equal number of calls going to each server.

Syntax

client.connect.host **=** <server> [,<server>] [...]

Examples

client.connect.host = acsp1.example.com, acsp2.example.com

9.7 Client Backup Host

Related Software Package	ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X			X	

Note: This parameter should not be used when using the auto-configuration feature.

The ***client.backup.host*** parameter specifies a list of ACSP Servers the client should connect to when an insufficient number of primary servers are available. See “Client Connect Minimum” on page 48. The server names are separated by commas. When activated, the ACSP Client will include these servers in the normal load balancing.

Note: Only available for the balanced load balancing scheme. See “Client Load Balancing Algorithm“ on page 50.

Syntax

***client.backup.host* =** <server> [,<server>] […]

Example

client.backup.host = acsp3.example.com, acsp4.example.com

9.8 Client Connect Minimum

Related Software Package	ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X			X	

The ***client.connect.min*** parameter specifies the minimum number of primary servers which is required. It is specified by the *client.connect.host* parameter, see “Client Connect Host” on page 47. When the number of active primary servers falls below this number, the secondary backup servers, which are specified by the *client.backup.host* parameter, will be activated.

The default is 1.

Note: Only available for the balanced load balancing scheme. See “Client Load Balancing Algorithm“ on page 50.

Syntax

***client.connect.min* =** nn

Example

client.connect.min = 1

9.9 Client Connect Port

Related Software Package	ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X			X	X

Note: This parameter should not be used when using the auto-configuration feature.

The ***client.connect.port*** parameter specifies the port number to connect to for the ACSP Client. The port number should specify a valid port for the client using the transport that is defined by the ***client.transport*** parameter.

The ACSP CCA Client uses the CCA protocol and the ACSP PKCS#11 Client uses the P11 protocol.

All ports will automatically be added to the default service name.

Syntax

client.connect.port = <port number>

Examples

If the server has defined the following ports and transports:

```
port.ssl-cca.9994 = ssl:cca
port.ssl-cca.9995 = ssl:p11
```

Use the following configuration parameters for the ACSP CCA Client:

```
client.transport=ssl
client.connect.port = 9994
```

or use the following configuration parameters for the ACSP PKCS#11 Client:

```
client.transport=ssl
client.connect.port = 9995
```

9.10 Client Definition of Server Host Groups

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	

Note: This parameter should not be used when using the auto-configuration feature.

To define the host names of the servers which are available on the network, use the **host.<name>** configuration parameter. You must use either the DNS name or IP address as host name. Not all the hosts that are listed, are assumed to be valid for all the defined protocols. A larger number of host names can be specified, and only those hosts defined on the ***client.connect*** will be used for connection.

Syntax

hostgroup.<name>= <hostname> [,<host-name>] […]

Here **<name>** is a logical name for the defined host group and **<hostname>** is the logical name of the grouped host.

Examples

```
host.olsen = 9.37.2.6
host.mvsf = mvsf.dk.ibm.com
```

```
hostgroup.prodcca=olsen,mvsf
```

9.11 Client Connect Maximum Time

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The ***client.connect.max.time*** parameter specifies the maximum amount of time to wait before the ACSP Client should give up delivering a call to the server. The time is specified in milliseconds (ms).

The default value is 30000 ms or 30 seconds.

Syntax

client.connect.max.time = nnnnn **|** **30000**

9.12 Client Connect Maximum Idle Time

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The ***client.connection.max.idle.time*** parameter specifies the maximum time in seconds that a connection can be idle, before it can be closed by the ACSP Client. At regular intervals the ACSP Client will trim the connection pool and close connections that have been idle for too long.

The default is 30 seconds.

Syntax

client.connection.max.idle.time = nn **|** **30**

9.13 Client Connection IO Timeout

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X		X	X	X

The ***client.connection.io.timeout.ms*** parameter specifies the time in milliseconds a read or write operation on a socket can block, before control is returned to the ACSP Client. Depending on the state of the socket, the ACSP Client will retry the operation or determine that the read/write operation has failed. The ACSP Client may retry the call before returning to the calling program. See parameter ***client.connection.retry.max*** on page 52.

The default is 30 seconds.

Syntax

client.connection.io.timeout.ms = nn **|** <u>**30000**</u>

Note: This parameter is not supported on i5/OS.

9.14 Client Connection Retry Interval

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X		X	X	

The ***client.connection.retry.interval*** parameter specifies the interval between retries from the ACSP Client in milliseconds. Longer intervals provide better opportunity for recovery in cases of transient downtime, but also increases the response time for the call.

The default is 100 ms.

Syntax

client.connection.retry.interval = nnn **|** <u>**100**</u>

9.15 Client Connection Retry Maximum

Related Software Package	ACSP Java Client, ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	

The ***client.connection.retry.max*** parameter specifies the maximum number of times delivery of a CCA call should be retried.

The default value is 0.

Syntax

client.connection.retry.max = nn | **<u>0</u>**

Examples

A value of 0, the default, will send a call to a server once. This is normal execution. The result of that will be returned to the calling application, regardless of success or failure.

A value of 1, will send a call to a server. This is normal execution. Should the call fail with any of the return/reason codes that are specified by ***client.retry.codes***, the call will be resent. The result of that will be returned to the calling application, regardless of success or failure.

A value of 2, will send a call to a server. This is normal execution. Should the call fail with any of the return/reason codes that are specified by ***client.retry.codes***, the call will be resent. Should the retried call fail again, the call will be resent a second time. The result of that will be returned to the calling application, regardless of success or failure.

9.16 Client Connection Time-Out

Related Software Package	ACSP Java Client, ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X		X	X	

During processing of calls to the ACSP CCA Clients, java and C, it may become necessary to open a connection to a server. Opening a connection adds extra time to the processing of a call.

The *client.connect.timeout.ms* parameter controls the maximum time a connection attempt is allowed to take, before aborting the connection attempt and considering it a failure.

If this parameter is set too low, connections may not have enough time to be successfully established. This leads to connection failures where there are none.

If this parameter is set too high, the client will wait longer than necessary before aborting a failed connection attempt.

The default is 10000 milliseconds (10 seconds).

Syntax

client.connect.timeout.ms= nn | **<u>10000</u>**

Note: The CCA client does not use this parameter for TCP connections.

9.17 Client Connections Maximum

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The ***client.connections.max*** parameter specifies the maximum number of connections that the ACSP Client can create to each server, before a calling thread is forced to wait for a connection to become available. The actual number of connections that are created will depend on the number of concurrent calls made to the application. A value of 0 means unrestricted.

The default value is 0.

Syntax

client.connections.max = nn | <u>**0**</u>

9.18 Client Connections Minimum

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The ***client.connections.min*** parameter specifies the minimum number of outgoing connections that the ACSP Client must maintain to each server during periods of little or no activity. Maintaining idle connections ensures that the initial calls after a period with no activity will not be delayed by the need to reestablish connections to one or more ACSP Servers.

The default is 0.

Syntax

client.connections.min = n | 0

9.19 Client Connections Balance interval

Related Software Package	ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The ACSP Client maintains a pool of connections to the active servers. To ensure a reasonable load balancing, the ACSP Client maintains an equal number of connections to each active server. To ensure that the connection pool remains balanced, even if a server is lost or existing connections are otherwise broken, the ACSP Client will regularly balance the connection pool.

The ***client.connections.balance.interval*** property specifies the interval in seconds at which the connection pool should be balanced.

The default is 300 seconds (5 minutes).

Syntax

client.connections.balance.interval = nn | **300**

9.20 Client Connections Poll interval

Related Software Package	ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The ACSP Client maintains a pool of connections to the active servers.
If a server experiences an error, the ACSP Client will take the server out of use. The ACSP Client will regularly poll the lost servers to determine if the servers have returned to working order.

The ***client.connections.poll.interval*** property specifies the interval in seconds, at which the connection pool should poll lost servers to determine if the servers have returned to working order

The default is 300 seconds (5 minutes).

Syntax

client.connections.poll.interval = nn | **300**

9.21 Client Definition of Server Host Names

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	

NOTE: This parameter should not be used when using the auto-configuration feature.

To define the host names of the servers available on the network use the **host.<name>** configuration parameters. You must use either the DNS names or IP addresses as host name. Not all the hosts which are listed, are assumed to be valid for all the defined protocols. A larger number of host names can be specified, and only the hosts defined on the ***client.connect*** will be used for connection.

Syntax

host.<name>= <hostname>

Here **<name>** is a logical name for the defined host, and **<hostname>** is the host name or IP address of the ACSP Server host.

Examples

```
host.local=localhost
host.olsen=169.254.25.129
host.mvsf=mvsf.dk.ibm.com
```

9.22 Client Port Definitions

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	

Note: This parameter should not be used when using the auto-configuration feature.

To define a port to connect to from the ACSP Client, the port must be defined using the following syntax:

Syntax

port.<name>.nnnn = *[transport:]protocol*

Here **<name>** is a logical assigned name, ***nnnn*** is the port number and ***protocol*** the protocol name to be assigned to the port.

Examples

```
port.tcp-ccc.9999 = ccc
port.tcp-cca.9997 = tcp:cca
port.tcp-acp.9900 = tcp:acp
```

To enable **SSL**/TLS communication for the port name, specify transport ***ssl*** in front of the protocol and separate with colon, as in the following example:

```
port.ssl-ccc.9996 = ssl:ccc
port.ssl-cca.9994 = ssl:cca
port.ssl-acp.9901 = ssl:acp
```

The ***<name>*** of the ports must be specified, to be used as reference for other parameters. The name syntax must not contain spaces and periods.

Note: Normally when ***ssl*** is used, the ports without ***ssl***, should be blocked in the firewall. This will allow only local host connections to access the server without ***ssl***.

9.23 Client Load Balancing Algorithm

Related Software Package	ACSP Java Client, ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
		X					X	

The ACSP Client on 'IBM i' has two different load balancing algorithms which can be specified by the ***client.load.balance.algorithm*** parameter.

The ***balanced*** algorithm is used by the ACSP CCA Client on all platforms and is described in more detail in the “ACSP CCA Load Balancing Basics” on page 31. On 'IBM i' it is one of two available algorithms. The balanced algorithm will ensure that all configured servers are handling an equal number of calls from the client. This is done by maintaining a count of calls in progress for each server, and dispatching calls to the server with the lowest call count. Even with multiple clients, this algorithm will keep all ACSP Servers loaded to the same degree.

The algorithm is optimized for multi-threaded applications with concurrent calls. If used by a single-threaded application performing sequential CCA calls, the balanced algorithm will balance the calls similar to a round-robin algorithm. The ACSP CCA Client will randomize the server list such that each client has a different order of servers.

On ''IBM i', the background threads that are used by this algorithm requires a job descriptor specifying the 'allow-multithreading' option.

The ***roundrobin*** algorithm can be used on 'IBM System i' by single-threaded applications that process calls sequentially, and it does not require a job descriptor with the 'allow-multithreading' option.

The default algorithm is *balanced*.

Syntax

client.load.balance.algorithm= roundrobin **|** **<u>balanced</u>**

9.24 Client Retry Codes on CCA Return/Reason Codes

Related Software Package	ACSP Java Client, ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	

The ***client.retry.codes*** parameter specifies which return-/reason-codes will cause the client to automatically retry the verb. Retries can be executed on a different server than the one used for the first execution attempt, thus preventing a possible server issue to affect the ACSP Client.

The ***client.retry.codes*** configuration parameter allows for definition of all the CCA and ACSP Client return/reason codes which must be retried by the client. The default setup already includes a list of codes which will be retried, and this setup will be adequate for most installations. However, the list can be changed by setting the configuration parameter.

When defining this parameter, be aware that you need to include the defaults also, if new return/reason codes are added.

The default configuration statement is:

```
8/=/753,8/=/1100-1115,
12/=/0/12/28/197/324/338/3006/3007/3014,/3015/3023/3048/3072/3078/3081,
12/=/3100-149,
12/=/6009/6016/6032/6036/10044/10048/10060/11024/11036/11040/11044,
12/=/11048/11052/11056/11060/11064/36068/36069/36156/36168/36173,
12/=/36174/36182/36189/36211,
16/=/4/336/337/556/708/709/712
```

Syntax

client.retry.codes= condition [,condition] [...]

Here **condition** is:

Retry on all return code (rc) greater than	rc/>
Retry on return code (rc) with reason codes greater than reason	rc/>/reason
Retry on return code (rc) with reason codes less than reason	rc/</reason
Retry on return code (rc) with reason code equal to reason	rc/=/reason
Retry on return code (rc) with reason codes sequence (reason-reason)	rc/=/reason-reason
Retry on return code (rc) with specific reason codes (reason/reason[/...])	rc/=/reason/reason/...

Make sure to specify all the default conditions, if new conditions are added.

Example

```
client.retry.codes = 8/=/1100-1115,12/=/338
```

9.25 Client Service Definitions

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X		X	X	X

The service definition is used to logically group a set of port names under a service name. The service name is a logical name, which is used find a port name for a specified host definition.

All ports are automatically added to a default service, which is used by the client connect definition, if no service is specified, see “Client Connect” on page 46.

Syntax47

service.<name> = port-name [,port-name] […]

Example

```
service.prod = ssl-cca,ssl-acp
```

9.26 Client Transport

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The ***client.transport*** parameter specifies if the connection should be encrypted using the SSL/TLS protocol, or if it should use no encryption.

To apply the transport scheme to use for client connections to the server you need to specify the ***client.transport*** parameter. The transport scheme can be either **tcp** or **ssl**.

The transport scheme is used to direct the protocols CCA, CCC, and so on, to the transport mechanism setup for communicating with the server.

Note: In the ACSP CCA and ACSP PKCS#11 Clients, the ***client.transport*** parameter is also used by the auto-configuration feature to determine if SSL/TLS is used for connection with the auto-configuration protocol (ACP).

The default is ***ssl***.

Syntax

client.transport= tcp | **ssl**

9.27 Client Unchecked Server Activation

Related Software Package	ACSP Java Client, ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	

The ***client.reactivate.unchecked*** parameter controls how the client behaves if all servers are lost.

When the client experiences problems with a server, the client will mark that server as down and stop using it. If all servers are marked down, the client cannot process calls. The parameter controls how the client attempts to recover the situation.

When set to *true* the client will unconditionally reactivate all servers, when it marks the last server down. This is done without checking if the servers are usable. It will cause subsequent calls to attempt to establish a connection to the servers. Until at least one server is recovered, all calls to the client will attempt to reconnect to each server. Because all servers are unusable, the calls will hang while waiting for the connections to fail. The delay is determined by the client connection time-out parameter and the number of servers including backups, see Client Connection Time-Out, on page 52. A timeout of 1 second, and 2 servers will result in a delay of 2 seconds per call.

Once a server is recovered, the client will quickly detect the server and begin sending calls to that server.

Set to *false*, the client will not reactivate servers unconditionally. In the event that all servers are marked down, subsequent calls will fail fast. This ensures that the client does not spend time on repeatedly connecting to an unusable server. The time it takes from a server is recovered, till the client detects that the server is usable is controlled by the client connections poll interval, see “Client Connections Poll interval“ on page 54. The poll interval is usually significantly less than the time needed to recover a failed server and is as such not a significant delay to recovery.

The default for this parameter is set to *true* for backwards compatibility with previous versions of the client. For applications with strict response-time requirements, the recommended setting of this parameter is *false*.

Syntax

client.reactivate.unchecked **=** false **|** **<u>true</u>**

9.28 Logging Level

Applies to	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	C

The log levels defined as the lowest level to be displayed are the following:

trace	This shows trace, debug, warning, info, error, and fatal messages in the log
debug	This shows debug, warning, info, error, and fatal messages in the log
info	This shows info, warning, error, and fatal messages in the log (default)
warn	This shows warning, error, and fatal messages in the log
error	This shows error, and fatal messages in the log
fatal	This shows fatal messages in the log

Syntax

loglevel = *level* **|** ***info***

9.29 Logging Max File Size

Applies to	ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X			X	X

The ***log.maxfilesize*** parameter specifies a maximum size in bytes of the log file. See also “Logging Target” below. When the file size has reached the limit, it is closed and renamed, and a new file with the same original name is opened. The renamed file has a time stamp inserted before the file name extension. A value of 0 which is the default, means no log rotation based on size. The value can be suffixed with KB or MB.

A warning is issued if value >= 2GB.

Note: The size may be a few log lines bigger that the limit, especially in multi-threaded clients.

Syntax

log.maxfilesize = n **| 0**{KB | MB}

where n is a number.

Examples

log.maxfilesize = 1048576
log.maxfilesize = 1024KB
log.maxfilesize = 1MB

All examples specify the same size limit.

9.30 Logging Configuration (Log4j)

Applies to		ACSP Java Client						
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	

The path and the file name to the Log4j configuration file can be specified with the ***logj4.configuration*** configuration parameter. The default is *log4j.properties* in the current directory, if it is present.

Syntax

log4j.configuration = <path/><*log file name*>

Example

log4j.configuration = ./config/log4j.properties

9.31 Logging Target

Applies to		ACSP CCA Client, ACSP PKCS#11 Client						
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X			X	X

The ***log.target*** parameter specifies where to write the log output to. Except for a few reserved words, the ***log.target*** parameter specifies a file to which log output is written.

It is recommended that the path that is given, is absolute. A relative path will work, but it will be relative to the work directory of the application, not the directory where ACSP Client has been installed.

Note: On IBM i, an absolute path must be used. In addition, the log.target must exist.

On AIX, Linux and zLinux, if ***log.target*** has the special value of *syslog*, the log data will be sent to the system SYSLOG facility. The application identifier is ***acsp-cca*** and is logged with *user facility* priority. The table below shows the mapping from ACSP Client log levels to *syslog levels*

ACSP Level	**Syslog Level**
FATAL	Emergency (emerg)
ERROR	Error (err)
WARN	Warning (warn)
INFO	Info (info)
DEBUG	Debug (debug)
TRACE	Debug (debug)

The special values of **stderr** and **stdout** will send the log output to stderr and stdout, respectively.

In addition, the value **server** is reserved for internal use and should not be used.

To enable log rotation based on time the file name may contain date/time formatting fields (%Y, %m %d, %F, %H, %M %S, %T) as accepted by the C library function strftime. When the name of the log file changes the logfile is closed and the new one is opened.

Note: Log rotation is not supported on IBM i.

Syntax

log.target = file

Examples

log.target = C:\acspc.log
log.target = /var/log/acspc.log
log.target = syslog
log.target = stderr
log.target = stdout

The following applies only to the IBM System i platform

log.target = qacsp/log(log)

9.32 SSL/TLS Local Context

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X		X	X	X

The **ssl.use.local.context** is used to decide whether the ACSP Client should establish a server connection using a local SSL context. If this property is not defined or set to **false,** the default JVM defined SSL context is used.

The ***ssl.use.local.context*** setting influences the way the SSL context is established. If it is set to **true,** the context is established by using the keystores that are set by the *ssl.keyStore* and the *ssl.trustStore* alone. The keystores which are provided by these properties, are therefore NOT shared with the settings that are provided by the *javax.net.ssl.keyStore* and the *javax.net.ssl.trustStore, both of* which are global for the JVM.

This property therefore makes it easier to use the ACSP Client within a Java application server which shares a common JVM, for example GlassFish Application Server.

Use **ssl.use.local.context=true** to enable use of local SSL context.

Syntax

ssl.use.local.context = true **|** **<u>false</u>**

9.33 SSL/TLS Keystore (Java)

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	X

The **ssl.keyStore** parameter is used to specify the keystore to use for private keys that are used for SSL/TLS authentication.

The keystore is a Java keystore (JKS) which should contain the public and private keys for the peer, client or server, itself. The private key is used during protocol negotiation to mask the symmetric secret key for encryption, when they are exchanged.

NOTE: Configuration parameters are case sensitive.

Syntax

ssl.keyStore = path/file

Example

```
ssl.keyStore = config/sample.ks
ssl.keyStorePassword = zaqwsx
```

Note: The *ssl.keyStore* is shadowed by the *javax.net.ssl.keyStore* if this property is not set. A message is written to the log if the above is done.

9.34 SSL/TLS Keystore Password (Java)

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	X

The **ssl.keyStorePassword** parameter is used to specify the password of the java keystore, specified by the **ssl.keyStore** parameter.

The keystore is a Java keystore (JKS) which holds the public and private keys for the peer itself. The private key is used during protocol negotiation to mask the symmetric secret key for encryption, when they are exchanged.

Note: Configuration parameters are case sensitive.

Syntax

ssl.keyStorePassword = <password>

Example

```
ssl.keyStore = config/sample.ks
ssl.keyStorePassword = zaqwsx
```

Note: The *ssl.keyStorePassword* is shadowed by the *javax.net.ssl.keyStorePassword.* A message is written to the log if this is done.

9.35 SSL/TLS Keystore (C)

Related Software Package	ACSP CCA Client, ACSP PKCS#11 Client

Platform or Operating System							API	
AIX	Linux	OS/400	Windows	zLinux	z/OS	Generic (Java)	CCA	PKCS#11
X	X		X	X			X	X

The **ssl.keystore** parameter specifies the location of a PKCS#12 keystore containing a private key and the corresponding client certificate.

It is recommended that the keystore path that is used, is absolute. A relative path will work, but it will be relative to the work directory of the application, not the directory where ACSP Clien has been installed.

The keystore path can also be specified by an environment variable named
ACSP_CLIENT_KEYSTORE
Please look in the "Setting the Environment Variables" on page 28 for information about this.

Note: If this parameter is specified, the parameters **ssl.private.key** and **ssl.certificate** are ignored.

Syntax

ssl.keystore = path/file

Examples

```
ssl.keystore = /opt/ibm/acsp-c-client/key.p12
ssl.keystore = c:\Program Files\IBM\IBM ACSP CCA Client\config\client1.p12
ssl.keystore.password=zaqwsx
```

9.36 SSL/TLS Keystore Password (C)

Related Software Package	ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X			X	X

The **ssl.keystore.password** parameter specifies the password of a PKCS#12 keystore which has been defined by the **ssl.keystore** parameter.

The **ssl.keystore.password** is normally specified together with the **ssl.keystore** parameter, but the password can sometimes be specified by either an environment variable (ACSP_CLIENT_KEYSTORE_PW) or via a program in the calling application. Please look in the "Setting the Environment Variables" on page 28 or in "ACSP-2115 IBM Advanced Cryptographic Service Provider - ACSP CCA and PKCS#11 Programmers Guide " for more information.

Syntax

ssl.keystore.password = <password>

Examples

```
ssl.keystore = /opt/ibm/acsp-c-client/key.p12
ssl.keystore = c:\Program Files\IBM\IBM ACSP CCA Client\config\client1.p12
ssl.keystore.password=zaqwsx
```

9.37 SSL/TLS Keystore Password File (C and Java)

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X			X	X

The **ssl.keystore.password.file** parameter specifies the file that contains the password of a PKCS#12 keystore which has been defined by the **ssl.keystore** parameter.

The **ssl.keystore.password.file** parameter prevents leaking of the password when copying the configuration file - e.g. for support purposes. It also allows tighter access control permissions on the password file than on the configuration file.

Syntax

ssl.keystore.password.file = path/file

Examples

```
ssl.keystore.password.file = /opt/ibm/acsp-cca-client/config/acsp.password
```

9.38 SSL/TLS Truststore (Java)

Related Software Package	ACSP Java Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X	X	X	X	

The ***ssl.trustStore*** parameter is used to specify the keystore to use for trusted certificates.

The truststore is a Java keystore (JKS) and must contain the approved root certificates, which will ultimately confirm or reject the authentications from peers.

Note: Configuration parameters are case sensitive.

Syntax

ssl.trustStore = path/file

Example

```
ssl.trustStore = config/sample.ts
```

Note: The *ssl.trustStore* is shadowed by the *javax.net.ssl.trustStore* if this property is not set. A message is written to the log if this is done.

9.39 SSL/TLS Truststore (C)

Related Software Package	ACSP CCA Client, ACSP PKCS#11 Client
Platform or Operating System	**API**

AIX	Linux	OS/400	Windows	zLinux	z/OS	Generic (Java)	CCA	PKCS#11
X	X		X	X			X	X

The ***ssl.truststore*** parameter points to a file containing the list of server certificates for the servers that the client should trust when establishing SSL/TLS connections. This file must be a file in the PEM format and must contain one or more X.509 server certificates.

It is recommended that the truststore path that is entered is absolute. A relative path will work, but it will be relative to the work directory of the application, not the directory where ACSP Client has been installed.

Syntax

***ssl.truststore* =** path/file

Examples

ssl.truststore = /opt/ibm/acsp-cca-client/config/server.pem
ssl.truststore = c:\Program Files\IBM\IBM ACSP CCA Client\config\server.pem

9.40 SSL/TLS Cipher Suites

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X	X	X	X	X

Cipher suites can be specified to make the ACSP Server accept or use specific sets of cryptographic algorithms for SSL/TLS handshaking by using the **ssl.ciphers** parameter.

If specified, provide a space or comma separated list of acceptable cipher suites to limit the ciphers eligible for handshake negotiation. This will prevent connections from clients not supporting the specified cipher suites, and ensure a minimum security level.

Syntax

***ssl.ciphers* =** <suitename> [, <suitename>] […]

The **<suitename>** is for example one or more of the names for the Java Cipher suites that are listed in the table below.

Example

ssl.ciphers =TLS_RSA_WITH_AES_128_CBC_SHA

In the ACSP Server and Java Client the available cipher-suites depend on the available security providers and SSL/TLS protocols.

OpenSSL Cipher Suite Name	Java Cipher Suite Equivalent
AES128-GCM-SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256
AES128-SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256
AES128-SHA	TLS_RSA_WITH_AES_128_CBC_SHA
AES256-GCM-SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384
AES256-SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256
AES256-SHA	TLS_RSA_WITH_AES_256_CBC_SHA
DHE-DSS-AES128-GCM-SHA256	TLS_DHE_DSS_WITH_AES_128_GCM_SHA256

DHE-DSS-AES128-SHA256	TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
DHE-DSS-AES256-GCM-SHA384	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
DHE-DSS-AES256-SHA256	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
DHE-RSA-AES128-GCM-SHA256	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
DHE-RSA-AES128-SHA256	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
DHE-RSA-AES256-GCM-SHA384	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
DHE-RSA-AES256-SHA256	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
ECDH-ECDSA-AES128-GCM-SHA256	TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
ECDH-ECDSA-AES128-SHA256	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
ECDH-ECDSA-AES256-GCM-SHA384	TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
ECDH-ECDSA-AES256-SHA384	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
ECDHE-ECDSA-AES128-GCM-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
ECDHE-ECDSA-AES128-SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
ECDHE-ECDSA-AES256-GCM-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
ECDHE-ECDSA-AES256-SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
ECDHE-RSA-AES128-GCM-SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
ECDHE-RSA-AES128-SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
ECDHE-RSA-AES256-GCM-SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
ECDHE-RSA-AES256-SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
ECDH-RSA-AES128-GCM-SHA256	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
ECDH-RSA-AES128-SHA256	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
ECDH-RSA-AES256-GCM-SHA384	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
ECDH-RSA-AES256-SHA384	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384

The full list of ciphers available in the ACSP CCA Client is described at https://www.openssl.org/docs/man1.0.1/apps/ciphers.html

For the list of ciphers available in the Java client, refer to the documentation for the JVM

The default value is 'TLS_RSA_WITH_AES_128_CBC_SHA' (Server/Java Client) and 'AES128-SHA' (CCA Client)

The default cipher suite has been chosen because this cipher suite contains a reasonable security level, and is available in the JVM default providers.

Note: For the clients, the specified ciphers must match at least some of the ciphers specified for the server.

Note: On i5/OS this parameter has been deprecated. Use the Digital Certificate Manager to configure cipher-suites instead.

See also "SSL/TLS Protocols" on page 67

9.41 SSL/TLS Protocols

Related Software Package	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X		X	X	X	X	X	X

The protocols to use for TLS are specified independently of the cipher suite to use. Using the **ssl.protocols** parameter, a subset of the supported protocols can be defined as available.

Valid protocol values are: **TLSv1**, **TLSv1.1**, and **TLSv1.2.**

If this parameter is not otherwise specified, the default value is used.

For the ACSP Server and Java client, the parameter accepts a comma-separated list of values. This list specifies all acceptable protocols, and lets the client and server negotiate the preferred protocol.

For the ACSP CCA Client, the parameter accepts only a single value. If a comma-separated list is specified, the ACSP CCA Client will issue a warning and use only the first value in the list.

Setting this parameter limits the protocols that are eligible for negotiation. This will prevent connections to peers that do not support the specified protocols, and will thus ensure a minimum security level.

The default value is *TLSv1.2*.

Syntax (Java Client)

ssl.protocols = <protocol name> [, <protocol name>] […]

Example 1

```
ssl.protocols =  TLSv1.1, TLSv1.2
```

Syntax (CCA Client and PKCS#11 Client)

ssl.protocols = <protocol name>

Example 2

```
ssl.protocols =  TLSv1.2
```

Note: For the clients, the specified protocols must match at least some of the protocols specified for the ACSP Server.

Note: This parameter is not supported on i5/OS. Use the Digital Certificate Manager to configure protocols instead.

See also “SSL/TLS Cipher Suites” on page 66

9.42 SSL/TLS Application ID

Related Software Package	ACSP CCA Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
		X					X	

The ***ssl.gskit.application.id*** property specifies an application ID to be used for the ACSP Client when running on the i5/OS platform. The application ID is used to access keys in the DCM (Digital Certificate Management)..

Note: This setting is used only on the i5/OS platform using GSKit.

Syntax

***ssl.gskit.application.id* =** appl-name

Example

ssl.gskit.application.id = ACSP

9.43 TCP No-Delay

Applies to			ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client					
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	X

The ***tcp.nodelay*** parameter controls the use of Nagle's algorithm for handling data in TCP connections.

If Nagle's algorithm is enabled, data may be buffered for a short time, while waiting for more data to be sent. This can increase TCP package size and reduce the overhead. It makes better use of available bandwidth, at the cost of a longer latency.

If Nagle's algorithm is disabled, data is sent without waiting for more data. This can reduce the latency of data, but can result in smaller TCP packages and greater overhead.

The default is ***true***, resulting in Nagle's algorithm being disabled.

Please refer to http://en.wikipedia.org/wiki/Nagle's_algorithm for the details about the Nagle algorithm.

Syntax

tcp.nodelay = *true* | *false*

9.44 TCP Keep Alive

Applies to	ACSP Java Client, ACSP CCA Client, ACSP PKCS#11 Client							
Platform or Operating System							**API**	
AIX	**Linux**	**OS/400**	**Windows**	**zLinux**	**z/OS**	**Generic (Java)**	**CCA**	**PKCS#11**
X	X	X	X	X	X	X	X	X

The ***tcp.keepalive*** parameter can be used to enable or disable TCP keep alive.

TCP keep alive is a feature of TCP which can help keep a connection alive during idle periods, when no other data is sent. The TCP keep alive algorithm will regularly send small probes, tcp packages that contain no data, to the connected peer. If a connected peer fails to acknowledge the probe, the probe will be retried a few times and ultimately the connection will be considered broken.

Firewalls and other networking equipment will sometimes break connections that appear idle or unused. The broken connection will behave as if a node, that is a gateway, router or switch, on the network path has failed. Because the server waits for data from a connection, the server will not detect the broken connection. The server will therefore continue to keep resources allocated for the connection. Using TCP keep alive, the connections appear active, and if they are broken, the server will detect the break and close the connection, freeing resources in the process.

The effect of the keep alive probes is to ensure that the connection appears active and therefore kept alive by firewalls.

The frequency at which TCP keep alive packages are sent depends on the platform that an application is running on. It is not uncommon for Operating Systems to have a two hour interval between keep alive probes.

For TCP keep alive to be effective, firewalls between ACSP Clients and ACSP Servers **must** be configured to allow connections to be idle for a longer time than the interval for the platform specific TCP keep alive probe.

There is no gain in using TCP keep alive on both server and client, although it does no harm to have TCP keep alive enabled on both ends of the connection.

The only effect of using TCP keep alive on the client is to help keep connections alive. Clients will detect broken connections on their own, regardless of whether TCP keep alive is enabled, and it will do so faster than TCP keep alive.

TCP keep alive has been enabled by default in the ACSP Server and in all ACSP Clients.

Syntax

tcp.keepalive = *true* | *false*